Some More Symmetric Key Ciphers

Information Security – Lecture 06

Aadil Zia Khan

Advanced Encryption Standard (AES)

- DES was vulnerable to bruteforcing the 56bit key
- Triple-DES had a larger key but was slow (especially if implemented in software)
- US NIST issued call for ciphers in 1997
- Rijndael was selected as the AES in 2000
- Unlike DES, AES is not a Feistel Cipher
    - It operates on entire data block in every round unlike Feistel which operates on halves at a time

Advanced Encryption Standard (AES)

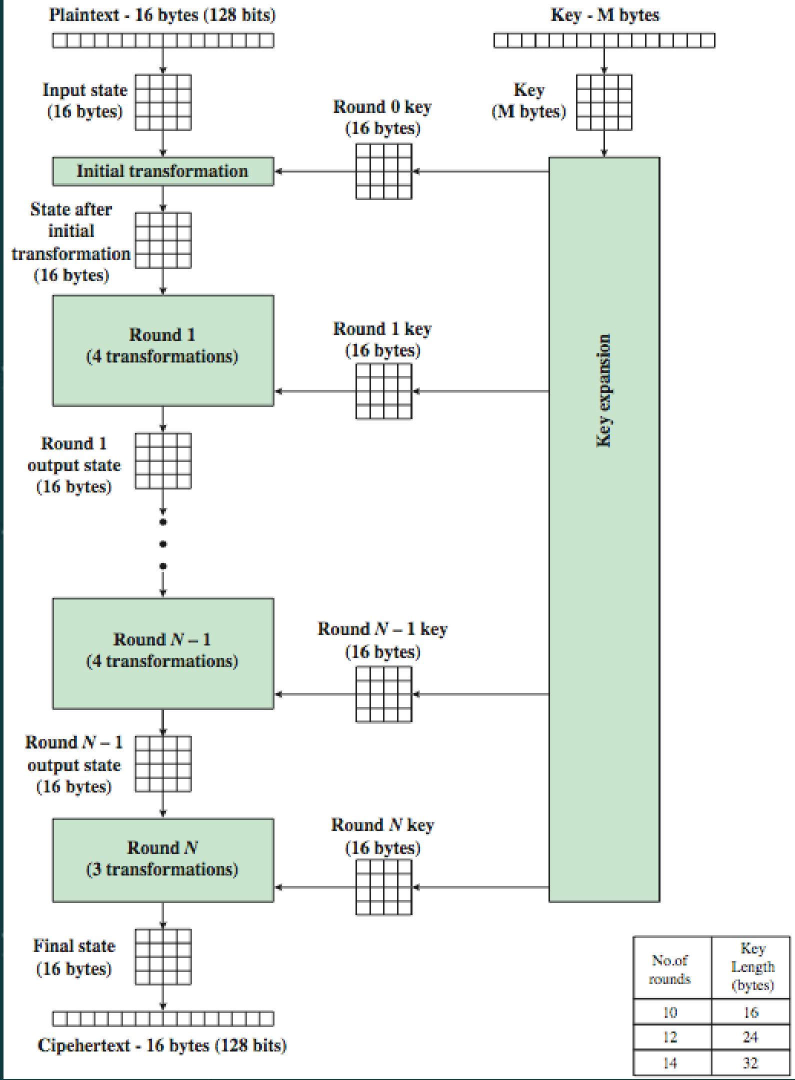
- Data split into 128bit blocks
- Each block treated as a 4x4 grid
- Key size can be 128/192/256 bits
- Key size determines the total number of rounds
  - 10/12/14 for sizes 128/192/256

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

Advanced Encryption Standard (AES)

Operations inside each round:

- Generate the round key of size 128bits
- XOR the round key and plaintext
- Substitute the bytes using a lookup table
- Left-shift the rows inside the 4x4 grid
  - first row by 0bytes, second by 1byte, third by 2bytes, fourth by 3bytes
- Mix the columns inside the 4x4 grid
  - Achieved through matrix multiplication
- XOR the resulting bytes with the round key



No.of rounds	Key Length (bytes)
10	16
12	24
14	32

Is AES Secure?

