University of Karlsruhe
Advanced Systems Seminar
Summer Term 2003

Symmetric Cryptography

Daniel Grund

2003/07/01

Abstract

Making systems secure is becoming more and more important nowadays. In most cases, isolation of sensitive data is not possible and systems therefore rely on cryptography for secure storage or transmission. This document will focus on classic cryptography, also called symmetric or secret key cryptography. It will present general concepts and some examples, allowing the reader to get a more detailed view of the internal work of ciphers. It will not discuss cryptanalysis.

Key Words: Stream Cipher, Block Cipher, DES, AES, One Time Pad, Electronic Code Book, Cipher Block Chaining, Output Feedback, Cipher Feedback

Contents

Chapter 1

Introduction

In symmetric cryptography, there are two main concepts: stream and block ciphers.

Both of them use a secret key for encryption and the same key or one that is easy to deduct for decryption. Assuming that all other cipher related information, namely the algorithm itself, its design principles and perhaps existing initilization values, are public and available to everyone, the whole security of a cipher rests entirely upon this secret key. This is called Kerckhoff's principle. This implies that the key space has to be large enough, making it impossible for attackers to do a brute force attack, i.e. searching iteratively for the correct key, in a reasonable time. Because this will always depend on available technology, the security of a cipher has to be reverified over time. Today's ciphers offer key sizes of at least 128 bits.

For performance and scalability reasons, most symmetric ciphers use only simple operations like XOR, other additions modulo a power of 2, modulo multilications, rotations, permutations...

1.1 Stream Ciphers

Stream ciphers are characterized by their ability to encrypt small amounts of data, making them suitable for applications where buffering is not possible, is limited or just not wanted. At the same time this ability leads directly to a vulnerability which has to be prevented: statistical or redundancy attacks are likely to be successful if the cipher encrypts only small blocks of data with the same function. Because of this, stream ciphers change their encryption function over time. Implemented in hardware, they show better performance than block ciphers and have a less complex circuitry. Offering limited or even no error propagation can be advantageous having a higher probability

of transmission errors. Although this class of algorithms has been thoroughly researched and analyzed, only few stream ciphers have been fully published and specified.

Members of this class are SEAL, One Time Pad, SOBER, Leviathan, and several others.

1.2 Block Ciphers

In contrast to stream ciphers, block ciphers encrypt groups of characters at once with a fixed encryption function. They are the most important ciphers in many crypto systems because of their versatility, being able to serve as part of pseudo random number generators, hash functions, message authentication codes and keystream generators. There are many block ciphers, but only few are suitable for many applications due to constraints of the target implementation platform.

Well-known examples are DES, AES, IDEA, SAFER, RC5, FEAL and many more.

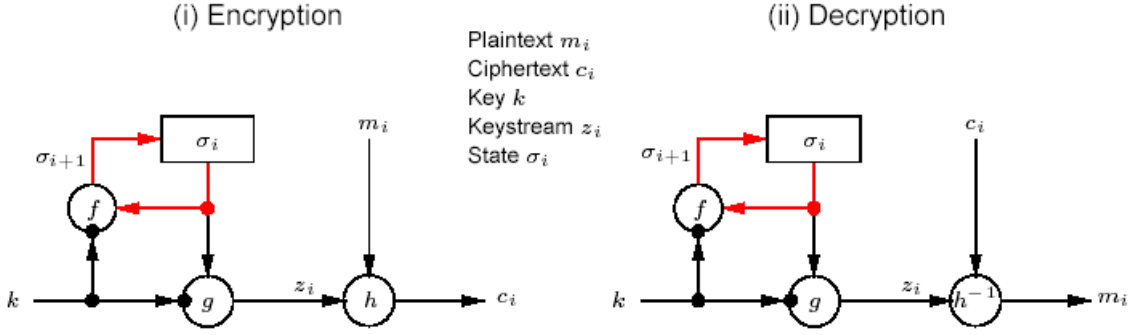
Chapter 2

Operation modes

Every cipher can operate in different modes. First presenting two modes for stream ciphers and four for the block ciphers, this chapter will end with a discussion of their pros and cons considering possible applications. In the following, sender and receiver are equivalent to encrypting and decrypting party, covering both, encryption over space and time. Nevertheless, stream ciphers tend to be used very often in transmission applications and hardly ever for secure storage.

