Computer Security

Modern Cryptography:
Symmetric Block Ciphers

Symmetric Ciphers

- A Symmetric Cipher has five constituents:
  - *Plaintext*
  - An *Encryption Algorithm*
  - A *single Secret Key*
  - *Ciphertext*
  - A *Decryption Algorithm*

Symmetric Ciphers

- Given:
  - Plaintext P
  - Secret Key K
  - Encryption Algorithm E_K
  - Ciphertext C
  - Decryption Algorithm D_K
- We have, as requirements:
  - For encryption:
    $C = E_K(P)$
  - and for decryption:
    $P = D_K(C)$

  so that the process is *reversible*.

- When the *same key* is used for encryption and decryption, the cipher is said to be a *Symmetric Cipher*.
- By necessity, this single key must be kept secret from all but the sender and recipient, so the cipher is also said to be a *Secret Key Cipher*.

Stream Ciphers & Block Ciphers

- A *Stream Cipher* is a cipher that encrypts a data stream one bit or one byte (octet) at a time.
  - Examples:
    - The autokeyed Vigenère cipher
    - The Vernam cipher
- A *Block Cipher* is a cipher which treats a fixed-sized block of plaintext as a whole, and from it creates a ciphertext block of equal length.
  - Typically, the block size is 64 bits (now considered small) or 128 bits
  - Using Modes of Operation (described later), a block cipher can be used to achieve the same effect as a stream cipher

Block Ciphers

- A block cipher operates on a plaintext block of n bits to produce a ciphertext of n bits.
- There are 2^n possible different plaintext blocks
- For the encryption to be *reversible*, each plaintext block must produce a unique ciphertext block
- However, if the block size is too small (e.g. $n = 4$), then the cipher is equivalent to a classical substitution cipher.

Shannon's *Diffusion* and *Confusion*

- Claude E. Shannon, in his seminal paper on cryptography*, suggested two methods for frustrating statistical cryptanalysis:
  - ***Diffusion***
    - Spreads the influence of individual plaintext or key bits over as much of the ciphertext as possible.
    - Hides statistical relationships and makes cryptanalysis more difficult
  - ***Confusion***
    - Hides any relationship between the plaintext, the ciphertext, and the key.
    - Good confusion makes the relationship statistics so complicated that even the most powerful cryptographic tools won't work.

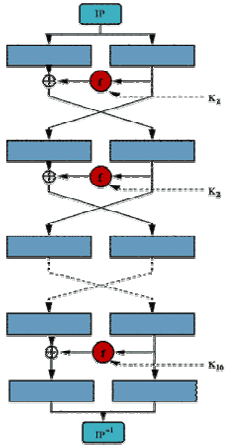
*C. E. Shannon, ***Communication theory of secrecy systems***, Bell System Technical Journal, 1949

The Feistel Cipher

- The *modern block cipher* was invented by Horst Feistel, around 1973. At the time, he was working for IBM.
  - Based on the concept of a *product cipher*, which uses two or more basic ciphers in sequence in such a way that the combined result is cryptographically stronger than any of the component ciphers.
  - In particular, Feistel proposed a cipher that alternates *substitutions* and *permutations*. This is a practical implementation of Shannon's *confusion* and *diffusion* principles.
  - So successful are diffusion and confusion, that they have become the cornerstone of modern block cipher design.

The Feistel Network

- The inputs to a Feistel Network are:
  - A plaintext block of length n bits (n is even)
  - A key K of m bits
- The plaintext is divided into 2 parts, L and R
  - The two halves pass through n ***rounds*** of processing, and finally combine to produce the ciphertext.
- The key, K, is used to generate n ***subkeys***, K_i, each of which is used in a round
  - The subkeys are distinct from each other and from the original key, K.



http://www.freesoft.org/CIE/Topics/143.htm

The Feistel Network

- A Feistel Network is an *iterated block cipher*, where the output of the *i*th round is determined from the output of the (*i*-1)th round:

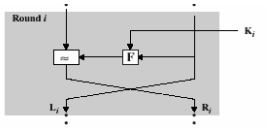
$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$

where K_i is the subkey used in the *i*th round,

and

f is an arbitrary round function



- Because XOR is used to combine the left half with the output of the round function, the operation is reversible:

$$L_{i-1} \oplus f(R_{i-1}, K_i) \oplus f(R_{i-1}, K_i) = L_{i-1}$$

The Feistel Network

- Implementations of a Feistel Network depend on a number of parameters:
  - Block size
    - Larger block sizes provide greater security, but reduced performance.
    - Block sizes of 64 bits or, more recently, 128 bits are reasonable
  - Key size
    - Larger key sizes provide greater security, but may reduce performance
    - Key sizes of 64 bits are now considered to be inadequate; 128 bits is now more common; 256 bits is better.
  - Number of rounds
    - The more rounds, the more security
    - 16 rounds is typical
  - Subkey generation
    - More complexity in how subkeys are generated from the input key provides greater security
  - Round function
    - Greater complexity yields greater resistance to cryptanalysis

The Feistel Network

- Interestingly:
  - A Feistel Network is guaranteed to be invertible as long as the inputs to *f* in each round can be reconstructed.
  - It doesn't matter what *f* is; *f* need not be invertible!
  - So we can design *f* to be as complicated as we wish.
  - Thus, we don't have to implement one algorithm for encryption and a different algorithm for decryption.

