Chapter 8: Basic Cryptography

- Classical Cryptography
- Public Key Cryptography
- Cryptographic Checksums

What is a Recommended Public Key Encryption Algorithm?

A. SHA-1

B. RSA

C. AES

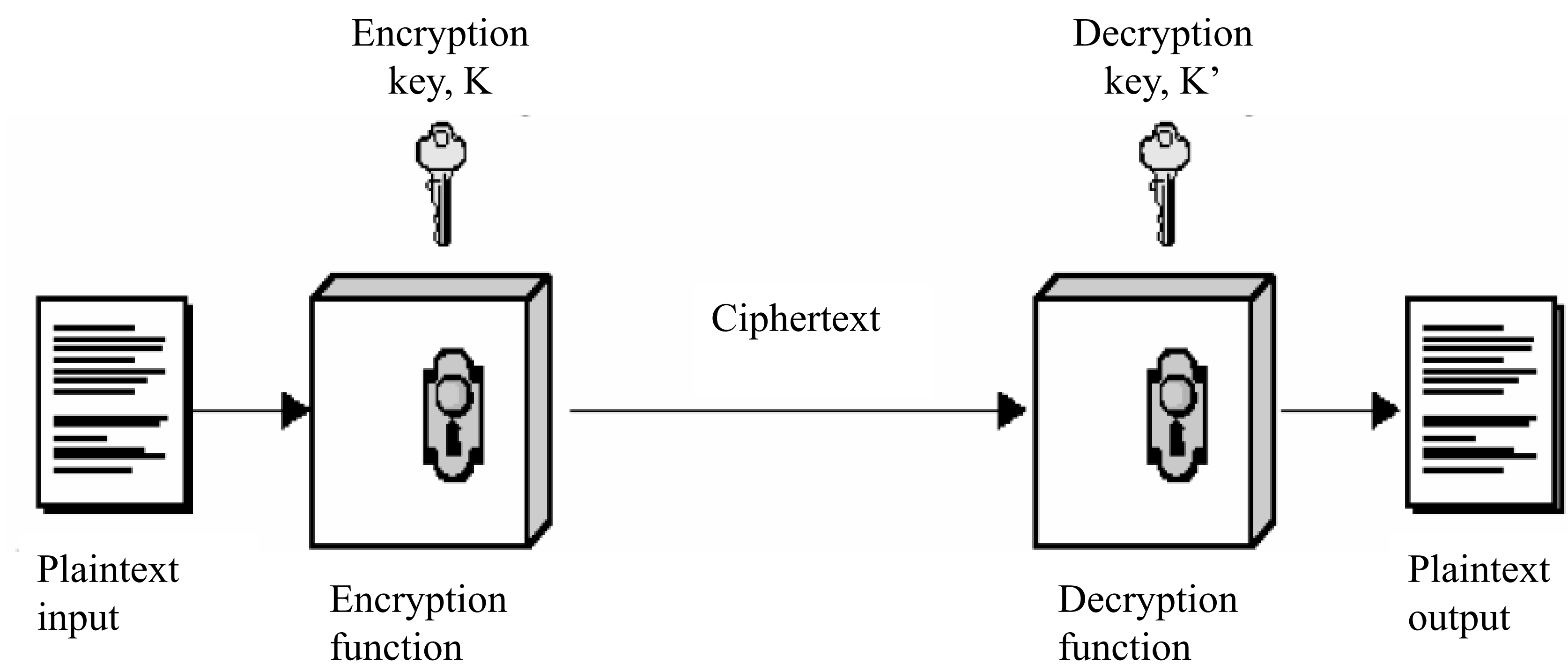
D. MD5

E. None of the above

Overview

- Cryptosystem
- Classical (symmetric) Cryptography
  - Rail-Fence Cipher
  - Cæsar cipher
  - DES
  - AES
- Public Key (asymmetric) Cryptography
  - RSA
- Cryptographic Checksums

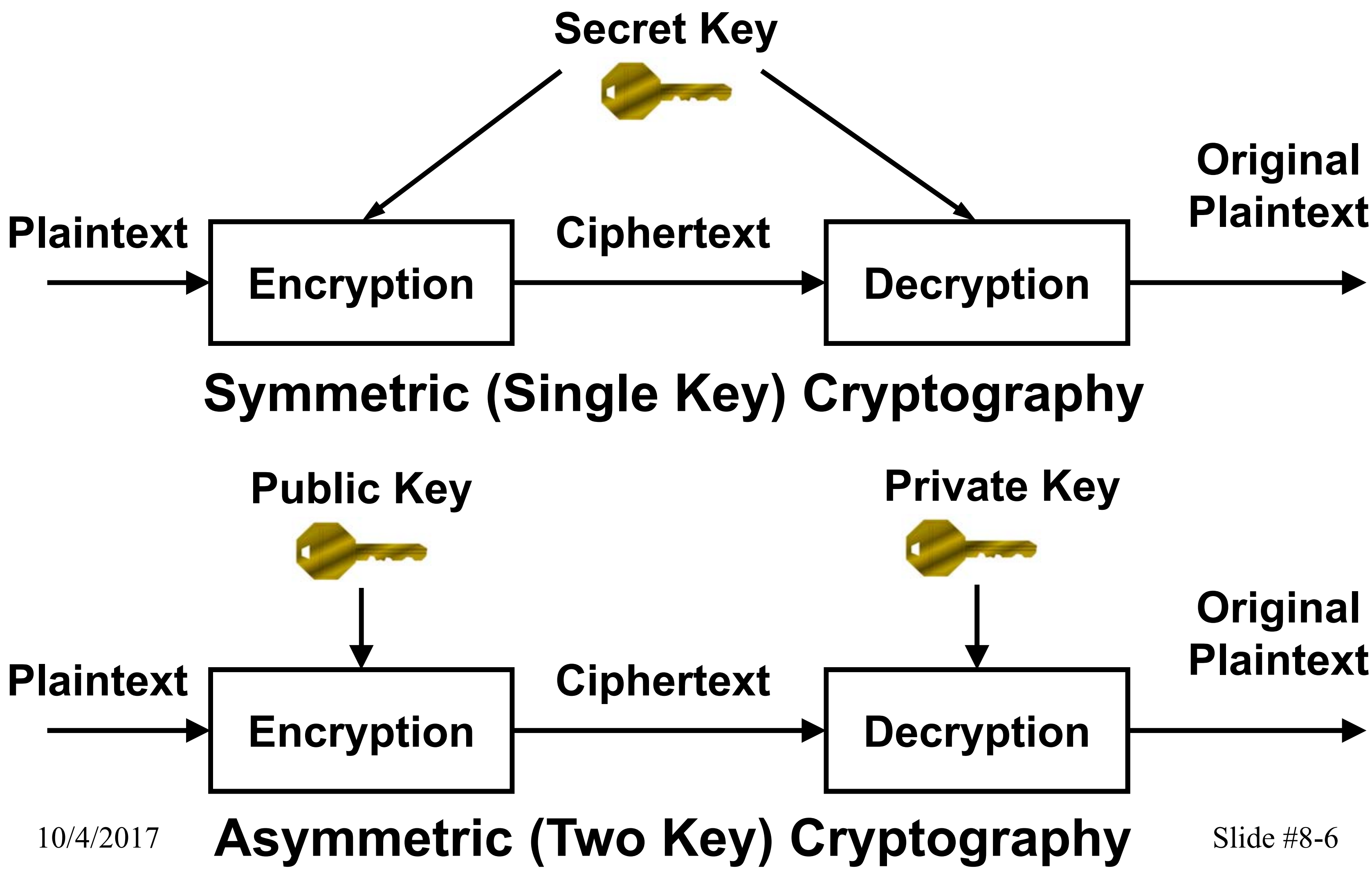
General Cipher Model



Cryptosystem

- Quintuple ($\mathcal{E}$, $\mathcal{D}$, $\mathcal{M}$, $\mathcal{K}$, $\mathcal{C}$)
  - $\mathcal{M}$ set of plaintexts
  - $\mathcal{K}$ set of keys
  - $\mathcal{C}$ set of ciphertexts
  - $\mathcal{E}$ set of encryption functions e: $\mathcal{M} \times \mathcal{K} \rightarrow \mathcal{C}$
  - $\mathcal{D}$ set of decryption functions d: $\mathcal{C} \times \mathcal{K} \rightarrow \mathcal{M}$

Comparison of Symmetric and Asymmetric Encryption



Example: Cæsar Cipher

Key: 3

Encryption function E_3:

in:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
out:	DEFGHIJKLMNOPQRSTUVWXYZABC

Decryption function D_3:

in:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
out:	XYZABCDEFGHIJKLMNOPQRSTUVW

Plaintext: HELLO WORLD

Ciphertext: KHOOR ZRUOG

Example: Cæsar Cipher

$\mathcal{M}$ = { sequences of letters, represented as 0..25 }

$\mathcal{K} = \{ i \mid i$ is an integer and $0 \le i \le 25 \}$

$\mathcal{E} = \{ E_k \mid k \in \mathcal{K}$ and for all letters m,

$E_k(m) = (m + k) \bmod 26 \}$

$\mathcal{D} = \{ D_k \mid k \in \mathcal{K}$ and for all letters c,

$D_k(c) = (26 + c - k) \bmod 26 \}$

$\mathcal{C} = \mathcal{M}$

Attacks

- Opponent whose goal is to break cryptosystem is the *adversary*
  - Assume adversary knows algorithm used, but not key
- Three types of attacks:
  - *ciphertext only*: adversary has only ciphertext; goal is to find plaintext, possibly key
  - *known plaintext*: adversary has ciphertext, corresponding plaintext; goal is to find key
  - *chosen plaintext*: adversary may supply plaintexts and obtain corresponding ciphertext; goal is to find key

Basis for Attacks

- Mathematical attacks
  - Based on analysis of underlying mathematics
- Statistical attacks
  - Make assumptions about the distribution of letters, pairs of letters (digrams), triplets of letters (trigrams), *etc*.
    - Called *models of the language*
  - Examine ciphertext, correlate properties with the assumptions.

