Chapter 3: Block Ciphers and AES

Math 495, Fall 2008

Hope College

September, 2008

Iterated Ciphers

- An *iterated cipher* is one in which the same encryption method is used several times in succession, with the key changing for each iteration.
- Each iteration is called a *round*, and the key used in a round is called the *round key* or *subkey*.
- The set of keys used is called the *key schedule*.
- We use the term *state* to refer to the data we are encrypting in each step.
- Each round, a *round function* is used to compute the next state is based on the round key and the current state.
- Thus, a round function g takes two inputs: the current state and the round key.

Iterated Ciphers

- Let Nr be the number of rounds of the cipher.
- We begin with a random key K of some length.
- Based on K, we construct Nr round keys, $K^1, K^2, \ldots k^{Nr}$.
- The algorithm used to construct the key schedule is public.
- The initial state w_0 is the plaintext.
- For each round, we define $w^r = g(w^{r-1}, K^r)$
- Clearly, for a fixed K, g must be one-to-one so that

  $$g^{-1}(g(w, y), y) = w,$$

  allowing us to perform the decryption.
- Let's see an example of the general process of encryption and decryption for $Nr = 4$ (On the board)

Key Schedule

- Given an initial key K, some mechanism is needed to construct a key schedule.
- Many methods can be used to do this.
- Certainly the security of the system is (in part) dependent on how the key schedule is constructed.
- We will just give one (not necessarily secure) example.

Key schedule example

- Assmume
  - We need to perform 5 rounds
  - Each round requires a 16 bit round key
  - The key has 20 bits.
- We can group the 20 bits in groups of 4 bits
- We remove the kth group for round k.

Example

- If $K = 0111\ 0101\ 1010\ 1011\ 1101$, then

$$\begin{aligned} K^1 &= 0101\ 1010\ 1011\ 1101 \\ K^2 &= 0111\ 1010\ 1011\ 1101 \\ K^3 &= 0111\ 0101\ 1011\ 1101 \\ K^4 &= 0111\ 0101\ 1010\ 1101 \\ K^5 &= 0111\ 0101\ 1010\ 1011 \end{aligned}$$

S-Box

- An S-box π_S is simply a permutation on binary strings of a given length.
- That is, $\pi_S : \{0,1\}^\ell \to \{0,1\}^\ell$ is a permutation.
- For convenience, we often represent the inputs and outputs of an S-box in a more compact representation.
- For instance, if $\ell = 4$, we can use hexadecimal notation.

Example

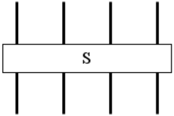
An example of an S-box with $\ell = 4$.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\pi_S(x)$	2	8	9	D	4	F	5	6	3	A	E	0	B	C	7	1

- How might you implement an S-box as part of an algorithm?
- How much space is required to store an S-box?

S-Box

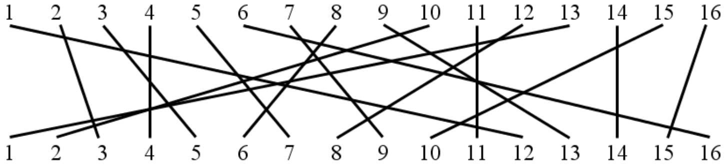
- S-boxes are represented in diagrams as a box with the appropriate number of inputs and outputs.



- The input and output bits can be thought of as traveling along the lines.
- Think of an S-box as being a device which simply replaces ℓ bits with another ℓ bits (in an invertible way, of course).
- Although this representation may lead you to believe that there is a there clear relationship between the individual bits of x and the bits of $\pi_S(x)$, in general you cannot think of it this way.

Permutations

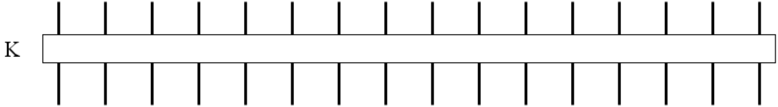
- You already know what a permutation is.
- A permutation can be visualized as follows:



- Think of this as a circuit that scrambles the bits.
- How much memory is required to store a permutation of k things?