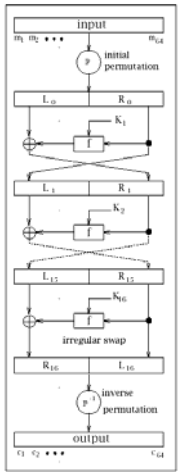
The Data Encryption Standard (DES)

- DES (also called the Data Encryption Algorithm, DEA) is the most widely used encryption scheme
  - In 1973, the National Bureau of Standards (NBS) -- now called the National Institute of Standards and Techology (NIST) -- issued a request for proposals for a national cipher standard.
  - IBM submitted a proposal based on Horst Feistel's work, known as *Lucifer*. It was adopted, with some modifications influenced by the NSA.
  - DES is a Feistel block cipher which operates on 64 bit blocks.
  - Lucifer originally used a key size of 128 bits, but DES reduced this to 56 bits* (causing lots of controversy and suspicion of NSA's involvement)

***Actually, the function expects 64 bits, but only 56 bits are used.**

The Data Encryption Standard (DES)

- There are 3 phases:
  - An initial permutation (IP)
  - 16 rounds of the same function, involving permutation and substitution functions
  - A final permutation (IP^{-1}), which is the inverse of the initial permutation function



http://www.itl.nist.gov/fipspubs/fip46-2.htm

The Data Encryption Standard (DES)

- The 64 bits of the input block to be enciphered are first subjected to the following permutation, called the *initial permutation* IP:

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

  - That is, the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit.

The Data Encryption Standard (DES)

- The permuted input block is then supplied as input to several rounds of complex key-dependent computation.
- The output of that computation, called the *preoutput*, is then subjected to the following permutation which is the inverse of the initial permutation:

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

The Data Encryption Standard (DES)

- According to our textbook:

  "Since the permutation appears to have no security value, it seems nearly certain that there is no security significance to this particular permutation." (Kaufman, Perlman, & Speciner, p. 67)

  and:

  "We hope you appreciate the time we spent staring at the numbers and discovering this completely useless structure." (p. 66)

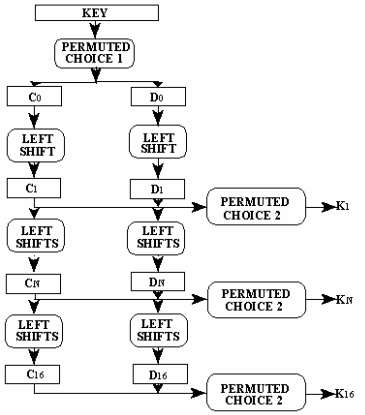
The Data Encryption Standard (DES)

- Per-Round Key Generation
  - The input key, K, is 64 bits -- however...
  - Every 8th bit of K is considered to be a parity bit, which makes the effective key length 64 - 8 = 56 bits.
  - This caused great controversy at the time of DES' adoption, with suspicions raised about the NSA intentionally weakening the design of DES because they already knew how to break it.

- DES performs a function on these bits to generate sixteen 48-bit subkeys, K_1, K_2, ... K_{16}, one for each round ...

The Data Encryption Standard (DES)

- First, an initial permutation* is performed on the useful 56 bits of K, to generate a 56-bit output. This is then divided into two 28-bit values, C_0 & D_0
- The two values are then fed through 16 rounds.
  - Rounds 1, 2, 9, and 16 perform a 1-bit rotate left with carry around to the right.
  - All other rounds use a 2-bit rotate left.
  - The output of each round is a subkey, K_i, permuted from the two values C_i (with 4 bits discarded) and D_i (with 4 bits discarded), producing a subkey of 48 bits.



*For details of the permutation, see the textbook.

A DES Round

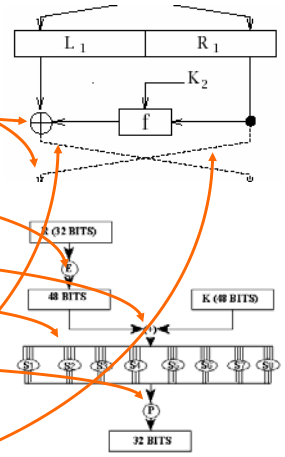
- For each round, i :

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$

where the function f:

- Expands the 32-bit input to 48 bits, using an *expansion permutation*
- XORs (mod 2) the result with K_i, producing a 48 bit value
- Passes the result through a set of eight *S-boxes* (substitution boxes)
- Then performs a permutation on the result, using a *P-box*
- The final result of f is XORed (mod 2) with L_i and the result of that passes into the next round as R_{i+1}
- The original R_i passes unchanged into the next round as L_{i+1}



DES Decryption

- As with any Feistel block cipher, decryption uses the same algorithm as encryption, but with the application of the subkeys reversed.
  - The various component operations of DES were chosen to make this work.