Classical Cryptography

- Sender, receiver share common key
  - Keys may be the same, or trivial to derive from one another
  - Sometimes called *symmetric cryptography*
- Two basic types
  - Transposition ciphers
  - Substitution ciphers
  - Combinations are called *product ciphers*

Transposition Cipher

- Rearrange letters in plaintext to produce ciphertext
- Example (Rail-Fence Cipher)
  - Plaintext is `HELLO WORLD`
  - Rearrange as

```
HLOOL
ELWRD
```

  - Ciphertext is `HLOOL ELWRD`

Attacking the Cipher

- Basic idea: permutation does not change the frequency of plaintext characters
- Anagramming
  – If 1-gram frequencies in the ciphertext match English frequencies, but other *n*-gram frequencies do not, probably transposition
  – Rearrange letters to form *n*-grams with highest frequencies

Example

- Ciphertext: `HLOOLELWRD`
- Frequencies of 2-grams beginning with H in English
  - HE 0.0305
  - HO 0.0043
  - HL, HW, HR, HD < 0.0010
- Frequencies of 2-grams ending in H
  - WH 0.0026
  - EH, LH, OH, RH, DH ≤ 0.0002
- Implies E follows H

Example

- Arrange so the H and E are adjacent

```
                              HE
                              LL
HLOOLELWRD      →             OW
                              OR
                              LD
```

- Read off across, then down, to get original plaintext

Substitution Ciphers

- Change characters in plaintext to produce ciphertext
- Example (Cæsar cipher)
  - Plaintext is `HELLO WORLD`
  - Change each letter to the third letter following it (X goes to A, Y to B, Z to C)
    - Key is 3, usually written as letter 'D'
  - Ciphertext is `KHOOR ZRUOG`

Attacking the Cipher

- Exhaustive search
  - If the key space is small enough, try all possible keys until you find the right one
  - Cæsar cipher has 26 possible keys
- Statistical analysis
  - Compare to 1-gram model of English

Statistical Attack

- Compute frequency of each letter in ciphertext:

G	0.1	H	0.1	K	0.1	O	0.3
R	0.2	U	0.1	Z	0.1		

- Apply 1-gram model of English
  - Frequency of characters (1-grams) in English is on next slide

Character Frequencies

a	0.080	h	0.060	n	0.070	t	0.090
b	0.015	i	0.065	o	0.080	u	0.030
c	0.030	j	0.005	p	0.020	v	0.010
d	0.040	k	0.005	q	0.002	w	0.015
e	0.130	l	0.035	r	0.065	x	0.005
f	0.020	m	0.030	s	0.060	y	0.020
g	0.015					z	0.002

Statistical Analysis

- $f(c)$ frequency of character c in ciphertext
- $\varphi(i)$ correlation of frequency of letters in ciphertext with corresponding letters in English, assuming key is i

G	0.1	H	0.1	K	0.1	O	0.3
R	0.2	U	0.1	Z	0.1		

  - $\varphi(i) = \Sigma_{0 \leq c \leq 25} f(c)p(c - i)$ so here,
    $\varphi(i) = 0.1p(6 - i) + 0.1p(7 - i) + 0.1p(10 - i) + 0.3p(14 - i) + 0.2p(17 - i) + 0.1p(20 - i) + 0.1p(25 - i)$
    - $p(x)$ is frequency of character x in English
- The correlation $\varphi(i)$ should be a maximum when the key i translates the ciphertext into English

Correlation: $\varphi(i)$ for $0 \leq i \leq 25$

i	**φ(*i*)**	***i***	**φ(*i*)**	***i***	**φ(*i*)**	***i***	**φ(*i*)**
0	0.0482	7	0.0442	13	0.0520	19	0.0315
1	0.0364	8	0.0202	14	0.0535	20	0.0302
2	0.0410	9	0.0267	15	0.0226	21	0.0517
3	0.0575	10	0.0635	16	0.0322	22	0.0380
4	0.0252	11	0.0262	17	0.0392	23	0.0370
5	0.0190	12	0.0325	18	0.0299	24	0.0316
6	0.0660					25	0.0430

The Result

- Most probable keys, based on φ:
  - $i = 6$, $\varphi(i) = 0.0660$
    - plaintext `EBIIL TLOLA`
  - $i = 10$, $\varphi(i) = 0.0635$
    - plaintext `AXEEH PHKEW`
  - $i = 3$, $\varphi(i) = 0.0575$
    - plaintext `HELLO WORLD`
  - $i = 14$, $\varphi(i) = 0.0535$
    - plaintext `WTAAD LDGAS`
- Only English phrase is for $i = 3$
  - That's the key (3 or 'D')

DES: History

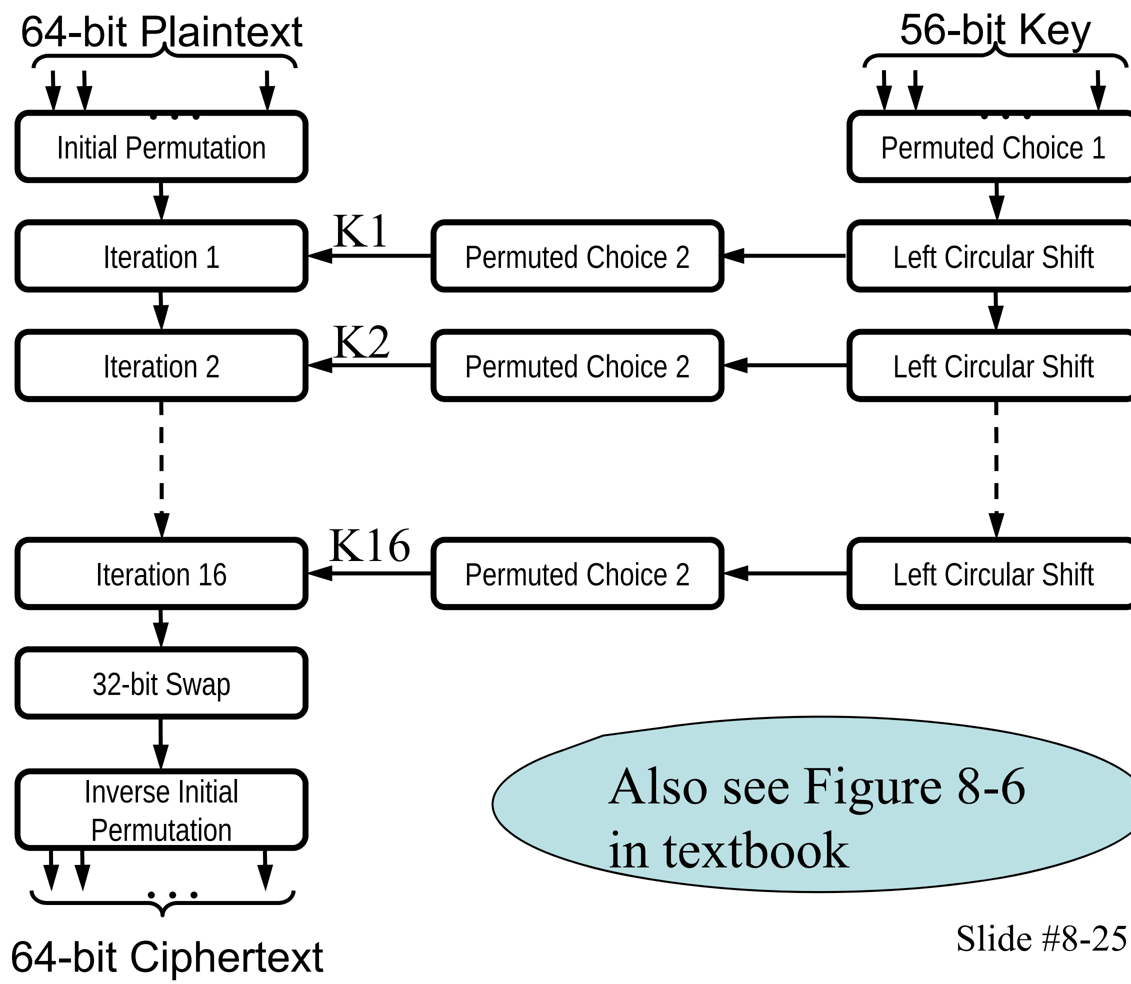
- The Data Encryption Standard (DES) was developed in the 1970s by the **National Bureau of Standards** (NBS)with the help of the **National Security Agency (NSA).**
- Its purpose is to provide a standard method for protecting *sensitive* commercial and unclassified data.
- IBM created the first draft of the algorithm, calling it **LUCIFER** with a 128-bit key.
- DES officially became a federal standard in November of 1976.
- Has been widely adopted.