2.1 Synchronous

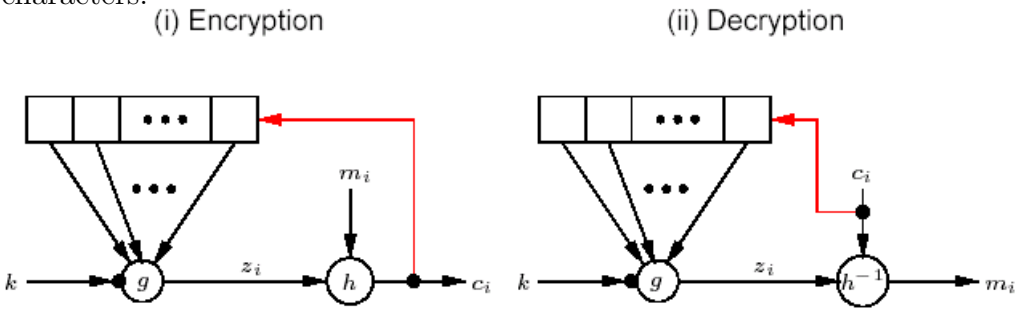
In the synchronous mode of a stream cipher, the plaintext is encrypted with the help of a key stream, which itself is generated with a stateful algorithm using the secret key as input and is independent of the plain- and ciphertext.



The decrypting party has to start with the same setup to produce the same keystream as input, to get out the original plaintext of the inverse h function. Both sides have to be in sync to match correct cipher and keystream characters. In the case $h(z, m) = z \bigoplus m = z + m \,(mod\, 2)$, one speaks of an additive stream cipher.

2.2 Asynchronous

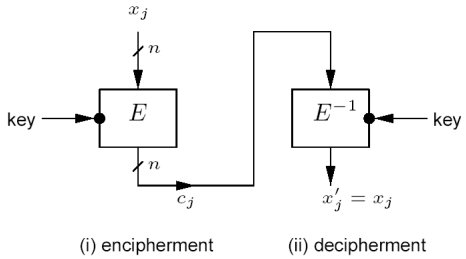
The main difference between the asynchronous mode and the synchronous mode is the dependence of the internal state on the last n output ciphertext characters.



So one could say that the synchronization information itself is transported within the ciphertext, which is why this mode is called 'self-synchronizing'.

2.3 Electronic Code Book

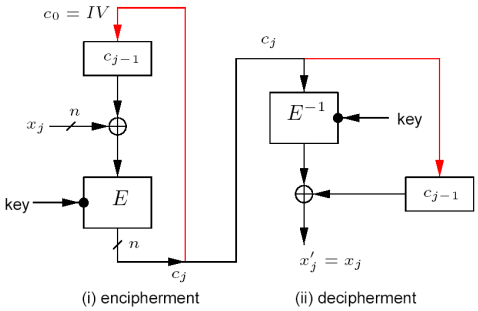
ECB is the simplest mode a block cipher can operate in.



If a block cipher does not produce longer ciphertext out of a plaintext (except for padding) and should be invertable, it has to implement a permutation depending on the secret key used.

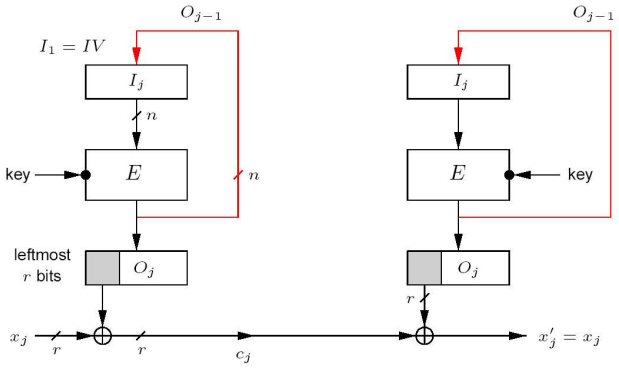
2.4 Cipher Block Chaining

If CBC is used, the plaintext blocks first get XORed with the previous output ciphertext block before they will be used as an input to the cipher. So all the blocks depend on prior cipher blocks chaining themselves together.



