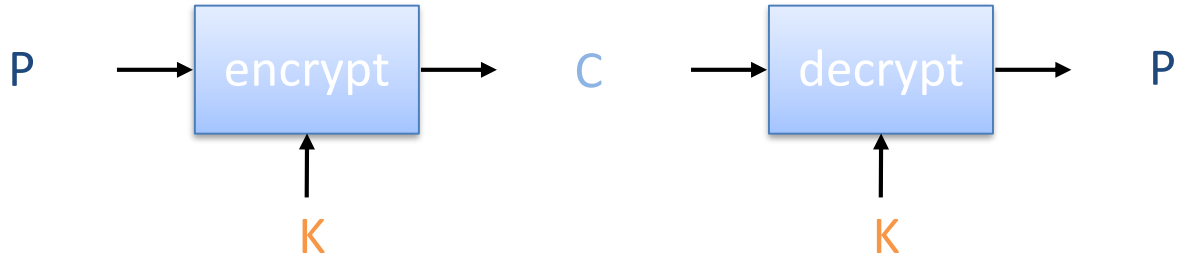
Cryptography

Symmetric Cryptosystem

- Scenario
  - Alice wants to send a message (plaintext P) to Bob.
  - The communication channel is insecure and can be eavesdropped
  - If Alice and Bob have previously agreed on a symmetric encryption scheme and a secret key K, the message can be sent encrypted (ciphertext C)
- Issues
  - What is a good symmetric encryption scheme?
  - What is the complexity of encrypting/decrypting?
  - What is the size of the ciphertext, relative to the plaintext?

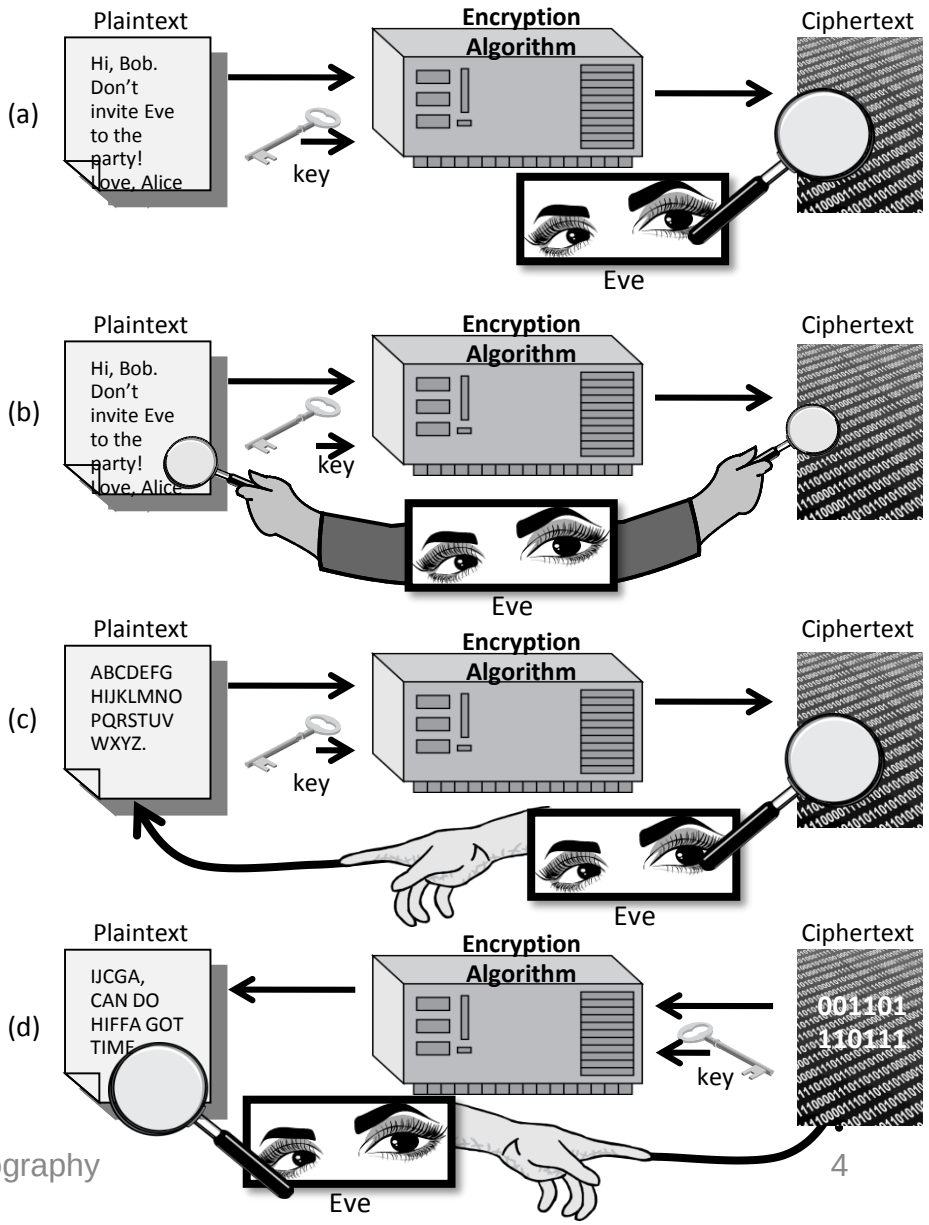


Basics

- Notation
  - Secret key K
  - Encryption function $E_K(P)$
  - Decryption function $D_K(C)$
  - Plaintext length typically the same as ciphertext length
  - Encryption and decryption are permutation functions (bijections) on the set of all n-bit arrays
- Efficiency
  - functions E_K and D_K should have efficient algorithms
- Consistency
  - Decrypting the ciphertext yields the plaintext
  - $D_K(E_K(P)) = P$

Attacks

- Attacker may have
  a) collection of ciphertexts (ciphertext only attack)
  b) collection of plaintext/ciphertext pairs (known plaintext attack)
  c) collection of plaintext/ciphertext pairs for plaintexts selected by the attacker (chosen plaintext attack)
  d) collection of plaintext/ciphertext pairs for ciphertexts selected by the attacker (chosen ciphertext attack)



Brute-Force Attack

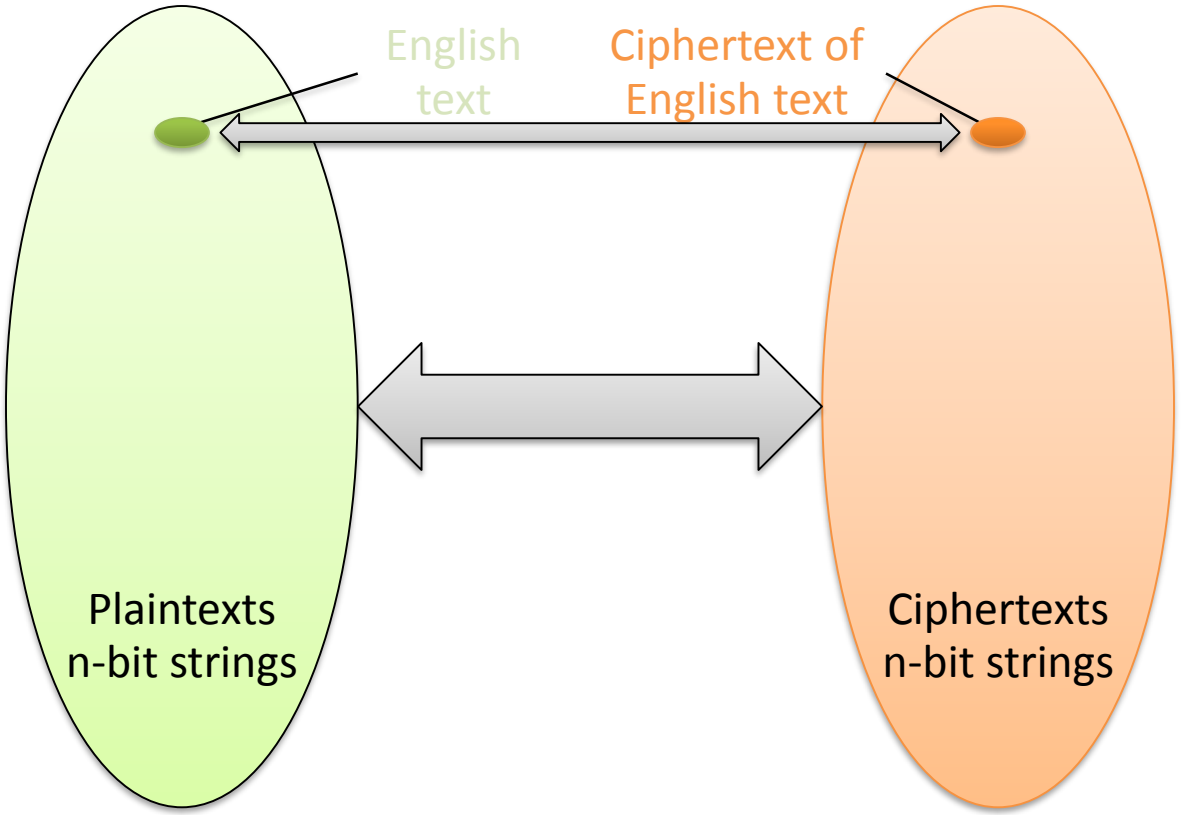
- Try all possible keys K and determine if $D_K(C)$ is a likely plaintext
  - Requires some knowledge of the structure of the plaintext (e.g., PDF file or email message)
- Key should be a sufficiently long random value to make exhaustive search attacks unfeasible

