Security Engineering

Lesson 8
Algorithm Types and Modes

Spring 2010
Dr. Marenglen Biba

Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Algorithm Types and Modes

- There are two basic types of symmetric algorithms: block ciphers and stream ciphers.
- **Block ciphers** operate on blocks of plaintext and ciphertext — usually of 64 bits but sometimes longer.
- **Stream ciphers** operate on streams of plaintext and ciphertext one bit or byte (sometimes even one 32-bit word) at a time.
- With a block cipher, the same plaintext block will always encrypt to the same ciphertext block, using the same key.
- With a stream cipher, the same plaintext bit or byte will encrypt to a different bit or byte every time it is encrypted.

Cryptographic mode

- A **cryptographic mode** usually combines the basic cipher, some sort of feedback, and some simple operations.
  - The operations are **simple** because the security is a function of the underlying cipher and not the mode.
  - Even more strongly, the cipher mode **should not compromise** the security of the underlying algorithm.
- There are other security considerations:
  - Patterns in the plaintext should be concealed,
  - Input to the cipher should be randomized,
  - Manipulation of the plaintext by introducing errors in the ciphertext should be difficult
  - Encryption of more than one message with the same key should be possible.

Electronic Codebook Mode

- **Electronic codebook** (ECB) mode is the most obvious way to use a block cipher: A block of plaintext encrypts into a block of ciphertext.
- Since the same block of plaintext always encrypts to the same block of ciphertext, it is theoretically possible to create a code book of plaintexts and corresponding ciphertexts.
- However, if the block size is 64 bits, the code book will have 2^{64} entries
  - much too large to precompute and store. And remember, every key has a different code book.

ECB Mode

- This is the easiest mode to work with.
- Each plaintext block is encrypted independently.
  - You don't have to encrypt a file linearly; you can encrypt the 10 blocks in the middle first, then the blocks at the end, and finally the blocks in the beginning.
- This is important for encrypted files that are accessed randomly, like a database. If a database is encrypted with ECB mode, then any record can be added, deleted, encrypted, or decrypted independently of any other record—assuming that a record consists of a discrete number of encryption blocks.
- And processing is parallizeable; if you have multiple encryption processors, they can encrypt or decrypt different blocks without regard for each other.

Problems of ECB Mode

- The problem with ECB mode is that if a cryptanalyst has the plaintext and ciphertext for several messages, he can start to compile a code book without knowing the key.
- In most real-world situations, fragments of messages tend to repeat.
- Different messages may have bit sequences in common.
- If a cryptanalyst learns that the plaintext block “5e081bc5” encrypts to the ciphertext block “7ea593a4,” he can immediately decrypt that ciphertext block whenever it appears in another message.

Problems of ECB Mode

- If the encrypted messages have a lot of redundancies, and these tend to show up in the same places in different messages, a cryptanalyst can get a lot of information.
  - He can mount statistical attacks on the underlying plaintext, irrespective of the strength of the block cipher.
- This vulnerability is greatest at the beginning and end of messages, where well-defined headers and footers contain information about the sender, receiver, date, and so on.
  - This problem is sometimes called stereotyped beginnings and stereotyped endings.

Positives of ECB Mode

- On the plus side, there is no security risk in encrypting multiple messages with the same key.
- In fact, each block can be looked at as a separate message encrypted with the same key.
- Bit errors in the ciphertext, when decrypted, will cause the entire plaintext block to decrypt incorrectly but will not affect the rest of the plaintext.
- However, if a ciphertext bit is accidentally lost or added, all subsequent ciphertext will decrypt incorrectly unless there is some kind of frame structure to realign the block boundaries.
  - Adding or removing a bit would shift the entire blocks!

Padding

- Most messages don't divide neatly into 64-bit (or whatever size) encryption blocks; there is usually a short block at the end. ECB requires 64-bit blocks.
- **Padding** is the way to deal with this problem.
- Pad the last block with some regular pattern—zeros, ones, alternating ones and zeros—to make it a complete block.
- If you need to delete the padding after decryption, add the number of padding bytes as the last byte of the last block.
- For example, assume the block size is 64 bits and the last block consists of 3 bytes (24 bits). Five bytes of padding are required to make the last block 64 bits; add 4 bytes of zeros and a final byte with the number 5. After decryption, delete the last 5 bytes of the last decryption block.

Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Block Replay

- A more serious problem with ECB mode is that an adversary could modify encrypted messages without knowing the key, or even the algorithm, in such a way as to fool the intended recipient.
- To illustrate the problem, consider a money transfer system that moves money between accounts in different banks. To make life easier for the bank's computer systems, banks agree on a standard message format for money transfer that looks like this:

Bank One: Sending	1.5 blocks
Bank Two: Receiving	1.5 blocks
Depositor's Name	6 blocks
Depositor's Account	2 blocks
Amount of Deposit	1 block

- A block corresponds to an 8-byte encryption block. The messages are encrypted using some block algorithm in ECB mode.

Block Replay

- Mallory, who is listening on the communications line between two banks, Bank of Alice and Bank of Bob, can record all of the encrypted messages from Bank of Alice to Bank of Bob.
- Then, he transfers $100 from Bank of Alice to his account in Bank of Bob. Later, he does it again.
- Using his computer, he examines the recorded messages looking for a pair of identical messages. These messages are the ones authorizing the $100 transfers to his account.
- If he finds more than one pair of identical messages (which is most likely in real life), he does another money transfer and records those results. Eventually he can isolate the message that authorized his money transaction.

Timestamps and Block Replay

- At first glance, the banks could easily prevent this by adding a timestamp to their messages.
- Still, using **block replay**, Mallory can get rich.
- He can pick out the eight ciphertext blocks that correspond to his own name and account number: blocks 5 through 12.
  - He intercepts random messages from Bank of Alice to Bank of Bob and replaces blocks 5 through 12 in the message with the bytes that correspond to his name and account number. Then he sends them on to Bank of Bob.
- He doesn't have to know who the original depositor was; he doesn't even have to know what the amount was.
- This will take longer than a day for the banks to catch. When they reconcile their transfers at the end of the day, everything will match.

Chaining

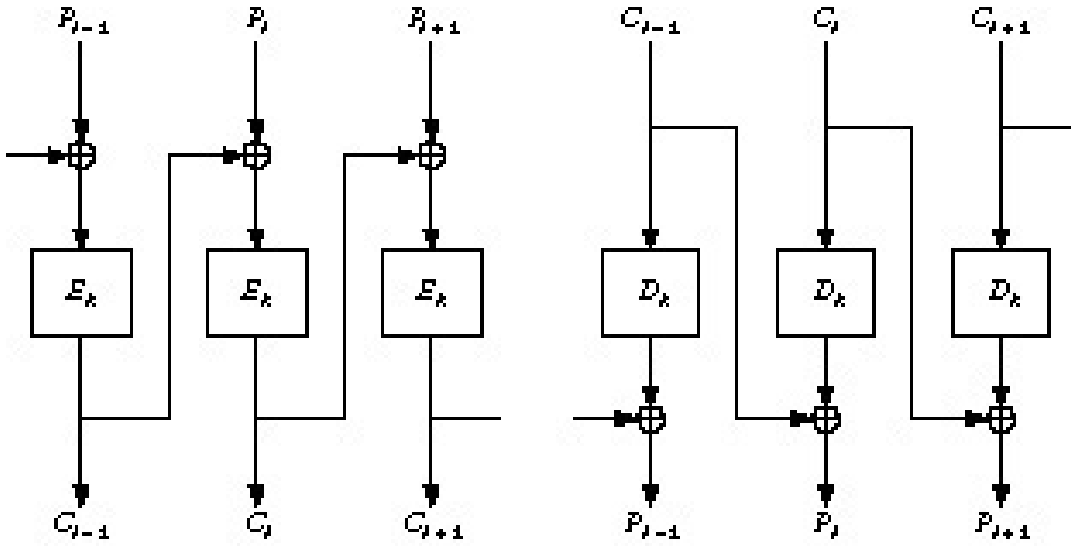
- Banks can minimize the problem by changing their keys frequently, but this only means that Mallory is going to have to work more quickly.
- Adding a MAC, however, will also solve the problem (A **message authentication code** (MAC), is a one-way hash function with the addition of a secret key.).
- Even so, this is a fundamental problem with ECB mode.
- Mallory can remove, repeat, or interchange blocks at will. The solution is a technique called chaining.