XOR

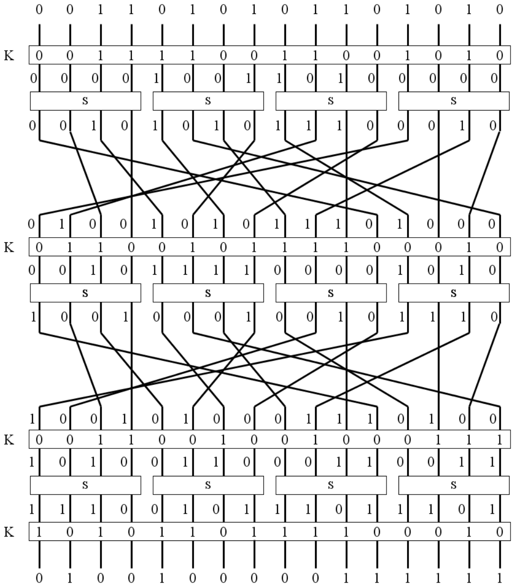
- Given a key K^r and the current state w, a desired operation is to *XOR* the key and state.
- That is, we want to compute $K^r \oplus w$.
- We can represent this as follows, where the inputs are the bits of w and the outputs are the bits of $K^r \oplus w$.



Substitution-Permutation Network Example

- You did the reading, so you already have an idea of what a *substitution-permutation network* is.
- You can re-read the book for the gory details, but briefly:
  - ℓ, m, and Nr are positive integers.
  - Plaintext and ciphertext are of length ℓm,
  - We define one or more S boxes $\pi_s : \{0,1\}^\ell \to \{0,1\}^\ell$
  - We define a permutation $\pi_p : \{1,\dots,\ell m\} \to \{1,\dots,\ell m\}$
  - K is a key consisting of at least ℓm bits from which Nr rounds keys can be derived.
  - We mix the above elements into a magical soup of Nr rounds.
- We now turn to an example SPN (with $\ell = m = 4$)

Substitution-Permutation Network Example



- DES is a type of iterated cipher called a *Feistel cipher*.
- As you would expect, the encryption occurs in rounds.
- Each round, the state is divided into two halves, L^i and R^i.
- The next state is computed as follows:

$$\begin{aligned} L^i &= R^{i-1} \\ R^i &= L^{i-1} \oplus f(R^{i-1}, K^i) \end{aligned}$$

  where K^i is the round key and f the *round function*.

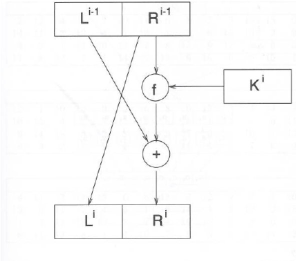


FIGURE 3.6
One round of DES encryption

- Each step is easily reversed:

$$\begin{aligned} R^{i-1} &= L^i \\ L^{i-1} &= R^i \oplus f(L^i, K^i) \end{aligned}$$

- Notice that f does not need to be invertible.

DES Overview

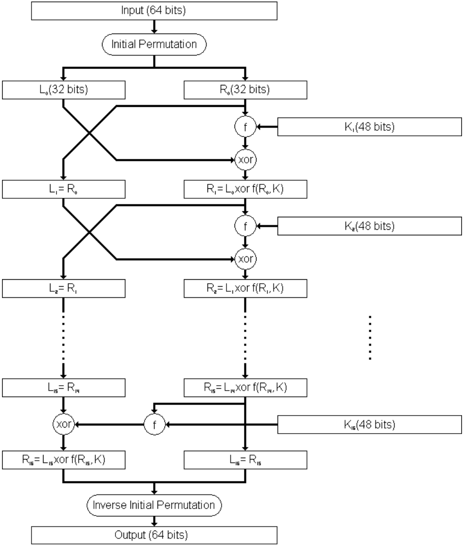


Figure from http://people.eku.edu/styere/Encrypt/JS-DES.html

DES Details

- We need to discuss some details:
  - What function f does DES use?
  - How are the round keys computed?
  - What is the *initial permutation*?
- The last question is easy to answer:

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

- We will discuss the round keys when we do an example.
- We turn to the function f.