2.5 Output Feedback

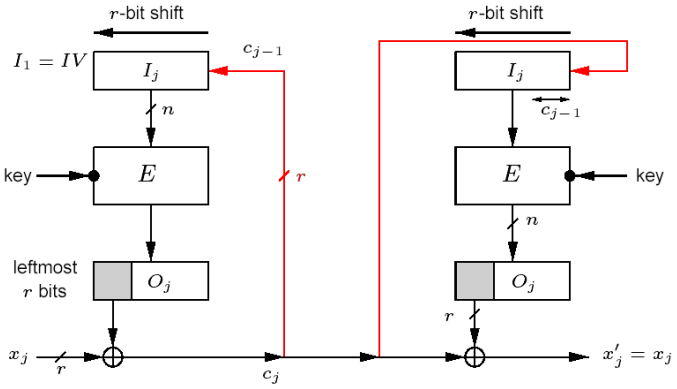
This mode is totally different. Every block cipher can be used as a stream cipher if using the OFB or CFB mode. Here the cipher works as a keystream generator.



The output cipher blocks are used to encrypt the plaintext stream. To encrypt r bits of plaintext, the left-most r bits of the cipher block are XORed with them and the block is fed back in the process, being the next input block for the cipher.

2.6 Cipher Feedback

The CFB mode can be seen as the feedback analogon to CBC, with the keystream generator dependent on the output ciphertext. Here the ciphertext is shifted into the input block of the cipher, leading to similar properties as with CBC.



2.7 Comparsion

This section will focus on the main properties of these modes and how they affect security and possible attacks on the ciphers using them.

First, comparing the two stream cipher modes, it should be clear that in synchronous mode, both the sender and receiver have to be synchronized, i.e. establishing the same internal state with the same secret key. If synchronization is lost, due to inserted or deleted characters, further decryption is impossible until extra mechanisms are used to resynchronize. The synchronous mode has no error propagation if characters are altered during transmission, i.e. this does not affect the decryption of other characters. Because of these properties, the receiver is likely to notice if an attacker tries to insert or delete ciphertext, because synchronization is lost. However, an attacker might be able to change the ciphertext if he knows the exact effect of this change. Therefore, offering data integrity requires additional techniques.

The asynchronous mode shows fewer problems. Because the decryption only depends on a fixed number of preceding ciphertext characters, synchronization is re-established automatically if characters are inserted or deleted; only a fixed number of characters is lost. Self-synchronizing ciphers have a limited error propagation: Insertion, deletion or change may only cause loss of a certain number of characters, after which decryption continues normally. This makes changes to the ciphertext detectable, but at the same time it makes it more difficult to notice insertion or deletion due to the resynchronization. The self-synchronizing mode has another important feature: diffusion of plaintext statistics. Because of the dependency on the ciphertext, redundancy or statistical attacks will fail more often compared to the synchronous mode.

The ECB mode encrypts larger blocks, making it more and more difficult to attack successfully with redundancy or statistical methods. But ECB is

not recommended for the following reasons:

- Under the same key, identical plaintext blocks imply identical ciphertext blocks and vice versa.
- The blocks are independent of others. Therefore malicious reordering, replay and deletion of ciphertext blocks is possible.

So ECB should be used only for messages not larger than one block.

CBC offers more security, because changing the IV or the first plaintext block will result in a completely different mapping of plaintext blocks on ciphertext blocks. The block n depends on all previous blocks, making it impossible to reorder ciphertext blocks without affecting decryption. Having a closer look at error propagation reveals that a single bit error in ciphertext block c_i will lead to incorrect decryption results for x'_i and x'_{i+1}. Block x'_i (with a typical cipher) will have an error rate about 50%, but x'_{i+1} will have bit errors at exactly the same positions as c_i did. Thus, an attacker may be able to make predictable bit changes to a block, although noticeable in x'_i. CBC has resynchronization capabilities: Having an error in c_i, c_{i+2} can be decrypted correctly, assuming c_{i+1} has no error.

In OFB, changing the IV will result in different ciphertext blocks. There is no error propagation, bit errors only affect the current character. Resynchronization after bit errors is possible. If reusing a key, the IV must be changed! If the same key and IV are used more than once, it is possible that an attacker gains $PlainText1 \bigoplus KeyStream$ and $PlainText2 \bigoplus KeyStream$ which reduces the cryptanalysis effort to break $PlainText1 \bigoplus PlainText2$, which is easier. A negative aspect is the decreased throughput, because n bits have to be computed first to encrypt r bits of plaintext. In OFB, these can be precomputed with key and IV known in advance.

