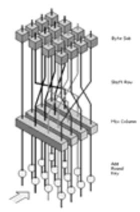
CNS
Lecture 6

Block ciphers -- AES (Rijndael)

Stream ciphers

Key management

review



assignment 5 A

assignment 6

In the news

- Microsoft IE Active X remote code execution
- Microsoft powerpoint remote code execution (0-day)
- MAX OS X multiple vulnerabilities
- OpenSSL ASN.1 remote buffer overflow
- gzip, firefox, Adobe flash player, …

You are here …

Attacks & Defenses
- Risk assessment✓
- Viruses✓
- Unix security✓
- authentication✓
- Network security
  - Firewalls,vpn,IPsec,IDS

Cryptography
- Random numbers✓
- Hash functions✓
  - MD5, SHA,RIPEMD
- Classical + stego✓
- Number theory
- Symmetric key
  - DES, Rijndael, RC5
- Public key
  - RSA, DSA, D-H,ECC

Applied crypto
- SSH
- PGP
- S/Mime
- SSL
- Kerberos
- IPsec

Block cipher modes

- ECB, CBC, CFB, OFB, CTR
- applies to all block ciphers
- Padding, chaining and IV's
- hide repeated plaintext
- different error/attack properties

encryption *does not* guarantee message integrity!

Padding, IV's, and key generation in OpenSSL

- **Encryption will pad to block size of cipher (DES: 8 bytes, AES 16)**
  - E.g., 3 bytes in ➔ 8 encrypted bytes out, 21 in ➔ 24 out
  - May want to pre-pad with random "salt" to obscure same message
- **OpenSSL standard API encryption pads with bytes of IV**
  - EVP methods pads with byte count (PKCS 5)
    and pre-pends 8-byte magic "Salted__" and 8 byte random salt
  - **openssl des-cbc**: 7 bytes in ➔ 24 encrypted bytes out, 8 bytes in ➔ 32 encrypted bytes out
- **Cipher may be attackable if attacker knows IV**
  - Best practice: derive IV from shared secret (asnmt 7) + nonce?
  - Need different IV when restarting encryption
  - OpenSSL EVP optionally derives IV from MD5 of key
- **Converting a password to a key**
  - Assignment 7 uses MD5
  - OpenSSL **EVP_BytesToKey()** generates key and IV from password using MD5

Block ciphers

Feistel
- DES
- Lucifer
- blowfish
- CAST

substitution and permutation

- performance (time/space) vs strength
- large keys
- strong subkey generation
- large blocks
- simple operations, non-linear functions (S-box, rotate)
- iterative, more rounds
- resist known attacks (diff./lin.)
- ciphertext should have uniform distribution (look random)

non-Feistel
- IDEA
- RC2, RC5
- AES (Rijndael)
- For non-feistel need invertible operations

AES

Other crypto bake-offs

Europe: NESSIE

Japan: CRYPTREC

Advanced Encryption Standard
- replace DES
- '97 call for algorithms
  - royalty free, publicly disclosed
  - 128-bit block symmetric key cipher
  - key-sizes: 128, 192, 256

evaluation criteria
- security
- cost/performance (memory, computational efficiency)
- architecture -- simplicity, flexibility
- hardware/software suitability

- '98 15 candidates
- '99 5 finalists (MARS, RC6, Rijndael, Serpent, Twofish)

Mars (IBM)

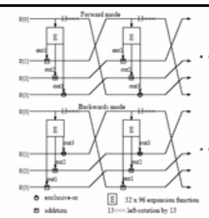


Figure 3: The type-3 Feistel network of the main keyed transformation

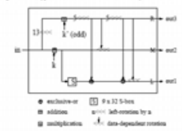


Figure 4: The E-function of the main keyed transformation

- Feistel (1/4, 3/4 rather than even split)
- 8 (unkeyed) pre/post-whitening rounds
  - addition, XOR, and 512x32 S-box (32-bit words)
- 16 keyed rounds
  - addition, XOR, and S-box, multiplication, data-dependent rotations, key addition
- S-box: pseudo-random and tested
- key schedule: linear transform and S-box, with pattern matching to eliminate weak subkeys

keys copied in to T[i]

$$T[i] = T[i] \oplus ((T[i-7 \bmod 15] \oplus T[i-2 \bmod 15]) \lll 3) \oplus (4i+j)$$

$$T[i] = (T[i] + S[\text{low 9 bits of } T[i-1 \bmod 15]]) \lll 9, \quad i = 0, 1, \ldots 14$$

$$K[10j+i] = T[4i \bmod 15], \; i = 0, 1, \ldots, 9$$

serpent

$$\hat{B}_0 := IP(P)$$
$$\hat{B}_{i+1} := R_i(\hat{B}_i)$$
$$C := FP(\hat{B}_{32})$$

where

$$R_i(X) = L(\hat{S}_i(X \oplus \hat{K}_i)) \qquad i = 0, \ldots, 30$$
$$R_i(X) = \hat{S}_i(X \oplus \hat{K}_i) \oplus \hat{K}_{32} \qquad i = 31$$

- substitution-linear transform network
- non-feistel – need inverse S-boxes and inverse transform
- 256-bit key, 128-bit blocks, 32-bit work unit
- bit-slice mode
- 8 4x4 DES-like S-boxes
  - pseudorandom with testing
- 32 rounds (most secure, slow)
  - XOR, S-boxes, linear transform
  - avalanche after 3 rounds
- key schedule: affine recurrence with S-boxes

  33 128-bit subkeys

Linear transform (L)

$$\begin{aligned}
X_0, X_1, X_2, X_3 &:= S_i(B_i \oplus K_i) \\
X_0 &:= X_0 \lll 13 \\
X_2 &:= X_2 \lll 3 \\
X_1 &:= X_1 \oplus X_0 \oplus X_2 \\
X_3 &:= X_3 \oplus X_2 \oplus (X_0 \ll 3) \\
X_1 &:= X_1 \lll 1 \\
X_3 &:= X_3 \lll 7 \\
X_0 &:= X_0 \oplus X_1 \oplus X_3 \\
X_2 &:= X_2 \oplus X_3 \oplus (X_1 \ll 7) \\
X_0 &:= X_0 \lll 5 \\
X_2 &:= X_2 \lll 22 \\
B_{i+1} &:= X_0, X_1, X_2, X_3
\end{aligned}$$

user key w_i

$$w_i := (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) \lll 11$$

$$\begin{aligned}
\{k_0, k_1, k_2, k_3\} &:= S_3(w_0, w_1, w_2, w_3) \\
\{k_4, k_5, k_6, k_7\} &:= S_2(w_4, w_5, w_6, w_7) \\
\{k_8, k_9, k_{10}, k_{11}\} &:= S_1(w_8, w_9, w_{10}, w_{11}) \\
\{k_{12}, k_{13}, k_{14}, k_{15}\} &:= S_0(w_{12}, w_{13}, w_{14}, w_{15}) \\
\{k_{16}, k_{17}, k_{18}, k_{19}\} &:= S_7(w_{16}, w_{17}, w_{18}, w_{19}) \\
&\ldots
\end{aligned}$$