DES round function f

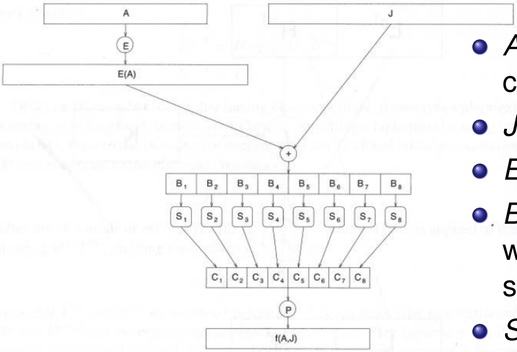


FIGURE 3.7
The DES f function

- A is the input (right half of current state)
- J is the round key
- E is the *expansion function*
- B_i are the XOR of the key with the expanded input, split into bytes
- S_i are the S-boxes–each is different
- C_i are the outputs of the S-boxes
- P is a permutation

DES expansion function

- The *expansion function* needs to take the 32 bits from the input A and expand them to 48 bits to be XORed with the round key.
- Uses 16 bits from A once, and 16 twice.
- Selects and permutes the 48 bit according the permutation E, outputting the result.

E bit-selection table

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

DES S-boxes

- The S-boxes in DES take 6 bits as input and output 4 bits.
- Label the input bits $B = b_1 b_2 b_3 b_4 b_5 b_6$
- $b_1 b_6$ corresponds to the row in the S-box in binary.
- $b_2 b_3 b_4 b_5$ corresponds to the column in binary.
- For instance, if $B = 101100$, then we look at row 2 (10_2) and column 6 (0110_2).
- Notice each row is a permutation of $\{0, 1, \ldots, 15\}$. This is important for security.

S_1

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

S_2

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

(and 6 more)

DES round function permutation

- The final step in the round function is to permute the 32 bits $C_1 C_2 \cdots C_8$ that came from the 8 S-boxes.
- The following permutation is used.

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

DES round review

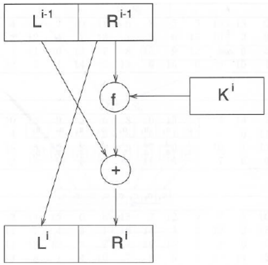


FIGURE 3.6
One round of DES encryption

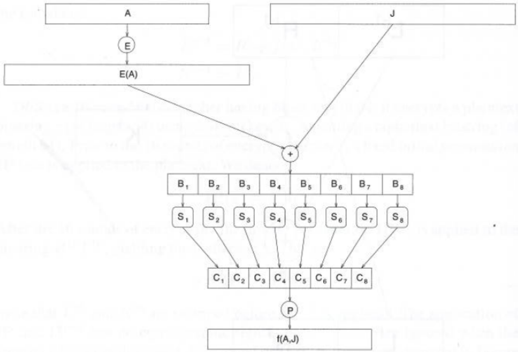


FIGURE 3.7
The DES f function

DES Example

- We will now see an example of DES in action.
- There is a great example, including how the round keys are computed, here:
  http://dhost.info/pasjagor/des/

Attacks on DES

- **Linear cryptanalysis:** One known plaintext attack requires 2^{43} plaintext-ciphertext pairs.
- **Differential cryptanalysis:** One attack requires 2^{47} chosen plaintexts.
- **Davies' attack:** Space/time trade-off.
  For instance, given 2^{52} known plaintexts, 24 bits of the key can be determined with 53% success rate.
- **Exhaustive key search:** Requires trying 2^{56} keys.
  In 2007, a machine was built for $10,000 that can crack a DES key in 6.4 days.
- The first three types of attacks are infeasible in practice.
- The fourth is not only practical, but DES is currently considered insecure because of it.
- The problem with DES is that the key space is too small.

DES Variations/Replacements

- Triple DES (TDES)
  - Apply DES 3 times with 3 different keys.
  - Two variations: *TDES-EEE* and *TDES-EDE* (E=encrypt, D=decrypt).
  - TDES-EDE is compatible with DES by choosing 3 identical keys.
  - The key length increases to 3*56 = 168 bits
  - Because of *meet-in-the-middle attacks*, the effective key length is more like 112 bits.
  - Considered secure through 2030 by NIST (National Institute of Standards and Technology).
- Advanced Encryption Standard (AES) has replaced DES as the standard.