Cipher Block Chaining Mode

- Chaining adds a feedback mechanism to a block cipher: The results of the encryption of previous blocks are fed back into the encryption of the current block.
- In other words, each block is used to modify the encryption of the next block.
- Each ciphertext block is dependent not just on the plaintext block that generated it but on all the previous plaintext blocks.
- In cipher block chaining (CBC) mode, the plaintext is XORed with the previous ciphertext block before it is encrypted.

Cipher Block Chaining

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



(a) CBC Encryption (b) CBC Decryption

Initialization Vector

- CBC mode forces identical plaintext blocks to encrypt to different ciphertext blocks only when some previous plaintext block is different.
- Two identical messages will still encrypt to the same ciphertext. Even worse, two messages that begin the same will encrypt in the same way up to the first difference.
  - Some messages have a common header: a letterhead, or a "From" line, or whatever. While block replay would still be impossible, this identical beginning might give a cryptanalyst some useful information.
- We can prevent this by using an initialization vector (IV),

Initialization Vector

- We can encrypt random data as the first block. This block of random data is called the initialization vector (IV), initializing variable, or initial chaining value.
  - The IV has no meaning; it's just there to make each message unique..
  - A timestamp often makes a good IV. Otherwise, use some random bits from someplace.
- With the addition of IVs, identical plaintext messages encrypt to different ciphertext messages.
- Thus, it is impossible for an eavesdropper to attempt block replay, and more difficult for him to build a code book.

Padding

- Padding works just like ECB mode, but in some applications the ciphertext has to be exactly the same size as the plaintext. Perhaps a plaintext file has to be encrypted and then replaced in the exact same memory location.
- In this case, you have to encrypt the last short block differently. Assume the last block has *j* bits. After encrypting the last full block, encrypt the ciphertext again, select the left-most *j* bits of the encrypted ciphertext, and XOR that with the short block to generate the ciphertext.

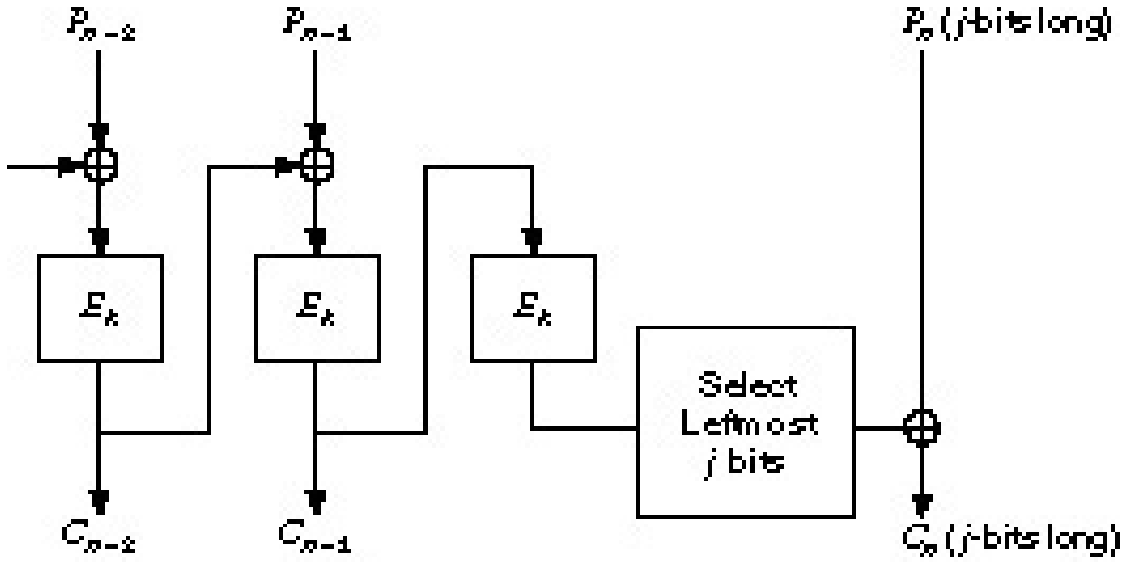


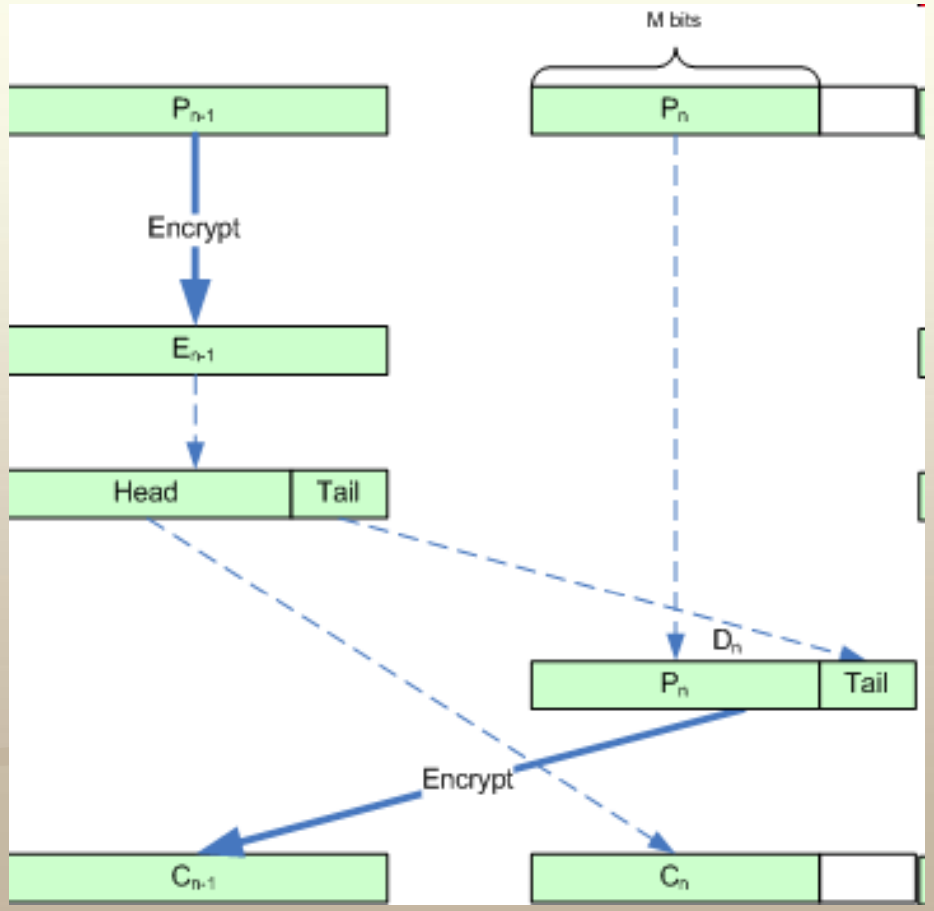
Figure 9.4 Encrypting the last short block in CBC mode.

Padding

- The weakness here is that while Mallory cannot recover the last plaintext block, he can change it systematically by changing individual bits in the ciphertext.
- If the last few bits of the ciphertext contain essential information, this is a weakness. If the last bits simply contain housekeeping information, it isn't a problem.
- Ciphertext stealing is a better way

Ciphertext stealing

- **Ciphertext stealing** (CTS) allows for processing of messages that are not evenly divisible into blocks without resulting in any expansion of the ciphertext, at the cost of slightly increased complexity.