Image by Michael Cote from http://commons.wikimedia.org/wiki/File:Bingo_cards.jpg

Encrypting English Text

- English text typically represented with 8-bit ASCII encoding
- A message with t characters corresponds to an n-bit array, with n = 8t

- Redundancy due to repeated words and patterns
  - E.g., “th”, “ing”
- English plaintexts are a very small subset of all n-bit arrays



Entropy of Natural Language

- Information content (entropy) of English: 1.25 bits per character
- t-character arrays that are English text:

$$(2^{1.25})^t = 2^{1.25\,t}$$

- n-bit arrays that are English text:

$$2^{1.25\,n/8} \approx 2^{0.16\,n}$$

- For a natural language, constant $\alpha < 1$ such that there are $2^{\alpha n}$ messages among all n-bit arrays
- Fraction (probability) of valid messages

$$2^{\alpha n} / 2^n = 1 / 2^{(1-\alpha)n}$$

- Brute-force decryption
  - Try all possible 2^k decryption keys
  - Stop when valid plaintext recognized
- Given a ciphertext, there are 2^k possible plaintexts
- Expected number of valid plaintexts

$$2^k / 2^{(1-\alpha)n}$$

- Expected unique valid plaintext , (no spurious keys) achieved at unicity distance

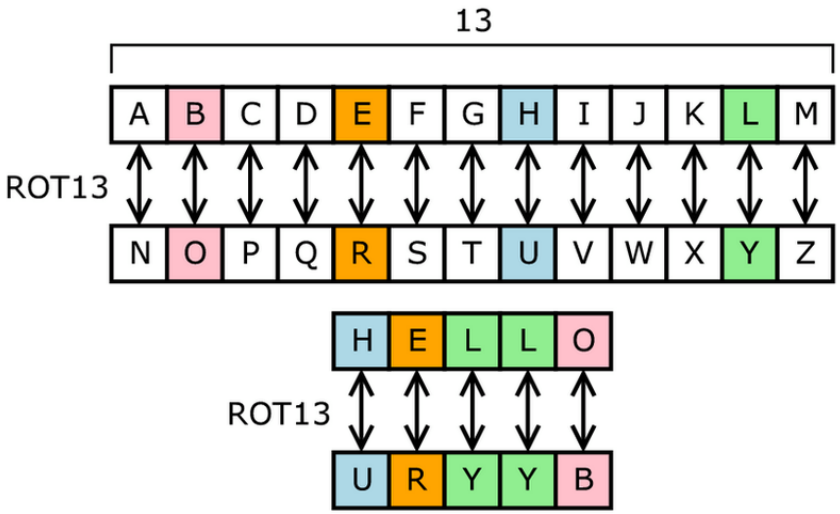
$$n = k / (1-\alpha)$$

- For English text and 256-bit keys, unicity distance is 304 bits

Substitution Ciphers

- Each letter is uniquely replaced by another.
- There are 26! possible substitution ciphers.
- There are more than 4.03×10^{26} such ciphers.

- One popular substitution "cipher" for some Internet posts is ROT13.



Public domain image from http://en.wikipedia.org/wiki/File:ROT13.png

Frequency Analysis

- Letters in a natural language, like English, are not uniformly distributed.
- Knowledge of letter frequencies, including pairs and triples can be used in cryptologic attacks against substitution ciphers.

a:	8.05%	b:	1.67%	c:	2.23%	d:	5.10%
e:	12.22%	f:	2.14%	g:	2.30%	h:	6.62%
i:	6.28%	j:	0.19%	k:	0.95%	l:	4.08%
m:	2.33%	n:	6.95%	o:	7.63%	p:	1.66%
q:	0.06%	r:	5.29%	s:	6.02%	t:	9.67%
u:	2.92%	v:	0.82%	w:	2.60%	x:	0.11%
y:	2.04%	z:	0.06%				

Letter frequencies in the book *The Adventures of Tom Sawyer*, by Twain.

Substitution Boxes

- Substitution can also be done on binary numbers.
- Such substitutions are usually described by substitution boxes, or S-boxes.

	00	01	10	11
00	0011	0100	1111	0001
01	1010	0110	0101	1011
10	1110	1101	0100	0010
11	0111	0000	1001	1100

(a)

	0	1	2	3
0	3	8	15	1
1	10	6	5	11
2	14	13	4	2
3	7	0	9	12

(b)

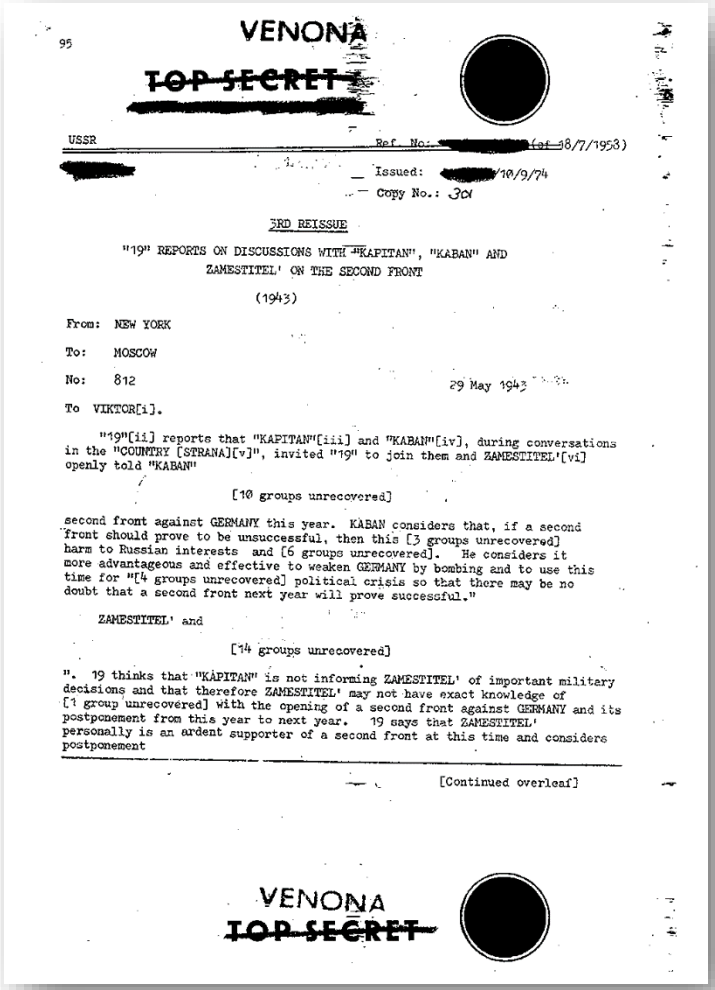
Figure 8.3: A 4-bit S-box (a) An S-box in binary. (b) The same S-box in decimal.

One-Time Pads

- There is one type of substitution cipher that is absolutely unbreakable.
  - The **one-time pad** was invented in 1917 by Joseph Mauborgne and Gilbert Vernam
  - We use a block of shift keys, (k_1, k_2, . . . , k_n), to encrypt a plaintext, M, of length n, with each shift key being chosen uniformly at random.
- Since each shift is random, every ciphertext is equally likely for any plaintext.

Weaknesses of the One-Time Pad

- In spite of their perfect security, one-time pads have some weaknesses
- The key has to be as long as the plaintext
- Keys can never be reused
  - Repeated use of one-time pads allowed the U.S. to break some of the communications of Soviet spies during the Cold War.



95

VENONA

~~TOP SECRET~~

USSR

Ref No: [redacted] (of 8/7/1953)

[redacted] Issued: [redacted] 10/9/74

Copy No.: 30

3RD REISSUE

"19" REPORTS ON DISCUSSIONS WITH "KAPITAN", "KABAN" AND ZAMESTITEL' ON THE SECOND FRONT

(1943)

From: NEW YORK

To: MOSCOW

No: 812

29 May 1943

To VIKTOR[i].