Overview of the DES

- A block cipher:
  - encrypts blocks of 64 bits using a 56 bit key
  - outputs 64 bits of ciphertext
- A product cipher
  - basic unit is the bit
  - performs both substitution and transposition (permutation) on the bits
- Cipher consists of 16 rounds (iterations) each with a round key generated from the user-supplied key

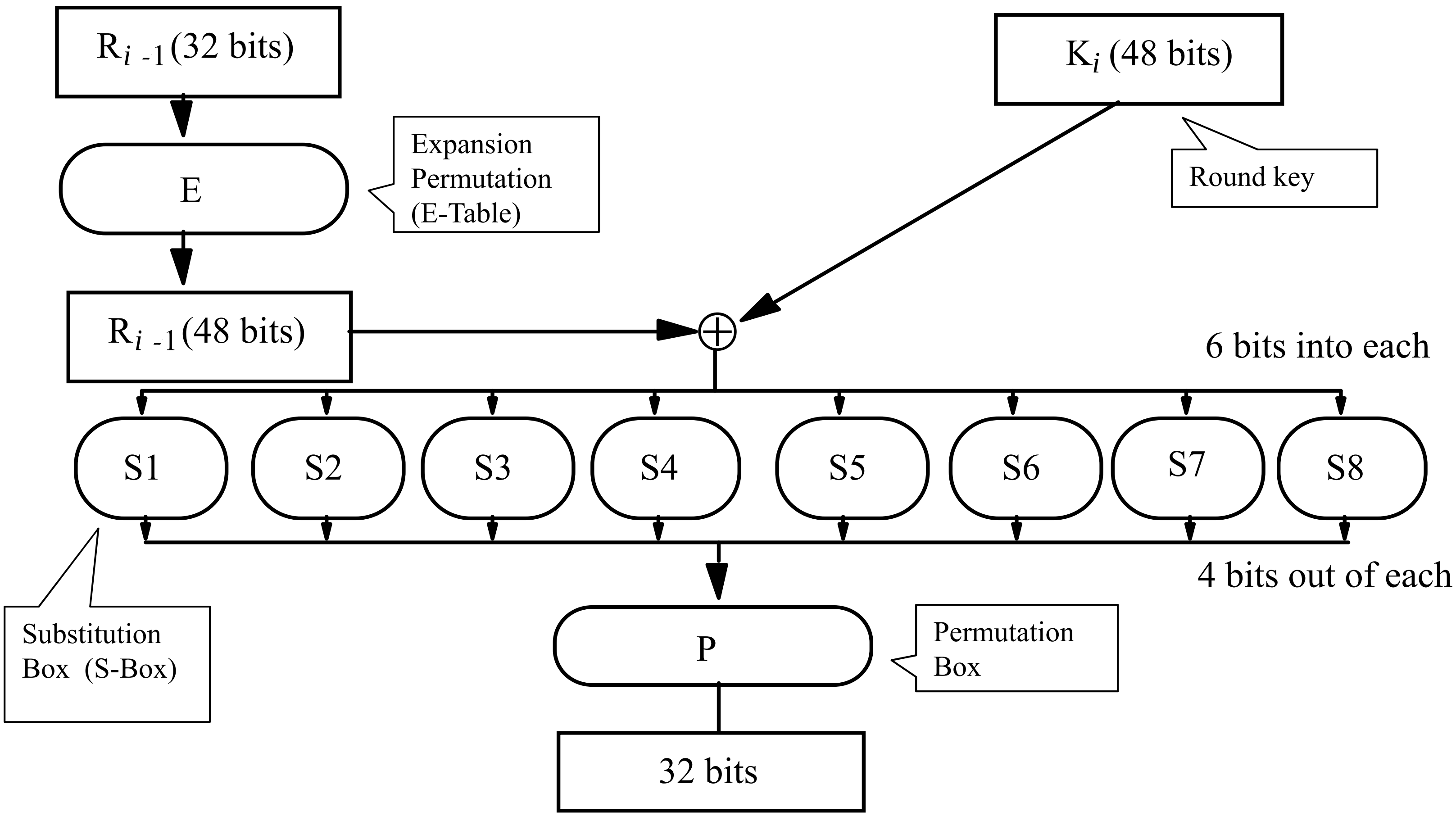
DES Encipherment

- A basic process in enciphering a 64-bit data block and a 56-bit key using the DES consists of:
  - An initial permutation (IP)
  - 16 rounds of a complex key dependent calculation **f**
  - A final permutation, being the inverse of IP



Also see Figure 8-6 in textbook

The *f* Function



Controversy

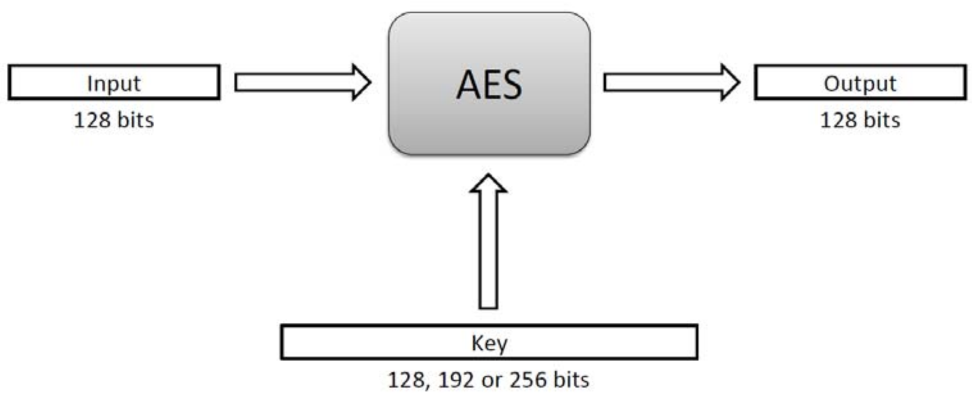
- Considered too weak
  - Key length of 56 bits is too short
  - Diffie, Hellman said in a few years technology would allow DES to be broken in days
    - Design using 1999 technology published
  - Design decisions not public
    - S-boxes may have backdoors, or inherent weaknesses?

Differential Cryptanalysis Attacks against DES

- A chosen ciphertext attack
  - Requires 2^{47} plaintext, ciphertext pairs
- Revealed several properties
  - Small changes in S-boxes reduce the number of pairs needed
  - Making every bit of the round keys independent does not impede attack
- Linear cryptanalysis improves result
  - Requires 2^{43} plaintext, ciphertext pairs

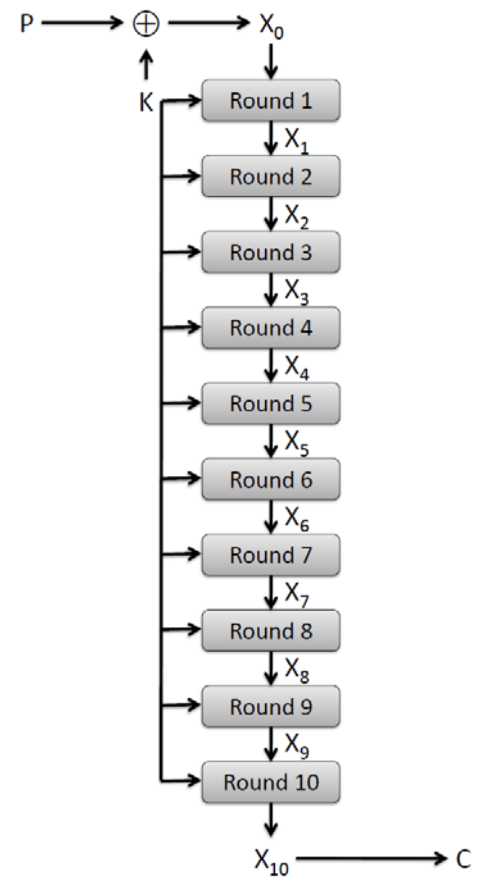
The Advanced Encryption Standard (AES)

- In 1997, the U.S. National Institute for Standards and Technology (NIST) put out a public call for a replacement to DES.
- It narrowed down the list of submissions to five finalists, and ultimately chose an algorithm that is now known as the **Advanced Encryption Standard** (**AES**).
- AES is a block cipher that operates on 128-bit blocks. It is designed to be used with keys that are 128, 192, or 256 bits long, yielding ciphers known as AES-128, AES-192, and AES-256.



AES Round Structure

- The 128-bit version of the AES encryption algorithm proceeds in ten rounds.
- Each round performs an invertible transformation on a 128-bit array, called **state**.
- The initial state X_0 is the XOR of the plaintext P with the key K:
- $X_0 = P \text{ XOR } K$.
- Round i (i = 1, …, 10) receives state X_{i-1} as input and produces state X_i.
- The ciphertext C is the output of the final round: $C = X_{10}$.