Birthday Paradox for CBC

- Finally, although plaintext patterns are concealed by chaining, very long messages will still have patterns.
- The birthday paradox predicts that there will be identical blocks after $2^{m/2}$ blocks, where m is the block size. For a 64-bit block size, that's about 34 gigabytes.
- A message has to be pretty long before this is a problem.

Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Stream ciphers

- Stream ciphers convert plaintext to ciphertext 1 bit at a time.
- A **keystream generator** (sometimes called a running-key generator) outputs a stream of bits: k_1, k_2, k_3,..., k_i.
- This keystream (sometimes called a running key) is XORed with a stream of plaintext bits, p_1, p_2, p_3,..., p_i, to produce the stream of ciphertext bits.
  - $c_i = p_i \oplus k_i$
- At the decryption end, the ciphertext bits are XORed with an identical keystream to recover the plaintext bits.
  - $p_i = c_i \oplus k_i$
- Since $p_i \oplus k_i \oplus k_i = p_i$, this works nicely.

Keystream generator

- The system's security depends entirely on the insides of the keystream generator.
- If the keystream generator outputs an endless stream of zeros, the ciphertext will equal the plaintext and the whole operation will be worthless.
- If the keystream generator spits out a repeating 16-bit pattern, the algorithm will be a simple XOR with negligible security.
- If the keystream generator spits out an endless stream of random (not pseudo-random, but real random) bits, you have a one-time pad and perfect security.
- The reality of stream cipher security lies somewhere between the simple XOR and the one-time pad.

Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Self-synchronizing stream cipher

- For a self-synchronizing stream cipher, each keystream bit is a function of a fixed number of previous ciphertext bits.
- The military calls this ciphertext auto key (CTAK). The basic idea was patented in 1946.
- In a self-synchronizing stream cipher. The internal state is a function of the previous *n* ciphertext bits.
- Since the internal state depends wholly on the previous *n* ciphertext bits, the decryption keystream generator will automatically synchronize with the encryption keystream generator after receiving *n* ciphertext bits.
- In smart implementations of this mode, each message begins with a random header *n* bits long. That header is encrypted, transmitted, and then decrypted.
  - After those *n* bits both keystream generators will be synchronized.

Security Problems of SSSC

- Self-synchronizing stream ciphers are also vulnerable to a playback attack.
- First Mallory records some ciphertext bits. Then, at a later time, he substitutes this recording into current traffic.
- After some initial garbage while the receiving end resynchronizes, the old ciphertext will decrypt as normal.
- The receiving end has no way of knowing that this is not current data, but old data being replayed. Unless timestamps are used,
  - Mallory can convince a bank to credit his account again and again, by replaying the same message (assuming the key hasn't been changed, of course).

Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Cipher-Feedback Mode

- Block ciphers can also be implemented as a self-synchronizing stream cipher; this is called cipher-feedback (CFB) mode.
- With CBC mode, encryption cannot begin until a complete block of data is received.
- This is a problem in some network applications. In a secure network environment, for example, a terminal must be able to transmit each character to the host as it is entered.
- When data has to be processed in byte-sized chunks, CBC mode just won't do.

CFB

- In CFB mode, data can be encrypted in units smaller than the block size.
- You can encrypt one ASCII character at a time (this is called **8-bit CFB**).
- You can encrypt data one bit at a time using 1-bit CFB, although using one complete encryption of a block cipher for a single bit seems like a whole lot of work; a stream cipher might be a better idea.
- You can also use 64-bit CFB, or any *n*-bit CFB where *n* is less than or equal to the block size.

Initialization Vector for CFB

- To initialize the CFB process, the input to the block algorithm must be initialized with an IV. Like the IV used in CBC mode, it need not be secret.
- The IV must be unique, though. (This is different from the IV in CBC mode, which should be unique but does not have to be.)
- If the IV in CFB is not unique, a cryptanalyst can recover the corresponding plaintext.
- The IV must be changed with every message.
- It can be a serial number, which increments after each message and does not repeat during the lifetime of the key. For data encrypted for storage, it can be a function of the index used to look up the data.

Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Synchronous Stream Ciphers

- In a synchronous stream cipher the keystream is generated independent of the message stream.
- The military calls this Key Auto-Key (KAK). On the encryption side, a keystream generator spits out keystream bits, one after the other.
- On the decryption side, another keystream generator spits out the identical keystream bits, one after the other.
- This works, as long as the two keystream generators are synchronized.
- If one of them skips a cycle or if a ciphertext bit gets lost during transmission, then every ciphertext character after the error will decrypt incorrectly.
- If this happens, the sender and receiver must resynchronize their keystream generators before they can proceed.

Insertion Attack

- Synchronous stream ciphers are vulnerable to an insertion attack.
- Mallory has recorded a ciphertext stream, but does not know the plaintext or the keystream used to encrypt the plaintext.

Original plaintext:	p_1	p_2	p_3	p_4	...
Original keystream:	k_1	k_2	k_3	k_4	...
Original ciphertext:	c_1	c_2	c_3	c_4	...

- Mallory inserts a single known bit, *p'*, into the plaintext after p_1 and then manages to get the modified plaintext encrypted with the same keystream. He records the resultant new ciphertext:

New plaintext:	p_1	p'	p_2	p_3	p_4	...
Original keystream:	k_1	k_2	k_3	k_4	k_5	...
Updated ciphertext:	c_1	c'_2	c'_3	c'_4	c'_5	...

Insertion Attack

- Assuming he knows the value of *p'*, he can determine the entire plaintext after that bit from the original ciphertext and new ciphertext:

$$k_2 = c'_2 \oplus p', \text{ and then } p_2 = c_2 \oplus k_2$$
$$k_3 = c'_3 \oplus p_2, \text{ and then } p_3 = c_3 \oplus k_3$$
$$k_4 = c'_4 \oplus p_3, \text{ and then } p_4 = c_4 \oplus k_4$$

- Mallory doesn't even have to know the exact position in which the bit was inserted; he can just compare the original and updated ciphertexts to see where they begin to differ.
- To protect against this attack, never use the same keystream to encrypt two different messages.

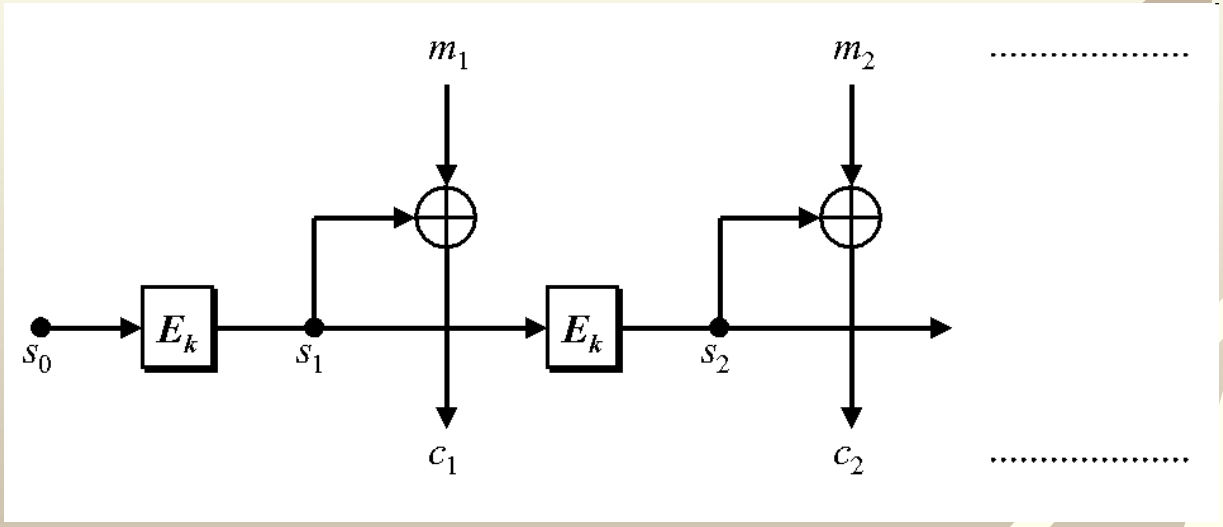
Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Output-feedback (OFB)

- Output-feedback (OFB) mode is a method of running a block cipher as a synchronous stream cipher.
- 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}.
- The encryption of a plaintext block is derived by taking the XOR of the plaintext block with the relevant data block.



$$c_i = m_i \,(+)\, s_i \qquad m_i = c_i \,(+)\, s_i \qquad s_i = E_k(s_{i-1})$$

Security Problems with OFB

- An analysis of OFB mode demonstrates that OFB should be used only when the feedback size is the same as the block size.
- For example, you should only use a 64-bit algorithm in 64-bit OFB mode. Even though the U.S. government authorizes other feedback sizes for DES, avoid them.
- OFB mode XORs a keystream with the text. This keystream will eventually repeat. It is important that it does not repeat with the same key; otherwise, there is no security.