"19"[ii] reports that "KAPITAN"[iii] and "KABAN"[iv], during conversations in the "COUNTRY [STRANA][v]", invited "19" to join them and ZAMESTITEL'[vi] openly told "KABAN"

[10 groups unrecovered]

second front against GERMANY this year. KABAN considers that, if a second front should prove to be unsuccessful, then this [3 groups unrecovered] harm to Russian interests and [6 groups unrecovered]. He considers it more advantageous and effective to weaken GERMANY by bombing and to use this time for "[4 groups unrecovered] political crisis so that there may be no doubt that a second front next year will prove successful."

ZAMESTITEL' and

[14 groups unrecovered]

". 19 thinks that "KAPITAN" is not informing ZAMESTITEL' of important military decisions and that therefore ZAMESTITEL' may not have exact knowledge of [1 group unrecovered] with the opening of a second front against GERMANY and its postponement from this year to next year. 19 says that ZAMESTITEL' personally is an ardent supporter of a second front at this time and considers postponement

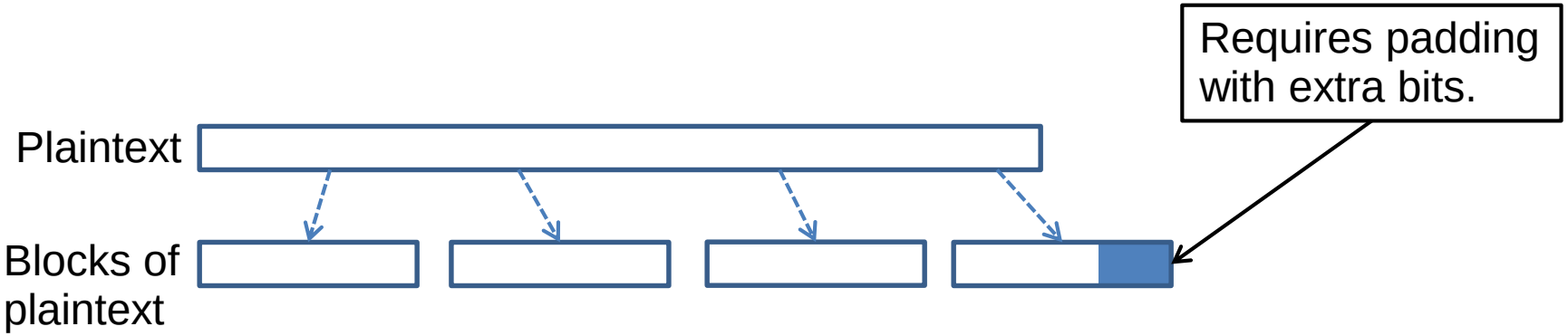
[Continued overleaf]

VENONA

~~TOP SECRET~~

Block Ciphers

- In a **block cipher:**
  - Plaintext and ciphertext have fixed length b (e.g., 128 bits)
  - A plaintext of length n is partitioned into a sequence of m **blocks**, P[0], …, P[m–1], where $n \leq bm < n + b$
- Each message is divided into a sequence of blocks and encrypted or decrypted in terms of its blocks.



Padding

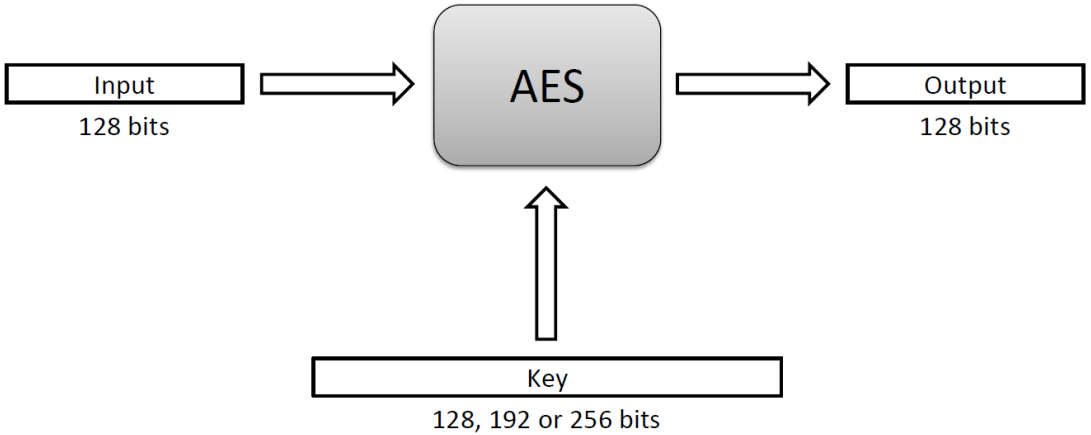
- Block ciphers require the length n of the plaintext to be a multiple of the block size b
- Padding the last block needs to be unambiguous (cannot just add zeroes)
- When the block size and plaintext length are a multiple of 8, a common padding method (PKCS5) is a sequence of identical bytes, each indicating the length (in bytes) of the padding
- Example for b = 128 (16 bytes)
  - Plaintext: “Roberto” (7 bytes)
  - Padded plaintext: “Roberto999999999” (16 bytes), where 9 denotes the number and not the character
- We need to always pad the last block, which may consist only of padding

Block Ciphers in Practice

- Data Encryption Standard (DES)
  - Developed by IBM and adopted by NIST in 1977
  - 64-bit blocks and 56-bit keys
  - Small key space makes exhaustive search attack feasible since late 90s
- Triple DES (3DES)
  - Nested application of DES with three different keys KA, KB, and KC
  - Effective key length is 168 bits, making exhaustive search attacks unfeasible
  - $C = E_{KC}(D_{KB}(E_{KA}(P)))$; $P = D_{KA}(E_{KB}(D_{KC}(C)))$
  - Equivalent to DES when KA=KB=KC (backward compatible)
- Advanced Encryption Standard (AES)
  - Selected by NIST in 2001 through open international competition and public discussion
  - 128-bit blocks and several possible key lengths: 128, 192 and 256 bits
  - Exhaustive search attack not currently possible
  - AES-256 is the symmetric encryption algorithm of choice

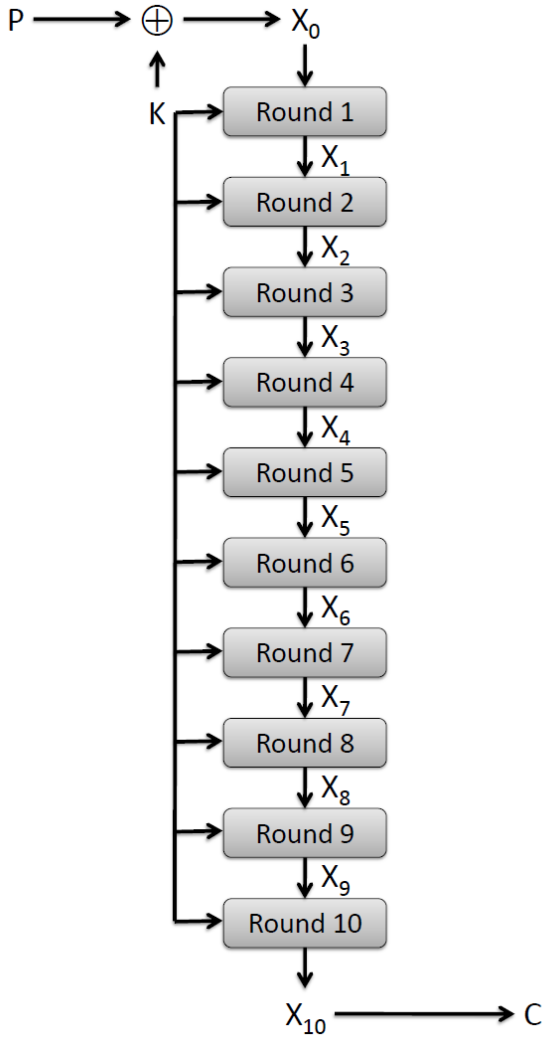
The Advanced Encryption Standard (AES)

- 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: $C = X_{10}$.