- RC6 (son of RC5)
- Twofish (son of Blowfish)

Rijndael

rhine-doll (or rain doll)

- son of Square cipher
- substitution-linear transform network

  non-feistel (need inverse)
- plaintext block (16 bytes) treated as 4x4 array (state array)
- 10/12/14 rounds (128/192/256 key size)

  S-box, row shifts, column mixing, key-XOR
- key schedule: S-box, constants with XOR, rotates
- 16x16 S-box (256 bytes)
- For decryption need inverse S-box and inverse mixcol
- Rijndael uses polynomial arithmetic (S boxes and mixcol)

  $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ **mod** $(x^4 + 1)$

  coefficients (bytes) a_i are in $GF(2^8)$

Table 5.3 AES Parameters

Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

Rijndael setup – key expansion

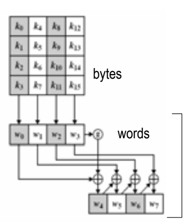


Figure 5.6 AES Key Expansion

- Key expansion (16B key → 176B subkeys)
  - *g*: byte rotate, S-box, XOR with $const_i$
  - Subkey (4 4-byte words) for each round plus one more subkey
  - For AES-128, 11 subkeys (10 rounds + 1)
- Rationale
  - Resistant to attacks
  - Round constants eliminate symmetry
  - Knowledge of some bits of key or subkey does not help calculating other subkeys
  - Diffusion
  - Speed
  - Can generate subkeys on the fly (time/space)

Rijndael setup -- S box setup

- S-box (and inverse S-box) (256-byte) can be calculated or pre-built (time vs memory) (permutation of all 256 bytes)

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

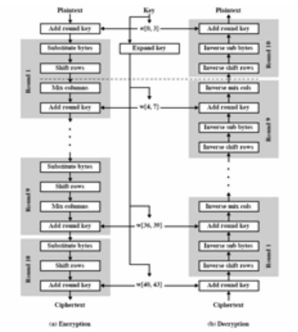
S-box creation:

1. initialize to 00 ... FF
2. calculate multiplicative inverse of each element over $GF(2^8)$ mod $x^8+x^4+x^3+x+1$
3. transform bits of byte with XOR with constant

Example: 97 → 88

- Rationale
  - Low correlation between input and output bits
  - Must be invertible
  - Step 3 prevents interpolation attacks and eliminates S(a)=a and S(a)=a'
  - Only non-linear transform of cipher

Encryption/decryption

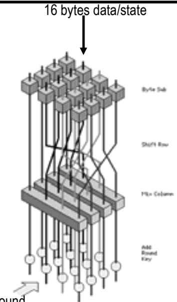


(a) Encryption (b) Decryption

Encryption is not identical to decryption.
Decryption is same sequence of
transformations, but using inverse transforme
Change of subkeys needed
Need rounds+1 subkeys
Note last round has no mixcolumn
First/last step is add round key … why?

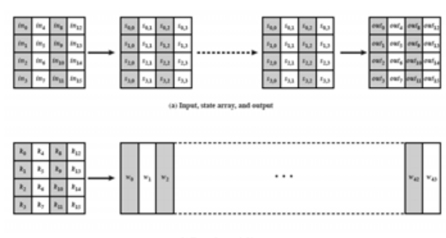
Rinjdael stages

- Byte substitution -- S-box (nonlinear, strength)
- Shift rows – permutation (diffusion)
- Mix columns – substitution using arithmetic over GF(2^8)
  - -multiplies data array with constant array
  - -multiplication is polynomial multiplication mod $x^4 + 1$ over GF(2^8)
  - -- easy in hardware and fast table-lookup in software (256-entry multiplication table)
  - -each element of a column is function of all the elements of the column (mixing)
- Add round key – XOR with round key (simple Vernam/XOR cipher)



-on 32-bit processor, round transformation can be done in set of table lookups
-table lookups contribute to speed of Rijndael and prevent timing/power attacks
- Potential for parallelism and each stage is reversible → decryption works

Rijndael data structures

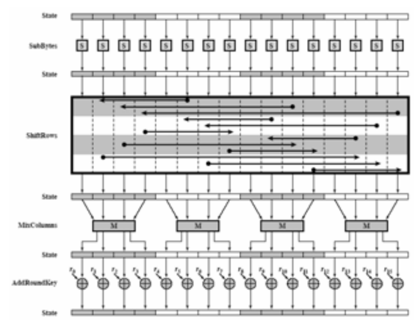


(a) Input, state array, and output

(b) Key and expanded key

- 128-bit (16 byte) data block treated at 4x4 byte array

Rijndael round



Row and column transformations

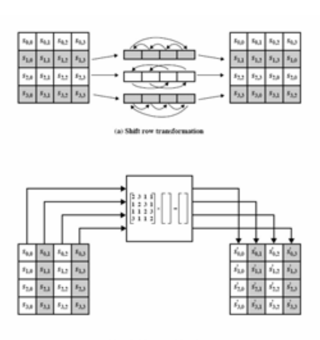
Row transform

- Row 1 -- unchanged
- Row 2 – rotate left 1 byte
- Row 3 – rotate right 2
- Row 4 – rotate right 1

The row transform shifts column values, the 4 bytes of one column are spread out to four different columns.

Mix column

each byte of a column is mapped into a new value that is a function of all four bytes in the column.
matrix multiply is over GF(2^8)

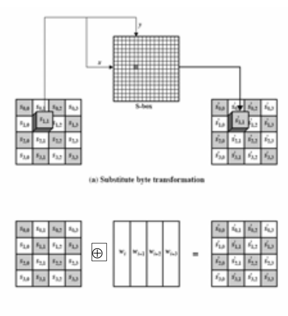


(a) Shift row transformation

(b) Mix column transformation

After a few rounds, all output bits depend on all input bits.