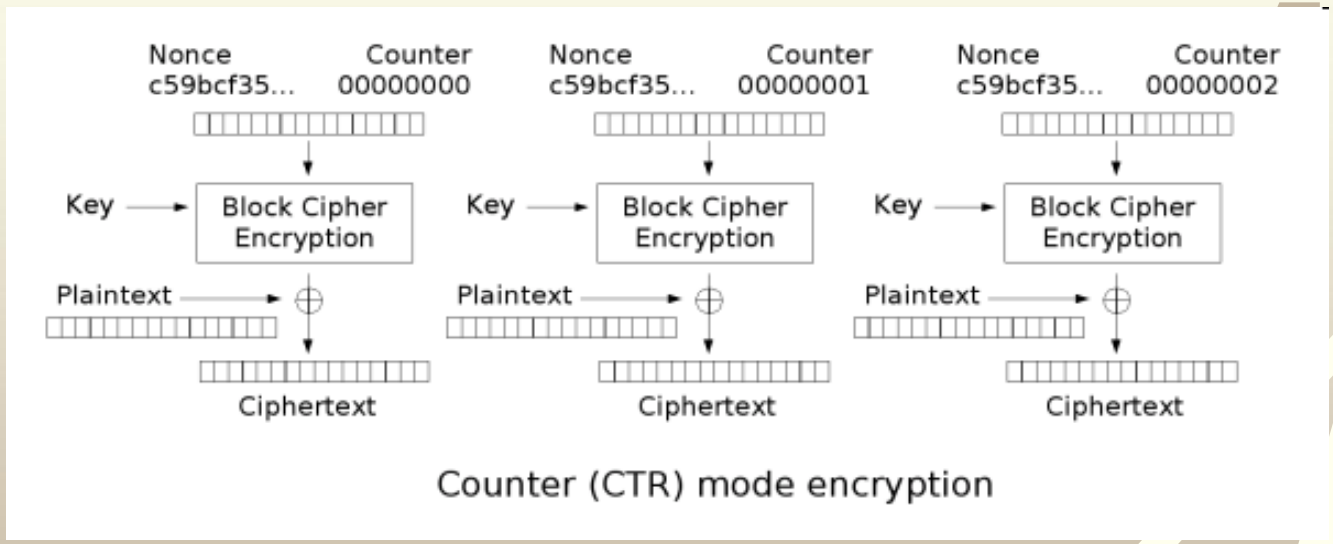
Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Counter Mode

- Block ciphers in counter mode use sequence numbers as the input to the algorithm.
- Instead of using the output of the encryption algorithm to fill the register, the input to the register is a counter.
- After each block encryption, the counter increments by some constant, typically one.



Counter (CTR) mode encryption

- Note that the nonce is the same thing as the initialization vector (IV) in the other graphs. The IV/nonce and the counter can be concatenated, added, or XORed together to produce the actual unique counter block for encryption.

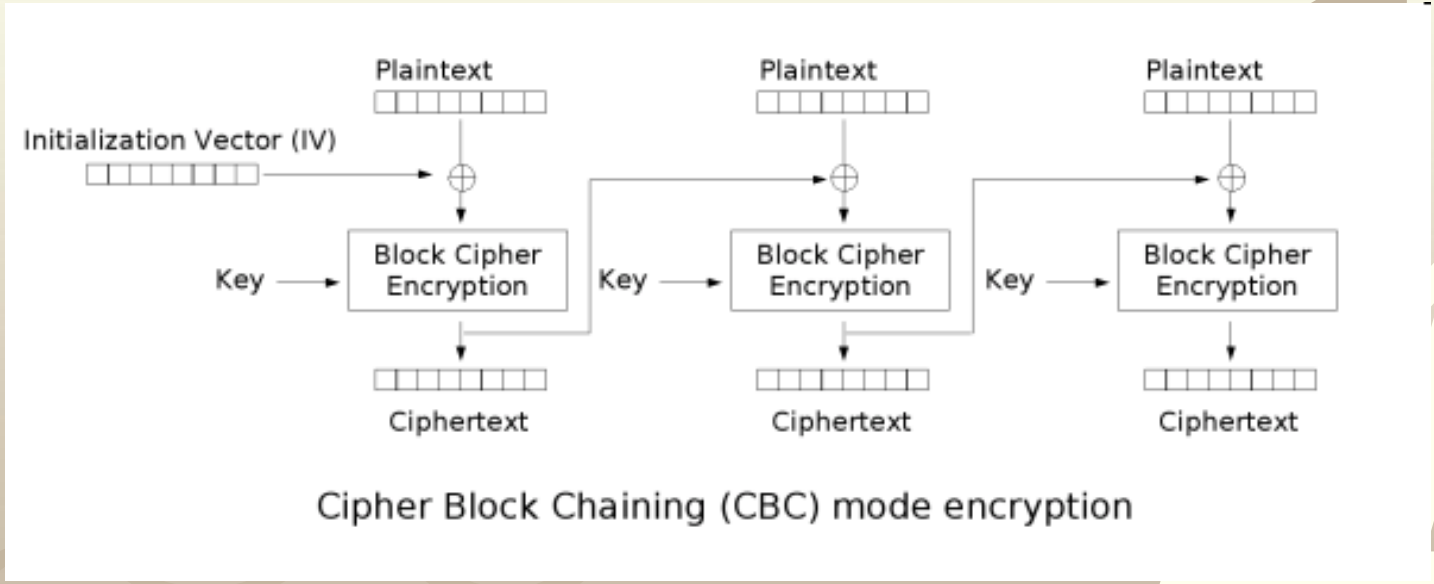
Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Block Chaining Mode

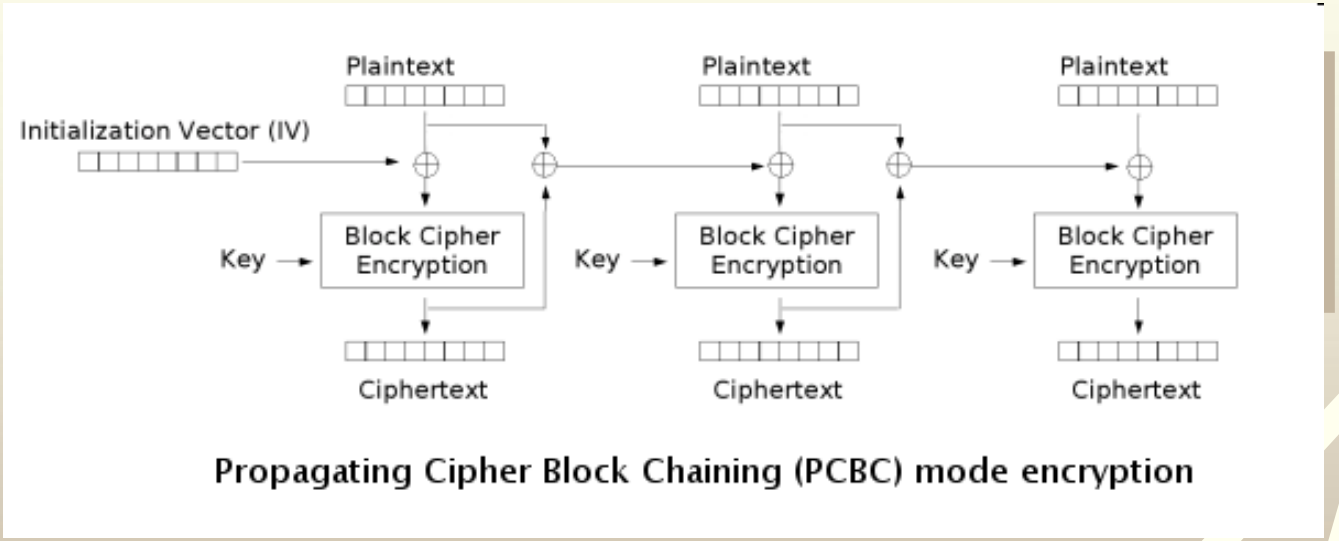
- CBC mode of operation was invented by IBM in 1976.
- To use a block algorithm in block chaining (BC) mode, simply XOR the input to the block cipher with the XOR of all the previous ciphertext blocks.
- As with CBC, an IV starts the process.



