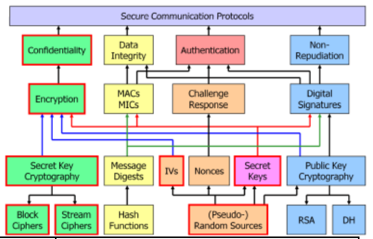
ISI - Secret Key Kryptography

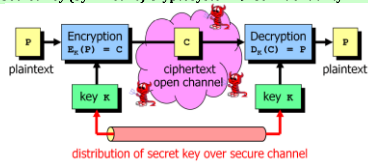
Building Blocks of Cryptography



DH	Diffie-Hellman public key cryptosystem
RSA	Rivest-Shamir-Adleman public key cryptosystem
IV	Initialization Vector, required to initialize symmetric encryption algorithms
Nonce	Random number, used in challenge-response protocols
MAC	Message Authentication Code, cryptographically secured checksum
MIC	Message Integrity Check - synonym for MAC

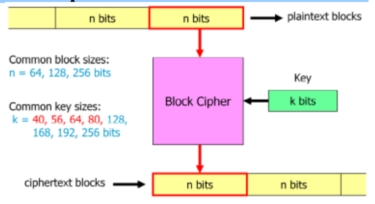
Shannon's Model of a Secrecy System

Secret Key (Symmetric) Cryptosystem → Confidentiality



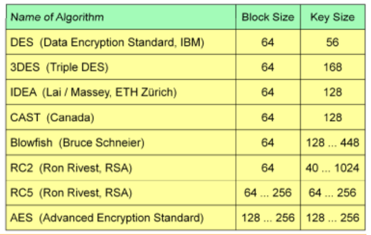
- The same, (pseudo-) randomly chosen key used for encryption and decryption
- Key must be kept absolutely secret
- Same key can be used for several messages, but should be changed periodically → **secret key distribution problem!**

Block Ciphers



- A block cipher cuts up a plaintext of arbitrary length into a series of blocks having a constant size of n bits. It then encrypts a single block of plaintext at a time and converts it into a block of ciphertext. Given any key, every possible plaintext block must result in a unique ciphertext block (so the mapping is 1:1) because otherwise, a ciphertext block could not be unambiguously decrypted. Feistel ciphers, a common class of block ciphers, achieve this by design

Some Block Ciphers



Name of Algorithm	Block Size	Key Size
DES (Data Encryption Standard, IBM)	64	56
3DES (Triple DES)	64	168
IDEA (Lai / Massey, ETH Zürich)	64	128
CAST (Canada)	64	128
Blowfish (Bruce Schneier)	64	128 ... 448
RC2 (Ron Rivest, RSA)	64	40 ... 1024
RC5 (Ron Rivest, RSA)	64 ... 256	64 ... 256
AES (Advanced Encryption Standard)	128 ... 256	128 ... 256

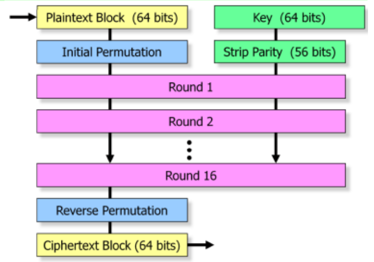
Most Popular Block Ciphers

- DES is still used, although you really shouldn't anymore
- 3DES is in widespread use
- IDEA is patented (by ASCOM, Switzerland), so it's not seen very frequently
- RC2, RC5, CAST and Blowfish are all used here and there, although they do not have major significance
- AES gets more and more used - if you need a block cipher, use AES unless you have good arguments not to use it!

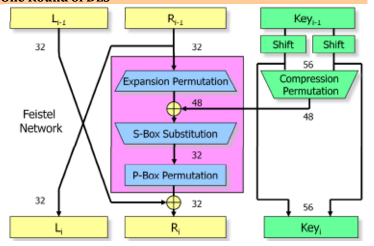
Data Encryption Standard (DES)

- Published in 1977 by the National Institute of Standards and Technology (NIST)
- The first widespread modern cipher, could be implemented in hard- and software, based on Shannon's principle of confusion and diffusion
- DES is conceptually a Feistel block cipher
- Based on IBM's Lucifer Cipher and inputs from the National Security Agency (NSA)
- Block size: 64 Bits
- Key size: 56 Bits
- Heavily cryptanalysed, no real weaknesses has been found
  - very well build, but little of the design ideas became public
- Involvement of NSA has always been subject of speculation
  - NSA might have built in a hidden backdoor
  - NSA has requested the key to be reduced from 64 to 56 Bits
- DES has been a highly successful cipher for years! (still is: 3DES)