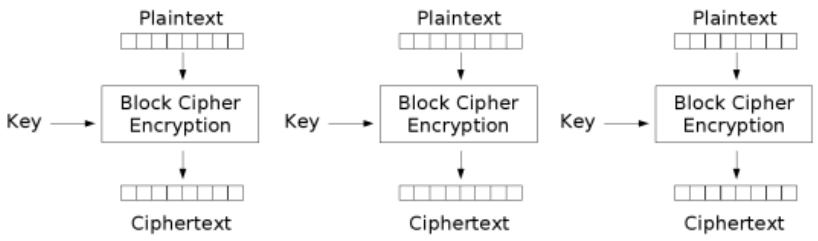
AES Rounds

- Each round is built from four basic steps:

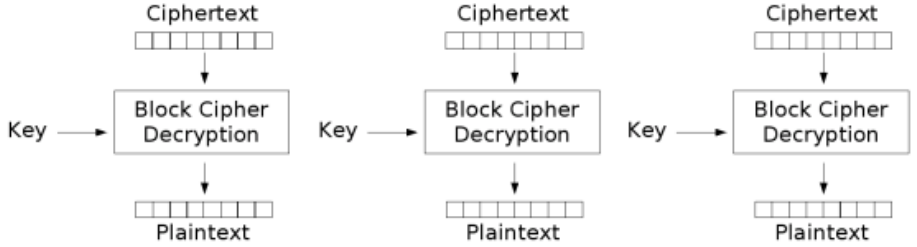
1. **SubBytes step**: an S-box substitution step
2. **ShiftRows step**: a permutation step
3. **MixColumns step**: a matrix multiplication step
4. **AddRoundKey step**: an XOR step with a **round key** derived from the 128-bit encryption key

Block Cipher Modes

- A block cipher mode describes the way a block cipher encrypts and decrypts a sequence of message blocks.
- Electronic Code Book (ECB) Mode (is the simplest):
  - Block P[i] encrypted into ciphertext block $C[i] = E_K(P[i])$
  - Block C[i] decrypted into plaintext block $M[i] = D_K(C[i])$



Electronic Codebook (ECB) mode encryption

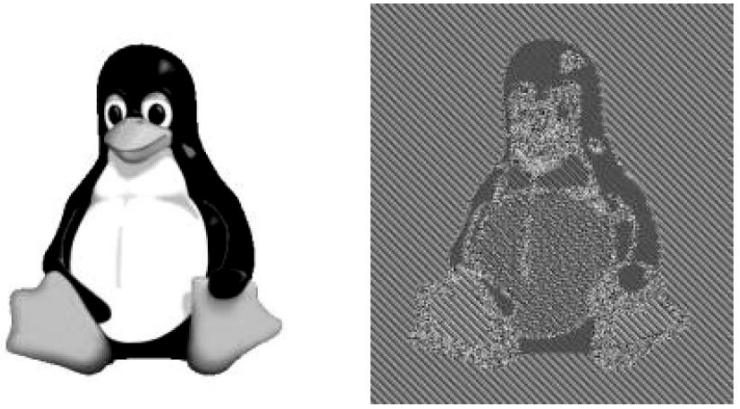


Electronic Codebook (ECB) mode decryption

Strengths and Weaknesses of ECB

- Strengths:
  - Is very simple
  - Allows for parallel encryptions of the blocks of a plaintext
  - Can tolerate the loss or damage of a block

- Weakness:
  - Documents and images are not suitable for ECB encryption since patters in the plaintext are repeated in the ciphertext:

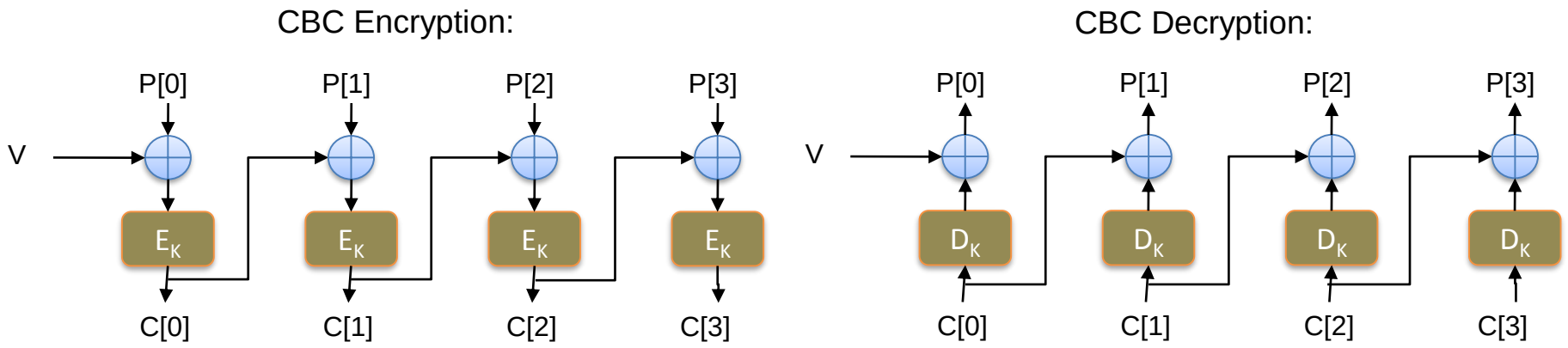


(a) (b)

Figure 8.6: How ECB mode can leave identifiable patterns in a sequence of blocks: (a) An image of Tux the penguin, the Linux mascot. (b) An encryption of the Tux image using ECB mode. (The image in (a) is by Larry Ewing, lewing@isc.tamu.edu, using The Gimp; the image in (b) is by Dr. Juzam. Both are used with permission via attribution.)

Cipher Block Chaining (CBC) Mode

- In Cipher Block Chaining (CBC) Mode
  - The previous ciphertext block is combined with the current plaintext block $C[i] = E_K(C[i-1] \oplus P[i])$
  - $C[-1] = V$, a random block separately transmitted encrypted (known as the initialization vector)
  - Decryption: $P[i] = C[i-1] \oplus D_K(C[i])$



Strengths and Weaknesses of CBC

- Strengths:
  - Doesn't show patterns in the plaintext
  - Is the most common mode
  - Is fast and relatively simple
- Weaknesses:
  - CBC requires the reliable transmission of all the blocks sequentially
  - CBC is not suitable for applications that allow packet losses (e.g., music and video streaming)

Java AES Encryption Example

- Source

  http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html

- Generate an AES key

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
SecretKey aesKey = keygen.generateKey();
```

- Create a cipher object for AES in ECB mode and PKCS5 padding

```
Cipher aesCipher;
aesCipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
```

- Encrypt

```
aesCipher.init(Cipher.ENCRYPT_MODE, aesKey);
byte[] plaintext = "My secret message".getBytes();
byte[] ciphertext = aesCipher.doFinal(plaintext);
```

- Decrypt