AES Rounds

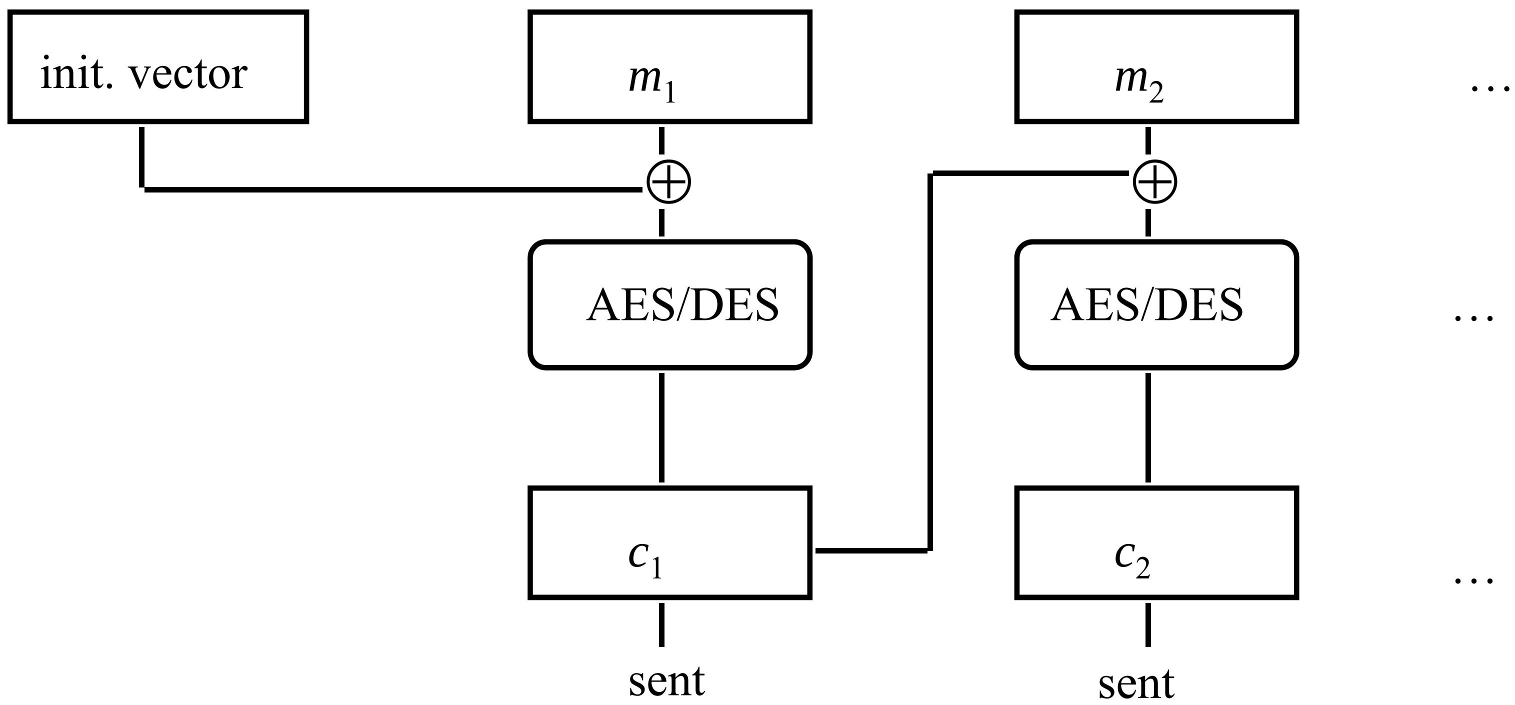
- Each round is built from four basic steps:

1. **SubBytes step**: an S-box substitution step
2. **ShiftRows step**: a permutation step
3. **MixColumns step**: a matrix multiplication step
4. **AddRoundKey step**: an XOR step with a **round key** derived from the 128-bit encryption key

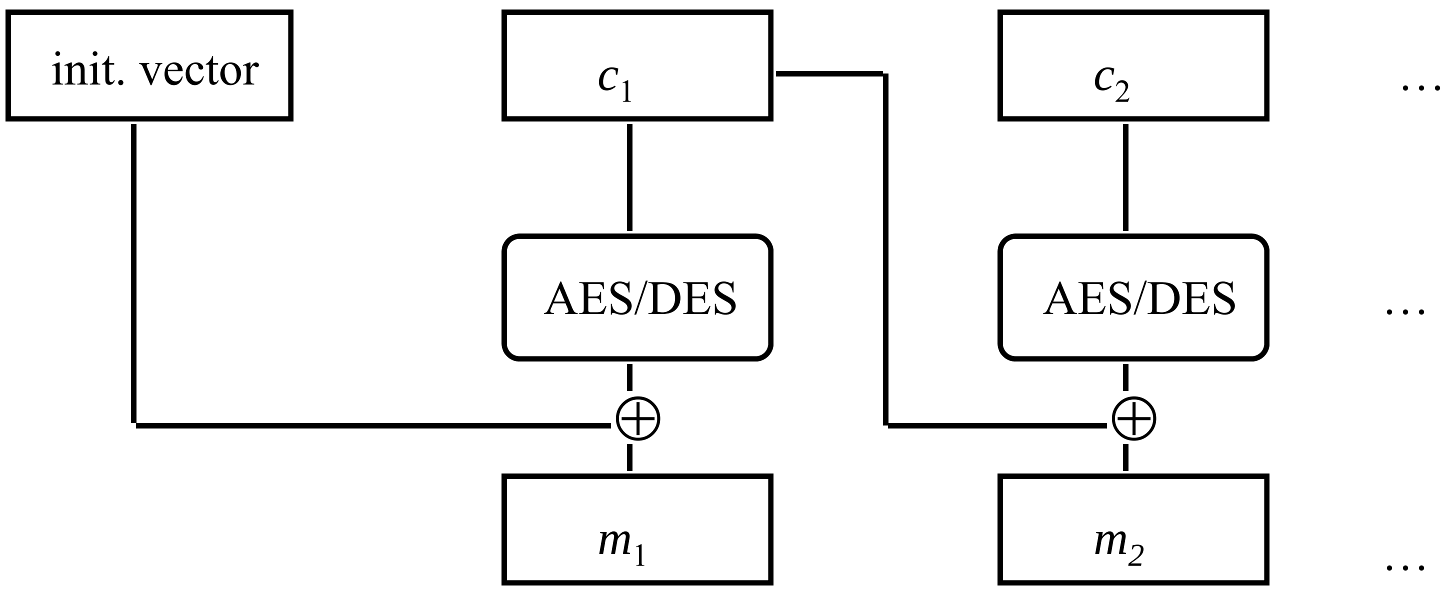
AES/DES Usage Modes

- Electronic Code Book Mode (ECB)
  - Encipher each block independently
  - The vanilla AES/DES
- Cipher Block Chaining Mode (CBC)
  - XOR each block with previous ciphertext block
  - Requires an initialization vector for the first one

CBC Mode Encryption



CBC Mode Decryption

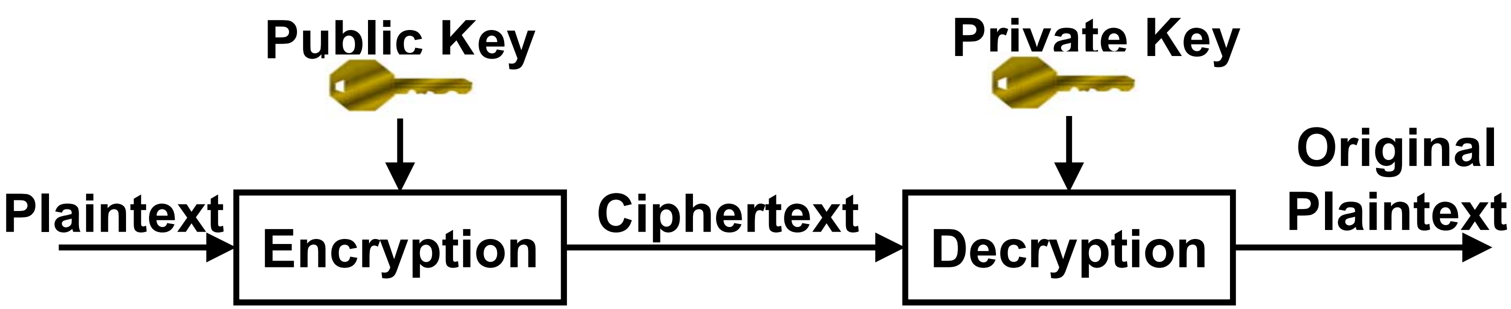


Other AES/DES Usage Modes

- Encrypt-Decrypt-Encrypt Mode (2 keys: k, k')
  - $c = \text{AES}_k(\text{AES}_{k'}^{-1}(\text{AES}_k(m)))$, or
  - $c = \text{DES}_k(\text{DES}_{k'}^{-1}(\text{DES}_k(m)))$
- Encrypt-Encrypt-Encrypt Mode (3 keys: k, k', k'')
  - $c = \text{AES}_k(\text{AES}_{k'}(\text{AES}_{k''}(m)))$, or
  - $c = \text{DES}_k(\text{DES}_{k'}(\text{DES}_{k''}(m)))$