16 Rounds of Confusion and Diffusion



One Round of DES



Feistel Networks

At the heart of each DES round is a Feistel network, named after the IBM scientist Horst Feistel. The 64 bit block of incoming plaintext is split into a right and a left half of 32 bits each. The right half becomes the left half of the output text block at the end of the round, but it also enters a black box where it is first expanded to 48 bits by an expansion permutation and then XOR-ed with a 48-bit wide key. The resulting sum then enters an array of eight S-boxes with 6 input lines and 4 output lines each, producing a 32 bit wide output which then gets permuted by a P-box. The S- and P-Box are those components that are most important for the security of DES, as they provide the core substitution (confusion) and transposition (diffusion) operations. The resulting output the black box is XOR-ed with the left half of the input text block and becomes the right half of the output text block. Each of the 16 DES rounds has a 48 bit key of its own, derived by continually shifting and permuting the full 56 bit key from round to round

DES Decryption

Looking at all the shifting, XOR-ing, substituting and permuting, one may thing that decrypting a ciphertext with DES is a completely different algorithm. In fact, it is not. As with all ciphers following the Feistel cipher design, nearly the same algorithm is used for decryption. We analyze this in detail by inverting (decrypting) a single round of DES, i.e. we want to derice L_{i-1} and R_{i-1} from L_i , R_i and K_{i-1}

- The illustration above shows that $L_i = R_{i-1}$, so $R_{i-1} = L_i$ → we already have R_{i-1}
- The illustration also shows that $R_i = L_{i-1}$ XOR $f(R_{i-1}, K_{i-1})$ where f() is the function performed by the "purple" box. Reforming this equation leads to $L_{i-1} = R_i$ XOR $f(R_{i-1}, K_{i-1})$. Since we know R_{i-1} to be equal to L_i from the step above and also know R_i and K_{i-1}, this allows to compute L_{i-1} and we have found both L_{i-1} and R_{i-1}

Note that decrypting DES does therefore not require to invert the "purple" box (where the actual "security of DES" is found). If one analyses decryption of DES even further, one can see that decryption of a DES round is exactly the same algorithm as encrypting a round when the two halves R and L are swapped, so feeding R_i|Li into a round and using key K_{i-1} produces R_{i-1}|L_{i-1} as its output. Applied to the entire DES algorithm, this means that decryption uses the same algorithm as encryption, with three minor modifications:

- The round keys are applied in reversed order
- After the initial permutation, the right and left halves of the permuted ciphertext must be swapped
- Before the reverse permutation is applied, the halves must be swapped again

Performance of DES

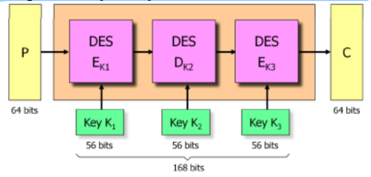
Although all these operations seem to be a lot of computational work, modern computers are capable of encrypting or decrypting "a few" million blocks per second

Security of DES

- Brute force attack is the best attack known
- 1977: Diffie, Hellman: with USD 20 million, a million-chip machine could be build to find a DES key of a plaintext-ciphertext pair in 12 hours
- 1998: EFF built a DES-breaking machine, the EFF DES Cracker for USD 250'000; can find a DES key in 4.5 days (90 billion keys/second)
- DES-Challenges by RSA Security to show limits of DES, given a ciphertext, USD 10'000 for the winner

→ DES is definitely no longer secure! → Triple DES

Triple DES (3DES)



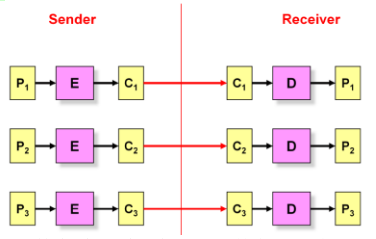
- This variant is also called DES-EDE3 (there is also DES-EEE3)
- True cryptographic strength of 3DES key is 2x56 bits = 112 bits
- Sometimes, K3 is chosen equal to K1 → two keys