```
aesCipher.init(Cipher.DECRYPT_MODE, aesKey);
byte[] plaintext1 = aesCipher.doFinal(ciphertext);
```

Stream Cipher

- Key stream
  - Pseudo-random sequence of bits S = S[0], S[1], S[2], …
  - Can be generated on-line one bit (or byte) at the time
- Stream cipher
  - XOR the plaintext with the key stream C[i] = S[i] $\oplus$ P[i]
  - Suitable for plaintext of arbitrary length generated on the fly, e.g., media stream
- Synchronous stream cipher
  - Key stream obtained only from the secret key K
  - Works for unreliable channels if plaintext has packets with sequence numbers
- Self-synchronizing stream cipher
  - Key stream obtained from the secret key and q previous ciphertexts
  - Lost packets cause a delay of q steps before decryption resumes

Key Stream Generation

- RC4
  - Designed in 1987 by Ron Rivest for RSA Security
  - Trade secret until 1994
  - Uses keys with up to 2,048 bits
  - Simple algorithm
- Block cipher in counter mode (CTR)
  - Use a block cipher with block size b
  - The secret key is a pair (K,t), where K a is key and t (counter) is a b-bit value
  - The key stream is the concatenation of ciphertexts

    $$E_K(t), E_K(t+1), E_K(t+2), \ldots$$

  - Can use a shorter counter concatenated with a random value
  - Synchronous stream cipher

Attacks on Stream Ciphers

- Repetition attack
  - if key stream reused, attacker obtains XOR of two plaintexts
- Insertion attack [Bayer Metzger, TODS 1976]
  - retransmission of the plaintext with
    - a chosen byte inserted by attacker
    - using the same key stream
  - e.g., email message resent with new message number

Original

P	P[i]	P[i+1]	P[i+2]	P[i+3]
S	S[i]	S[i+1]	S[i+2]	S[i+3]
C	C[i]	C[i+1]	C[i+2]	C[i+3]

Retransmission

P	P[i]	X	P[i+1]	P[i+2]
S	S[i]	S[i+1]	S[i+2]	S[i+3]
C	C[i]	C′[i+1]	C′[i+2]	C′[i+3]

Public Key Encryption

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 $\boldsymbol{n}$:

$$\boldsymbol{n} = \boldsymbol{p_1}^{\boldsymbol{e}_1} \times \ldots \times \boldsymbol{p_k}^{\boldsymbol{e}_k}$$

- 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:

$$\gcd(18, 30) = 6 \qquad \gcd(0, 20) = 20$$
$$\gcd(-21, 49) = 7$$

- 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 n

  $$r = a \bmod n$$

  equivalent to

  $$a = r + kn$$

  and

  $$r = a - \lfloor a/n \rfloor n$$

- 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:

  $$\gcd(a, b) = \gcd(b, a \bmod b)$$

- 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

  $\gcd(a, b) = \gcd(b, a \bmod b)$

- 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 n are the set

$$Z_n = \{0, 1, 2, \ldots, (n-1)\}$$

- Let x and y be two elements of Z_n such that

$$xy \bmod n = 1$$

We say that y is the multiplicative inverse of x in Z_n and we write $y = x^{-1}$

- Example:
  - Multiplicative inverses of the residues modulo 11

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

Multiplicative Inverses (2)

Theorem

An element x of Z_n has a multiplicative inverse if and only if x and n are relatively prime

- Example
  - The elements of Z_{10} with a multiplicative inverse are $1, 3, 7, 9$

Corollary

If p is prime, every nonzero residue in Z_p has a multiplicative inverse

Theorem

A variation of Euclid's GCD algorithm computes the multiplicative inverse of an element x of 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

$xx^{-1} mod\ n=1$

RSA

- By Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- Basic idea: prime multiplication is very easy, integer factorization seems to be unfeasible.
  - nb. exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA Key Setup

- Step 1: select p, q
  - selecting two large primes at random p, q
  - computing their system modulus $N=p.q$
    - Eluer totient: $\varnothing(N)=(p-1)(q-1)$
- Step 2: select the encryption key e
  - e and $\varnothing(N)$ are relative prime
  - $\gcd(e,\varnothing(N))=1$, $1<e<\varnothing(N)$
- Step 3: select the decryption key d
  - $e.d=1 \bmod \varnothing(N)$ and $0 \le d \le N$
- Public encryption key: PK={e,N}
- Secret private decryption key: SK={d,p,q}

RSA Encryption/Decryption

- to encrypt a message M the sender:
  - obtains **public key** of recipient $PK=\{e,N\}$
  - Encrypt: $C=M^e \bmod N$, where $0 \leq M < N$
- to decrypt the ciphertext C the owner:
  - uses their private key $SK=\{d,p,q\}$
  - Decrypt: $M=C^d \bmod N$

- note that the message M must be smaller than the modulus N (block if needed)

Step 1: Generate Primes

- Get a pseudo random number
- Use Fermat's Little Theorem to test for prime
  - For prime n and any a, $a^n \bmod n = a$
  - For composite n and any a, $a^n \bmod n \neq a$
  - BUT
  - If $a^n \bmod n = a$, n could be a composite

Step2: Generate Exponents e and d

- For public exponent, e, pick any prime
  - Common choices are 3, 17 and 65537 ($2^{16} + 1$)

Step3: Extended Euclidean Algorithm

- For secret exponent, d, compute the modular inverse of e mod ϕ
  - Use Extended Euclidean Algorithm
- To find inverse of e mod n:
  - Find quotient and remainder of n/e at each step
  - Also carry an auxiliary number $u_i = u_{i-2} - u_{i-1}q_{i-2} \bmod n$
  - Initialize $u_0 = 0$ and $u_1 = 1$
  - For each step use the previous e as the current n and the previous remainder as the current e
  - Repeat until e = 0 and the auxiliary number is the inverse of e mod n

Step3: Inverses Modulo a Number

4 and 1/4 are inverses because $4 * 1/4 = 1$

In modulo world, want to find x such that

$1 = a*x \pmod n$

Also written $a^{-1} = x \pmod n$

- Has unique solution if a and n are relatively prime.
- If a and n aren't relatively prime, has no solution.

$4x = 1 \pmod 7$ <=> Finding x, k such that $4x = 7k + 1$

Inverse of 5 modulo 14 = 3

2 has no inverse modulo 14.

RSA Example 1

- Setup:
  - $n = pq$, with p and q primes
  - e relatively prime to $\phi(n) = (p - 1)(q - 1)$
  - d inverse of e in $Z_{\phi(n)}$: $e.d=1 \bmod \varnothing(N)$
- 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:
    - $p = 2,\ q = 5$
      - $n = 2 \cdot 5 = 10$
      - $\phi(n) = (2-1) \cdot (5-1) = 4$
    - $e = 3$ (relatively prime to 4
    - $d = 7$ (inverse of e)
  - Keys:
    - public key: (10, 3)
    - private key: 7
  - Encryption:
    - $M = 8$
    - $C = 8^3 \bmod 10 = 2$
  - Decryption:
    - $C = 2^7 \bmod 10 = 8$

RSA Example 2

- Setup:
  - $n = pq$, with p and q primes
  - e relatively prime to $\phi(n) = (p - 1)(q - 1)$
  - d inverse of e in $Z_{\phi(n)}$: `e.d=1 mod ∅(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:
    - $C = 66^{77} \bmod 119 = 19$

https://www.wolframalpha.com/input/?i=19%5E5+mod+119

Complete RSA Example 3

- 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

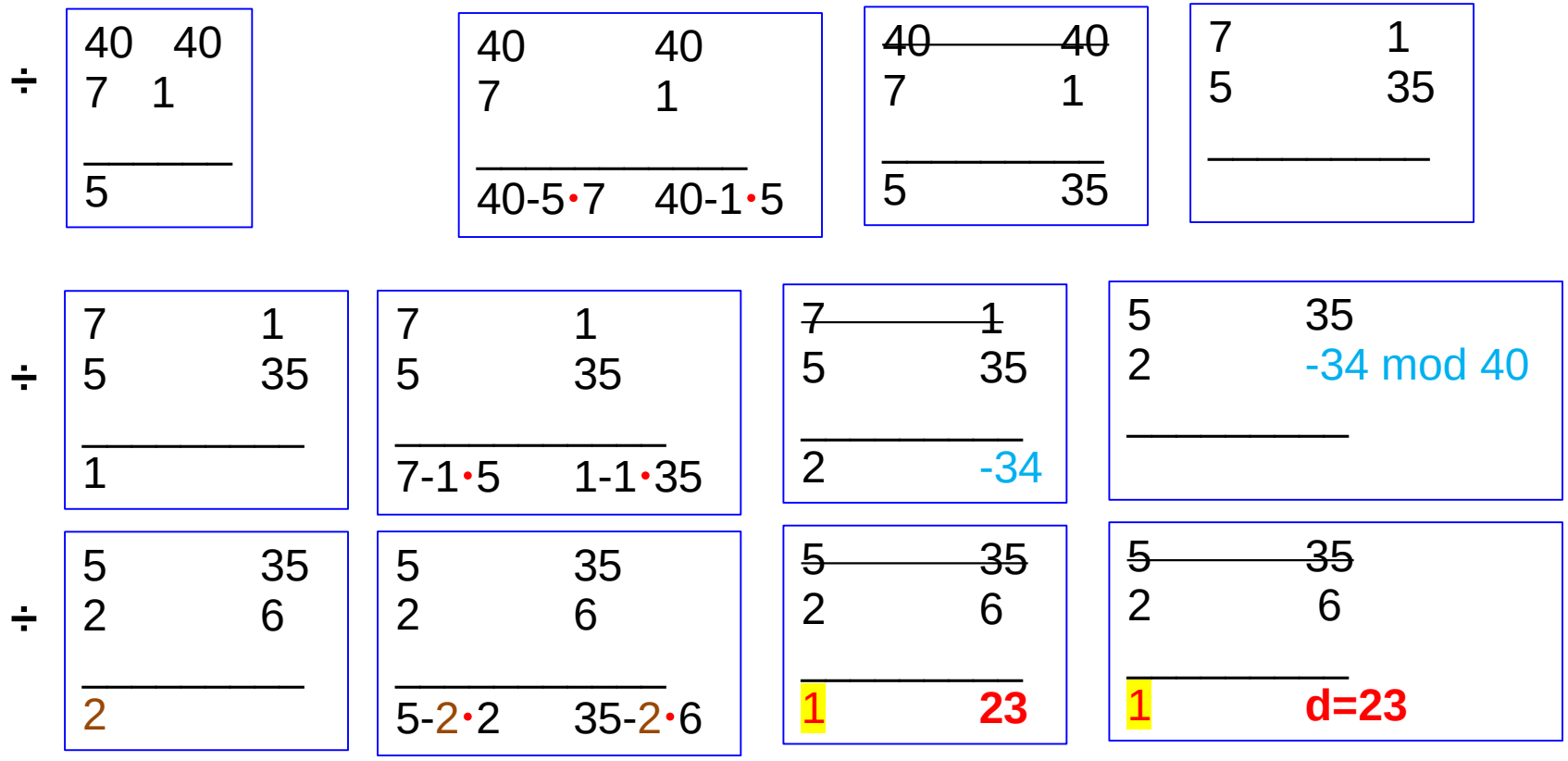
RSA Example 4

- Setup:
  - $n = pq$, with p and q primes
  - e relatively prime to $\phi(n) = (p - 1)(q - 1)$
  - d inverse of e in $Z_{\phi(n)}$: e.d=1 mod ∅(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:
  - $M = C^d \bmod n$

- ***Select*** $p = 5,\ q = 11$
  - $n = 5 \cdot 11 = 55$
  - $\phi(n) = 4 \cdot 10 = 40$
- ***Select*** $e = 7$
- Find d such that $e \cdot d \bmod \phi = 1$
  - $7 \cdot d \bmod 40 = 1$
    - Extended Euclidean algorithm

https://www.wolframalpha.com/input/?i=19%5E5+mod+119

Find d s.t.7·d mod 40 =1



- $e \cdot d \mod \phi = 1$
- $7 \cdot 23 \mod 40 = 1$

- Keys:
  - public key: (55, 7)
  - private key: 23

RSA Example 4

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

- ***Select*** $p = 5$, $q = 11$
  - $n = 5 \cdot 11 = 55$
  - $\phi(n) = 4 \cdot 10 = 40$
- ***Select*** $e = 7$
- Find d such that $e \cdot d \mod \phi = 1$
  - $7 \cdot d \mod 40 = 1$
    - Extended Euclidean algorithm
    - **d=23**
    - Encryption:
      - $M = 8$
      - $C = 8^7 \mod 55 = 2$
    - Decryption:
      - $C = 2^{23} \mod 55 = 8$

https://www.wolframalpha.com/input/?i=8%5E7+mod+55

Does this seem easy to crack?

- If n is 256 bits or shorter, you can crack the keys in a couple of hours on your own computer
- Typical keys are 1024-2048 bits long
- It is possible that 1024 bits keys will be breakable soon
- The current recommendation is that keys be 2048 bits

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

Private-key versus public-key cryptography

- The prime advantage of public-key cryptography is increased security - the private keys do not ever need to be transmitted or revealed to anyone.

- Public key cryptography is not meant to replace secret-key cryptography, but rather to supplement it, to make it more secure.

- Example RSA and DES are usually combined as follows
  1. The message is encrypted with a random DES key
  2. DES-key is encrypted with RSA
  3. DES-encrypted message and RSA-encrypted DES-key are sent.

This protocol is called RSA digital envelope.

- In software (hardware) DES is generally about 100 (1000) times faster than RSA.

If *n* users communicate with secrete-key cryptography, they need *n* (*n* - *1*) / *2* keys. In the case they use public key cryptography *2n* keys are sufficient.

Public-key cryptography allows spontaneous communication.

Private-key versus public-key cryptography

• If RSA is used for digital signature then the public key is usually much smaller than private key => verification is faster.

• An RSA signature is superior to a handwritten signature because it attests both to the contents of the message as well as to the identity of the signer.

As long as a secure hash function is used there is no way to take someone's signature from one document and attach it to another, or to after the signed message in any way.

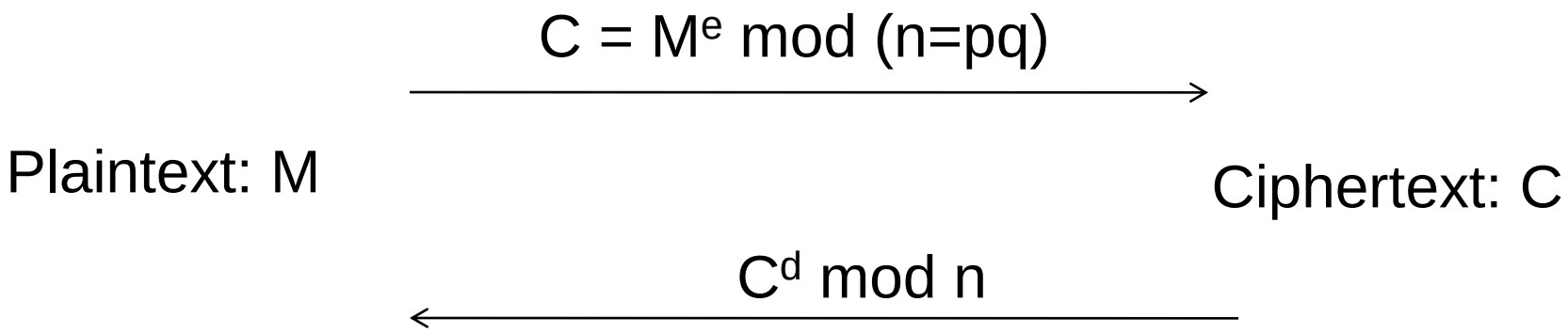
The slightest change in a signed message will cause the digital signature verification process to fail.

• Digital signature are the exact tool necessary to convert even the most important paper based documents to digital form and to have them only in the digital form.

The RSA trapdoor 1-to-1 function

- **Parameters:** N=pq. N ≈1024 bits. p,q ≈512 bits.