Public Key Cryptography

- Two keys
  - *Private key* known only to individual
  - *Public key* available to anyone
    - Public key, private key inverses



Requirements

1. It must be computationally easy to encipher or decipher a message given the appropriate key

2. It must be computationally infeasible to derive the private key from the public key

3. It must be computationally infeasible to determine the private key from a chosen plaintext attack

RSA

- Exponentiation cipher
- Relies on the difficulty of determining the number of numbers relatively prime to a large integer *n*

Facts About Numbers

- Prime number $\boldsymbol{p}$:
  - $\boldsymbol{p}$ is an integer
  - $\boldsymbol{p} \geq 2$
  - The only divisors of $\boldsymbol{p}$ are 1 and $\boldsymbol{p}$
- Examples
  - 2, 7, 19 are primes
  - –3, 0, 1, 6 are not primes
- Prime decomposition of a positive integer $\boldsymbol{n}$:

$$\boldsymbol{n} = \boldsymbol{p}_1^{\boldsymbol{e}_1} \times \ldots \times \boldsymbol{p}_{\boldsymbol{k}}^{\boldsymbol{e}_{\boldsymbol{k}}}$$

- Example:
  - $200 = 2^3 \times 5^2$

Fundamental Theorem of Arithmetic

The prime decomposition of a positive integer is unique

Greatest Common Divisor

- The greatest common divisor (GCD) of two positive integers $\boldsymbol{a}$ and $\boldsymbol{b}$, denoted gcd($\boldsymbol{a}$, $\boldsymbol{b}$), is the largest positive integer that divides both $\boldsymbol{a}$ and $\boldsymbol{b}$
- The above definition is extended to arbitrary integers
- Examples:

  gcd(18, 30) = 6 gcd(0, 20) = 20

  gcd(–21, 49) = 7
- Two integers a and b are said to be relatively prime if

$$\gcd(\boldsymbol{a}, \boldsymbol{b}) = 1$$

- Example:
  - Integers 15 and 28 are relatively prime

Modular Arithmetic

- Modulo operator for a positive integer $\boldsymbol{n}$

  $$\boldsymbol{r} = \boldsymbol{a} \bmod \boldsymbol{n}$$

  equivalent to

  $$\boldsymbol{a} = \boldsymbol{r} + \boldsymbol{kn}$$

  and

  $$\boldsymbol{r} = \boldsymbol{a} - \lfloor \boldsymbol{a}/\boldsymbol{n} \rfloor \boldsymbol{n}$$

- Example:

  $29 \bmod 13 = 3$ $\quad$ $13 \bmod 13 = 0$ $\quad$ $-1 \bmod 13 = 12$

  $29 = 3 + 2\times 13$ $\quad$ $13 = 0 + 1\times 13$ $\quad$ $12 = -1 + 1\times 13$

- Modulo and GCD:

  $$\gcd(\boldsymbol{a}, \boldsymbol{b}) = \gcd(\boldsymbol{b}, \boldsymbol{a} \bmod \boldsymbol{b})$$

- Example:

  $\gcd(21, 12) = 3$ $\quad$ $\gcd(12, 21 \bmod 12) = \gcd(12, 9) = 3$

Euclid's GCD Algorithm

- Euclid's algorithm for computing the GCD repeatedly applies the formula

  gcd($\boldsymbol{a}$, $\boldsymbol{b}$) = gcd($\boldsymbol{b}$, $\boldsymbol{a}$ mod $\boldsymbol{b}$)

- Example
  - gcd(412, 260) = 4

Algorithm ***EuclidGCD***(***a, b***)
  Input integers ***a*** and ***b***
  Output gcd(***a***, ***b***)

  if ***b*** = 0
    return ***a***
  else
    return ***EuclidGCD***(***b, a*** mod ***b***)

a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0

Multiplicative Inverses (1)

- The residues modulo a positive integer $\boldsymbol{n}$ are the set

  $\boldsymbol{Z_n} = \{0, 1, 2, \ldots, (\boldsymbol{n} - 1)\}$
- Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two elements of $\boldsymbol{Z_n}$ such that

  $\boldsymbol{x} * \boldsymbol{y} \bmod \boldsymbol{n} = 1$

  We say that $\boldsymbol{y}$ is the multiplicative inverse of $\boldsymbol{x}$ in $\boldsymbol{Z_n}$ and we write $\boldsymbol{y} = \boldsymbol{x}^{-1}$
- Example:
  - Multiplicative inverses of the residues modulo 11

$\boldsymbol{x}$	0	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{x}^{-1}$		1	6	4	3	9	2	8	7	5	10

Multiplicative Inverses (2)

Theorem

An element $\boldsymbol{x}$ of $\boldsymbol{Z_n}$ has a multiplicative inverse if and only if $\boldsymbol{x}$ and $\boldsymbol{n}$ are relatively prime

- Example
  - The elements of $\boldsymbol{Z}_{10}$ with a multiplicative inverse are 1, 3, 7, 9

Corollary

If $\boldsymbol{p}$ is prime, every nonzero residue in $\boldsymbol{Z_p}$ has a multiplicative inverse

Theorem

A variation of Euclid's GCD algorithm computes the multiplicative inverse of an element $\boldsymbol{x}$ of $\boldsymbol{Z_n}$ or determines that it does not exist

x	0	1	2	3	4	5	6	7	8	9
x^{-1}		1		7				3		9

Modular Inverse by Extended Euclid's Algorithm

- To test the existence of and compute the inverse of $\boldsymbol{x} \in \boldsymbol{Z_n}$, we execute the extended Euclid's algorithm on the input pair (***n***, ***x***)
- Let (***d***, ***i***, ***j***) = GCD (***n***, ***x***) be the triplet returned
  - $\boldsymbol{d} = \boldsymbol{i}*\boldsymbol{n} + \boldsymbol{j}*\boldsymbol{x}$

  Case 1: $\boldsymbol{d} = 1$

  j is the inverse of ***x*** in $\boldsymbol{Z_n}$

  Case 2: $\boldsymbol{d} > 1$

  x has no inverse in $\boldsymbol{Z_n}$

Extended Euclid's Algorithm

Algorithm GCD(a, b):

  if b = 0, then /*we assume a > b */

    return (a, 1, 0)

  Let q = ⌊a/b⌋

  Let (d, k, m) = GCD(b, a mod b)

  return (d, m, k-m*q)

The floor operator.
E.g., ⌊10/4⌋ = 2

Example: the Inverse of 5 in $\boldsymbol{Z_{96}}$

- Let (***d, i, j***) = GCD (***96, 5***) be the triplet returned
  - $\boldsymbol{d = i*96 + j*5}$

  Case 1: $\boldsymbol{d} = 1$

  $\boldsymbol{j}$ is the inverse of $\boldsymbol{5}$ in $\boldsymbol{Z_n}$

  Case 2: $\boldsymbol{d} > 1$

  $\boldsymbol{5}$ has no inverse in $\boldsymbol{Z_n}$

- What is GCD(96, 5) ?

GCD (96, 5)

a=96, b=5

q=96/5=19

    GCD(5, 96 mod 5) = GCD(5, 1)

        q'=5/1=5

        GCD(1, 5 mod 1) = GCD(1, 0)

            base case: GCD(1,0) = (1,1,0) = (d',k',m')

        return (d',m',k'- m'*q')= (1,0,1-0*5)=(1,0,1)

    GCD(5, 1)=(1, 0, 1)=(d, k, m)

return (d, m, k-m*q) = (1, 1, 0-1*19)=(1, 1, -19)

Example: the Inverse of 5 in $\boldsymbol{Z_{96}}$

- To test the existence of and compute the inverse of $x \in Z_n$, we execute the extended Euclid's algorithm on the input pair (n, x)
- Let (d, i, j) = GCD $(96, 5)$be the triplet returned
  - $d = i*96 + j*5$

  Case 1: $d = 1$

  j is the inverse of 5 in Z_n

  Case 2: $d > 1$

  5 has no inverse in Z_n

- Therefore, GCD(96,5)=(1, 1, -19), so d = 1, i = 1, j = -19
- Because d = 1, inverse exists and it is j or -19.
- -19 mod 96 = 77 because

  -19 = 77 – 96 = 77 + (-1)*96

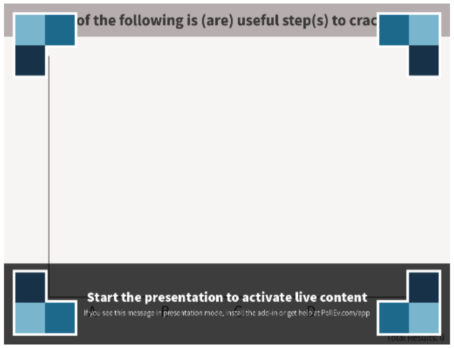
RSA Cryptosystem

- Setup:
  - $n = pq$, with p and q primes
  - e relatively prime to $\phi(n) = (p - 1)(q - 1)$
  - d inverse of e in $Z_{\phi(n)}$
- Keys:
  - Public key: $K_E = (n, e)$
  - Private key: $K_D = d$
- Encryption:
  - Plaintext M in Z_n
  - $C = M^e \bmod n$
- Decryption:
  - $M = C^d \bmod n$

- Example
  - Setup:
    - $p = 7$, $q = 17$
    - $n = 7 \cdot 17 = 119$
    - $\phi(n) = 6 \cdot 16 = 96$
    - $e = 5$
    - $d = 77$
  - Keys:
    - public key: (119, 5)
    - private key: 77
  - Encryption:
    - $M = 19$
    - $C = 19^5 \bmod 119 = 66$
  - Decryption:
    - $C = 66^{77} \bmod 119 = 19$

Attacking RSA

Given a public key (119, 5) and incepted message 66, which of the following is (are) useful step(s) for an attacker who wants to decrypt the message?