Substitute byte and add round key



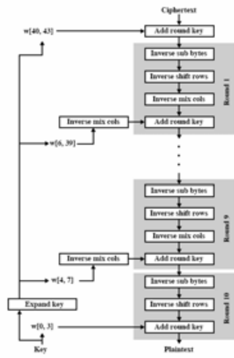
(a) Substitute byte transformation

(b) Add Round Key Transformation

Rijndael decryption

- Each stage is reversible
  - alter direction of shift rows
  - invert Sbox
  - Invert mixcolumn (mod $x^4 + 1$)
    - $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$
    - $a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$
- Modified round key
  - pre-calculate when making subkeys

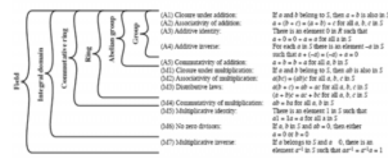
So decryption differs slightly from encryption



The mathematics of cryptography

Finite (discrete) mathematics

- Modular arithmetic (shift ciphers, polyalphabets, Hill cipher)
- Primes and prime factors, greatest common divisor, BIG integer libraries
- Linear transforms (row/column transpositions, linear algebra)
- Exponentiation/discrete logs (D-H, RSA)
- Polynomial arithmetic (CRC, AES, LFSR, ECC)
- Elliptic curves (ECC)



Finite field of dreams

- What we'd like is arithmetic over a finite field
  - Computers do better with finite (discrete) arithmetic
  - Field is associative, commutative, etc, with additive inverse, multiplicative inverse
  - Works for arithmetic mod a prime, e.g (5/4) mod 7 = 3
  - But computer "words" are usually powers of 2, (5/4) mod 8 = ☹
- Stay tuned … corn fields, wheat fields, Galois fields ☺

w	$-w$	w^{-1}
0	0	—
1	7	1
2	6	—
3	5	3
4	4	—
5	3	5
6	2	—
7	1	7

(c) Additive and multiplicative inverses modulo 8

w	$-w$	w^{-1}
0	0	—
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

(c) Additive and multiplicative inverses modulo 7

Rijndael and polynomial arithmetic

- Rijndael utilizes polynomial arithmetic in two ways (really one)
  - Invertible arithmetic (finite field) over 8-bit numbers (+ ×)
    - invertible S-box
    - linear transforms (mixcolumn)
  - 4-byte arithmetic (constant poly is relatively prime to $x^4 + 1$)
    - Invert mixcolumn mod $(x^4 + 1)$
    - $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$

Polynomial arithmetic used by CRC's, Rijndael, LFSR's, ECC

Polynomial arithmetic

- addition and multiplication

  $(3x^4 + 5x^2) + (x^3 + 6x^2) = 3x^4 + x^3 + 11x^2$

  $(3x^4 + 2x^3)(x + 1) = 3x^5 + 5x^4 + 2x^3$

- division: $3x^7 + x^5 + x^2 - 2$ divided by $x^4 - 1$
  equals $3x^3 + x$ with remainder $3x^3 + x^2 + x - 2$
- Coefficients can be integers or mod p (Z_p)
  $((3x^4 + 5x^2) + (x^3 + 6x^2)) \bmod 7 = (3x^4 + x^3 + 11x^2) \bmod 7 = 3x^4 + x^3 + 4x^2$

see CRC reading or Rijndael spec or text Ch 4 and Ch. 5 appendix

Polynomial arithmetic over GF(2)

Galois fields: finite fields of order p^n, written $GF(p^n)$

- hardware influence: use <u>coefficients</u> mod 2
  - Think of each bit as a coefficient
  - addition/subtraction is XOR, multiply is AND
- polynomial: $x^3 + x + 1$ is 1011
  $(x + 1)(x^2 + x) = x^3 + x^2 + x^2 + x = x^3 + x$
- Hardware/software: fast, XOR and table lookups

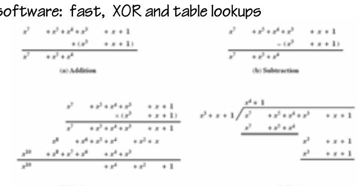


Figure 4.4 Examples of Polynomial Arithmetic over GF(2)

Modular polynomial arithmetic over GF(2^n)

- Do polynomial multiplication mod an irreducible polynomial
- Notion of prime/irreducible polynomials

  f(x) is irreducible if it cannot be expressed as product of two polynomials

  GF(2^8) has 30 irreducible polynomials, Rijndael uses the first one ☺

  $x^8 + x^4 + x^3 + x + 1$

- Primitive polynomials (a subset of irreducible polynomials) generate all elements of an extension field from a base field (used in LFSR)

  Over GF(2^n) there are $\phi(2^n - 1) / n$ primitive polynomials

  $\phi(n)$ number of elements relatively prime to n (Euler's totient)

n	primitive polynomials
1	$x+1$
2	$x^2 + x + 1$
3	$x^3 + x + 1$, $x^3 + x^2 + 1$
4	$x^4 + x + 1$, $x^4 + x^3 + 1$ ($x^4 + x^3 + x^2 + 1$ is irreducible but not primitive)

poly arithmetic in GF(2^8)

- Extended Euclid algorithm can be used to find the multiplicative inverse of a polynomial (exists if mod irreducible polynomial)

Table 4.7 Extended Euclid [$(x^7 + x + 1)$, $(x^8 + x^4 + x^3 + x + 1)$]

Initialization	$A1(x) = 1$; $A2(x) = 0$; $A3(x) = x^8 + x^4 + x^3 + x + 1$ $B1(x) = 0$; $B2(x) = 1$; $B3(x) = x^7 + x + 1$
Iteration 1	$Q(x) = x$ $A1(x) = 0$; $A2(x) = 1$; $A3(x) = x^7 + x + 1$ $B1(x) = 1$; $B2(x) = x$; $B3(x) = x^4 + x^3 + x^2 + 1$
Iteration 2	$Q(x) = x^3 + x^2 + 1$ $A1(x) = 1$; $A2(x) = x$; $A3(x) = x^4 + x^3 + x^2 + 1$ $B1(x) = x^3 + x^2 + 1$; $B2(x) = x^4 + x^3 + x + 1$; $B3(x) = x$
Iteration 3	$Q(x) = x^3 + x^2 + x$ $A1(x) = x^3 + x^2 + 1$; $A2(x) = x^4 + x^3 + x + 1$; $A3(x) = x$ $B1(x) = x^6 + x^2 + x + 1$; $B2(x) = x^7$; $B3(x) = 1$
Iteration 4	$B3(x) = \gcd[(x^7 + x + 1), (x^8 + x^4 + x^3 + x + 1)] = 1$ $B2(x) = (x^7 + x + 1)^{-1} \bmod (x^8 + x^4 + x^3 + x + 1) = x^7$

Result: $(x^7 + x + 1)^{-1} = x^7$

or 10000011 x 1000000 = 00000001

modular polynomial arithmetic over GF(2^3)

Finite field GF(2^3) – only two irreducible polynomials: $x^3 + x^2 + 1$ $x^3 + x + 1$