Advanced Encryption Standard (AES)

- 21 cryptosystems were submitted to replace DES
- Only 15 met all of the initial criteria
- The list was narrowed down to 5 choices:
  - *MARS*
  - *RC6*
  - *Rijndael*
  - *Serpent*
  - *Twofish*
- *Rijndael*, invented by Daemon and Rijmen from Belgium, was chosen.
- AES has a block length of 128 bits
- Three key lengths are allowed: 128 (10 rounds), 192 (12 rounds), and 256 (14 rounds)

AES Overview

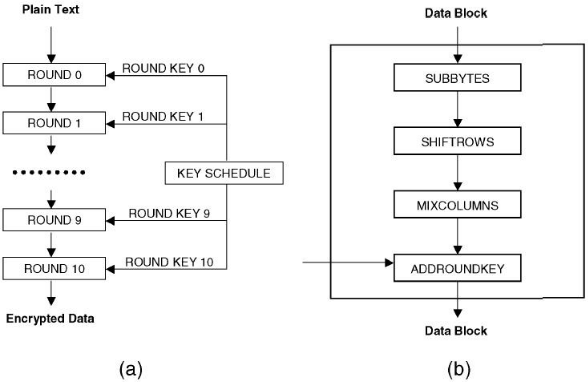


Fig. 1. (a) The data-path for data block and key size of 128 bits, (b) generic structure of one internal round.

Image borrowed from http://www.ecs.umass.edu/ece/koren/FaultTolerantSystems/simulator/AES/help.htm

AES Details

- The state, consisting of 128 bits, is represented by a four by four array of bytes:

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

- The plaintext is mapped into the initial state as follows

$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|}\hline x_0 & x_4 & x_8 & x_{12} \\ \hline x_1 & x_5 & x_9 & x_{13} \\ \hline x_2 & x_6 & x_{10} & x_{14} \\ \hline x_3 & x_7 & x_{11} & x_{15} \\ \hline \end{array}$$

- In round 0, AddRoundKey is performed.
- In rounds 1 through 9, SubBytes, ShiftRows, MixColumns and AddRoundKey are performed.
- In round 10, SubBytes, ShiftRows, and AddRoundKey are performed.

AES SubBytes

- SubBytes is an *S*-box that acts independently on each byte of the state.

	Y															
X	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

- **Example:** The byte 01101101 is represented by $6D$ in hexadecimal, and maps to $3C$, which is 00111100.
- You can check that the *S*-box is, in fact, a permutation.
- The *S*-box can be defined algebraically–we won't go there.

SHIFTROWS

- Shift the ith row of the state $i - 1$ positions to the left.

$$
\begin{array}{|c|c|c|c|}
\hline
s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline
s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline
s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline
s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline
\end{array}
\leftarrow
\begin{array}{|c|c|c|c|}
\hline
s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline
s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ \hline
s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ \hline
s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \\ \hline
\end{array}
$$

ADDROUNDKEY

- Simply the XOR of the current state with the round key.

- The most complicated step

Algorithm 3.5: MixColumn(c)

external FieldMult, BinaryToField, FieldToBinary
for $i \leftarrow 0$ to 3
  do $t_i \leftarrow$ BinaryToField($s_{i,c}$)
$u_0 \leftarrow$ FieldMult$(x, t_0) \oplus$ FieldMult$(x+1, t_1) \oplus t_2 \oplus t_3$
$u_1 \leftarrow$ FieldMult$(x, t_1) \oplus$ FieldMult$(x+1, t_2) \oplus t_3 \oplus t_0$
$u_2 \leftarrow$ FieldMult$(x, t_2) \oplus$ FieldMult$(x+1, t_3) \oplus t_0 \oplus t_1$
$u_3 \leftarrow$ FieldMult$(x, t_3) \oplus$ FieldMult$(x+1, t_0) \oplus t_1 \oplus t_2$
for $i \leftarrow 0$ to 3
  do $s_{i,c} \leftarrow$ FieldToBinary(u_i)

- FieldMult multiplies two elements in a finite field.
- Essentially, the bits in a byte are interpreted as coefficients of a polynomial, and the method multiplies the polynomials modulo another polynomial.
- In this case, we are multiplying by the polynomials x and $x + 1$.
- Notice that this operations acts on the columns independently.