Cipher Block Chaining (CBC) mode encryption

Propagating Cipher Block Chaining Mode

- Propagating cipher block chaining (PCBC) mode is similar to CBC mode, except that both the previous plaintext block and the previous ciphertext block are XORed with the current plaintext block before encryption (or after decryption)



Propagating Cipher Block Chaining (PCBC) mode encryption

- PCBC was used in Kerberos version 4 to perform encryption.

Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Choosing a Cipher Mode

- If simplicity and speed are your main concerns, ECB is the easiest and fastest mode to use a block cipher. It is also the weakest. Besides being vulnerable to replay attacks, an algorithm in ECB mode is the easiest to cryptanalyze.
- For encrypting random data, such as other keys, ECB is a good mode to use. Since the data is short and random, none of the shortcomings of ECB matter for this application.
- For normal plaintext, use CBC, CFB, or OFB. Which mode you choose depends on your specific requirements. Table 9.1 gives a summary of the security and efficiency of the various modes.
- CBC is generally best for encrypting files. The increase in security is significant; and while there are sometimes bit errors in stored data, there are almost never synchronization errors. If your application is software-based, CBC is almost always the best choice.

Comparison: ECB Vs. CBC

ECB:	CBC:
***Security*:**	***Security*:**
- Plaintext patterns are not concealed.	+ Plaintext patterns are concealed by XORing with previous ciphertext block.
- Input to the block cipher is not randomized; it is the same as the plaintext.	+Input to the block cipher is randomized by XORing with the previous ciphertext block.
+More than one message can be encrypted with the same key.	+ More than one message can be encrypted with the same key.
- Plaintext is easy to manipulate, blocks can be removed, repeated, or interchanged.	+/- Plaintext is somewhat difficult to manipulate; blocks can be removed from the beginning and end of the message, bits of the first block can be changed, and repetition allows some controlled changes.
***Efficiency*:**	***Efficiency*:**
+ Speed is the same as the block cipher.	+ Speed is the same as the block cipher.
Ciphertext is up to one block longer than the plaintext, due to padding.	- Ciphertext is up to one block longer than the plaintext, not counting the IV.
- No preprocessing is possible.	- No preprocessing is possible.
+ Processing is parallelizable.	+/- Encryptions not parallelizable; decryption is parallelizable and has a random-access property.

Comparison: CFB Vs. OFB/Counter

CFB:

***Security*:**

+ Plaintext patterns are concealed.

+ Input to the block cipher is randomized.

+ More than one message can be encrypted with the same key provided that a different IV is used.

+/- Plaintext is somewhat difficult to manipulate;blocks can be removed from the beginning and end of the message, bits of the first block can be changed, and repetition allows some controlled changes.

***Efficiency*:**

+ Speed is the same as the block cipher.

- Ciphertext is the same size as the plaintext, not counting the

OFB/Counter:

***Security*:**

+ Plaintext patterns are concealed.

+ Input to the block cipher is randomized.

+ More than one message can be encrypted with the same key, provided that a different IV is used.

- Plaintext is very easy to manipulate, any change in ciphertext directly affects the plaintext.

***Efficiency*:**

+ Speed is the same as the block cipher.

- Ciphertext is the same size as the plaintext, not counting the

Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Interleaving

- With most modes, encryption of a bit (or block) depends on the encryption of the previous bits (or blocks).
- This can often make it impossible to parallelize encryption.
- For example, consider a hardware box that does encryption in CBC mode. Even if the box contains four encryption chips, only one can work at any time. The next chip needs the results of the previous chip before it starts working.
- The solution is to **interleave** multiple encryption streams. (This is not multiple encryption!!).
- Instead of a single CBC chain, use four. The first, fifth, and every fourth block thereafter are encrypted in CBC mode with one IV.
- The second, sixth, and every fourth block thereafter are encrypted in CBC mode with another IV, and so on. The total IV is much longer than it would have been without interleaving.

Interleaving

- Three parallel streams interleaved in CFB mode.
- The idea can also work in CBC and OFB modes, and with any number of parallel streams.
- Just remember that each stream needs its own IV. Don't share.

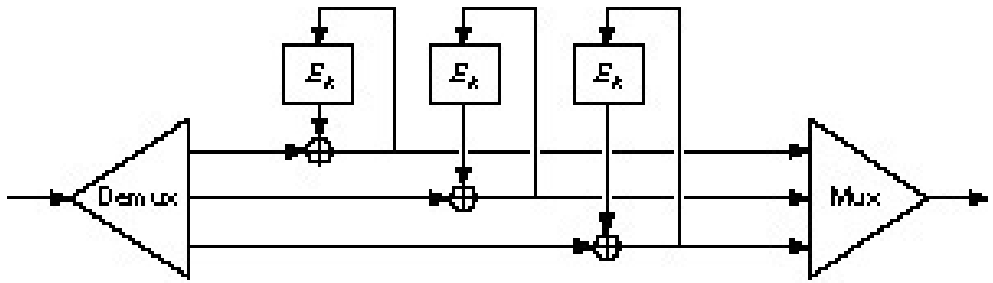


Figure 9.16 Interleaving three CFB encryptions.

Outline

Algorithm Types and Modes

- 9.1 Electronic Codebook Mode
- 9.2 Block Replay
- 9.3 Cipher Block Chaining Mode
- 9.4 Stream Ciphers
- 9.5 Self-Synchronizing Stream Ciphers
- 9.6 Cipher-Feedback Mode
- 9.7 Synchronous Stream Ciphers
- 9.8 Output-Feedback Mode
- 9.9 Counter Mode
- 9.10 Other Block-Cipher Modes
- 9.11 Choosing a Cipher Mode
- 9.12 Interleaving
- 9.13 Block Ciphers versus Stream Ciphers

Block Ciphers Vs. Stream Ciphers

- Although block and stream ciphers are very different, block ciphers can be implemented as stream ciphers and stream ciphers can be implemented as block ciphers.
- In the real world, block ciphers seem to be more general (i.e., they can be used in any of the modes) and stream ciphers seem to be easier to analyze mathematically.
- There is a large body of theoretical work on the analysis and design of stream ciphers
- They have been used by the world's militaries since the invention of electronics.

Block Vs. Stream: Implementation

- Differences between stream ciphers and block ciphers are in the implementation.
- Stream ciphers that only encrypt and decrypt data one bit at a time are not really suitable for software implementation.
- Block ciphers can be easier to implement in software, because they often avoid time-consuming bit manipulations and they operate on data in computer-sized blocks.
- On the other hand, stream ciphers can be more suitable for hardware implementation because they can be implemented very efficiently in silicon.