- No known practical attack that would allow someone without knowledge of the key to read data encrypted by AES

Block Cipher - Modes of Operation

- Electronic Codebook Mode
  - Each block of plaintext bits is encoded independently using the same key
  - Repeating keys is not very secure
  - Typically used for secure transmission of single values - e.g.,an encryption key

- Cipher Block Chaining Mode
  - To encrypt a block, you first XOR the current plaintext block with the preceding block's ciphertext
  - For the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext
  - Using this approach, the repeating patterns of bits are not exposed – unlike Electronic Codebook Mode

Block Cipher - Modes of Operation

- Cipher Feedback Mode / Output Feedback Mode
  - Same as Cipher Block Chaining Mode – difference being that the size doesn't have to be a fixed block
  - Convert any block cipher into a stream cipher – no need to pad a message
  - Can operate in real time
  - Each character can be encrypted and transmitted immediately

- Counter Mode
  - A counter equal to the plaintext block size is used
  - The counter is encrypted using the key and then XORed with the plaintext block to produce the ciphertext block - there is no chaining
  - Counter is incremented for each block – starting counter value may be random

Why Need Random Number Generation

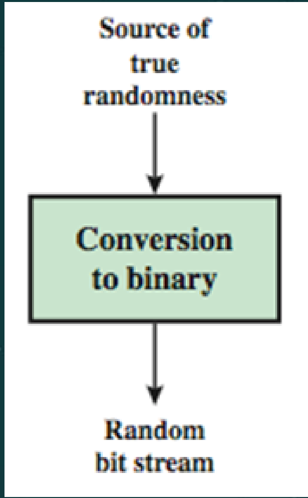
- Several security algorithms based on cryptography make use of random numbers
  - Generation of keys for the public-key encryption algorithms
  - Generation of a stream key for symmetric stream cipher
  - Generation of a symmetric key for use as a temporary session key for networking apps
  - Needed for key distribution protocols

Randomness Criteria

- Following criteria are used to validate that a sequence of bits is indeed random
  - Uniform distribution: distribution of bits in the sequence should be uniform – frequency of occurrence of ones and zeros should be approximately the same
  - Independence: no one subsequence in the sequence can be inferred from the others

True Random Number Generation

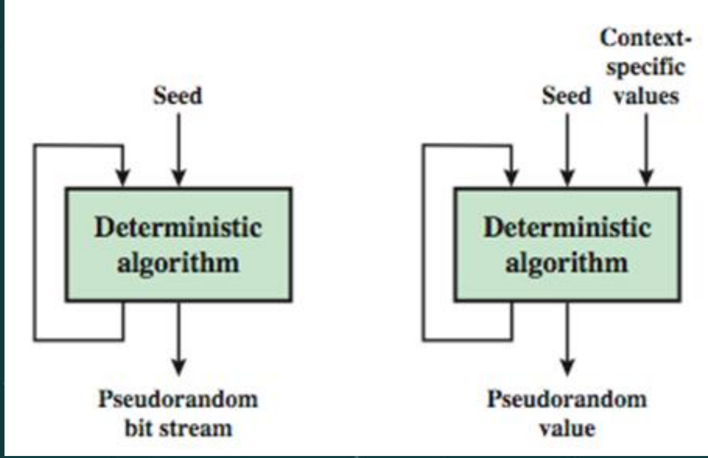
- Takes as input a source that is random - source referred to as an entropy source
- The entropy source is drawn from the physical environment of the computer
    - Keystroke timing patterns
    - Disk electrical activity
    - Mouse movements
    - Instantaneous values of the system clock
    - Intel has developed a commercially available chip that samples thermal noise
- True Random Number Generator (TRNG) then produces a random binary output using the input
    - May simply involve conversion of an analog source to a binary output with some additional processing



Time to Apply Your Knowledge

- How can you generate a random number between 0 and 7?
  - Note that the number between 0 and 7 can be represented using three bits
  - Toss a coin three times – let head represent 0 and tail represent 1
  - E.g.
    - HHH represents 000 which is 0
    - THT represents 101 which is 5
    - TTT represents 111 which is 7