Advanced Encryption Standard (AES)

- DES is no longer considered to be secure
  - Key length is too short
  - Triple DES is secure, but relatively slow
- The need for a new Advanced Encryption Standard
  - The National Institute of Standards and Technology (NIST) started a public contest for AES in 1977 and selected Rijndael in Oct. 2000
  - On Nov. 26 2001, AES was officially published as the U.S. Federal Information Processing Standard FIPS PUBS 197
- Requirements for AES
  - AES shall be publicly defined
  - AES shall be a secret key (symmetric) block cipher
  - AES shall be implementable efficiently in both hardware and software
  - AES shall have a block size of n = 128 bits
  - AES shall have flexible key sizes of k = 128, 192, and 256 bits
  - AES shall be freely available (no patents like IDEA etc.)
- **If possible, AES should be used today! (128-bit key is fine)**

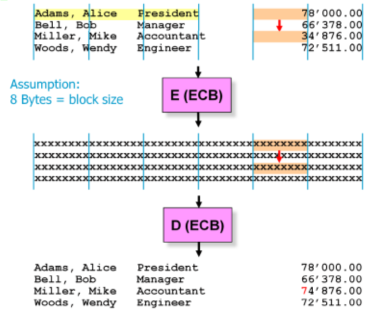
Block Cipher Modes I

Electronic Code Book Mode (ECB)



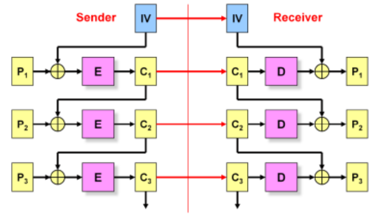
In ECB block cipher mode, a plaintext input block is mapped statically to a ciphertext output block. Decryption for the receiver is straightforward as he simply decrypts one ciphertext block after the other to receive the plaintext blocks. While ECB mode is simple, there are a few problems. One is that the same block of ciphertext always decrypts to the same block of plaintext. This is already a potential weakness because if the attacker sees two identical blocks of ciphertext encrypted with the same key (for instance as part of a larger message), he directly knows the corresponding plaintext blocks must also be identical. If the attacker manages to collect many corresponding plaintext-ciphertext blocks that were generated with the same key and if he has sufficient memory resources available, he can start building a lookup table or electronic code book (this is where the mode has its name from). For every ciphertext block he receives later, he can check the table if the corresponding plaintext is available. If the table contains reasonably many plaintext-ciphertext blocks, this may allow the attacker to at least decrypt parts (some blocks) of an encrypted message. Of course, the table gets useless as soon as sender and receiver change the key

Electronic Code Book Mode - Attack


Adams, Alice President 78'000.00
Bell, Bob Manager 66'378.00
Miller, Mike Accountant 34'876.00
Woods, Wendy Engineer 72'511.00
Assumption:
8 Bytes = block size
E (ECB)
D (ECB)
Adams, Alice President 78'000.00
Bell, Bob Manager 66'378.00
Miller, Mike Accountant 74'876.00
Woods, Wendy Engineer 72'511.00

Block Cipher Modes II

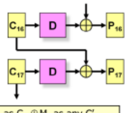
Cipher Block Chaining Mode (CBC)



In order to inhibit block replay attacks and codebook compilation, modern block ciphers are usually run in cipher block chaining mode. Each plaintext block is XOR-ed with the previous ciphertext block before encryption, so that identical plaintext blocks occurring in the same message show up as different ciphertext blocks. With CBC-mode, a ciphertext block Ci is not only dependent on the plaintext block Pi and the key (as in ECB mode), but depends on the IV, all previous and the current plaintext block P1..Pi and the key. Even if two identical messages are encrypted with the same key, they still result in different ciphertexts if the initialisation vectors (see below) differ. At the receiving side each block coming out of the decryption algorithm must first be XOR-ed with the previously received ciphertext block in order to recover the plaintext. A single bit error occurring over the transmission channel will result in the loss of one whole plaintext block plus a single bit error in the immediately following plaintext block. Error propagation is therefore restricted to two plaintext blocks.
Any CBC-encrypted message must be initialised with an initialisation vector (IV) that is openly transmitted over the insecure channel at the beginning of the session. In order to avoid replay attacks an IV value should be used only (for a given secret key) once and never be used again. This can be achieved either by assigning a monotonically increasing counter or a random value to the IV