The Avalanche Effect

- We would like to ensure a particular behavior, called an *Avalanche Effect,* in any good encryption algorithm:
  - A small change in either the plaintext or the key should produce a significant change in the ciphertext.
  - In particular, a change in one bit of the plaintext or one bit of the key should change many bits in the ciphertext
- If this were not the case, it could provide a way to reduce the size of the plaintext or keyspace to be searched by an attacker.
- DES exhibits a strong avalanche effect.

Weak and Semi-Weak Keys in DES

- It turns out that there are 16 keys that should not be used with DES:
  - 4 *Weak Keys,* where C_0 and D_0 are *all ones* or *all zeroes*
    - Weak keys are their own inverses (encrypting with one is the same as decrypting with the other)
  - 12 Semi-Weak Keys, where C_0 and D_0 are *alternating ones and zeros* or *alternating zeros and ones*
    - Each semi-weak key is the inverse of one of the other semi-weak keys

The Data Encryption Standard (DES)

- When DES was adopted, there was much controversy over the 56-bit key size.
  - In 1977, Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption devices, each of which could perform one encryption per microsecond. They estimated that a machine costing about $20M in 1977 could crack DES in about 10 hours.
- DES was finally cracked in 1998 by the *Electronic Frontier Foundation*, which built special hardware, using custom chips. The *EFF DES Cracker*, built for less than $250,000, took less than 3 days to crack DES.
- DES is now known to be insufficient for today's environment.

See the Electronic Frontier Foundation's web site documenting this.

Block Cipher Modes of Operation

- A block cipher can encrypt a single block of a fixed size (*n* bits)
- So, how do we encrypt long messages?
  - First, *if the plaintext is not an exact multiple of the block cipher's block size*, we have to perform some form of ***padding***
    - Such padding must be *reversible*, in order to allow decryption
    - The most obvious scheme -- appending zero bytes to *P* until the length is suitable -- is *not reversible*.
  - Once we have suitably padded the plaintext, we then must use some mechanism to create concatenated blocks of ciphertext from the concatenated plaintext blocks. These are called ***Block Cipher Modes of Operation***, and are specified in a FIPS standard (see FIPS PUB 81, DES MODES OF OPERATION*) . Note that they can be used with any block cipher.

*http://www.itl.nist.gov/fipspubs/fip81.htm

Block Cipher Modes of Operation

- Ferguson and Schneier* suggest the following:
  - Let P be the plaintext, $l(P)$ be the length of P in bytes, and b be the block size of the block cipher in bytes
  - Using one of the following padding schemes:
    - Append to P a single byte with a value of 128, and then as many zero bytes as necessary to make the overall size a multiple of b.
    - Determine the number, n, of padding bytes needed.

      $1 \le n \le b$ and $n + l(P)$ is a multiple of b

      Pad P by appending n bytes, each with value n.

**Practical Cryptography*, by Niels Ferguson & Bruce Schneier, Wiley

Electronic Codebook Mode (ECB)

- The most straightforward way of encrypting a longer plaintext is to use ***Electronic Codebook mode***, or ***ECB***.

$C_i = E_K(P_i)$ for $i = 1, \ldots k$

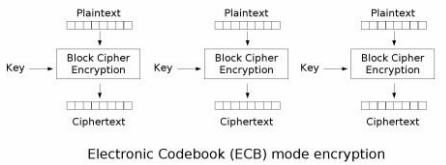
where C_i is the ith ciphertext block

P_i is the ith plaintext block

E is the encryption algorithm

K is the key

k is the number of plaintext blocks



Electronic Codebook (ECB) mode encryption

`http://en.wikipedia.org/wiki/Electronic_codebook`

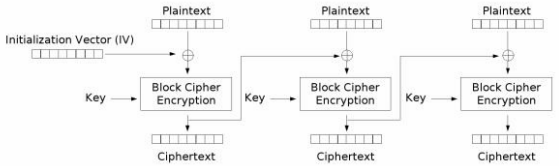
Electronic Codebook Mode (ECB)

- Unfortunately, ECB has serious shortcomings:
  - If two plaintext blocks are the same, then the corresponding ciphertext blocks will be the same, which is visible to an attacker. Depending on the structure of the message, this can leak a lot of information.
  - *It is strongly recommended that ECB not be used for message encryption.*

To see a particularly striking example of how weak ECB can be, go to `http://en.wikipedia.org/wiki/Electronic_codebook`

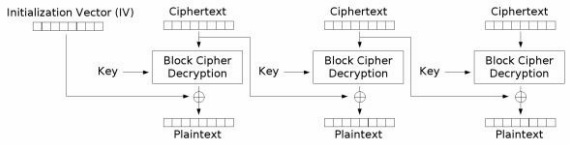
Cipher Block Chaining Mode (CBC)

- Cipher Block Chaining mode (CBC) avoids the problems with ECB by XORing each plaintext block with the previous ciphertext block:



Cipher Block Chaining (CBC) mode encryption

$$C_i = E_K(P_i \oplus C_{i-1})$$



Cipher Block Chaining (CBC) mode decryption

Cipher Block Chaining Mode (CBC)

- This raises the issue of what to do for the first plaintext block, which does not have a previous ciphertext block.
- We use an ***Initialization Vector (IV)***
  - What should we use for an IV?
    - A *Fixed IV* is not a good idea, since it reintroduces the ECB problem for the first plaintext block. Messages often start with similar or identical blocks, so this is to be discouraged.
    - A *Counter IV* (e.g. 0, 1, ...) is not a good idea, either, because it can open up the block to easier attack

Cipher Block Chaining Mode (CBC)

- What should we use for an IV?
  - A *Random IV* is better, but how will the recipient of the message know the random number used?
    - One solution is to generate a random IV and prepend it as the first block of the plaintext.
    - To do this, we need a cryptographic strength random number generator, which is not easy to implement
    - Also, this adds one ciphertext block to the size of every message, which is never a good idea, especially for short messages.