Block Vs. Stream: Implementation

- It makes sense for a hardware encryption device on a digital communications channel to encrypt the individual bits as they go by. This is what the device sees.
- On the other hand, it makes no sense for a software encryption device to encrypt each individual bit separately.
- There are some specific instances where bit- and byte-wise encryption might be necessary in a computer system — encrypting the link between the keyboard and the CPU, for example — but generally the encryption block should be at least the width of the data bus.

Break

Using Algorithms

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

General Security

- Think of security — data security, communications security, information security, whatever — as a chain.
- The security of the entire system is only as strong as the weakest link.
- Everything has to be secure: cryptographic algorithms, protocols, key management, and more.
- If your algorithms are great but your random-number generator stinks, any smart cryptanalyst is going to attack your system through the random-number generation.
- If you patch that hole but forget to securely erase a memory location that contains the key, a cryptanalyst will break your system via that route.
- If you do everything right and accidentally e-mail a copy of your secure files to *The Wall Street Journal*, you might as well not have bothered.

Designer's task

- It's not fair. As the designer of a secure system, you have to think of every possible means of attack and protect against them all, but a cryptanalyst only has to find one hole in your security and exploit it.
- Cryptography is only a part of security, and often a very small part.
- It is the mathematics of making a system secure, which is different from actually making a system secure.
- Cryptography has its "size queens": people who spend so much time arguing about how long a key should be that they forget about everything else.

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

Choosing an Algorithm

When it comes to evaluating and choosing algorithms, people have several alternatives:

- They can choose a published algorithm, based on the belief that a published algorithm has been scrutinized by many cryptographers; if no one has broken the algorithm yet, then it must be pretty good.
- They can trust a manufacturer, based on the belief that a well-known manufacturer has a reputation to uphold and is unlikely to risk that reputation by selling equipment or programs with inferior algorithms.
- They can trust a private consultant, based on the belief that an impartial consultant is best equipped to make a reliable evaluation of different algorithms.
- They can trust the government, based on the belief that the government is trustworthy and wouldn't steer its citizens wrong.
- They can write their own algorithms, based on the belief that their cryptographic ability is second-to-none and that they should trust nobody but themselves.

Choosing an Algorithm

- Putting your trust in a single manufacturer, consultant, or government is asking for trouble.
- Most people who call themselves security consultants usually don't know anything about encryption. Most security product manufacturers are no better.
- The algorithms in this course are public. Most have appeared in the open literature and many have been cryptanalyzed by experts in the field.
- We don't have access to the cryptanalysis done by any of the myriad military security organizations in the world (which are probably better than the academic institutions — they've been doing it longer and are better funded), so it is possible that these algorithms are easier to break than it appears.
- The hole in all this reasoning is that we don't know the abilities of the various military cryptanalysis organizations. ☺

NSA capability

- What algorithms can the NSA break? For the majority of us, there's really no way of knowing.
- If you are arrested with a DES-encrypted computer hard drive, the FBI is unlikely to introduce the decrypted plaintext at your trial; the fact that they can break an algorithm is often a bigger secret than any information that is recovered.
- A good working assumption is that the NSA can read any message that it chooses, but that it cannot read all messages that it chooses.
  - The NSA is limited by resources, and has to pick and choose among its various targets.
- In any case, the best most of us can do is to choose among public algorithms that have withstood a reasonable amount of public scrutiny and cryptanalysis.

Algorithms for Export

- Algorithms for export out of the United States must be approved by the U.S. government.
- It is widely believed that these export-approved algorithms can be broken by the NSA.
- NSA gets a copy of the source code, but the algorithm's details remain secret from everyone else.
- Certainly no one advertises any of these deliberate weaknesses, but beware if you buy a U.S. encryption product that has been approved for export.

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

Public Vs. Symmetric

- Which is better, public-key cryptography or symmetric cryptography? This question doesn't make any sense, but has been debated since public-key cryptography was invented.
- The debate assumes that the two types of cryptography can be compared on an equal footing. They can't.
- Needham and Schroeder pointed out that the number and length of messages are far greater with public-key algorithms than with symmetric algorithms.
  - Their conclusion was that the symmetric algorithm was more efficient than the public-key algorithm.
  - While true, this analysis overlooks the significant security benefits of public-key cryptography.

RSA Vs. DES

- Whitfield Diffie wrote:
  - "In viewing public-key cryptography as a new form of cryptosystem rather than a new form of key management, I set the stage for criticism on grounds of both security and performance. Opponents were quick to point out that the RSA system ran about one-thousandth as fast as DES and required keys about ten times as large. Although it had been obvious from the beginning that the use of public key systems could be limited to exchanging keys for conventional [symmetric] cryptography, it was not immediately clear that this was necessary. In this context, the proposal to build *hybrid* systems was hailed as a discovery in its own right."

Public Vs. Symmetric

- Public-key cryptography and symmetric cryptography are different sorts of animals; they solve different sorts of problems.
- Symmetric cryptography is best for encrypting data.
- It is orders of magnitude faster and is not susceptible to chosen-ciphertext attacks.
- Public-key cryptography can do things that symmetric cryptography can’t; it is best for key management and a myriad of protocols.

Classes of Algorithms

0011

Classes of Algorithms

Algorithm	Confidentiality	Authentication	Integrity	Key Management
Symmetric encryption algorithms	Yes	No	No	Yes
Public-key encryption algorithms	Yes	No	No	Yes
Digital signature algorithms	No	Yes	Yes	No
Key-agreement algorithms	Yes	Optional	No	Yes
One-way hash functions	No	No	Yes	No
Message authentication codes	No	Yes	Yes	No

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

Encrypting Communications Channels

- This is the classic Alice and Bob problem: Alice wants to send Bob a secure message. What does she do? She encrypts the message.
- In theory, this encryption can take place at any layer in the OSI (Open Systems Interconnect) communications model.
- In practice, it takes place either at the lowest layers (one and two) or at higher layers.
- If it takes place at the lowest layers, it is called link-by-link encryption; everything going through a particular data link is encrypted.
- If it takes place at higher layers, it is called end-to-end encryption; the data are encrypted selectively and stay encrypted until they are decrypted by the intended final recipient.
- Each approach has its own benefits and drawbacks.

Link-by-Link Encryption

- The easiest place to add encryption is at the physical layer. This is called link-by-link encryption.
- The interfaces to the physical layer are generally standardized and it is easy to connect hardware encryption devices at this point.
- These devices encrypt all data passing through them, including data, routing information, and protocol information.
- They can be used on any type of digital communication link.
- On the other hand, any intelligent switching or storing nodes between the sender and the receiver need to decrypt the data stream before processing it.

Link Encryption

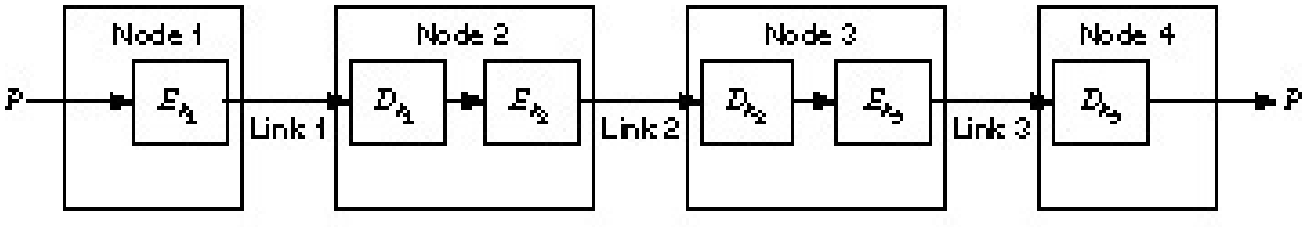


Figure 10.1 Link encryption.

Link Encryption

- This type of encryption is very effective. Because everything is encrypted, a cryptanalyst can get no information about the structure of the information.
- He has no idea who is talking to whom, how long the messages they are sending are, what times of day they communicate, and so on.
- This is called traffic-flow security: the enemy is not only denied access to the information, but also access to the knowledge of where and how much information is flowing.
- Security does not depend on any traffic management techniques.
- Key management is also simple; only the two endpoints of the line need a common key, and they can change their key independently from the rest of the network.