Cipher Block Chaining Mode - Attack

- Attack on the salary list revisited, using a cipher in CBC mode
  - Mike again wants to change the 3 in his salary to 7 (block 17, P_{17})
  - Simply replaying ciphertext blocks won't work this time
- $P_{17} = D_K(C_{17}) \oplus C_{16}$
  - To modify P_{17}, Mike can modify C_{17} or C_{16}
  - Modifying C_{17} is pointless as Mike cannot predict its decrypted value
  - Modifying C_{16} is the better approach



- We replace C_{16} with C'_{16}

$$\begin{aligned} P'_{17} &= D_K(C_{17}) \oplus C'_{16} \\ &= D_K(C_{17}) \oplus (C_{16} \oplus M) \\ &= (D_K(C_{17}) \oplus C_{16}) \oplus M \\ &= P_{17} \oplus M \end{aligned}$$

We write C'_{16} as $C_{16} \oplus M$, as any C'_{16} can be generated from C_{16} by XOR-ing it with an appropriate bit string M

By replacing C_{16} with $C_{16} \oplus M$, P_{17} becomes $P_{17} \oplus M$ after decryption

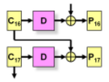
While CBC mode avoids the problem of replaying ciphertext blocks, it is still possible to modify the message in transit in a reasonable way. We consider again Mike, who wants to increase the leading number of his salary from 3 to 7. The 3 of Mike's salary is in plaintext block 17 (P17), this 3 should be changed to 7 (P'17). To see that this is possible, one must realise that
P17 = DK(C17) XOR C16 _ P17 XOR M = DK(C17) XOR C16 XOR M = DK(C17) XOR (C16 XOR M), which means that by XOR-ing C16 with any bit-string M (we write C'16 for C16 XOR M) that has the length of the block size, P17 will also be XOR-ed with M when decrypting the ciphertext. Consequently, if the original content of the plaintext block P17 is known, it can be changed at will by the attacker. Assuming P17 contains 8 ASCII-encoded characters with 7 spaces and a 3 (00110011) and the attacker wants to change this to " 7", which contains 7 spaces and a 7 (00110111), he must choose M = 00...00|00000100, which means 56 zero-bits (to not change the spaces) and the byte 00000100. As a side effect, P16 will be garbled too: P'16 = DK(C'16) XOR C15 = DK(C16 XOR M) XOR C15. There's nothing the attacker can do to avoid this, expect hoping that the recipient won't get suspicious by a "few random looking bytes" in the message. To prevent such attacks, one must make use of mechanisms to protect the integrity of a message, which will be discussed later in this course.

- All Mike has to do is pick M such that by XOR-ing it with P_{17} (which is "   3"), the result becomes "   7"
- We assume the characters are ASCII-encoded:

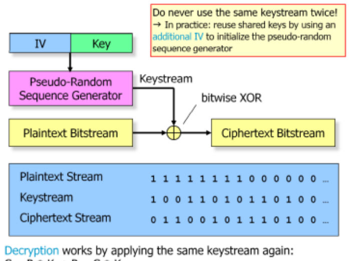
Original block P_{17}:
```
00100000 00100000 00100000 00100000 00100000 00100000 00100000 00110011
```
Desired block P_{17}:
```
00100000 00100000 00100000 00100000 00100000 00100000 00100000 00110111
```
Only one bit has to be changed, so M is:
```
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000100
```
→ So Mike only has to change the 3rd last bit in C_{16} to achieve his goal

- Side effect: P_{16} will also be affected (garbled), possible output after decryption:

```
Adams, Alice    Presdident                   78'000.00
Bell, Bob       Manager                      66'378.00
Miller, Mike    Accountan&=Mç #1^            74'876.00
Woods, Wendy    Engineer                     72'511.00
```

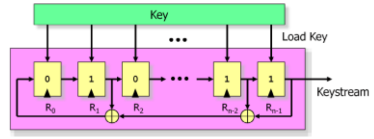


Stream Ciphers


Do never use the same keystream twice!
→ In practice: reuse shared keys by using an additional IV to initialize the pseudo-random sequence generator
IV
Key
Pseudo-Random Sequence Generator
Keystream
bitwise XOR
Plaintext Bitstream
Ciphertext Bitstream
Plaintext Stream 1 1 1 1 1 1 1 1 0 0 0 0 0 0 ...
Keystream 1 0 0 1 1 0 1 0 1 1 0 1 0 0 ...
Ciphertext Stream 0 1 1 0 0 1 0 1 1 1 0 1 0 0 ...