Cipher Block Chaining Mode (CBC)

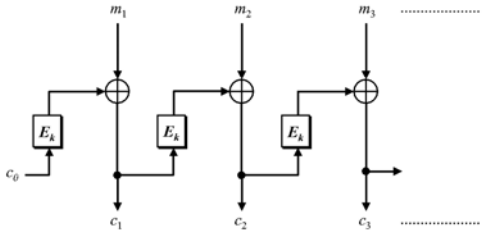
- What should we use for an IV?
  - A *Nonce*-Generated IV* is a better solution.
    - Each message to be encrypted is given a unique number called a ***nonce***.
    - In security, ***nonce*** is a contraction of "number used once", and its value must be unique; that is, the value should never be used again.

***nonce**
1 : the one, particular, or present occasion, purpose, or use <for the *nonce*>
2 : the time being

Cipher Feedback Mode (CFB)

- In CFB mode, the previous ciphertext block is encrypted and the output produced is combined with the plaintext block using XOR to produce the current ciphertext block.
- An initialization vector c_0 is used as a *seed* for the process.

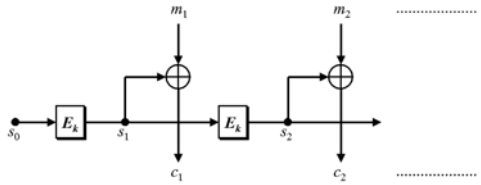
$$C_i = P_i \oplus E_K(C_{i-1})$$



Output Feedback Mode (OFB)

- OFB mode is similar to CFB mode except that the quantity XORed with each plaintext block is generated independently of both the plaintext and ciphertext.
- An initialization vector s_0 is used as a *seed* for a sequence of data blocks s_i, and each data block s_i is derived from the encryption of the previous data block s_{i-1}.

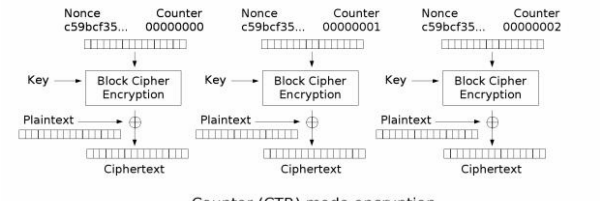
$$C_i = P_i \oplus S_i$$
$$S_i = E_K(S_{i-1})$$



CFB and OFB Modes

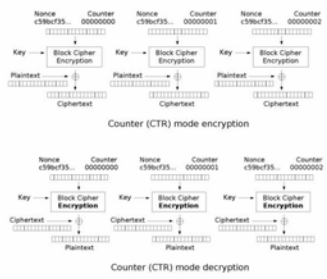
- The CFB and OFB modes *make the block cipher into a stream cipher*
  - They generate keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext.
  - Just as with other stream ciphers, flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location.
- With CFB, a keystream block is computed by encrypting the previous ciphertext block.
- OFB generates the next keystream block by encrypting the last one.

Counter Mode (CTR)



Counter Mode (CTR)

- Counter Mode (CTR) block ciphers use sequence numbers as the input to the algorithm.
- CTR mode has existed for a long time.
- It was not standardized as an official DES mode, but has recently been standardized by NIST.
- Like OFB, it is a stream cipher.



$$K_i = E_K(\text{Nonce} \,||\, i) \quad \text{for } i = 1, \ldots k$$
$$C_i = P_i \oplus K_i$$

What Mode to Use?

- Schneier* suggests:
  - For encrypting random data, such as other keys, ECB is a good choice
  - For normal plaintext, use CBC, CFB or OFB
  - To encrypt files, use CBC
- Ferguson & Schneier (published later):
  - Only recommend use of CBC or CTR
  - Compare CBC and CTR, with CTR coming out on top in most respects.

**Applied Cryptography (2nd Edition)* by Bruce Schneier, Wiley

Triple DES (3DES)

- DES was cracked, and is now considered to be too weak for today's world.
  - ***Triple DES (3DES)*** was an attempt to increase the strength of encryption without having to move to an entirely different algorithm.
  - Two keys are used, K_1 and K_2
  - Each plaintext block P_i is subjected to the following:

$$C_i = E_{K_1}(D_{K_2}(E_{K_1}(P_i)))$$

  - 3DES was adopted as the encrypt-decrypt-encrypt EDE mode of the DES cipher algorithm. The choice of EDE with the keys was deliberate to avoid cryptanalysis problems.
  - The problem with 3DES is that it is slow.