Table 4.6 Polynomial Arithmetic Modulo $(x^3 + x + 1)$

		000	001	010	011	100	101	110	111
	+	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
000	0	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
001	1	1	0	$x+1$	x	x^2+1	x^2	x^2+x+1	x^2+x
010	x	x	$x+1$	0	1	x^2+x	x^2+x+1	x^2	x^2+1
011	$x+1$	$x+1$	x	1	0	x^2+x+1	x^2+x	x^2+1	x^2
100	x^2	x^2	x^2+1	x^2+x	x^2+x+1	0	1	x	$x+1$
101	x^2+1	x^2+1	x^2	x^2+x+1	x^2+x	1	0	$x+1$	x
110	x^2+x	x^2+x	x^2+x+1	x^2	x^2+1	x	$x+1$	0	1
111	x^2+x+1	x^2+x+1	x^2+x	x^2+1	x^2	$x+1$	x	1	0

(a) Addition

Addition is just XOR

		000	001	010	011	100	101	110	111
	×	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
000	0	0	0	0	0	0	0	0	0
001	1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
010	x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
011	$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
100	x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
101	x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
110	x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
111	x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

(b) Multiplication

Multiplication by x (010): shift left, if bit shifted out is 1 XOR in 011

If $m(x) = x^3 + x + 1$ then $x^3 \bmod m(x) = m(x) - x^3 = x + 1$ ↔ 011

Modular poly arithmetic comparison (3-bit)

Table 4.1 Arithmetic Modulo 8

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

(a) Addition modulo 8

w	−w	w^{-1}
0	0	—
1	7	1
2	6	—
3	5	3
4	4	—
5	3	5
6	2	—
7	1	7

(c) Additive and multiplicative inverses modulo 8

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

(b) Multiplication modulo 8

$(5/4) \bmod 8 = (5 * 4^{-1}) \bmod 8$

☹ no solution

Table 4.5 Arithmetic in GF(2^3)

		000	001	010	011	100	101	110	111
	+	0	1	2	3	4	5	6	7
000	0	0	1	2	3	4	5	6	7
001	1	1	0	3	2	5	4	7	6
010	2	2	3	0	1	6	7	4	5
011	3	3	2	1	0	7	6	5	4
100	4	4	5	6	7	0	1	2	3
101	5	5	4	7	6	1	0	3	2
110	6	6	7	4	5	2	3	0	1
111	7	7	6	5	4	3	2	1	0

(a) Addition

w	−w	w^{-1}
0	0	—
1	1	1
2	2	5
3	3	6
4	4	7
5	5	2
6	6	3
7	7	4

(c) Additive and multiplicative inverses

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

(b) Multiplication

mod $x^3 + x + 1$

See previous slide

Multiplication is just shift and XOR

Every element has a multiplicative inverse

Now 5/4 has a solution ☺ = 2

Rijndael poly arithmetic

- Coefficients of polynomial in GF(2^8) represented by 8-bit number (byte)
- Addition of two polynomial is just XOR of the two 8-bit numbers

  $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$

  01010111 + 10000011 = 11010100
- Multiplication is more complicated (need modular reduction), but still is just shifts and XORs

  Rijndael prime poly: $x^8 + x^4 + x^3 + x + 1$

  For multiplication by x, a shift and XOR in 00011011

Mixcolumn does modular poly multiply mod $(x^4 + 1)$ (e.g. 4 bytes, each in GF(2^8))

$(\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}) \bmod (x^4 + 1)$

but it is implemented as multiply and add on GF(2^8)

CRC's and polynomial arithmetic -- sidebar

10011 ⟌ 1010100011010101

- CRC is remainder in dividing message by polynomial

  Think of message as long string of bits (polynomial)
- Can be implemented with XOR's, shifts, and a table lookup

  Fast in hardware. in C

```
r=0; while (len--) r = (r<<8) ^ t[(r >> 24) ^ *p++];
```

- CRC polynomials chosen to detect "common" errors
  - All single bit errors
  - All 2-bit errors
  - N-bit errors bursts
  - Worry about what errors are not detected?
- Some popular CRC polynomials:

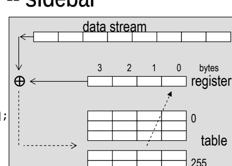
  16 bits:

  (16,12,5,0) X25 standard $x^{16} + x^{12} + x^5 + 1$

  (16,15,2,0) "CRC-16"

  32 bits:

  (32,26,23,22,16,12,11,10,8,7,5,4,2,1,0) Ethernet



1. Shift the register left by one byte, reading in a new message byte.
2. XOR the top byte just rotated out of the register with the next message byte to yield an index into the table ([0,255]).
3. XOR the table value into the register.
4. Goto 1 iff more message bytes.

Rijndael in C

```
/* BC byte count   rk  round key   S  sbox*/
/* plaintext in a  */
KeyAddition(a,rk[0],BC);

/* ROUNDS-1 ordinary rounds */
for(r = 1; r < ROUNDS; r++) {
    Substitution(a,S,BC);
    ShiftRow(a,0,BC);
    MixColumn(a,BC);
    KeyAddition(a,rk[r],BC);
}

/* Last round is special: there is no MixColumn */
Substitution(a,S,BC);
ShiftRow(a,0,BC);
KeyAddition(a,rk[ROUNDS],BC);
```


Rijndael/AES in OpenSSL

- Command line (uses EVP mode, prepend magic and salt, pad with byte count)
  - aes-128-cbc aes-128-ecb aes-192-cbc aes-192-ecb aes-256-cbc aes-256-ecb

```
openssl  aes-256-cbc  -in plain.txt -out enc.dat  -pass pass:boo
```

- API

```
#define AES_BLOCK_SIZE 16
int AES_set_encrypt_key(const unsigned char *userKey, const int bits,
        AES_KEY *key);
int AES_set_decrypt_key(const unsigned char *userKey, const int bits,
        AES_KEY *key);
void AES_cbc_encrypt(const unsigned char *in, unsigned char *out,
        const unsigned long length, const AES_KEY *key,
        unsigned char *iv, const int enc);
```

`enc is either` AES_ENCRYPT or AES_DECRYPT

Result of encryption will be rounded up to a multiple of block size (16) with IV padding. (e.g., you encrypt 3 bytes, you get 16 bytes out)

IV will be updated.