Link-by-Link Encryption

Link-by-Link Encryption: Advantages and Disadvantages

Advantages:

Easier operation, since it can be made transparent to the user. That is, everything is encrypted before being sent over the link.

Only one set of keys per link is required.

Provides traffic-flow security, since any routing information is encrypted.

Encryption is online.

Disadvantages:

Data is exposed in the intermediate nodes.

- The biggest problem with encryption at the physical layer is that each physical link in the network needs to be encrypted: Leaving any link unencrypted jeopardizes the security of the entire network.
- If the network is large, the cost may quickly become prohibitive for this kind of encryption.

End-to-End Encryption

- Another approach is to put encryption equipment between the network layer and the transport layer.
- The encryption device must understand the data according to the protocols up to layer three and encrypt only the transport data units, which are then recombined with the unencrypted routing information and sent to lower layers for transmission.
- This approach avoids the encryption/decryption problem at the physical layer.
- By providing end-to-end encryption, the data remains encrypted until it reaches its final destination

End-to-End Encryption

- The primary problem with end-to-end encryption is that the routing information for the data is not encrypted;
  - a good cryptanalyst can learn much from who is talking to whom, at what times and for how long, without ever knowing the contents of those conversations.
- Key management is also more difficult, since individual users must make sure they have common keys.

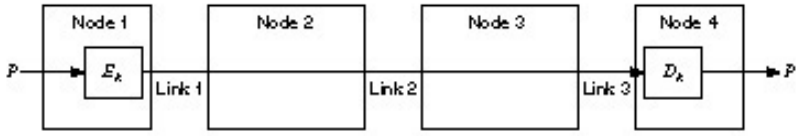


Figure 10.2 *End-to-end encryption.*

End-to-End Encryption

- If encryption takes place at a high layer of the communications architecture, like the applications layer or the presentation layer, then it can be independent of the type of communication network used.
- It is still end-to-end encryption, but the encryption implementation does not have to bother about line codes, synchronization between modems, physical interfaces, and so forth.

End-to-End Encryption: Traffic Analysis

- The major disadvantage of end-to-end encryption is that it allows traffic analysis.
  - Traffic analysis is the analysis of encrypted messages: where they come from, where they go to, how long they are, when they are sent, how frequent or infrequent they are, whether they coincide with outside events like meetings, and more.
- A lot of good information is buried in that data, and a cryptanalyst will want to get his hands on it.

End-to-End Encryption: Advantages and Disadvantages

Advantages:

Higher secrecy level.

Disadvantages:

Requires a more complex key-management system.

Traffic analysis is possible, since routing information is not encrypted.

Encryption is offline.

Combining the Two

- Combining link-by-link and end-to-end encryption. while most expensive, is the most effective way of securing a network.
- Encryption of each physical link makes any analysis of the routing information impossible, while end-to-end encryption reduces the threat of unencrypted data at the various nodes in the network.
- Key management for the two schemes can be completely separate: The network managers can take care of encryption at the physical level, while the individual users have responsibility for end-to-end encryption.

Comparing link with end

Comparing Link-by-Link and End-to-End Encryption

Link-by-link encryption	End-to-end encryption
Security within Hosts	
Message exposed in sending host	Message encrypted in sending host
Message exposed in intermediate nodes	Message encrypted inintermediate nodes
Role of User	
Applied by sending host	Applied by sending process
Invisible to user	User applies encryption
Host maintains encryption	User must find algorithm
One facility for all users	User selects encryption
Can be done in hardware	More easily done in software
All or no messages encrypted	User chooses to encrypt or not, for each message
Implementation Concerns	
Requires one key per host pair	Requires one key per user pair
Requires encryption hardware or software at each host	Requires encryption hardware or software at each node
Provides node authentication	Provides user authentication

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

Data storage

- In communications channels, messages in transit have no intrinsic value. If Bob doesn't receive a particular message, Alice can always resend it.
- This is not true for data encrypted for storage. If Alice can't decrypt her message, she can't go back in time and re-encrypt it. She has lost it forever.
- This means that encryption applications for data storage should have some mechanisms to prevent unrecoverable errors.
- The encryption key has the same value as the message, only it is smaller.
- In effect, cryptography converts large secrets into smaller ones. Being smaller, they can be easily lost.
- Key management procedures should assume that the same keys will be used again and again, and that data may sit on a disk for years before being decrypted.

Encrypting data for storage

- Other problems particular to encrypting computer data for storage were listed in:
  - The data may also exist in plaintext form, either on another disk, in another computer, or on paper. There is much more opportunity for a cryptanalyst to perform a known-plaintext attack.
  - In database applications, pieces of data may be smaller than the block size of most algorithms. This will cause the ciphertext to be considerably larger than the plaintext.
  - The speed of I/O devices demands fast encryption and decryption, and will probably require encryption hardware. In some applications, special high-speed algorithms may be required.
  - Safe, long-term storage for keys is required.
  - Key management is much more complicated, since different people need access to different files, different portions of the same file, and so forth.

Dereferencing Keys

- When encrypting a large hard drive, you have two options.
- You can encrypt all the data using a single key.
  - This gives a cryptanalyst a large amount of ciphertext to analyze and makes it impossible to allow multiple users to see only parts of the drive.
- Or, you can encrypt each file with a different key, forcing users to memorize a different key for each file.
- Another solution is to encrypt each file with a separate key, and to encrypt the keys with another key known by the users.
  - Each user only has to remember that one key.
  - Different users can have different subsets of the file-encryption keys encrypted with their key.
  - And there can even be a master key under which every file-encryption key is encrypted. This is even more secure because the file-encryption keys are random and less susceptible to a dictionary attack.

Driver-Level vs. File-Level Encryption

- There are two ways to encrypt a hard drive: at the file level and at the driver level.
- Encryption at the file level means that every file is encrypted separately. To use a file that's been encrypted, you must first decrypt the file, then use it, and then re-encrypt it.
- Driver-level encryption maintains a logical drive on the user's machine that has all data on it encrypted.
- If done well, this can provide security that, beyond choosing good passwords, requires little worry on the part of the user.
  - The driver must be considerably more complex than a simple file-encryption program, however, because it must deal with the issues of being an installed device driver, allocation of new sectors to files, recycling of old sectors from files, random-access read and update requests for any data on the logical disk, and so on.
  - Typically, the driver prompts the user for a password before starting up. This is used to generate the master decryption key, which may then be used to decrypt actual decryption keys used on different data.

File-Level Vs. Driver-Level

Comparing File-Level and Driver-Level Encryption

File-Level Encryption	Driver-Level Encryption
Benefits:	
Ease of implementation and use. Flexible. Relatively small performance penalty. Users can move files between different machines without problems. Users can back files up without problems.	Temporary files, work files, and so forth can be kept on the secure drive. It's harder to forget to re-encrypt something on this kind of system.
Security Issues:	
Potential leakage through security-unconscious programs. (Program may write file to disk for temporary storage, for example.) Bad implementations may always re-encrypt with same key for same password.	Lots of things can go wrong with a device-driver or memory-resident program. Bad implementations will allow chosen-plaintext, or even chosen-ciphertext attacks. If whole system is master-keyed under one password, loss of that password means that the attacker gets everything. A more limited set of ciphers can reasonably be used for this kind of application. For example, OFB stream ciphers would not work.
Usability Problems:	
User has to figure out what to do. There may be different passwords for different files. Manual encryption of selected files is the only access control.	There will be a performance penalty. The driver may interact in weird ways with Windows, OS/2 DOS emulation, device drivers, and so on.

Providing Random Access to an Encrypted Drive

- Most systems expect to be able to access individual disk sectors randomly. This adds some complication for using many stream ciphers and block ciphers in any chaining mode. Several solutions are possible.
- Use the sector address to generate a unique IV for each sector being encrypted or decrypted.
- The drawback is that each sector will always be encrypted with the same IV. Make sure this is not a security problem.
- You can use a block cipher with a large enough block size that it can encrypt the whole sector at once.