International Data Encryption Algorithm (IDEA)

- Introduced in 1990, by Xuejia Lai and James Massey of ETH Zurich*, under the name Proposed Encryption Standard (PES)
- The algorithm was subjected to cryptanalysis, and some shortcomings were identified. The authors published a strengthened algorithm called Improved Proposed Encryption Standard (IPES) in 1991.
- In 1992, they renamed it International Data Encryption Algorithm (IDEA)
- IDEA is part of PGP (Pretty Good Privacy)

****E**idgenössische **T**echnische **H**ochschule Zürich,
The Swiss Federal Institute of Technology, Zurich

International Data Encryption Algorithm (IDEA)

- IDEA is similar to DES in some ways:
  - Both have rounds
  - Both have a complicated "mangler" function which does not have to be reversible for decryption to work.
- With DES, the same keys are used in reverse for decryption
- With IDEA, the encryption and decryption keys are related in a more complex manner.
- IDEA is a block cipher with a:
  - Block size of 64 bits
  - Key size of 128 bits

International Data Encryption Algorithm (IDEA)

- IDEA is patented by the Swiss company Ascom, but they have been generous in allowing free non-commercial use of their algorithm.
- IDEA avoids the use of any lookup tables or S-boxes (no bit-level permutations)
- Each primitive operation in IDEA maps two 16-bit quantities into a 16-bit quantity
  - By comparison, each DES S-box maps a 6-bit quantity into a 4-bit quantity
- The primitive operations in IDEA are efficient in computers -- even in 16-bit processors

International Data Encryption Algorithm (IDEA)

- IDEA uses 3 operations to create a mapping:
  - $\oplus$ - bitwise exclusive OR
  - $+$ - slightly modified ADD
  - $\otimes$ - slightly modified MULTIPLY

  – The addition is done **mod 2^{16}**

International Data Encryption Algorithm (IDEA)

- The multiplication is done by first calculating the 32-bit result from the two 16-bit inputs, and then taking the remainder **mod $(2^{16} + 1)$**
  - Multiplication mod $(2^{16} + 1)$ is *reversible*, in that every number between 1 and 2^{16} has an inverse in the range 1 to 2^{16} because $2^{16} + 1$ (65,537) *is prime*
  - 0 would not have a multiplicative inverse, and 2^{16} (a valid remainder in mod $2^{16} + 1$ arithmetic) cannot be expressed in 16 bits, so in IDEA a 16-bit number containing all zeros is treated as an encoding for 2^{16}
  - Note that:
    - $(2^4 + 1 = 17)$ and $(2^8 + 1 = 257)$ are both prime, but
    - $(2^{32} + 1)$ is ***not*** prime, so IDEA cannot easily be extended to a 128-bit block size

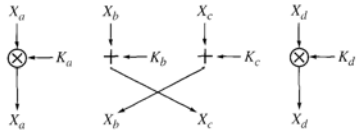
International Data Encryption Algorithm (IDEA)

- The 64-bit plaintext block is divided into 4 16-bit sub-blocks, X_a, X_b, X_c, X_d, which become the inputs to the first round.
- The 128-bit key is expanded into 52 16-bit subkeys, K_i
- There are 17 rounds, even and odd:
  - Odd rounds use four of the keys, K_a, K_b, K_c, K_d
    - For example, round 1 uses K_1, K_2, K_3, K_4
  - Even rounds use two keys, K_e and K_f
    - For example, round 2 uses K_5 and K_6

(Some explanations talk about 8 rounds, where each of those rounds combines the work of two of the above rounds. It can get confusing if you mix different explanations!)

IDEA Rounds

- Odd round: $X_a \otimes K_a \rightarrow X_a$
  $X_b + K_b \rightarrow X_c$
  $X_c + K_c \rightarrow X_b$
  $X_d \otimes K_d \rightarrow X_d$



- Even round:

Mangler function : $f(Y_{\text{in}}, Z_{\text{in}}, K_e, K_f)$

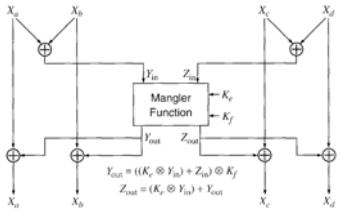
$X_a \oplus X_b \rightarrow Y_{\text{in}}$, $X_c \oplus X_d \rightarrow Z_{\text{in}}$

$((K_e \otimes Y_{\text{in}}) + Z_{\text{in}}) \otimes K_f \rightarrow Y_{\text{out}}$

$(K_e \otimes Y_{\text{in}}) + Y_{\text{out}} \rightarrow Z_{\text{out}}$

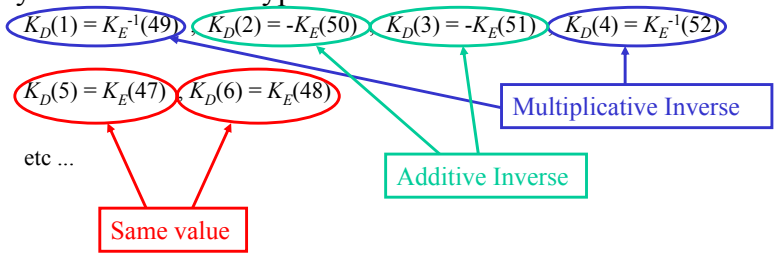
$X_a \oplus Y_{\text{out}} \rightarrow X_a$, $X_b \oplus Y_{\text{out}} \rightarrow X_b$

$X_c \oplus Z_{\text{out}} \rightarrow X_c$, $X_d \oplus Z_{\text{out}} \rightarrow X_d$



IDEA Rounds

- For decryption:
  - For odd rounds, we perform the multiplications with the inverses of the K_i keys, mod $2^{16} + 1$
  - *An even round is its own inverse -- use the same keys*
- Key schedule for decryption:



International Data Encryption Algorithm (IDEA)

- IDEA has been thoroughly cryptanalyzed, and is considered to be a secure cipher -- much better than DES
- However, it has not caught on as much as did DES, and perhaps it is now overshadowed by...