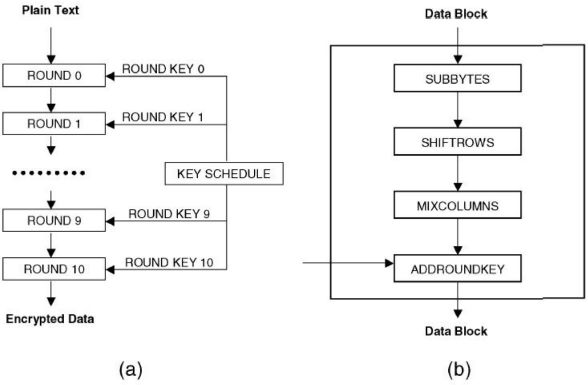


Fig. 1. (a) The data-path for data block and key size of 128 bits, (b) generic structure of one internal round.

- We still need to explore how to compute the round keys.

Image borrowed from http://www.ecs.umass.edu/ece/koren/FaultTolerantSystems/simulator/AES/help.htm

AES Keys

- AES round keys are 128 bits–or 4 words long.
- Recall that 1 word = 4 bytes = 32 bits.
- There are 11 rounds, each of which needs 4 words.
- The KEYEXPANSION algorithm produces an array of 44 words given the initial 128 bit key.
- Taken 4 words at a time, that gives 11 round keys
- Each of the 4 words of a round key is placed in columns of a 4 by 4 array to correspond to the state.

AES key schedule

- The key schedule algorithm uses the following algorithms.

RotWord

- Given a word (4 bytes), left cyclic shift 1 byte.
- That is,

$$\text{RotWord}(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0).$$

SubWord

- Let $B_i' = \text{SubBytes}(B_i)$.
- Then

$$\text{SubWord}(B_0, B_1, B_2, B_3) = (B_0', B_1', B_2', B_3').$$

- That is, we simply apply the S-box to each of the bytes.

AES key schedule algorithm

- The KeyExpansion algorithm produces the array of 44 words.

Algorithm 3.6: KEYEXPANSION(key)

external ROTWORD, SUBWORD
$RCon[1] \leftarrow 01000000$
$RCon[2] \leftarrow 02000000$
$RCon[3] \leftarrow 04000000$
$RCon[4] \leftarrow 08000000$
$RCon[5] \leftarrow 10000000$
$RCon[6] \leftarrow 20000000$
$RCon[7] \leftarrow 40000000$
$RCon[8] \leftarrow 80000000$
$RCon[9] \leftarrow 1B000000$
$RCon[10] \leftarrow 36000000$
for $i \leftarrow 0$ **to** 3
  do $w[i] \leftarrow (key[4i], key[4i+1], key[4i+2], key[4i+3])$
for $i \leftarrow 4$ **to** 43

$$\textbf{do} \begin{cases} temp \leftarrow w[i-1] \\ \textbf{if } i \equiv 0 \pmod 4 \\ \quad \textbf{then } temp \leftarrow \text{SUBWORD}(\text{ROTWORD}(temp)) \oplus RCon[i/4] \\ w[i] \leftarrow w[i-4] \oplus temp \end{cases}$$

return $(w[0], \ldots, w[43])$

AES key schedule example

- K = `A802 56F4 ED3C 812A D4AC 97CF BB42 5398`
- We will attempt to compute (part of) the AES key schedule.