A. Compute 66^5 mod 119

B. Factor 119 to get two prime numbers $\boldsymbol{p}$ and $\boldsymbol{q}$

C. Compute 119^{66} mod 5

D. Compute $\phi(119) = (\boldsymbol{p} + 1)\ (\boldsymbol{q} - 1)$

E. Compute the inverse of 5 in $\boldsymbol{Z}_{\phi}$ for some ϕ

Complete RSA Example

- Setup:
  - $p = 5$, $q = 11$
  - $n = 5 \cdot 11 = 55$
  - $\phi(n) = 4 \cdot 10 = 40$
  - $e = 3$
  - $d = 27$ $(3 \cdot 27 = 81 = 2 \cdot 40 + 1)$
- Encryption
  - $C = M^3 \bmod 55$
- Decryption
  - $M = C^{27} \bmod 55$

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	1	8	27	9	15	51	13	17	14	10	11	23	52	49	20	26	18	2
M	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
C	39	25	21	33	12	19	5	31	48	7	24	50	36	43	22	34	30	16
M	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
C	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

Security

- Security of RSA based on difficulty of factoring
  - Widely believed
  - Best known algorithm takes exponential time
- RSA Security factoring challenge (discontinued)
- In 1999, 512-bit challenge factored in 4 months using 35.7 CPU-years
  - 160 175-400 MHz SGI and Sun
  - 8 250 MHz SGI Origin
  - 120 300-450 MHz Pentium II
  - 4 500 MHz Digital/Compaq
- In 2005, a team of researchers factored the RSA-640 challenge number using 30 2.2GHz CPU years
- In 2004, the prize for factoring RSA-2048 was $200,000
- Current practice is 2,048-bit keys
- Estimated resources needed to factor a number within one year

Length (bits)	PCs	Memory
430	1	128MB
760	215,000	4GB
1,020	342×10^{6}	170GB
1,620	1.6×10^{15}	120TB

Correctness

- We show the correctness of the RSA cryptosystem for the case when the plaintext $\boldsymbol{M}$ does not divide $\boldsymbol{n}$
- Namely, we show that

$$(\boldsymbol{M}^{\boldsymbol{e}})^{\boldsymbol{d}} \bmod \boldsymbol{n} = \boldsymbol{M}$$

- Since $\boldsymbol{ed} \bmod \phi(\boldsymbol{n}) = 1$, there is an integer $\boldsymbol{k}$ such that

$$\boldsymbol{ed} = \boldsymbol{k}\phi(\boldsymbol{n}) + 1$$

- Since $\boldsymbol{M}$ does not divide $\boldsymbol{n}$, by Euler's theorem we have

$$\boldsymbol{M}^{\phi(\boldsymbol{n})} \bmod \boldsymbol{n} = 1$$

- Thus, we obtain

$$
\begin{aligned}
(\boldsymbol{M}^{\boldsymbol{e}})^{\boldsymbol{d}} \bmod \boldsymbol{n} &= \\
&\boldsymbol{M}^{\boldsymbol{ed}} \bmod \boldsymbol{n} = \\
&\boldsymbol{M}^{\boldsymbol{k}\phi(\boldsymbol{n}) + 1} \bmod \boldsymbol{n} = \\
&\boldsymbol{M}\boldsymbol{M}^{\boldsymbol{k}\phi(\boldsymbol{n})} \bmod \boldsymbol{n} = \\
&\boldsymbol{M}\,(\boldsymbol{M}^{\phi(\boldsymbol{n})})^{\boldsymbol{k}} \bmod \boldsymbol{n} = \\
&\boldsymbol{M}\,(\boldsymbol{M}^{\phi(\boldsymbol{n})} \bmod \boldsymbol{n})^{\boldsymbol{k}} \bmod \boldsymbol{n} = \\
&\boldsymbol{M}\,(1)^{\boldsymbol{k}} \bmod \boldsymbol{n} = \\
&\boldsymbol{M} \bmod \boldsymbol{n} = \\
&\boldsymbol{M}
\end{aligned}
$$

- Proof of correctness can be extended to the case when the plaintext $\boldsymbol{M}$ divides $\boldsymbol{n}$

Example: Confidentiality

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Alice chooses $e = 17$, making $d = 53$
- Bob wants to send Alice secret message HELLO (07 04 11 11 14)
  - $07^{17} \bmod 77 = 28$
  - $04^{17} \bmod 77 = 16$
  - $11^{17} \bmod 77 = 44$
  - $11^{17} \bmod 77 = 44$
  - $14^{17} \bmod 77 = 42$
- Bob sends 28 16 44 44 42

Example

- Alice receives 28 16 44 44 42
- Alice uses private key, *d* = 53, to decrypt message:
  - 28^{53} mod 77 = 07
  - 16^{53} mod 77 = 04
  - 44^{53} mod 77 = 11
  - 44^{53} mod 77 = 11
  - 42^{53} mod 77 = 14
- Alice translates message to letters to read HELLO
  - No one else could read it, as only Alice knows her private key and that is needed for decryption

Security Services

- Confidentiality
  - Use public key to encipher, private key to decipher
  - Only the owner of the private key knows it, so *text enciphered with public key can be read only by the owner of the private key*
- Authentication
  - Use private key to encipher, public key to decipher
  - Only the owner of the private key knows it, so *text enciphered with private key must have been generated by the owner*

More Security Services

- Integrity
  - Enciphered letters cannot be changed undetectably without knowing private key
- Non-Repudiation
  - Message enciphered with private key came from someone who knew it

Warnings

- Encipher message in blocks considerably larger than the examples here
  - If 1 character per block, RSA can be broken using statistical attacks (just like classical cryptosystems)
  - Attacker cannot alter letters, but can rearrange them and alter message meaning
    - Example: reverse enciphered message of text ON to get NO

Cryptographic Checksums (Hash Functions)

- Mathematical function to generate a set of k bits from a set of n bits (where $k \leq n$).
  - k is smaller than n except in unusual circumstances
- Can be used for checking integrity
- Example: ASCII parity bit
  - ASCII has 7 bits; 8th bit is "parity"
  - Even parity: even number of 1 bits
  - Odd parity: odd number of 1 bits

Example Use

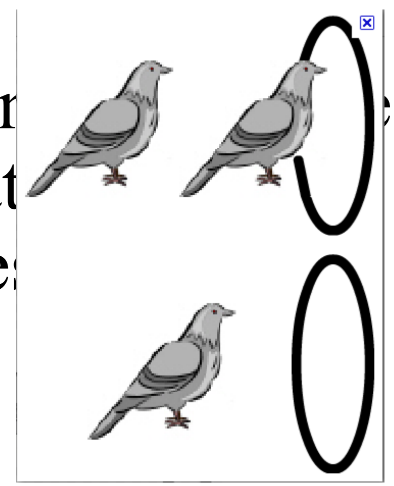
- Bob receives “10111101” as bits.
  - Sender is using even parity; 6 1 bits, so character was received correctly
    - Note: could be garbled, but 2 bits would need to have been changed to preserve parity
  - Sender is using odd parity; even number of 1 bits, so character was not received correctly

Definition

- Cryptographic checksum (Hash function, message digest function) $h: A \rightarrow B$:
  1. For any $x \in A$, $h(x)$ is easy to compute
  2. For any $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$
  3. It is computationally infeasible to find two inputs x, $x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$
     – Alternate form (stronger): Given any $x \in A$, it is computationally infeasible to find a different $x' \in A$ such that $h(x) = h(x')$.