The Advanced Encryption Standard (AES)

- In January, 1997, NIST announced a contest to select a new encryption standard to be used for protecting sensitive, non-classified, U.S. government information.
  - Proposals were accepted from anyone, anywhere in the world
  - 5 finalists were chosen from 15 submissions:
    - MARS, RC6, Rijndael, Serpent, and Twofish

The Advanced Encryption Standard (AES)

- Proposals had to meet a number of specific evaluation criteria. The initial criteria were:
  - **Security:**
    - High effort required to cryptanalyze the algorithm
  - **Cost:**
    - Practical in a wide range of applications
  - **Algorithm and Implementation Characteristics:**
    - Flexibility, suitability for a variety of hardware and software implementations, simplicity (to make the analysis of security easier)

The Advanced Encryption Standard (AES)

- The final criteria used to pick from the 5 finalists were:
  - General Security: Public worldwide security analyses were published
  - Software Implementations
  - Restricted-Space Implementations
  - Hardware Implementations
  - Attacks on Implementations
  - Encryption vs Decryption
  - Key Agility
  - Other Versatility and Flexibility
  - Potential for Instruction-level Parallelism

The Advanced Encryption Standard (AES)

- The DES selection process:
  - Was done in relative secrecy
  - The details and motivation for the algorithm was secret
  - Led to lots of suspicion about the role of NSA and whether DES was truly secure
- In contrast, the AES selection process:
  - Was open
  - NSA was specifically excluded from proposing, although they could provide advice
  - A detailed explanation and analysis of the algorithms was part of the process.
  - This resulted in more confidence about AES

The Advanced Encryption Standard (AES) "Rijndael"

- NIST chose a submission called ***"Rijndael"*** by two Belgian cryptographers -- Joan Daemen & Vincent Rijmen
- In 2001, they published this as the new ***Advanced Encryption Standard (AES)****, ultimately replacing DES.

*http://csrc.nist.gov/CryptoToolkit/aes/rijndael/
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

The Advanced Encryption Standard (AES) "Rijndael"

- From the authors of Rijndael:
  (http://www.esat.kuleuven.ac.be/~rijmen/rijndael/)

  1) How is that pronounced ?

  If you're Dutch, Flemish, Indonesian, Surinamer or South-African, it's pronounced like you think it should be. Otherwise, you could pronounce it like "Reign Dahl", "Rain Doll", "Rhine Dahl". We're not picky. As long as you make it sound different from "Region Deal".

  2) Why did you choose this name ?

  Because we were both fed up with people mutilating the pronunciation of the names "Daemen" and "Rijmen". (There are two messages in this answer.)

  3) Can't you give it another name ? (Propose it as a tweak !)

  Dutch is a wonderful language. Currently we are debating about the names "Herfstvrucht", "Angstschreeuw" and "Koeieuier". Other suggestions are welcome of course. Derek Brown, Toronto, Ontario, Canada, proposes "bob".

The Mathematics of Rijndael

- Rijndael uses arithmetic in the *Galois Field* GF(2^8), a.k.a. GF(256) (the finite field of order 256)
  - Recall that:
    - It can be shown that the order of a finite field (number of elements in the field) must be a power of a prime, p^n, where n is a positive integer.
  - A nice property of GF(2^8) is that each element of the field can be represented by an octet.

The Mathematics of Rijndael

- The bits in the octet are the coefficients of a polynomial over $\mathbf{Z}_2$ modulo the *irreducible* $\mathbf{Z}_2$ polynomial:*

  $m(x) = x^8 + x^4 + x^3 + x + 1$

*A polynomial is called **irreducible** if its only divisors are one and itself. By analogy to integers, an irreducible polynomial is also called a **prime polynomial**.

The Mathematics of Rijndael

- Byte values are represented as polynomials with the least significant bit being the the coefficient of x^0, and the most significant bit the coefficient of x^7.
  - For example, {01100011} identifies the specific field element:

    $x^6 + x^5 + x + 1$
- Some finite field operations involve one additional bit to the left of an 8-bit byte. When this extra bit is present, it appears as {01} to the left of the other 8 bits:

  {01} {00011011}

The Mathematics of Rijndael

- Addition in a finite field is achieved by "adding" the coefficients for the corresponding powers in the polynomials for the two elements.
- Addition is s performed using an XOR operation, denoted by $\oplus$ (recall that addition modulo 2 is equivalent to XOR)
- Subtraction of polynomials is identical to addition of polynomials (recall that, too?)