  e – encryption exponent. $\gcd(e, \varphi(N)) = 1$.
  $\varphi(N)=(p-1)(q-1)$
- **Encryption:** $C=RSA(M) = M^e \pmod N$ where $M \in Z_N^*$

- **Trapdoor:** d – decryption exponent.
  Where $e \cdot d = 1 \pmod{\varphi(N)}$
- **Decryption:** $M=RSA(C)^d = M^{ed} = M^{k\varphi(N)+1} =$
  $M^1 \cdot (M^{\varphi(N)})^q = M^1 \cdot (1)^q = M^1 = M \bmod N$



From n, difficult to figure out p,q

From (n,e), difficult to figure d.

From (n,e) and C, difficult to figure out M s.t. $C = M^e$

Correctness

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

$$(M^e)^d \bmod n = M$$

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

$$ed = k\phi(n) + 1$$

- Since M does not divide n, by Euler's theorem we have

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

- Thus, we obtain

$$(M^e)^d \bmod n =$$
$$M^{ed} \bmod n =$$
$$M^{k\phi(n)+1} \bmod n =$$
$$MM^{k\phi(n)} \bmod n =$$
$$M\,(M^{\phi(n)})^k \bmod n =$$
$$M\,(M^{\phi(n)} \bmod n)^k \bmod n =$$
$$M\,(1)^k \bmod n =$$
$$M \bmod n =$$
$$M$$

- Proof of correctness can be extended to the case when the plaintext M divides n

Algorithmic Issues

- The implementation of the RSA cryptosystem requires various algorithms
- Overall
  - –Representation of integers of arbitrarily large size and arithmetic operations on them
- Encryption
  - –Modular power
- Decryption
  - –Modular power

- Setup
  - –Generation of random numbers with a given number of bits (to generate candidates $\boldsymbol{p}$ and $\boldsymbol{q}$)
  - –Primality testing (to check that candidates $\boldsymbol{p}$ and $\boldsymbol{q}$ are prime)
  - –Computation of the GCD (to verify that $\boldsymbol{e}$ and $\phi(\boldsymbol{n})$ are relatively prime)
  - –Computation of the multiplicative inverse (to compute $\boldsymbol{d}$ from $\boldsymbol{e}$)

Cryptographic Hash Functions

Hash Functions

- A hash function h maps a plaintext x to a fixed-length value x = h(P) called hash value or digest of P
  - A collision is a pair of plaintexts P and Q that map to the same hash value, h(P) = h(Q)
  - Collisions are unavoidable
  - For efficiency, the computation of the hash function should take time proportional to the length of the input plaintext
- Hash table
  - Search data structure based on storing items in locations associated with their hash value
  - Chaining or open addressing deal with collisions
  - Domain of hash values proportional to the expected number of items to be stored
  - The hash function should spread plaintexts uniformly over the possible hash values to achieve constant expected search time

Cryptographic Hash Functions

- A cryptographic hash function satisfies additional properties
  - Preimage resistance (aka one-way)
    - Given a hash value x, it is hard to find a plaintext P such that h(P) = x
  - Second preimage resistance (aka weak collision resistance)
    - Given a plaintext P, it is hard to find a plaintext Q such that h(Q) = h(P)
  - Collision resistance (aka strong collision resistance)
    - It is hard to find a pair of plaintexts P and Q such that h(Q) = h(P)
- Collision resistance implies second preimage resistance
- Hash values of at least 256 bits recommended to defend against brute-force attacks
- A random oracle is a theoretical model for a cryptographic hash function from a finite input domain $\mathcal{P}$ to a finite output domain $\mathcal{X}$
  - Pick randomly and uniformly a function h: $\mathcal{P} \rightarrow \mathcal{X}$ over all possible such functions
  - Provide only oracle access to h: one can obtain hash values for given plaintexts, but no other information about the function h itself

Birthday Attack

- The brute-force birthday attack aims at finding a collision for a hash function h
  - Randomly generate a sequence of plaintexts X_1, X_2, X_3,…
  - For each X_i compute $y_i = h(X_i)$ and test whether $y_i = y_j$ for some $j < i$
  - Stop as soon as a collision has been found
- If there are m possible hash values, the probability that the i-th plaintext does not collide with any of the previous i –1 plaintexts is $1 – (i – 1)/m$
- The probability F_k that the attack fails (no collisions) after k plaintexts is

$$F_k = (1 – 1/m)\ (1 – 2/m)\ (1 – 3/m)\ \ldots\ (1 – (k – 1)/m)$$

- Using the standard approximation $1 – x \approx e^{-x}$

$$F_k \approx e^{-(1/m + 2/m + 3/m + \ldots + (k-1)/m)} = e^{-k(k-1)/2m}$$

- The attack succeeds/fails with probability ½ when $F_k = ½$, that is,

$$e^{-k(k-1)/2m} = ½$$

$$k \approx 1.17\ m^{½}$$

- We conclude that a hash function with b-bit values provides about b/2 bits of security

Message-Digest Algorithm 5 (MD5)

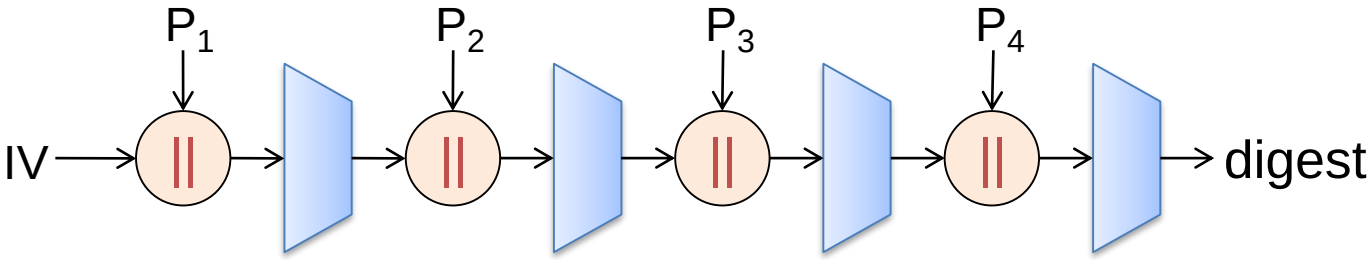
- Developed by Ron Rivest in 1991
- Uses 128-bit hash values
- Still widely used in legacy applications although considered insecure
- Various severe vulnerabilities discovered
- Chosen-prefix collisions attacks found by Marc Stevens, Arjen Lenstra and Benne de Weger
  - Start with two arbitrary plaintexts P and Q