Algorithm 3.6: KEYEXPANSION(key)

external ROTWORD, SUBWORD

$RCon[1] \leftarrow$ 01000000
$RCon[2] \leftarrow$ 02000000
$RCon[3] \leftarrow$ 04000000
$RCon[4] \leftarrow$ 08000000
$RCon[5] \leftarrow$ 10000000
$RCon[6] \leftarrow$ 20000000
$RCon[7] \leftarrow$ 40000000
$RCon[8] \leftarrow$ 80000000
$RCon[9] \leftarrow$ 1B000000
$RCon[10] \leftarrow$ 36000000
for $i \leftarrow 0$ **to** 3
  do $w[i] \leftarrow (key[4i], key[4i+1], key[4i+2], key[4i+3])$
for $i \leftarrow 4$ **to** 43
  do $\begin{cases} temp \leftarrow w[i-1] \\ \textbf{if } i \equiv 0 \pmod 4 \\ \quad \textbf{then } temp \leftarrow \text{SUBWORD}(\text{ROTWORD}(temp)) \oplus RCon[i/4] \\ w[i] \leftarrow w[i-4] \oplus temp \end{cases}$
return $(w[0], \ldots, w[43])$

AES Security

- As far as we know, AES is secure against all known attacks.
- As far as we know...

Block cipher modes of operation

- The following are modes of operation that can be used with AES, DES, and other block ciphers.
  - *electronic codebook mode* (ECB mode)
  - *cipher feedback mode* (CFB mode)
  - cipher block chaining mode (CBC mode)
  - output feedback mode (OFB mode)
  - *counter mode*
  - *counter with cipher-block chaining mode* (CCM mode)
- We will briefly describe each of these.

ECB Mode

- This is simply the way we have thought about applying block ciphers up to this point–apply the encryption to each plaintext block with the same key.
- Thus, we encrypt as follows

$$
\begin{aligned}
y_1 &= e_K(x_1) \\
y_2 &= e_K(x_2) \\
y_3 &= e_K(x_3) \\
&\vdots \\
y_i &= e_K(x_i) \\
&\vdots
\end{aligned}
$$

- Before we encrypt a plaintext, we XOR it with the previous cipher text.
- We begin by defining an *initialization vector*, IV, and set $y_0 = IV$
- Then we encrypt as follows

$$\begin{aligned} y_1 &= e_K(y_0 \oplus x_1) \\ y_2 &= e_K(y_1 \oplus x_2) \\ &\vdots \\ y_i &= e_K(y_{i-1} \oplus x_i) \\ &\vdots \end{aligned}$$

- CBC mode is often used to provide a *message authentication code* (MAC).

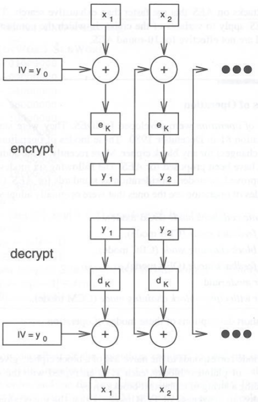


FIGURE 3.9
CBC mode

OFB Mode

- OFB is a synchronous stream cipher.
- We define $z_0 = IV$, an initialization vector.
- The keystream is computed using the encryption function

$$z_i = e_K(z_{i-1})$$

- The ciphertext is produced by XORing the plaintext with the key:

$$y_i = x_i \oplus z_i$$

- Decryption is simple–compute the key stream as above, and then

$$x_i = y_i \oplus z_i$$

- It should be evident that encryption and decryption are actually the same algorithm.

- CFB is a synchronous stream cipher.
- We define $y_0 = IV$, an initialization vector.
- Each keystream element is computed by encrypting the previous ciphertext block.
- That is,

$$z_i = e_K(y_{i-1})$$

- The ciphertext is produced by XORing the plaintext with the key:

$$y_i = x_i \oplus z_i$$

- Encryption and decryption are the same algorithm.

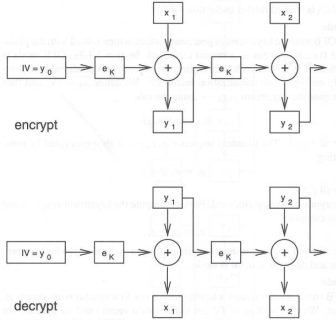


FIGURE 3.10
CFB mode

counter Mode

- Given plaintext block length of m, let *counter*, denoted *ctr*, be a bitstring of length m.
- We define a sequence of bitstrings T_1, $T_2, \ldots$, by

$$T_i = ctr + i - 1 \bmod 2^m.$$

- The ciphertext is produced by XORing the plaintext with the encryptions of the produced bitstrings:

$$y_i = x_i \oplus e_K(T_i)$$

CCM Mode

- A combination of counter mode and CBC-mode.