The Mathematics of Rijndael

- For example:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 + 0 \quad \text{(polynomial notation)}$$

$$\{01010111\} \oplus \{10000011\} = \{11010100\} \quad \text{(binary notation)}$$

$$\{57\} \oplus \{83\} = \{d4\} \quad \text{(hexadecimal notation)}$$

are all equivalent.

The Mathematics of Rijndael

- Multiplication in Rijndael is the multiplication of polynomials *modulo the irreducible polynomial:*

  $m(x) = x^8 + x^4 + x^3 + x + 1$

  or {01}{1b} in hexadecimal notation.

- For example:

$\{57\} \bullet \{83\} = \{c1\}$, because:

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) =$$
$$x^{13} + x^{11} + x^9 + x^8 + x^7 +$$
$$x^7 + x^5 + x^3 + x_2 + x +$$
$$x^6 + x^4 + x^2 + x + 1$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

and:

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x_4 + x^3 + 1$$
$$\text{modulo } (x^8 + x^4 + x^3 + x + 1)$$
$$= x^7 + x^6 + 1$$

The Mathematics of Rijndael

- The modular reduction by $m(x)$ ensures that the result will be a binary polynomial of degree less than 8, and thus can be represented in a byte.
- This multiplication is *associative*
- The element {01} is the *multiplicative identity*.
- For any non-zero binary polynomial $b(x)$ of degree less than 8, the multiplicative inverse of $b(x)$, denoted by $b^{-1}(x)$ can be found using the Extended Euclidean algorithm. This computes polynomials $a(x)$ and $c(x)$ such that:

$$b(x)a(x) + m(x)c(x) = 1,$$
$$\text{hence } a(x) \bullet b(x) \bmod m(x) = 1,$$
$$\text{or } b^{-1}(x) = a(x) \bmod m(x)$$

The Mathematics of Rijndael

- It follows from the above that the set of 256 possible byte values, with XOR used as addition, and the multiplication defined as above, has the structure of the finite field $GF(2^8)$

The Mathematics of Rijndael

- Rijndael also uses polynomials over $GF(2^8)$, taken modulo the $GF(2^8)$ *non-irreducible polynomial* $x^4 + 1$.
  - These polynomials are represented as 4-vectors of octets, with the coefficient of 1 being the first octet in the 4-vector.
  - With this representation, multiplication is simply a rotation.
- Finally, for one of its operations composing its "S-box", Rijndael treats octets as polynomials over $\mathbf{Z}_2$ modulo the *non-irreducible polynomial* $x^8 + 1$

The Advanced Encryption Standard (AES) "Rijndael"

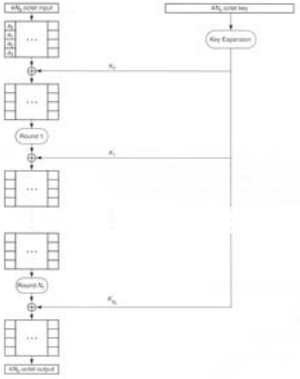
- Originally, Rijndael defined a symmetric block cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits
- The accepted AES allows key sizes of 128, 192, or 256 bits, but restricts the block size to 128 bits
- AES is *not* a Feistel cipher.

The Advanced Encryption Standard (AES) "Rijndael"

- The basic structure provides flexibility with the use of 3 parameters:
  - N_b, the block size (the number of 32-bit words, or 4-octet columns, in a plaintext block). For AES, $N_b = 4$, since the block size is 128 bits
  - N_k, the key size (the number of 32-bit words in an encryption key)
    - 128 bits => $N_k = 4$; 192 bits => $N_k = 6$; 256 bits => $N_k = 8$
  - N_r, the number of rounds
    - Needs to be larger for longer keys
    - Needs to be larger for larger block sizes
    - Rijndael specifies $N_r = 6 + \max(N_b, N_k)$
      - 128-bit key => 10 rounds
      - 192-bit key => 12 rounds
      - 256-bit key => 14 rounds

The Basic Rijndael Structure

- Internally, the algorithm's operations are performed on a 2-dimensional array of bytes called the ***State***.
- The State consists of 4 rows of bytes, each containing N_b bytes, where N_b is the block length/32.
  - In AES, $N_b = 128/32 = 4$ so the State is a 4x4 array of bytes.



The Rijndael State

- At the start of the cipher and inverse cipher, the array of input bytes (plaintext) is copied into the State array.
- The cipher or inverse cipher operations are then conducted on this State array.
- Finally, the State's final value is copied to the output array.

input bytes

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

→

State array

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

→

output bytes

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

The Rijndael Cipher Algorithm

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])              // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                          // See Sec. 5.1.1
      ShiftRows(state)                         // See Sec. 5.1.2
      MixColumns(state)                        // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```


The Rijndael Cipher Algorithm

- **AddRoundKey()**
  - A Round Key is added to the state using XOR
- **SubBytes()**
  - uses S-box to perform a non-linear byte-by-byte substitution of State
- **ShiftRows()**
  - processes the State by cyclically shifting the last three rows of the State by different offsets
- **MixColumns()**
  - takes all the columns of the State and mixes their data, independently of one another, making use of arithmetic over GF(2^8)

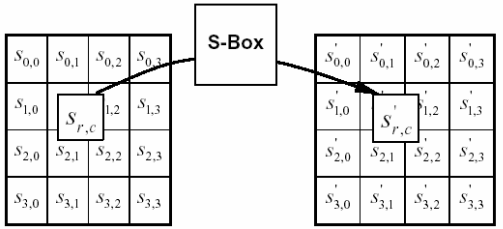

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])              // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                          // See Sec. 5.1.1
      ShiftRows(state)                         // See Sec. 5.1.2
      MixColumns(state)                        // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

The Rijndael Cipher Algorithm

- Only AddRoundKey() makes use of the key
- The other three functions are used to produce diffusion and confusion
- The final round omits MixColumns transformation.