Collisions

- If $x \neq x'$ and $h(x) = h(x')$, x and x' are a *collision*
  - Pigeonhole principle: if there are n containers for n+1 objects, then at least one container will have 2 objects in it.
  - Application: if there are 32 files ar
    cryptographic checksum values, at
    value corresponds to at least 4 file



Keys and Hash Functions

- Keyed hash function: requires cryptographic key as part of the computation
  - DES in chaining mode: encipher message, use last *n* bits. Requires a key to encipher, so it is a keyed cryptographic checksum.
- Keyless hash function: requires no cryptographic key
  - MD5 and SHA-1 are best known; others include MD4, HAVAL, and Snefru

Key Points

- Two main types of cryptosystems: classical and public key
- Classical cryptosystems encipher and decipher using the same key
  - Or one key is easily derived from the other
- Public key cryptosystems encipher and decipher using different keys
  - Computationally infeasible to derive one from the other
- Cryptographic checksums provide a check on integrity

Chapter 9: Key Management

- Session and Interchange Keys
- Key Exchange
- Cryptographic Key Infrastructure
- Revoking Keys
- Digital Signatures

Notation

- $X \rightarrow Y : \{ Z \| W \} k_{X,Y}$
  - X sends Y the message produced by concatenating Z and W enciphered by key $k_{X,Y}$, which is shared by users X and Y
- $A \rightarrow T : \{ Z \} k_A \| \{ W \} k_{A,T}$
  - A sends T a message consisting of the concatenation of Z enciphered using k_A, A's key, and W enciphered using $k_{A,T}$, the key shared by A and T
- r_1, r_2 nonces (nonrepeating random numbers)

Session, Interchange Keys

- Alice wants to send a message *m* to Bob
  - Assume public key encryption
  - Alice generates a random cryptographic key k_s and uses it to encipher *m*
    - To be used for this message *only*
    - Called a *session key*
  - She enciphers k_s with Bob's public key k_B
    - k_B enciphers all session keys Alice uses to communicate with Bob
    - Called an interchange *key*
  - Alice sends $\{ m \} k_s \{ k_s \} k_B$

Benefits of Session Keys

- Limits amount of traffic enciphered with single key
  - Standard practice, to decrease the amount of information an attacker can obtain from the traffic
- Prevents some attacks (e.g., forward search)
  - Example: Alice will send Bob message that is either "BUY" or "SELL". Eve computes possible ciphertexts { "BUY" } k_B and { "SELL" } k_B. Eve intercepts enciphered message, compares, and gets plaintext at once

Classical (Symmetric) Key Exchange

- Bootstrap problem: how do Alice, Bob begin?
  - Alice can't send it to Bob in the clear!
- Assume trusted third party, Cathy
  - Alice and Cathy share secret key k_A
  - Bob and Cathy share secret key k_B
- Use this to exchange shared key k_s

Simple Protocol

$$\text{Alice} \xrightarrow{\{\text{ request for session key to Bob }\}\, k_A} \text{Cathy}$$

$$\text{Alice} \xleftarrow{\{\, k_s \,\}\, k_A \,\|\, \{\, k_s \,\}\, k_B} \text{Cathy}$$

$$\text{Alice} \xrightarrow{\{\, k_s \,\}\, k_B} \text{Bob}$$

$$\text{Alice} \xrightarrow{\{\text{ “Pay \$500 to Dan” }\}\, k_s} \text{Bob}$$

Problems

- How does Bob know he is talking to Alice?
  - Replay attack: Eve records message from Alice to Bob, later replays it; Bob may think he's talking to Alice, but he isn't

Replay Attack

Alice $\xrightarrow{\{ \text{ request for session key to Bob } \} k_A}$ Cathy

Alice $\xleftarrow{\{ k_s \} k_A \,\|\, \{ k_s \} k_B}$ Cathy

Alice $\xrightarrow{\{ k_s \} k_B}$ Bob

$\quad\to$ Eve

Alice $\xrightarrow{\{ \text{“Pay \$500 to Dan”} \} k_s}$ Bob

$\quad\to$ Eve

Eve $\xrightarrow{\{ k_s \} k_B}$ Bob

Eve $\xrightarrow{\{ \text{“Pay \$500 to Dan”} \} k_s}$ Bob

Acknowledgement: Matt Bishop

Problems

- How does Bob know he is talking to Alice?
  - Replay attack: Eve records message from Alice to Bob, later replays it; Bob may think he's talking to Alice, but he isn't
  - Session key reuse: Eve replays message from Alice to Bob, so Bob re-uses session key
- Protocols must provide authentication and defense against replay

Needham-Schroeder

$$\text{Alice} \xrightarrow{\text{Alice} \,\|\, \text{Bob} \,\|\, r_1} \text{Cathy}$$

$$\text{Alice} \xleftarrow{\{\, \text{Alice} \,\|\, \text{Bob} \,\|\, r_1 \,\|\, k_s \,\|\, \{\, \text{Alice} \,\|\, k_s \,\}\, k_B \,\}\, k_A} \text{Cathy}$$

$$\text{Alice} \xrightarrow{\{\, \text{Alice} \,\|\, k_s \,\}\, k_B} \text{Bob}$$

$$\text{Alice} \xleftarrow{\{\, r_2 \,\}\, k_s} \text{Bob}$$

$$\text{Alice} \xrightarrow{\{\, r_2 - 1 \,\}\, k_s} \text{Bob}$$

Acknowledgement: Matt Bishop

Argument: Alice talking to Bob

- Second message $\{ \text{Alice} \,\|\, \text{Bob} \,\|\, r_1 \,\|\, k_s \,\|\, \{ \text{Alice} \,\|\, k_s \} \, k_B \} \, k_A$
  - Enciphered using key (k_A) only she and Cathy knows
    - So Cathy enciphered it
  - Response to first message $\text{Alice} \,\|\, \text{Bob} \,\|\, r_1$
    - As r_1 in it matches r_1 in first message
- Third message $\{ \text{Alice} \,\|\, k_s \} \, k_B$
  - Alice knows only Bob can read it
    - As only Bob can derive session key from message
  - Any messages enciphered with that key are from Bob

Argument: Bob talking to Alice

- Third message $\{ \text{Alice} \,\|\, k_s \} \, k_B$
  - Enciphered using key only he and Cathy know
    - So Cathy enciphered it
  - Names Alice, session key
    - Cathy provided session key, says Alice is other party
- Fourth message $\{ r_2 \} \, k_s$
  - Uses session key to determine if it is replay from Eve
    - If not, Alice will respond correctly in fifth message $\{ r_2 - 1 \} \, k_s$
    - If so, Eve can't decipher r_2 and so can't respond, or responds incorrectly

Public Key Key Exchange

- Here interchange keys known
  - e_A, e_B Alice and Bob's public keys known to all
  - d_A, d_B Alice and Bob's private keys known only to owner
- Simple protocol
  - k_s is desired session key

$$\text{Alice} \xrightarrow{\{ k_s \}\, e_B} \text{Bob}$$

Problem and Solution

- Vulnerable to forgery or replay
  - Because e_B known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
  - k_s is desired session key

$$\text{Alice} \xrightarrow{\{\ \{\ k_s\ \}\ d_A\ \}\ e_B} \text{Bob}$$

Notes

- Can include message enciphered with k_s
- Assumes Bob has Alice's public key, and *vice versa*
  - If not, each must get it from public server
  - If keys not bound to identity of owner, attacker Eve can launch a *man-in-the-middle* attack (next slide; Cathy is public server providing public keys)

Man-in-the-Middle Attack

Alice —send Bob's public key→ | *Eve intercepts request* ···→ Cathy

Eve —send Bob's public key→ Cathy

Eve ←e_B— Cathy

Alice ←e_E— Eve

Alice —$\{ k_s \}\ e_E$→ | *Eve intercepts message* ···→ Bob

Solution to this (binding identity to keys): public key infrastructure (PKI) → Bob

Cryptographic Key Infrastructure

- Goal: bind identity to key
- Classical: not possible, as all keys are shared
  - Use protocols to agree on a shared key (see earlier)
- Public key: bind identity to public key
  - Crucial as people will use key to communicate with principal whose identity is bound to key
  - Erroneous binding means no secrecy between principals
  - Assume principal identified by an acceptable name

Certificates

- Create token (message) containing
  - Identity of principal (here, Alice)
  - Corresponding public key
  - Timestamp (when issued)
  - Other information (perhaps identity of signer)

  signed by trusted authority (here, Cathy)

$$C_A = \{ e_A \,||\, \text{Alice} \,||\, T \}\, d_C$$

Use

- Bob gets Alice’s certificate
  - If he knows Cathy’s public key, he can decipher the certificate
    - When was certificate issued?
    - Is the principal Alice?
  - Now Bob has Alice’s public key
- Problem: Bob needs Cathy’s public key to validate certificate
  - Problem pushed “up” a level
  - One approach: signature chains

X.509 Chains

- Some certificate components in X.509v3:
  - Version
  - Serial number
  - Signature algorithm identifier: hash algorithm
  - Issuer’s name; uniquely identifies issuer
  - Interval of validity
  - Subject’s name; uniquely identifies subject
  - Subject’s public key
  - Signature: enciphered hash

X.509 Certificate Validation

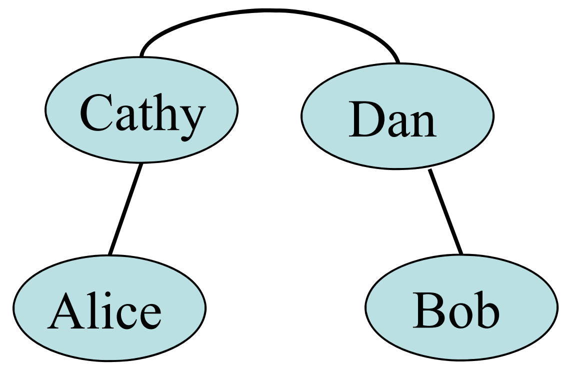
- Obtain issuer’s public key
  - The one for the particular signature algorithm
- Decipher signature
  - Gives hash of certificate
- Recompute hash from certificate and compare
  - If they differ, there’s a problem
- Check interval of validity
  - This confirms that certificate is current