You need to set up the key specifically for encryption and again for decryption, since AES has a different key schedule for encryption and decryption

example

```
#include <openssl/aes.h>

        static const unsigned char key16[16]=
                {0x12,0x34,0x56,0x78,0x9a,0xbc,0xde,0xf0,
                 0x34,0x56,0x78,0x9a,0xbc,0xde,0xf0,0x12};
        unsigned char iv[16],tmpiv[16];

main()
{
        char out[4096],in[4096], *str="123456789abcdefghij";
        AES_KEY aeskey;
         int lth;

        lth = strlen(str) + 1;
         strncpy(in,str, lth);

        AES_set_encrypt_key(key16,128,&aeskey);
        memcpy(tmpiv,iv,sizeof(iv));
        AES_cbc_encrypt(in,out, lth, &aeskey, tmpiv,AES_ENCRYPT);
        AES_set_decrypt_key(key16,128,&aeskey);
        memcpy(tmpiv,iv,sizeof(iv));  //reset IV
        AES_cbc_encrypt(out,in, sizeof(out),&aeskey, tmpiv,AES_DECRYPT);
        printf("%s\n",in);
}
```

See ~dunigan/cns06/aes.c and assignment 7

AES and java (see ~dunigan/cns06/aes.java)

```
 // Get the KeyGenerator
KeyGenerator kgen = KeyGenerator.getInstance("AES");
kgen.init(128); // 192 and 256 bits may not be available
 SecretKey skey = kgen.generateKey();
byte[] raw = skey.getEncoded();
SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
// Instantiate the cipher
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
byte[] encrypted =
   cipher.doFinal((args.length == 0 ?
     "This is just an example" : args[0]).getBytes());
System.out.println("encrypted string: " + asHex(encrypted));
```


Rijndael in hardware

- 8-bit processor (smartcard)
  - addroundkey is a bytewise XOR
  - shiftrows is byte rotates
  - subbytes is table lookup (256 byte table)
  - mixcolumns is XORs and a table lookup (256 byte table)
- 32-bit processor
  - Operate on 32-bit words rather than bytes
  - 4 table lookups and four XORs per column per round (fast)
  - Need 4 256-word (1024 byte) tables
- Can trade off memory space for computation time

AES selection criteria

- security
- software implementations C/java, 8/32/64-bit processors
- ROM/RAM requirements
- hardware implementations ASIC/FPGA
- instruction-level parallelism
- speed
- susceptibility to timing/power attacks
- encryption vs decryption
- key agility (switch keys quickly)
- versatility (block/key/round sizes)

AES assessment

- high security: mars, serpent, twofish
- software: rijndael good 8-64, rc6 good, serpent slow
- key sched: rijndael fast, twofish slow
- space: rijndael/serpent good. mars not.
- hardware: serpent/rijndael good. mars average.
- attacks: serpent/rijndael good. twofish ok. rc6/mars bad
  enc/dec: twofish, mars, rc6 good. rijndael ok. serpent last.
- key agility: twofish/serpent good. rijndael ok. rc6 last.
- parallelism: rijndael best.

- Votes:
  rijndael(86), serpent(59), twofish(31), rc6(23), mars(13)

Choosing AES
(Table from Twofish Paper)

Cipher	Speed (32)	Speed (8)	Safety Factor	Simplicity (code size)
Serpent	62	69	3.56	341 KB
MARS	23	34	1.90	85 KB
RC6	15	43	1.18	48 KB
Rijndael	18	20	1.11	98 KB
Twofish	16	18	2.67	104 KB

(cycles/byte encrypt)

Rijndael AES evaluation

General Security
Rijndael has no known security attacks. Rijndael uses S-boxes as nonlinear components. Rijndael appears to have an adequate security margin, but has received some criticism suggesting that its mathematical structure may lead to attacks. On the other hand, the simple structure may have facilitated its security analysis during the timeframe of the AES development process.

Software Implementations
Rijndael performs encryption and decryption very well across a variety of platforms, including 8-bit and 64-bit platforms, and DSPs. However, there is a decrease in performance with the higher key sizes because of the increased number of rounds that are performed. Rijndael's high inherent parallelism facilitates the efficient use of processor resources, resulting in very good software performance even when implemented in a mode not capable of interleaving. Rijndael's key setup time is fast.

Restricted-Space Environments
In general, Rijndael is very well suited for restricted-space environments where either encryption or decryption is implemented (but not both). It has very low RAM and ROM requirements. A drawback is that ROM requirements will increase if both encryption and decryption are implemented simultaneously, although it appears to remain suitable for these environments. The key schedule for decryption is separate from encryption.

Hardware Implementations
Rijndael has the highest throughput of any of the finalists for feedback modes and second highest for non-feedback modes. For the 192 and 256-bit key sizes, throughput falls in standard and unrolled implementations because of the additional number of rounds. For fully pipelined implementations, the area requirement increases, but the throughput is unaffected.

Rijndael AES evaluation cont.

Attacks on Implementations
The operations used by Rijndael are among the easiest to defend against power and timing attacks. The use of masking techniques to provide Rijndael with some defense against these attacks does not cause significant performance degradation relative to the other finalists, and its RAM requirement remains reasonable. Rijndael appears to gain a major speed advantage over its competitors when such protections are considered.

Encryption vs. Decryption
The encryption and decryption functions in Rijndael differ. One FPGA study reports that the implementation of both encryption and decryption takes about 60% more space than the implementation of encryption alone. Rijndael's speed does not vary significantly between encryption and decryption, although the key setup performance is slower for decryption than for encryption.

Key Agility
Rijndael supports on-the-fly subkey computation for encryption. Rijndael requires a one-time execution of the key schedule to generate all subkeys prior to the first decryption with a specific key. This places a slight resource burden on the key agility of Rijndael.

Other Versatility and Flexibility
Rijndael fully supports block sizes and key sizes of 128 bits, 192 bits and 256 bits, in any combination. In principle, the Rijndael structure can accommodate any block sizes and key sizes that are multiples of 32, as well as changes in the number of rounds that are specified.

Potential for Instruction-Level Parallelism
Rijndael has an excellent potential for parallelism for a single block encryption.

AES vs DES

- Can you match the DES steps with the Rijndael steps?

DES steps	Rijndael
Generate sub-keys	Shift-rows
Permutations	Mix column
XOR sub-key with right half	Byte substitution
XOR f() (S-box) with right half	Add round key
Swap left and right	Generate round keys

DES: substitution & permutation (tuned only for hardware)
AES (Rijndael): substitution & linear transform

Whirlpool (hash function)

- Uses AES-like encryption function (W) to mix bits
  - Based on polynomial arithmetic but fast (shift's and XOR's)
  - Added to OpenSSL 0.9.9
- 512-bit hash (faster than SHA-512)
- More secure? ... test of time

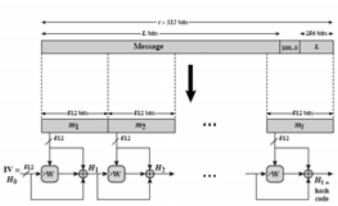


Figure 12.6 Message Digest Generation Using Whirlpool

AES and whirlpool hash

Table 12.2 Comparison of Whirlpool Block Cipher W and AES

	W	AES
Block size (bits)	512	128
Key size (bits)	512	128, 192, or 256
Matrix orientation	Input is mapped row-wise	Input is mapped column-wise
Number of rounds	10	10, 12, or 14
Key expansion	W round function	dedicated expansion algorithm
$GF(2^8)$ polynomial	$x^8 + x^4 + x^3 + x^2 + 1$ (011D)	$x^8 + x^4 + x^3 + x + 1$ (011B)
Origin of S-box	recursive structure	multiplicative inverse in $GF(2^8)$ plus affine transformation
Origin of round constants	Successive entries of the S-box	elements 2^i of $GF(2^8)$
Diffusion layer	right multiplication by 8×8 circulant MDS matrix (1, 1, 4, 1, 8, 5, 2, 9) - mix rows	left multiplication by 4×4 circulant MDS matrix (2, 3, 1, 1) - mix columns
Permutation	shift columns	shift rows