  - One can compute suffixes S1 and S2 such that P||S1 and Q||S2 collide under MD5 by making 250 hash evaluations
  - Using this approach, a pair of different executable files or PDF documents with the same MD5 hash can be computed

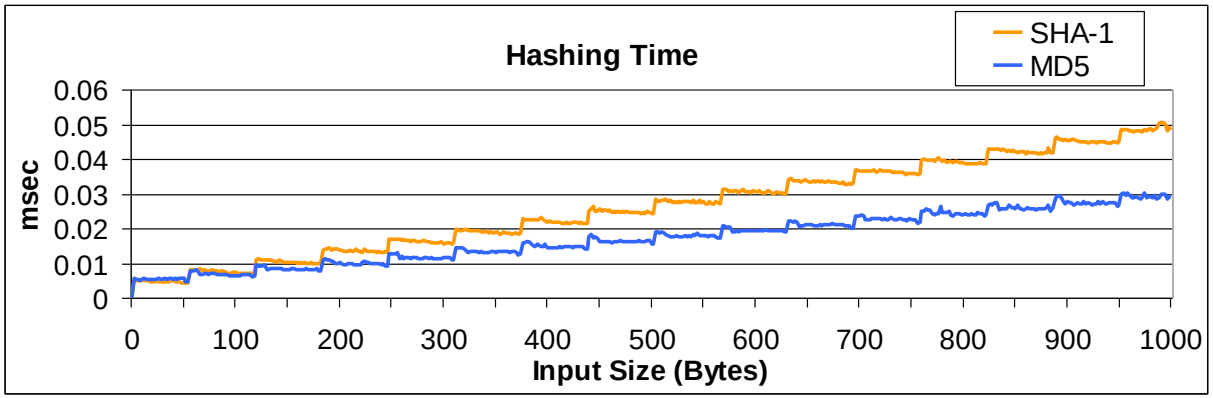
Secure Hash Algorithm (SHA)

- Developed by NSA and approved as a federal standard by NIST
- SHA-0 and SHA-1 (1993)
  - 160-bits
  - Considered insecure
  - Still found in legacy applications
  - Vulnerabilities less severe than those of MD5
- SHA-2 family (2002)
  - 256 bits (SHA-256) or 512 bits (SHA-512)
  - Still considered secure despite published attack techniques
- Public competition for SHA-3 announced in 2007

Iterated Hash Function

- A compression function works on input values of fixed length
- An iterated hash function extends a compression function to inputs of arbitrary length
  - padding, initialization vector, and chain of compression functions
  - inherits collision resistance of compression function
- MD5 and SHA are iterated hash functions





Data Integrity: Applications of Cryptographic Hash Functions

Message Authentication Code (MAC)

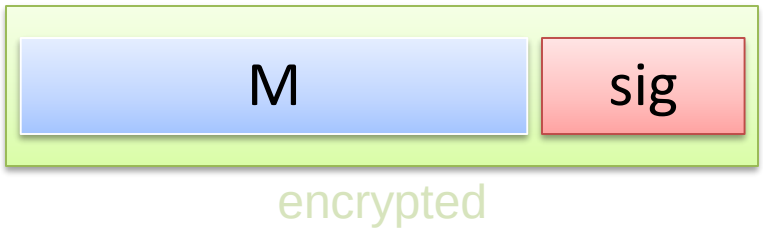
- Cryptographic hash function h(K,M) with two inputs:
  - Secret key K
  - Message M
- Message integrity with MAC
  - Sequence of messages transmitted over insecure channel
  - Secret key K shared by sender and recipient
  - Sender computes MAC c = h(K,M) and transmits it along with message M
  - Receiver recomputes MAC from received message and compares it with received MAC
  - Attacker cannot compute correct MAC for a forged message
  - More efficient than signing each message
  - Secret key can be sent in a separate encrypted and signed message

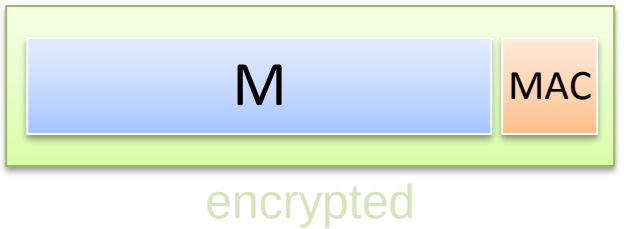
HMAC

- Building a MAC from a cryptographic hash function is not immediate
- Because of the iterative construction of standard hash functions, the following MAC constructions are insecure:
  - $h(K \| M)$
  - $h(M \| K)$
  - $h(K \| M \| K)$
- HMAC provides a secure construction:
  - $h(K \oplus A \| h(K \oplus B \| M))$
  - A and B are constants
  - Internet standard used, e.g., in IPSEC
  - HMAC security is the same as that of the underlying cryptographic hash function

Securing a Communication Channel

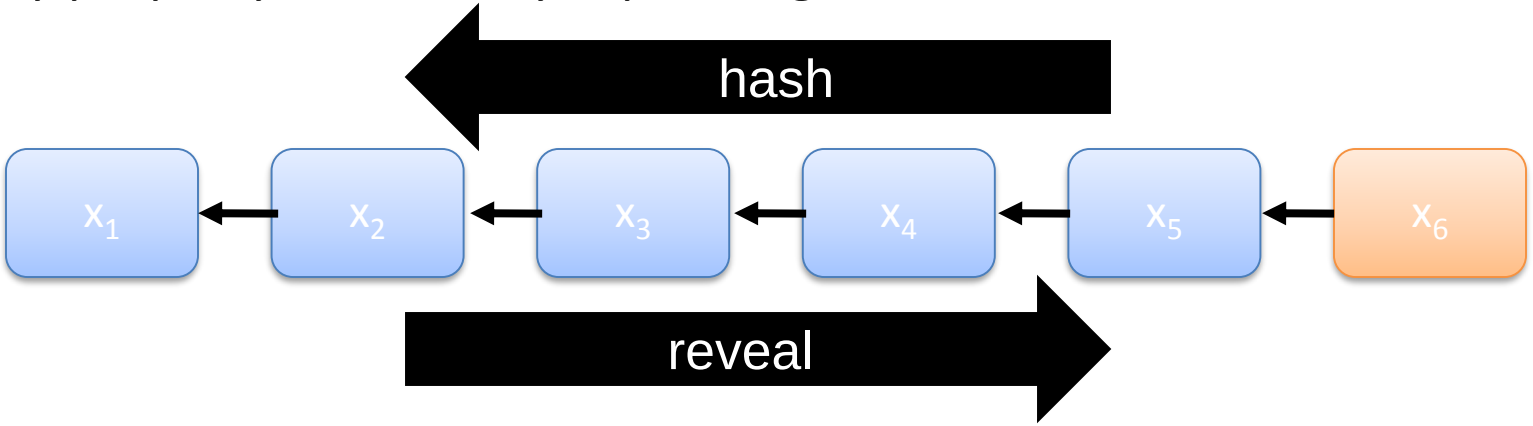
- Assuring both integrity and confidentiality of messages transmitted over an insecure channel
- Sign and encrypt
  - The encrypted pair (message, signature) is transmitted
- MAC and encrypt
  - The encrypted pair (message, MAC) is transmitted
  - Secret key for MAC can be sent in separate message
  - More efficient than sign and encrypt
  - MAC is shorter and faster to compute than signature and verification
- Alternatively, signing or applying MAC could be done on encrypted message





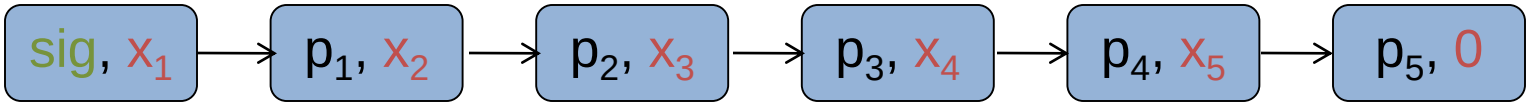
Hash Chain

- Repeated cryptographic hashing starting from a random value r
  - $x_n = r$
  - $x_i = h(x_{i+1})$ for $i = n-1 \ldots 1$
- Sequence $x_1\ x_2\ \ldots\ x_n$ is pseudo-random
- Applications
  - One-time passwords
  - Incremental micropayments (PayWord)
- Key property for security is preimage resistance of hash function



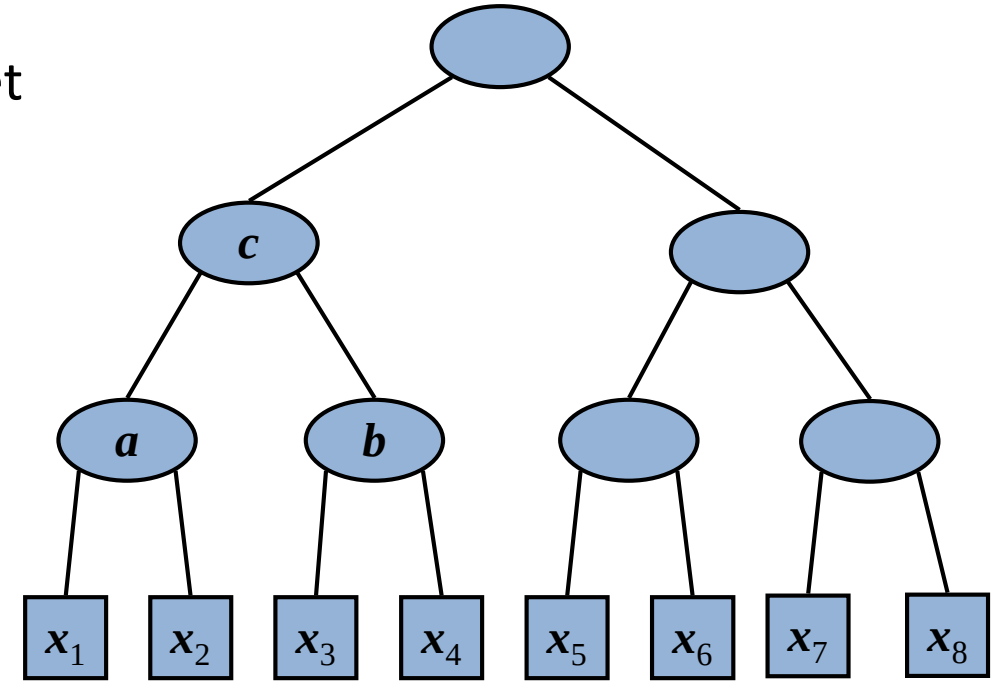
Validation Chain

- Validation chain over a sequence of plaintexts
  $p_1, p_2, \ldots, p_n$
  - $x_{n+1} = 0$
  - $x_i = h(p_i \,||\, x_{i+1})$ for i = n … 1
- Incremental stream authentication [Gennaro Rohatgi]
  - transmit signed x_1
  - transmit packets $(p_1, x_2), (p_2, x_3), \ldots, (p_{n-1}, x_n), (p_n, x_{n+1})$
  - each packet contains the hash of the next packet
  - the integrity of the first hash implies the integrity of the rest
  - any prefix of the stream is signed and cannot be repudiated
  - constant overhead (one hash per plaintext)
  - one signature (slow), n hash computations (fast)
  - offline method, requires reliable transmission



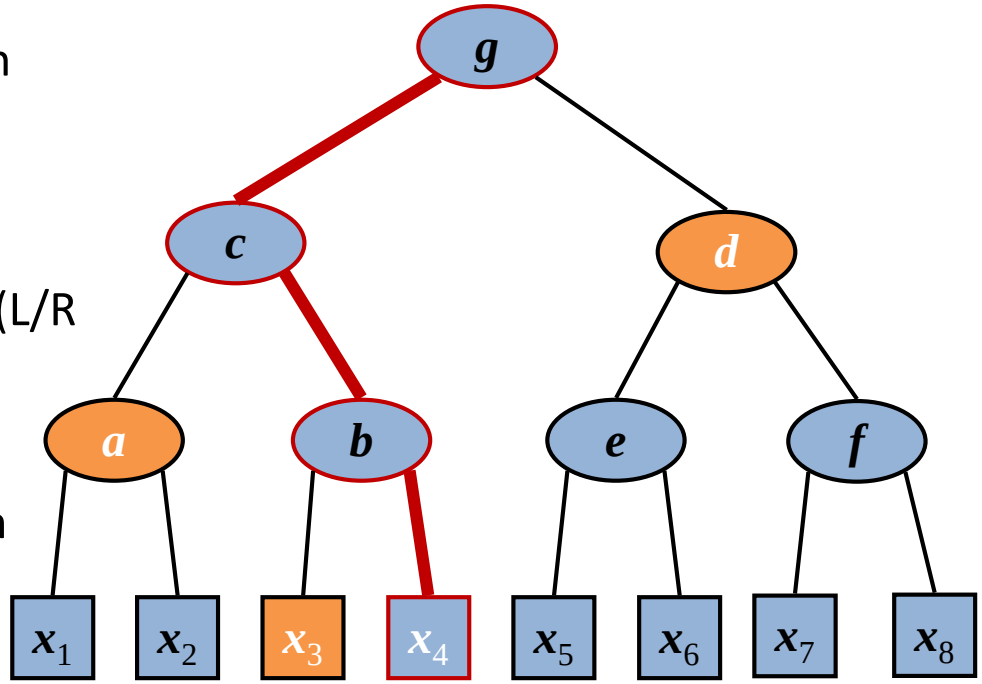
Hash Tree

- Balanced binary tree defining a hierarchical hashing scheme over a set of items
  - $a = h(x_1, x_2)$
  - $b = h(x_3, x_4)$
  - $c = h(a, b)$
  - …
- The root hash is a hierarchical digest of entire set
- [Merkle]



Hash Tree Authentication

- Assumptions
  - Collision resistant hash function
  - Root hash is known
- Membership proof of an item
  - path from the item to the root (L/R sequence) plus hash values of sibling nodes
  - logarithmic size and verification time
- Example
  - $g = h(h(a, h(x_3, x_4)), d)$
  - The proof of x_4 is the sequence $[(x_3, L), (a, L), (d, R)]$



Stream Authentication with Packet Losses

- Sequence of plaintexts to be transmitted
  $p_1, p_2, \ldots, p_n$
- Build a hash tree on top of items (i, p_i)
- Transmit the signed root hash
- For each item p_i, transmit packet
  $(i, p_i, \text{proof}(i,p_i))$
- Logarithmic space overhead and verification time per packet
- Lost packets do not prevent authentication of future packets
- Off-line scheme