This is different in CFB. Here the computation has to be done online. In this 'chaining' mode, reordering is not possible as in CBC. Resynchronization is possible after $\lceil n/r \rceil$ correct blocks; (this results in having a correct value in the shift register). Errors are propagated to the next $\lceil n/r \rceil$ blocks, and reduced throughput applies as in OFB.

In all block modes, recovering from loss or insertion of bits which alter the block boundaries, is not possible. Although the chained modes (asynchronous, CBC, CFB) can be considered more secure, the choice of mode depends on the application. With a high transmission error rate, it could be better to use non-chaining modes to reduce the costs of retransmission. Another possible requirement which makes CBC less handy to use is the random access to the encrypted data.

Chapter 3

Examples

Now that some general concepts have been explained, the rest of this paper will focus on examples, beginning with

3.1 DES

3.1.1 History

In 1972 the American NBS (National Bureau of Standards), now called the NIST, started a program called "Secure Data Storage and Transmission". They searched for a uniform algorithm which could be implemented efficiently on different target systems. The first incoming proposals in 1973 did not fit the requirements, and so it took another year before IBM commited the LUCIFER algorithm. LUCIFER was a 128 bit block cipher considered to be strong enough. However, (imho) it seemed that the NSA (National Security Agency) considered it too strong, and therefore a weakened version of LUCIFER was published by the NSA in 1975. The main modification was to reduce the key size to 56 bits. Although the design decisions were kept secret and reduced key size was criticized, this algorithm was certified as the 'Data Encryptoin Standard' in 1977. In 1983 and 1988 it was recertified, although in 1987 the first deficiencies occured. Because there were no other alternatives, the algorithm was recertified, although it was obvious that a key space of 2^{56} was becoming too small. This was proven by a cluster of workstations in 1994 breaking a key in 'only' 50 days, topped by a dedicated hardware attack ($250 000 US) revealing the correct key after 56 hours.

3.1.2 The Algorithm

The algorithm itself is a 64 bit block cipher using a 56 bit effective key. Internally it uses the 'Feistel' structure and 16 rounds of computation with 48 bit roundkeys. This is reflected in pseudocode as follows:

```
DES(Block, CipherKey) {
  KeyExpansion(CipherKey, RoundKey);
  IPerm(Block);
  for i=1..15 do Round(Block, RoundKey[i]);
  FinalRound(Block, RoundKey[16]);
  InvIPerm(Block);
}
```

All operations are presented in more detail below.

3.1.3 Key Expansion

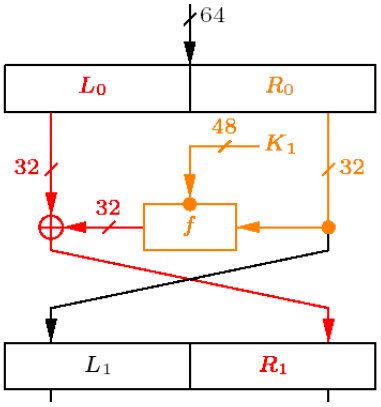
From the original 64 bit key, the 8 parity bits have to be removed. This is done by a permuted choice (PC1): bits are chosen and permuted. The resulting 56 bits are split in two equally sized halves C_0 and D_0. To get the 16 roundkeys, these two halves are independently rotated left ($C_i = C_{i-1}\ rol\ \{1|2\} D_i = D_{i-1}\ rol\ \{1|2\}$) and another permuted choice (PC2) is applied to the concatenated halves: $Roundkey_i = PC2(C_i D_i)$.

3.1.4 Initial Permutation

This operation simply permutes single bits of the block. One might ask why this operation is necessary. It is not, because all operations after the last and prior to the first operation in a cipher, which use the key, can simply be peeled off without knowledge of the key. Therefore IPerm and InvIPerm do not contribute to the security of the cipher. One might think that the whole purpose of this operations is to slow down software implementations, decreasing throughput and increasing the time of brute force attacks. In hardware, one can have permutations in O(0) by just crossing the wires.

