CS 494/594
Computer and Network Security

Dr. Jinyuan (Stella) Sun
Dept. of Electrical Engineering and Computer Science
University of Tennessee
Fall 2010
Secret Key Cryptography

- Modes of operation
- Stream cipher
Encrypting A Large Message

How to encrypt a message > 64 bits?
- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Output Feedback Mode (OFB)
- Cipher Feedback Mode (CFB)
- Counter Mode (CTR)
**ECB Mode**

- **ECB Encryption**
  - Message is broken into 64-bit blocks
  - Each block is independently encoded with the same secret key

- **ECB Decryption**
  - Decrypt with secret key
  - Reassemble blocks

• Message is broken into 64-bit blocks
• Each block is independently encoded with the same secret key
Pros and Cons of ECB

• Suitable for use in secure transmission of single values (e.g. an encryption key)
• Error in one received ciphertext block does not affect the correct decryption of other ciphertext blocks
• Identical plaintext blocks produce identical ciphertext blocks resulting in recognizable pattern

• Ciphertext blocks can be easily rearranged or modified
CBC Mode

CBC Encryption

- Selects a random number: IV (initialization vector) that is XORed with the first plaintext block. Why?
- Then generates its own random numbers: the ciphertext from the previous block, XORed with the next plaintext block

CBC Decryption
Pros and Cons of CBC

- Suitable for use in general-purpose block-oriented transmission, and authentication
- The same block repeating in the plaintext will not cause repeats in the ciphertext
- Subject to modification attack: (but error propagates)
  - Subject to ciphertext block rearranging attack
- IV: needs to be shared between sender and receiver, either a fixed value or sent encrypted (How to encrypt?)
AES Example: ECB vs. CBC

AES in ECB mode

AES in CBC mode

Similar plaintext blocks produce similar ciphertext blocks (not good!)
Output Feedback Mode (OFB)

- OFB is a stream cipher: encryption is done by XORing plaintext with one-time pad
- One-time pad: $b_0|b_1|b_2|b_3\ldots$, where $b_0$ is a random 64-bit IV, $b_1$ is the secret key encrypted $b_0$, and so on…
Pros and Cons of OFB

- Suitable for use in stream-oriented transmission over noisy channel (e.g., satellite communication)
- One-time pad can be generated in advance, only XOR operations are performed in real-time
- Bit errors do not propagate: error in one ciphertext block only grables the corresponding plaintext block
- Message can arrive in arbitrarily sized chunks, get encrypted and transmitted immediately
- Plaintext modification attack: if attacker knows <plaintext, ciphertext>, he can XOR the plaintext and ciphertext, and XOR the result with any message of his choosing
- Must not reuse the same IV or secret key (Why?)
Cipher Feedback Mode (CFB)

k-bit CFB

- Similar to OFB
- k bits shifted in the register are the k bits of ciphertext from the previous block (k can be any number: 1, 8, 64, 128, etc.)
Pros and Cons of CFB

• Suitable for use in general-purpose stream-oriented transmission, and authentication
• Less subject to tampering: with k-bit CFB, the change of any k-bit of plaintext in a predictable way will cause unpredictably garbling the next b/k blocks
• One-time pad cannot be pre-computed, encryption needs to be done in real-time
• Error in a k-bit ciphertext block propagates: it garbles the next b/k plaintext blocks
Counter Mode (CTR)

- Similar to OFB
- Instead of chaining the encryption of one-time pad, the IV is incremented and encrypted to get successive blocks of the one-time pad
Pros and Cons of CTR

• Suitable for use in general-purpose block-oriented transmission, and high speed encryption
• One-time pad can be pre-computed
• Decrypting at any point rather than the beginning: ideal for random access applications
• Hardware/software efficiency: parallel encryption/decryption on multiple blocks of plaintext or ciphertext
• Provable security: at least as secure as other modes
• Simplicity: unlike ECB and CBC, no decryption algorithm is needed in CTR
• Must not reuse the same IV or key, same as OFB
• An attacker could get the XOR of two plaintext blocks by XORing the two corresponding ciphertext blocks
Generating MACs

• Integrity: protect against undetected modifications, cannot be guaranteed by any mode of operation if attacker knows the plaintext
• Plaintext + CBC residue (when message not secret)
Privacy and Integrity

- Privacy: CBC encryption
- Integrity: CBC residue
- Ciphertext + CBC residue?
- Encrypt \{plaintext + CBC residue\}?
- Encrypt \{plaintext + CRC\}?
Privacy and Integrity (Cont’d)

- Privacy: CBC encryption + Integrity: CBC residue, but with different keys
- CBC + weak cryptographic checksum
- CBC + CBC residue with related keys
- CBC + cryptographic hash
- OCB: offset codebook mode
3DES: CBC Outside vs. Inside

CBC on the outside
(Why this one?)

CBC on the inside
Stream Ciphers

- A key is input into a pseudorandom generator to produce a pseudorandom keystream
- Pseudorandom stream: unpredictable without knowing key
- Keystream is bitwise XORed with plaintext stream
Design Considerations

- The encryption sequence should have a large period without repetitions
- The keystream should approximate the properties of a true random number stream as close as possible
- Input key need be sufficiently long
- When properly designed, a stream cipher can be as secure as block cipher of comparable key length
- Advantage of stream ciphers: almost always faster and use far less code than block ciphers
RC4

- Designed by Ron Rivest in 1987 for RSA security
- Variable key-size stream cipher with byte-oriented applications
- Popular uses: SSL/TLS (Secure Sockets Layer/Transport Layer Security), WEP (Wired Equivalent Privacy) protocol and the newer WiFi Protected Access (WPA)
- A variable-length key is used to initialize a 256-byte state vector $S$
- A byte $k$ is generated from $S$ by selecting one of the 255 entries for encryption/decryption
- The entries in $S$ are permuted after generating each $k$
RC4 (Cont’d)
RC4 Keystream Generation

```c
/* Initialization */
for i = 0 to 255 do
    S[i] = i;
    T[i] = K[i mod keylen];

/* Initial Permutation of S */
j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    Swap (S[i], S[j]);

/* Stream Generation */
i, j = 0;
while (true)
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
    Swap (S[i], S[j]);
    t = (S[i] + S[j]) mod 256;
    k = S[t];
```
Strength of RC4

- No practical attack on RC4 is known
- Must not reuse key
- A known vulnerability in WEP: relevant to the generation of the key input to RC4 but not RC4 itself
Reading Assignments

- [Kaufman] Chapter 4