Block cipher advances

- Variable key length
- Mixed operators (non-linear) (Bent functions)
- Key/Data-dependent rotations (RC5)
- Key-dependent S boxes (blowfish)
- Round-dependent functions
- Whitening (XOR key material before first round and after last round)
- Complex sub-key generation (blowfish)
- Variable block lengths and rounds and substitution
- Operate on both halves (blowfish/RC5)
- Mitigate linear/differential cryptanalysis
- Optimized for hardware/software

Block cipher summary

Algorithm	Key Length	Width (bits)	Rounds	Cycles	Clocks/Byte
Twofish	variable	128	16	8	18.1
Blowfish	variable	64	16	8	19.8
Square	128	128	8	8	20.3
RC5-32/16	variable	64	32	16	24.8
CAST-128	128	64	16	8	29.5
DES	56	64	16	8	43
Serpent	128, 192, 256	128	32	32	45
SAFER (S)K-128	128	64	8	8	52
FEAL-32	64, 128	64	32	16	65
IDEA	128	64	8	8	74
Triple-DES	112	64	48	24	116

Table 9: Performance of different block ciphers (on a Pentium)

substitution/permutation

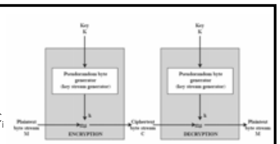
- 64-bit or 128-bit blocks
- DES
  - small key ☹ (use 3DES)
  - test of time
  - widely available
  - S boxes are strength
- IDEA -- optimized for hardware/software
- blowfish -- key-dependent S-boxes, fast
- RC5 -- input-dependent rotations
- CAST -- nonlinear S-boxes, round-dependent functions
- Rijndael – S-boxes, optimized for hardware/software, AES winner ☺

Need key,padding, IV, and ECB/CBC/OFB/CFB/CTR

CCM – encryption and authentication

- Use 128-bit encryption (AES) to do both authentication and encryption in one pass using same key (RFC 3610, NIST 800-38C)
- Use AES-CBC to calculate authenticator over message and nonce (e.g., message number)
- Append authenticator to message
- Use AES in counter mode to encrypt message and authenticator
- Two encryptions per message block
- Many parameters to select, prepend mode to message
- Used in 802.16 (wireless MAN, WiMAX)
- Though it is best practice to use different key for authentication and encryption, it's OK here to use one key because of "shared" nonce
- Remember encryption does not provide authentication

Stream ciphers



- encrypt a byte/bit at a time (telecomm)
  - XOR plaintext with keystream $P_i \oplus K_i \rightarrow C_i$
    Decrypt $C_i \oplus K_i \rightarrow P_i$
    keystream == pseudorandom number generator
- efficient in hardware
- much theoretical analysis (LFSR's)
- faster than block ciphers (hardware)
- synchronous (independent of plain/cipher), pad/OFB
- asynchronous (feedback) CFB
- easily misused ☹
- examples (many proprietary)
  - one-time pad
  - hash PRNG's
  - OFB/CFB (can make a block a stream)
  - PKZIP
  - RC4 (in 802.11, PPTP, Lotus Notes, CDPD, SQL, ssh,WORD/Excel)
  - A5 (3 LFSR's) in Europe's GSM cell phone, US cellular ORYX 3 32-bit LFSRs
  - E0 (4 LFSR's) for bluetooth

Encryption with a hash function

- (pre) compute a (pseudo) one-time pad (keystream)
  b_1 = Hash(key, IV)
  b_i = Hash(key,b_{i-1})
- $c_i = p_i \oplus b_i$
- $p_i = c_i \oplus b_i$
- why IV? why use key each time?
- stream cipher (byte at a time)
- exportable
- used by RADIUS/TACACS+
- error properties:
  - change a bit in c_i?
  - lose a c_i?

Stream cipher from a block cipher

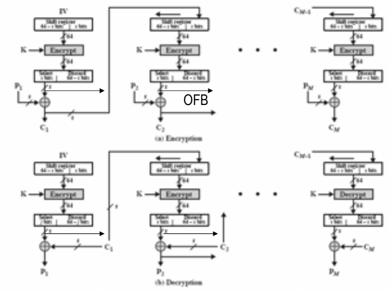


Figure 3.13 s-bit Cipher Feedback (CFB) Mode

- Either OFB or CFB mode or CTR
- Can pre-compute key stream in OFB or CTR

RC4

trademark of RSA

- used in several products (WEP, SSL, WORD)
- fast
- synchronous, 8x8 S box (evolves)

Cipher	Key Length	Speed (Mbps)
DES	56	9
3DES	168	3
RC2	variable	0.9
RC4	variable	45

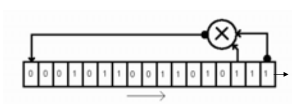
```
fill S box with 0 to 255 and take key K
j=0
for i = 0 to 255      // rearrange Sbox according to key
  j = (j + S_i + K_i)  mod 256
  swap S_i and S_j

//stream generation
i,j=0
while (true)
  i = (i+1) mod 256
  j = (j+ S_i) mod 256
  swap S_i and S_j           // rearrange Sbox
  t = (S_i + S_j)  mod 256
  b = S_t
XOR plain/cipher text  byte with b
```


LFSR keystream

linear feedback shift registers

- efficient in hardware (shift XOR)
- where to tap (connection polynomial)
- full period (2^n - 1) if polynomial is primitive mod 2
- example, $x^4 + x + 1$ (1,0,0,1)
- n-bit "key" is initial setting (seed)
- can solve single LFSR, so use several
- shrinking generator -- use output of first LFSR to select/drop bit of 2nd LFSR
- GSM's A5 uses XOR of 3 LFSRs
- US cellular ORYX 3 32-bit LFSRs

applet animation



$x^{17} + x^{15} + 1$

Figure 6.5: The LFSR $(4, 1 + D + D^4)$ of Example 6.10.

sequence. The following tables show the contents of the stages D_3, D_2, D_1, D_0 at the end of each unit of time t when the initial state is [0, 1, 1, 0].

t	D_3	D_2	D_1	D_0
0	0	1	1	0
1	0	0	1	1
2	1	0	0	1
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	0	0	0
7	1	1	0	0