- Decryption works by applying the same keystream again: $C = P \oplus K \rightarrow P = C \oplus K$

Pseudo-Random Sequences with Linear Feedback Shift Registers (LFSRs)



- Maximum possible sequence length is $2^n - 1$ with n registers
- LFSRs are often used as building blocks for stream ciphers
- GSM A5/1 is a cipher with 3 LFSRs of lengths 19, 22 and 23

Stream Ciphers - RC 4 Internal state of 256 registers (8 bits wide)

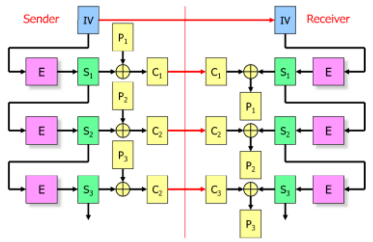
```
// java class definition

public class RC4 {
  private final static int stateSize = 256;
  private int state[];
  private int index1;
  private int index2;

  // constructor

  public RC4(int key[]) {
    state = new int[stateSize];

    // initialises the internal state, sets all possible
    // byte-values from 0..255 once in the 256 registers,
    // shuffled according to the key
    this.loadKey(key);
  }

  ...
}
```

Simple state update by swapping registers

```
// returns the next byte of the keystream

public int getByte() {
  int swap, byteIndex;

  // compute next index
  index1 = (index1 + 1) % stateSize;
  index2 = (index2 + state[index1]) % stateSize;

  // swap contents of state[index1] and state[index2]
  swap = state[index1];
  state[index1] = state[index2];
  state[index2] = swap;

  // return next byte
  byteIndex = (state[index1] + state[index2]) % stateSize;
  return state[byteIndex];
}
```

Stream Cipher with Block Cipher: Block Cipher Modes III

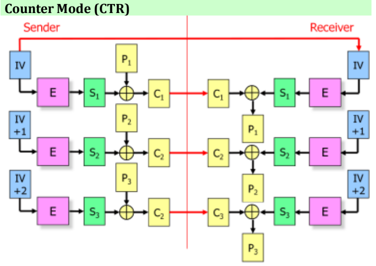
Output Feedback Mode (OFB)



A block cipher in output feedback mode works as a key stream generator producing a pseudorandom key sequence a block at a time. By XOR-ing the key stream with the plaintext, the block cipher actually works as a stream cipher. Note that the individual block cipher boxes (denoted with E) still receive the shared secret key as an input, although this is not illustrated above. So the generated key sequence is based on and is unambiguously determined by the shared secret key. One advantage of this mode is that you can use a well-established block cipher that has demonstrated to produce ciphertext that is statistically independent of the plaintext to produce a (most probably) highly random keystream without having to rely on another keystream generator

Stream Cipher with Block Cipher: Block Cipher Modes IV

Counter Mode (CTR)



A block cipher in counter mode works as a key stream generator producing a pseudo-random key sequence a block at a time. By XOR-ing the key stream with the plaintext the block cipher actually works as a stream cipher. Compared to the similar OFB mode, the CTR mode has the advantage that decoding can be started at any point in the data stream without precomputing the whole key stream up to this point. CTR also allows to parallelize the encryption/decryption of a large plaintext/ciphertext. Counter mode is often used over unreliable communication channels (WLAN, GSM, UMTS) or protocols (UDP-based RTP)

Summary

- Block ciphers split the plaintext into equally-sized blocks and encrypt them block-by-block
- Most popular block ciphers: DES (no longer secure), 3DES, AES
- Two important block cipher modes
    - Electronic Codebook Mode (ECB)
        - Every plaintext block is statically mapped to a ciphertext block, has security problems
    - Cipher Block Chaining Mode (CBC)
        - Avoids most problems of ECB by chaining the individual encryptions together
- Stream ciphers generate a keystream, which is XOR-ed bit-by-bit with the plaintext bitstream
- Popular stream ciphers: RC4
- Stream ciphers can be implemented with block ciphers
    - Output Feedback Mode (OFB), Counter Mode (CTR)