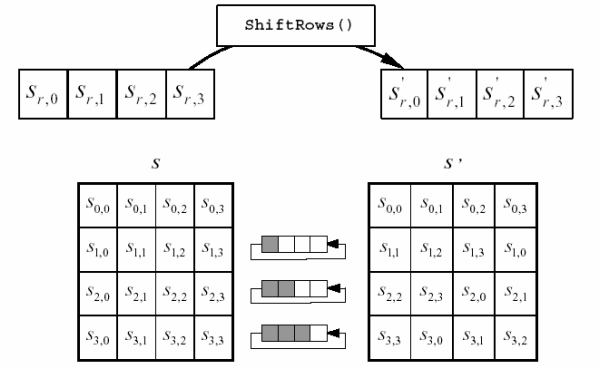
SubBytes()

- The **SubBytes()** transformation is a non-linear substitution that operates independently on each byte of the State using a substitution table ("S-box").
- The S-box is invertible



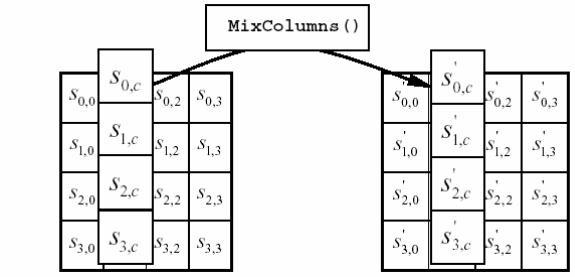
ShiftRows()

- ShiftRows() cyclically shifts the last three rows in the State



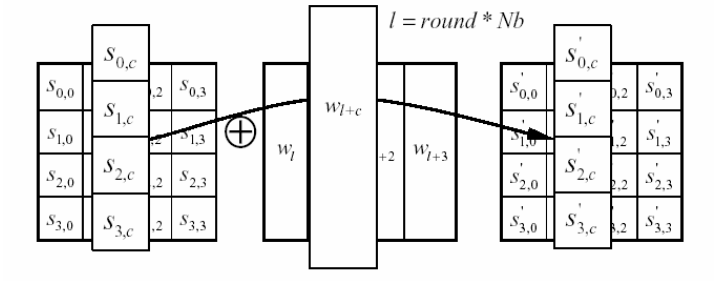
MixColumns()

- MixColumns() operates on the State column-by-column



AddRoundKey()

- AddRoundKey() XORS each column of the State with a word from the key schedule.

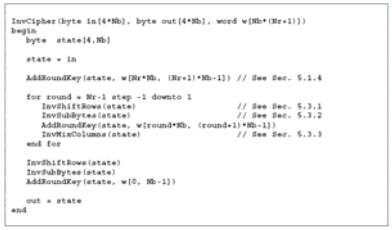


Inverse Cipher Algorithm

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)                    // See Sec. 5.3.1
      InvSubBytes(state)                     // See Sec. 5.3.2
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)                   // See Sec. 5.3.3
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```


The AES Inverse Cipher

- Decryption algorithm uses the expanded key in reverse order
- All the functions are easily reversible and their inverse form is used in decryption
- The decryption algorithm is not identical to the encryption algorithm
- The final round again omits a stage.



Rijndael Key Expansion

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Rijndael Key Expansion

- Starts with the key arranged as N_k 4-octet columns and iteratively generates the next N_k columns of the expanded key.
- **SubWord()** takes a 4-byte input and applies the S-box to each of the 4 bytes to produce an output word.
- **RotWord()** takes a word $[a_0a_1a_2a_3]$ as input and performs a cyclic permutation on it, returning $[a_1a_2a_3a_0]$

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```


Some Other Symmetric Block Ciphers

- Serpent
  - An AES finalist – conservative; slower than Rijndael
- Blowfish
  - Designed by Bruce Schneier; unpatented algorithm; C code in the public domain; fast; compact; simple; variable key length up to 448 bits.
- Twofish
  - AES finalist; Designed by Bruce Schneier et. al. from Counterpane Systems; 128-bit block; 128-, 192-, or 256-bit key; 16 rounds
- RC5
  - Designed by Ron Rivest (of RSA fame); variable block size, variable key size, and variable number of rounds

Summary

- Whew!
- We've "done" symmetric block ciphers in a fair amount of detail.
- Bruce Schneier, in his book *Applied Cryptography, Second Edition,* Wiley documents a lot more of them, and about them, including C code that is available either for free, or for a nominal charge.