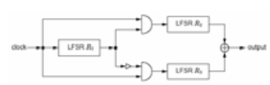
t	D_3	D_2	D_1	D_0
8	1	1	1	0
9	1	1	1	1
10	0	1	1	1
11	1	0	1	1
12	0	1	0	1
13	1	0	1	0
14	1	1	0	1
15	0	1	1	0

The output sequence is s = 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, . . . , and is periodic with period 15 (see Definition 5.25). □

Combining LFSR's

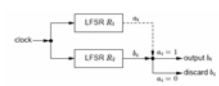
Alternating step generator

select R_2 or R_3 based on R_1

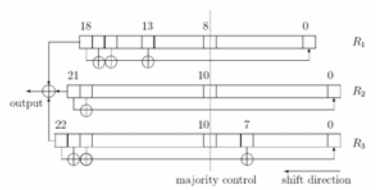


Shrinking generator

select R_2 only when R_1 outputs a 1



GSM A5



- 3 LFSR's with periods (19,22, and 23 → 64-bit key)
  $x^{19} + x^5 + x^2 + x + 1$ $x^{22} + x + 1$ $x^{23} + x^{16} + x^2 + x + 1$
- Output "clocked" by majority function from taps at 8, 10, and 10
  - Clocked means register is shifted with its new feedback input
- Without clocking, period would be $(2^{19} - 1)(2^{22} - 1)(2^{23} - 1)$, but experiments show really only $4/3\ (2^{23} - 1)$
- Only 70% of seeds produce different keystreams

SNOW

- Version 1 weak, version 2 better
- LFSR (16x32 bits) plus finite state machine (FSM)
- 32-bit operations /output
- 8x8 bit S box
- 128 or 256 bit key
- 128-bit IV

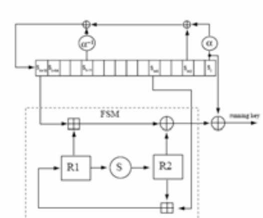
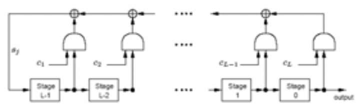


Fig. 2. A schematic picture of SNOW 2.0

LFSR summary



- Fast/simple in hardware
- Subject to correlation attacks with known plaintext
- Need non-linear combinations
- Use "secret" connection polynomial (c_i)

Polynomial arithmetic over GF(2^n) used in Rijndael, CRC's, ECC, and LFSR's

Stream ciphers

- Byte or bit based
- Efficient in hardware (LFSR) based on XOR $c_i = p_i \oplus s_i$
- The ultimate: ONE-TIME PAD, everything else repeats ☹
- Hash-based PRNG's are good approximations
- TROUBLE if you re-use the key stream!
  - If plaintext/ciphertext pair known, you have the keystream
  - If PRNG period too short (WEP/RC4 GSM), key stream will repeat
  - If you have multiple plaintext's encrypted with same keystream (Microsoft Excel/WORD), you can XOR ciphertexts and with word/character frequencies derive plaintexts $P1_i \oplus P2_i$
    see Dawson/Nielsen

Use a block cipher if you can.

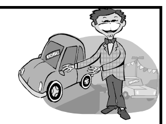
Choosing a cipher

- *depends on application*
- type: stream or block
- block mode: CBC, ECB, OFB, CFB, CTR
- compact (smart card)
- strength (key length, lifetime, test of time)
- license?
- availability/portability
- performance
- error properties (mods/losses) – you need separate integrity check (hash)
- tested, widely used
- Worry about padding and IV
- OpenSSL: DES, 3DES, DESX, blowfish, AES, RC4, Cast
- don't build your own or buy snake oil

Performance	(MBs)
MD5	204
RIPEM	53
SHA	73
Panama	302
IDEA	19
Skipjack	20
DES	21
3DES	10
RC5	59
Blowfish	64
Rijndael	62
RC4	113

2.1GHz pentium 4

Snake oil



- pseudo-mathematical gobbledygook
  - unique in-house developed incremental base shift algorithm
  - virtual matrix of binary values which is infinity in size in theory
  - utilizes DGNT bulk encryption method
- new mathematics -- chaotic functions, neural nets, zeta functions
- revolutionary breakthrough
- proprietary crypto -- trust us
- extreme cluelessness -- unbreakable
- ridiculous key lengths
- one-time pads
- unsubstantiated claims – " *scientifically acclaimed … military grade*"
- security proofs – " *proved as secure as a one-time pad*"
- exportable
- "*not broken by Tom's cns06 students*"
  - cracking contests don't guarantee security

PAIN

Does symmetric key encryption provide:

- Privacy?
- Authenticity?
- Integrity?
- Non-repudiation?
- Availability?
- Virus protection?

Key management

- key generation
- key length
- key lifetime, archiving
- key distribution

PKI (later)

- PKI issues (later)
- key recovery
- protecting keys

Key generation

choose strong keys

- passwords/phrases
  - –length
  - –mixture: upper, lower, special, numerics
  - –good generator/verifier
  - –dictionary attacks
- random keys
  - –unpredictable
  - –random sources (keystrokes, system info, /dev/random)
  - –mixing (MD5, X9.17)
  - –watch out for rand()

Key length

Size matters

- depends on value of information and resources of attacker
- depends on lifetime of secrets
- assume algorithm perfect, then brute force
- one more bit of key, doubles attacker's work factor (exponential)

type	**lifetime**	**key lth bits**
tactical military	minutes	56-64
product announce	days	64
business plan	years	64
trade secrets	decades	112
nuclear secrets	40 yrs	128
spy IDs	50 yrs	128
personal info	50 yrs	128

- 128-bit key:
  using all the computers in the world, and if they could do a million encryptions/sec, it would take a million times the age of the universe!
  (AES key sizes: 128, 192, 256)

Brute force key attacks

symmetric key

- time and cost
- software, FPGA, ASIC
- hacker, corporate, government
- 40-bit key: \$400 FPGA, 5 hours
- EFF DES cracker \$250K, 3 days
- DES key breaking (\$1M/4 hr.) within budget of large corporation or criminal organization
- \$300M, DES keys in 12 seconds
- 75-bit, \$10M/6 yrs, \$300M/70 days
- recommend 75-90 bit keys today , 128

Brute force

- equivalent resistance to attack (key length in bits)

Symmetric	Public
56	384
64	512
80	768
112	1792
128	2304

ref. Schneier

- public key vulnerable to improvements in factoring algorithms (or discrete logs)

Key lifetime

- lifetime is a function of keylength (work factor for brute force)
- the more a key is used, the greater the loss if compromised
- the longer a key is used, the more likely it will be compromised
- lifetime of info (message, signature, file)
- amount of data encrypted can determine lifetime
  - –bad guy accumulates ciphertext for cryptanalysis
  - –for DES, don't send more than 2^{38} bits under the same key
    at 1 Gbit/sec, 5 minutes
- key hierarchy (master key, session key)
  - –Use master key only to encrypt temporary session keys

Updating keys

- Kerberos/PEM/PGP/PKI include ticket/key lifetime fields
- password aging
- public keys, typically 2 yrs max
- may need key archive (key id with material), or re-key material
- archive CRL's too (PKI Certificate Revocation List ...later)
- risk in distributing new keys
  - –need (secure) key renewal, key update protocol
  - –Perfect forward secrecy (don't use old key to generate/send new key)

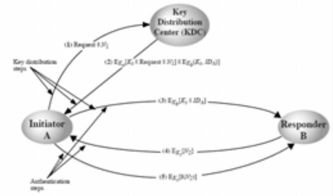
Key distribution

How to get keys (code books, one-time pads) to the end users?