Pseudorandom Number Generation

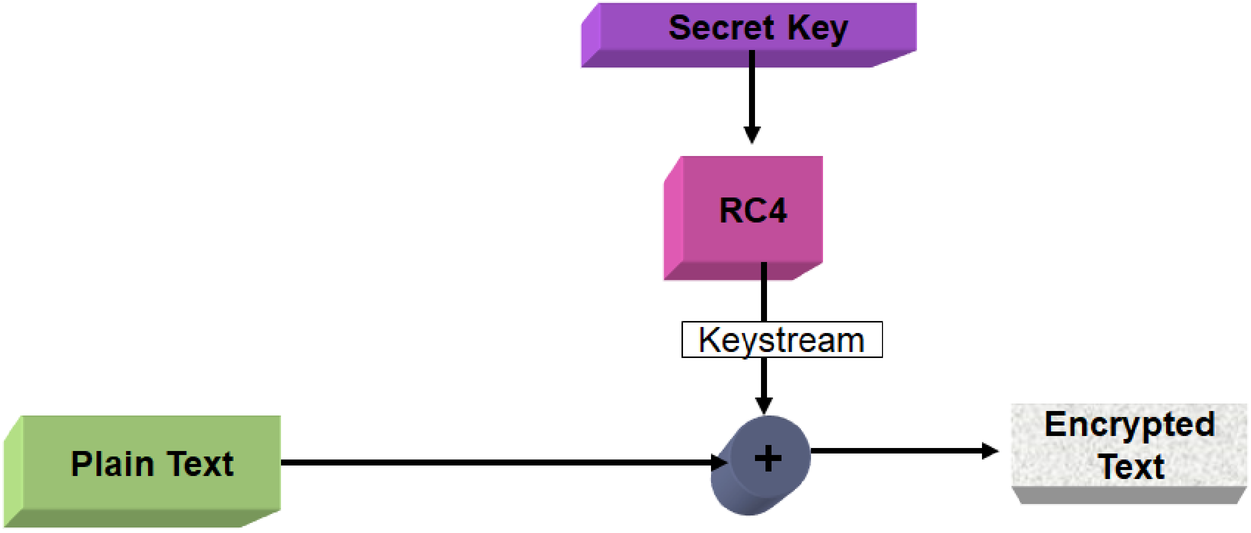


- Takes as input a fixed value, called the seed, and produces a sequence of output bits using a deterministic algorithm
    - Not really random
    - Adversary who knows the algorithm and the seed can reproduce the entire pseudorandom bit stream
- Pseudorandom Number Generator (PRNG): produces an open-ended sequence of bits
- Pseudorandom Function (PRF): produces a pseudorandom string of bits of some fixed length
    - Examples are symmetric encryption keys and nonces – (*nonce is an arbitrary number used only once in a cryptographic communication*)
    - Usually the PRF takes as input a seed plus some context specific values, such as a user/application ID

Stream Cipher: RC4

- A symmetric key encryption algorithm invented by Ron Rivest
  - A proprietary cipher owned by RSA, kept secret
  - Leaked anonymously in Cyberpunks mailing list in 1994
- Used in
  - SSL/TLS (Secure socket, transport layer security) between web browsers and servers
  - Wireless LAN

Stream Cipher: RC4



Cryptographically very strong and easy to implement

RC4: The Fine Details
Initializations

- Use a variable-length key K of size between 8 and 2048 bits
- Create a byte array S of size 256 – this is the state array
  - Entries of S are set equal to the values from 0 through 255 in ascending order
    - S[0] = 0, S[1] = 1, . . . , S[255] = 255
- Create a temporary byte array T and copy the key to it
  - if the length of K is 256 bytes copy all of it to T - for a larger key of length ignore the remaining bits – for a smaller key, repeat it as many times as needed

RC4: The Fine Details
Initial Permutation

```
/* 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]);
```

- Once the S array is initialized, the input key is no longer used

RC4: The Fine Details
Pseudorandom Stream Generation

- After S[255] is reached, the process continues, starting over again at S[0]
- Value of k will be used for encryption

```
/* 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];
```

RC4: The Fine Details
Encryption

- To encrypt, XOR the value k with the next byte of plaintext
- To decrypt, XOR the value k with the next byte of ciphertext.

Is RC4 Secure?

- Since RC4 simply XORs the plaintext with a random stream, it is vulnerable to a Reused key attack
  - If a key is reused, or the random number sequence repeats after a certain number of bits, the encryption can be broken using the simple known attacks on the XOR cipher

- Bit-flipping attack
  - Suppose an adversary knows the exact content of a part of the message – e.g., he knows a portion of an electronics fund transfer message contains the ASCII string "$1000.00“
  - He can change that to *"$9500.00"* by XORing that portion of the ciphertext with : *"$1000.00" xor "$9500.00"*.
    - Ciphertext consists of $1000.00 XOR key
    - XORing it with $1000.00 would give the key
    - XORing it with $9500.00 would give the ciphertext where the transaction is now of $9500 instead
    - The original message has now been changed

Galois Field

- It is finite set
- Any mathematical operation results in a value that exists within the finite set
- e.g., XORing two 1byte values would give a result that is also 1byte – no overflow or underflow
- e.g., Doing any mathematical operation followed by mod 256 would give a result that would remain in the range 0 to 255