Issuers

- *Certification Authority (CA)*: entity that issues certificates
  - Multiple issuers pose validation problem
  - Alice’s CA is Cathy; Bob’s CA is Don; how can Alice validate Bob’s certificate?
  - Have Cathy and Don cross-certify
    - Each issues certificate for the other

Validation and Cross-Certifying

- Certificates:
  - Cathy<<Alice>>
  - Dan<<Bob>
  - Cathy<<Dan>>
  - Dan<<Cathy>>



- Alice validates Bob’s certificate
  - Alice obtains Cathy<<Dan>>
  - Alice uses (known) public key of Cathy to validate Cathy<<Dan>>
  - Alice uses Cathy<<Dan>> to validate Dan<<Bob>>

Acknowledgement: Matt Bishop

Key Revocation

- Certificates invalidated *before* expiration
  - Usually due to compromised key
  - May be due to change in circumstance (*e.g.*, someone leaving company)
- Problems
  - Entity revoking certificate authorized to do so
  - Revocation information circulates to everyone fast enough
    - Network delays, infrastructure problems may delay information

CRLs

- *Certificate revocation list* lists certificates that are revoked
- X.509: only certificate issuer can revoke certificate
  - Added to CRL

PGP Chains

- OpenPGP certificates structured into packets
  - One public key packet
  - Zero or more signature packets
- Public key packet:
  - Version (3 or 4; 3 compatible with all versions of PGP, 4 not compatible with older versions of PGP)
  - Creation time
  - Validity period (not present in version 3)
  - Public key algorithm, associated parameters
  - Public key

OpenPGP Signature Packet

- Version 3 signature packet
  - Version (3)
  - Signature type (level of trust)
  - Creation time (when next fields hashed)
  - Signer's key identifier (identifies key to encipher hash)
  - Public key algorithm (used to encipher hash)
  - Hash algorithm
  - Part of signed hash (used for quick check)
  - Signature (enciphered hash)
- Version 4 packet more complex

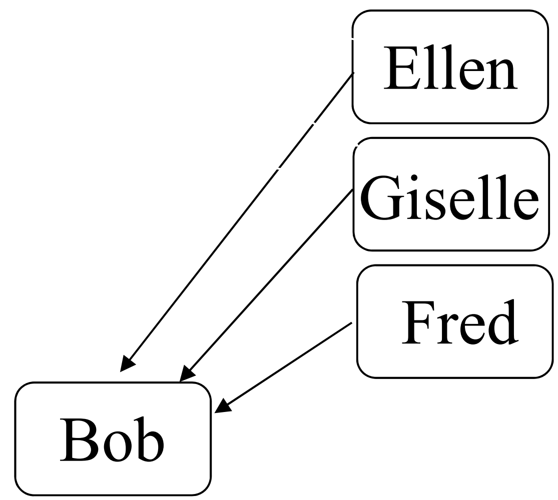
Signing

- Single certificate may have multiple signatures
- Notion of "trust" embedded in each signature
  - Range from "untrusted" to "ultimate trust"
  - Signer defines meaning of trust level (no standards!)
- All version 4 keys signed by subject
  - Called "self-signing"

Validating Certificates

- Alice needs to validate Bob's OpenPGP cert
  - Alice does not know any of the signers: Fred, Giselle, or Ellen

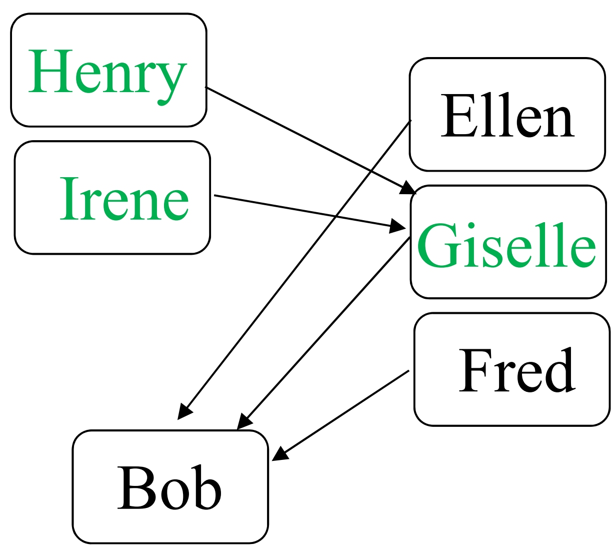
Arrows show signatures
Self signatures not shown



Validating Certificates

- Alice needs to validate Bob's OpenPGP cert
  - Alice does not know any of the signers: Fred, Giselle, or Ellen
- Alice gets Giselle's cert
  - Alice knows Henry slightly, but his signature is at "casual" level of trust

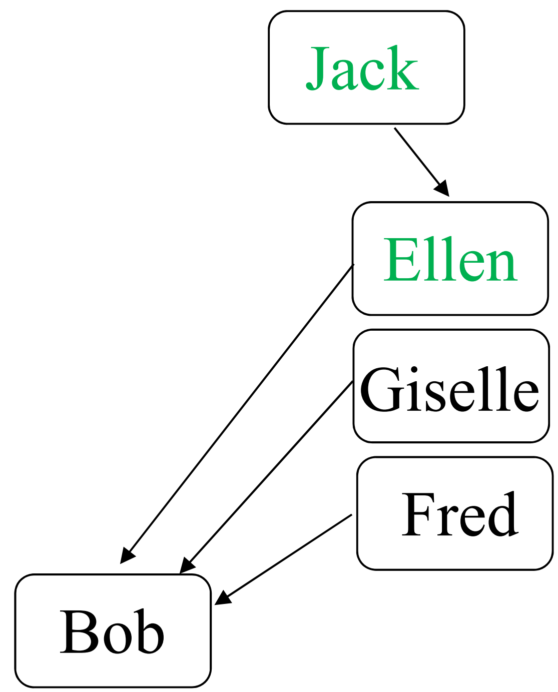
Arrows show signatures
Self signatures not shown



Validating Certificates

- Alice needs to validate Bob's OpenPGP cert
  - Alice does not know any of the signers: Fred, Giselle, or Ellen
- Alice gets Giselle's cert
  - Alice knows Henry slightly, but his signature is at "casual" level of trust
- Alice gets Ellen's cert
  - Alice knows Jack, so uses his cert to validate Ellen's, then hers to validate Bob's

Arrows show signatures
Self signatures not shown



Digital Signature

- Construct that authenticated origin, contents of message in a manner provable to a disinterested third party (“judge”)
- Sender cannot deny having sent message (service is “non-repudiation”)
  - Limited to *technical* proofs
    - Inability to deny one’s cryptographic key was used to sign
  - One could claim the cryptographic key was stolen or compromised
    - Legal proofs, *etc.*, probably required; not dealt with here

Classical Digital Signatures

- Require trusted third party
  - Alice, Bob each share keys with trusted party Cathy
- To resolve dispute, judge gets $\{ m \} k_{Alice}$, $\{ m \} k_{Bob}$, and has Cathy decipher them; if messages matched, contract was signed

Alice $\xrightarrow{\{ m \} k_{Alice}}$ Bob

Cathy $\xleftarrow{\{ m \} k_{Alice}}$ Bob

Cathy $\xrightarrow{\{ m \} k_{Bob}}$ Bob

Public Key Digital Signatures

- Alice's keys are d_{Alice}, e_{Alice}
- Alice sends Bob

$$m \,\|\, \{ m \} \, d_{Alice}$$

- In case of dispute, judge computes

$$\{ \{ m \} \, d_{Alice} \} \, e_{Alice}$$

- and if it is m, Alice signed message
  - She's the only one who knows d_{Alice}!

RSA Digital Signatures

- Use private key to encipher message
  - Protocol for use is *critical*
- Key points:
  - Never sign random documents, and when signing, always sign hash and never document
    - Mathematical properties can be turned against signer

Attack if Random Documents Are Signed

- Example: Alice, Bob communicating
  - $n_A = 95$, $e_A = 59$, $d_A = 11$
  - $n_B = 77$, $e_B = 53$, $d_B = 17$
- 26 contracts, numbered 00 to 25
  - Alice has Bob sign 05 and 17:
    - $c = m^{d_B} \bmod n_B = 05^{17} \bmod 77 = 3$
    - $c = m^{d_B} \bmod n_B = 17^{17} \bmod 77 = 19$
  - Alice computes $05 \times 17 \bmod 77 = 08$; corresponding signature is $03 \times 19 \bmod 77 = 57$; claims Bob signed 08
  - Judge computes $c^{e_B} \bmod n_B = 57^{53} \bmod 77 = 08$
    - Signature validated; Bob is toast

RSA Digital Signatures

- Use private key to encipher message
  - Protocol for use is *critical*
- Key points:
  - Never sign random documents, and when signing, always sign hash and never document
    - Mathematical properties can be turned against signer
  - Sign message first, then encipher
    - Otherwise, the recipient can change public keys to forge signatures on a different message
    - Example in the textbook

Key Points

- Key management critical to effective use of cryptosystems
  - Different levels of keys (session *vs*. interchange)
- Keys need infrastructure to identify holders, allow revoking

- Digital signatures provide integrity of origin and content

  Much easier with public key cryptosystems than with classical cryptosystems