- courier ☹
  - Codebook (enigma) get key of the day

- secret keys -- out of band, splitting
- third party generates key and delivers to A and B
- KDC (Kerberos/DCE) -- must be secure
- key update using old key ☹or key encrypting key/master key ☺
- ISAKMP/Oakley, SKIP, Photuris (IPsec)
  set up key on behalf of application
- Diffie-Hellman (perfect forward secrecy)
- public key cryptography

KDC



key distribution center

- N^2 keys for N nodes
  - one for every pair ☹
- KDC:
  - Just N KDC keys for N users
  - KDC generates session key for Bob and Alice, sends it to them encrypted with their respective KDC key
  - KDC must be secure
- protocol for requesting session key
- More on key distribution when we look at Kerberos

Key recovery

key recovery is when I can find out my key.
Key escrow is when you can find out my key.

- good business sense
- key escrow
  - Clipper chip – big brother →
  - clipper III -- and export carrot
- commercial recovery (Entrust)
- secret sharing (m,n) threshold
  - split secret into n parts (n > m)
  - any m of them can be used to reconstruct secret
  - several algorithms (Schneier) (Shamir: m-degree polynomial)
- part of a PKI? -- encrypt message key with CA's key too?

Key escrow the NSA way

Escrowed Encryption Standard (EES)

- FIPS 185
- permit decrypting bad guys messages
- key would be split in half
- two government agencies would hold halves
- court order would allow gathering two halves
- inducement: US let you use bigger key, if you use EES !
- bad public relations
  - rushed announcement, which agencies?
  - not pre-released to security community
  - not clearly justified
  - no provision for review by Congress, public,...
- Clipper chip – an implementation

Clipper chip

- Alice and Bob create a session key k (Diffie-Hellman, KEA,...)
- clipper encrypts message M with Skipjack $E_k(M)$
- LEAF is transmitted with encrypted message
  Law Enforcement Agent's Field (128 bits)

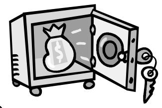
  $E_f(E_u(k),id,ac)$

  - **f** secret family key (chips know, Key Escrow Decrypt Processor knows)
  - **id** is chip id (32b)
  - **ac** authentication code (16b)(used by receiving chip to verify authentic LEAF) receiver won't "decrypt" if LEAF is "bad"
  - **u** chip-specific key
    $u = k1 \oplus k2$ is split between two agencies → given K1 and k2, construct u
- MYK-80 1MB/sec (includes hardware random generator)
- Example, telephone encryptor, link encryptor, FORTEZZA card
- e.g, wiretap captures encrypted message and LEAF, court order gets you k1 and k2

skipjack

- NSA encryption for Clipper chip (FORTEZZA)
- algorithm secret (til '98), hardware tamper-resistant
- reviewed by panel of experts
- strength not dependent on secrecy of algorithm
- design started in '85
- evaluation completed in '90
- specs
  iterative block cipher
  64-bit block
  80 bit key
  32 rounds, XOR
  4x16 bit shift register/counter with 4-round Feistel using 8x8 S box

Protecting keys



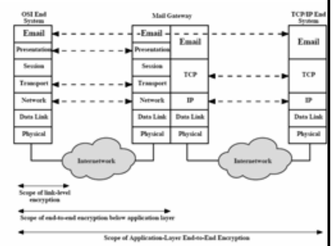
- don't PostIt
- one-time passwords or encrypted channel (ssh)
- encrypt private key
- key renewal
- worry about trojan horses, keyboard sniffers
- strong protection of KDC and CA's
- strong protection of backups
- tamperproof hardware
  - token/Fortezza card
  - CA Certificate Signing Unit (CSU)

Where to encrypt?

link layer
- encrypting modem, net board (wireless)
- transparent,fast
- suitable for private net
- protects only one link (pt-to-pt)
- info may be exposed in OS

network/transport layer
- swIPe, IPv6(IPsec)
- transparent
- selectable (policy)
- appl./host/net keying
- works over public net
- virtual private network (VPN)
- system layer: encrypting file systems (EFS/CFS)

application layer
- end-to-end over public net
- custom applications (PGP, ssh, ssl)
- intrusive, but flexible
- API for application development
- key for every logical circuit



Traffic analysis

encrypted traffic threats

- covert channels
- who's talking to whom
- frequency, event correlation
- quantity, length, patterns of messages
- countermeasures
  - padding messages
  - continuous/random traffic

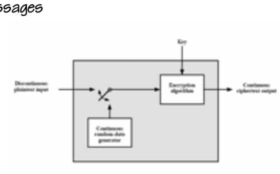


Figure 7.6 Traffic-Padding Encryption Device

review

Attacks & Defenses
- Risk assessment✓
- Viruses✓
- Unix security✓
- authentication✓
- Network security
  Firewalls,vpn,IPsec,IDS

Cryptography
- Random numbers✓
- Hash functions✓
  MD5, SHA,RIPEMD
- Classical + stego✓
- Number theory
- Symmetric key✓
  DES, Rijndael, RC5
- Public key
  RSA, DSA, D-H,ECC

Applied crypto
- SSH
- PGP
- S/Mime
- SSL
- Kerberos
- IPsec

Review

- Lectures

	Lectures
1.	Risk, viruses
2.	UNIX vulnerabilities
3.	Authentication & hashing
4.	Random #s classical crypto
5.	Block ciphers DES, RC5
6.	AES, stream ciphers RC4, LFSR
7.	**MIDTERM** ☹
8.	Public key crypto RSA, D-H
9.	ECC, PKCS, ssh/pgp
10.	PKI, SSL
11.	Network vulnerabilities
12.	Network defenses, IDS, firewalls
13.	IPsec, VPN, Kerberos, secure OS
14.	Secure coding, crypto APIs
15.	review

http://www.cs.utk.edu/~dunigan/cns06/midrvw.txt

Next time …

Mid-term

part 1: take-home (analysis of MISTY1 cipher), work on your own

see midterm (class7.html)

part 2: in class (open book/notes/…) next Tuesday, you'll need your textbook