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

Hardware

- Until very recently, all encryption products were in the form of specialized hardware. These encryption/decryption boxes plugged into a communications line and encrypted all the data going across that line.
- Although software encryption is becoming more prevalent today, hardware is still the embodiment of choice for military and serious commercial applications.
- The NSA, for example, only authorizes encryption in hardware.
- There are several reasons why this is so. => next slides

Hardware Vs. Software: Speed

- The first reason is speed.
- As we will see, encryption algorithms consist of many complicated operations on plaintext bits. These are not the sorts of operations that are built into your run-of-the-mill computer.
- The two most common encryption algorithms, DES and RSA, run inefficiently on general-purpose processors.
- While some cryptographers have tried to make their algorithms more suitable for software implementation, specialized hardware will always win a speed race.

Hardware Vs. Software: Security

- The second reason is security.
- An encryption algorithm running on a generalized computer has no physical protection. Mallory can go in with various debugging tools and surreptitiously modify the algorithm without anyone ever realizing it.
- Hardware encryption devices can be securely encapsulated to prevent this.
- Tamperproof boxes can prevent someone from modifying a hardware encryption device.
- Special-purpose VLSI chips can be coated with a chemical such that any attempt to access their interior will result in the destruction of the chip's logic.
- The U.S. government's Clipper and Capstone chips) are designed to be tamperproof. The chips can be designed so that it is impossible for Mallory to read the unencrypted key.

Hardware Vs. Software: Installation

- The final reason for the prevalence of hardware is the ease of installation.
- Most encryption applications don’t involve general-purpose computers.
- People may wish to encrypt their telephone conversations, facsimile transmissions, or data links.
- It is cheaper to put special-purpose encryption hardware in the telephones, facsimile machines, and modems than it is to put in a microprocessor and software.

TEMPEST

- Electromagnetic radiation can sometimes reveal what is going on inside a piece of electronic equipment.
- Dedicated encryption boxes can be shielded, so that they leak no compromising information.
- General-purpose computers can be shielded as well, but it is a far more complex problem.
- The U.S. military calls this **TEMPEST**; it's a subject well beyond the scope of this book.

Encryption Hardware

- The three basic kinds of encryption hardware on the market today are:
  - self-contained encryption modules (that perform functions such as password verification and key management for banks)
  - dedicated encryption boxes for communications links
  - boards that plug into personal computers.

Encryption Hardware

- Some encryption boxes are designed for certain types of communications links, such as T-1 encryption boxes.
- PC-board encryptors usually encrypt everything written to the hard disk and can be configured to encrypt everything sent to the floppy disk and serial port as well.
  - These boards are not shielded against electromagnetic radiation or physical interference, since there would be no benefit in protecting the boards if the computer remained unaffected.
- More companies are starting to put encryption hardware into their communications equipment. Secure telephones, facsimile machines, and modems are all available.

Software Encryption

- Any encryption algorithm can be implemented in software.
- The disadvantages are in speed, cost, and ease of modification (or manipulation).
- The advantages are in flexibility and portability, ease of use, and ease of upgrade.
- The algorithms written in C (at the end of the book) can be implemented, with little modification, on any computer.
- They can be inexpensively copied and installed on many machines.
- They can be incorporated into larger applications, such as communications programs or word processors.

Software Encryption

- Software encryption programs are popular and are available for all major operating systems.
- These are meant to protect individual files; the user generally has to manually encrypt and decrypt specific files.
- It is important that the key management scheme be secure:
  - The keys should not be stored on disk anywhere (or even written to a place in memory from where the processor swaps out to disk).
  - Keys and unencrypted files should be erased after encryption.

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

Compression, Encoding, and Encryption

- Using a data compression algorithm together with an encryption algorithm makes sense for two reasons:
  - Cryptanalysis relies on exploiting redundancies in the plaintext; compressing a file before encryption reduces these redundancies.
  - Encryption is time-consuming; compressing a file before encryption speeds up the entire process.
- The important thing to remember is to compress before encryption.
- If the encryption algorithm is any good, the ciphertext will not be compressible; it will look like random data. (This makes a reasonable test of an encryption algorithm; if the ciphertext can be compressed, then the algorithm probably isn't very good.)

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

Detecting Encryption

- How does Eve detect an encrypted file?
- Other file encryptors just produce a ciphertext file of seemingly random bits. How can she distinguish it from any other file of seemingly random bits? There is no sure way, but Eve can try a number of things:
  - Examine the file. ASCII text is easy to spot. Other file formats, such as TIFF, TeX, C, Postscript, G3 facsimile, or Microsoft Excel, have standard identifying characteristics. Executable code is detectable, as well. UNIX files often have “magic numbers” that can be detected.
  - Try to uncompress the file, using the major compression algorithms. If the file is compressed (and not encrypted), this should yield the original file.
  - Try to compress the file. If the file is ciphertext (and the algorithm is good), then the probability that the file can be appreciably compressed by a general-purpose compression routine is small. (By appreciably, I mean more than 1 or 2 percent.) If it is something else (a binary image or a binary data file, for example) it probably can be compressed.

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

Hiding Ciphertext in Ciphertext

- Wouldn't it be nice to be able to encrypt a file such that there are two possible decryptions, each with a different key.
- Alice could encrypt a real message to Bob in one of the keys and some innocuous message in the other key.
- If Alice were caught, she could surrender the key to the innocuous message and keep the real key secret.
- The easiest way to do this is with one-time pads. Let *P* be the plaintext, *D* the dummy plaintext, *C* the ciphertext, *K* the real key, and *K'* the dummy key. Alice encrypts *P*:
  - $P \oplus K = C$
- Alice and Bob share *K*, so Bob can decrypt *C*:
  - $C \oplus K = P$
- If the secret police ever force them to surrender their key, they don't surrender *K*, but instead surrender:
  - $K' = C \oplus D$
- The police then recover the dummy plaintext:
  - $C \oplus K' = D$
- Since these are one-time pads and *K* is completely random, there is no way to prove that *K'* was not the real key.

Outline

Using Algorithms

- 10.1 Choosing an Algorithm
- 10.2 Public-Key Cryptography versus Symmetric Cryptography
- 10.3 Encrypting Communications Channels
- 10.4 Encrypting Data for Storage
- 10.5 Hardware Encryption versus Software Encryption
- 10.6 Compression, Encoding, and Encryption
- 10.7 Detecting Encryption
- 10.8 Hiding Ciphertext in Ciphertext
- 10.9 Destroying Information

Destroying Information

- When you delete a file on most computers, the file isn't really deleted.
- The only thing deleted is an entry in the disk's index file, telling the machine that the file is there. Many software vendors have made a fortune selling file-recovery software that recovers files after they have been deleted.
- And there's yet another worry: Virtual memory means your computer can read and write memory to disk any time.
  - Even if you don't save it, you never know when a sensitive document you are working on is shipped off to disk. This means that even if you never save your plaintext data, your computer might do it for you.
- To erase a file so that file-recovery software cannot read it, you have to physically write over all of the file's bits on the disk.

Overwritting

According to the National Computer Security Center:

- Overwriting is a process by which unclassified data are written to storage locations that previously held sensitive data.... To purge the...storage media, the DoD requires overwriting with a pattern, then its complement, and finally with another pattern; e.g., overwrite first with 0011 0101, followed by 1100 1010, then 1001 0111. The number of times an overwrite must be accomplished depends on the storage media, sometimes on its sensitivity, and sometimes on different DoD component requirements. In any case, a purge is not complete until a final overwrite is made using unclassified data.

Overwritting

- Most commercial programs that claim to implement the DoD standard overwrite three times: first with all ones, then with all zeros, and finally with a repeating one-zero pattern.
- Schneier recommends overwriting a deleted file seven times: the first time with all ones, the second time with all zeros, and five times with a cryptographically secure pseudo-random sequence.

End of Lecture