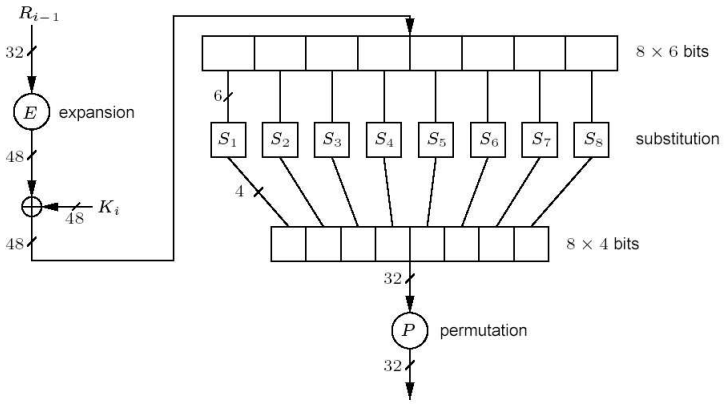
3.1.5 A DES Round

The below image shows the 'Feistel' structure.



This is one round of DES. The 64 bit input block is split into two halves L_i and R_i. The new left side is just the old right side $L_i = R_{i-1}$, and $R_i = L_{i-1} \bigoplus f(R_{i-1}, RoundKey_i)$. This round structure is valid for the first 15 rounds. The last round does not swap the output halves. The single point where keys are used is the round function f upon which almost all the strength of the cipher is founded.

3.1.6 The Round Function



The input of 32 bits is first expanded to 48 bits; each bit is chosen once, some bits twice. Next the roundkey is applied by a simple XOR. The resulting 48 bits serve as input for 8 different substitution boxes. These boxes are nonlinear elements, which can be implemented as a table lookup. Two bits specify the row, the remaing ones the column of a 4x16 matrix, where 4-bit entries represent the output of this step. The last operation is a permutation on the output of the s-boxes. This permutation and the s-boxes are designed in such a way that after a few rounds, every bit of the output block depends on every bit of the key and all the other bits of that block. This is called the 'avalanche effect'. After the final round and the inverse initial permutation, encryption is finished.

3.1.7 Decryption

Due to the Feistel structre, the same algorithm can be used for decryption by just reversing the order of the roundkeys used. The initial permutation of the decryption nullifies the inverse initial permutation of the encryption, resulting in (R_{16}, L_{16}) being the input to the first decryption round. The output of this first round is $(L_{16}, R_{16} \bigoplus f(L_{16}, Roundkey_{16})$. Knowing from encryption that $L_{16} = R_{15}$ and $R_{16} = L_{15} \bigoplus f(R_{15}, Roundkey_{16})$ this term can be reduced to (R_{15}, L_{15}). 14 rounds lead to (R_1, L_1) and after the last non-swapping round, the input results in (L_0, R_0), the original plaintext. This scheme is completely independent of the function f used.

3.1.8 Security of DES

The keyspace of only 56 bits is far too small for the technologies available today, allowing brute force attacks. Due to the good design of the s-boxes and the permutation, DES is strong against differential attacks, but linear attacks on the cipher remain possible. Although they are very unlikely to succeed because they need a large amount of known plaintext ciphertext pairs under the same key, it would be better if these did not exist. The best attack of this class needs 2^{43} pairs, which is better than brute force. Another obscurity in DES is the existence of weak and semi-weak keys. For a weak key K, the following holds: DES(DES(x, K), K) = x. For semi-weak keys K1 and K2 DES(DES(x, K1), K2) = x.

3.1.9 Improvements on DES

Because the DES operation does not result in a group, multiple encryption is possible. However, double encryption does not result in an effective keyspace of 2^{112}, but only 2^{57}. This is caused by the 'meet in the middle' attack. This is a known plaintext attack, i.e. the attacker knows at least one pair of plain and ciphertext and wants to get the key. The attacker encrypts the plaintext with all possible keys, 2^{56} operations. Then he decrypts the ciphertext with all possible keys, again 2^{56}. By now searching for equal blocks in the 2 computed sets and identifying false positives, he knows the key. To improve security, at least triple DES has to be chosen, even though a kind of meet in the middle attacks reduce the strength to 2^{108}.

3.2 AES

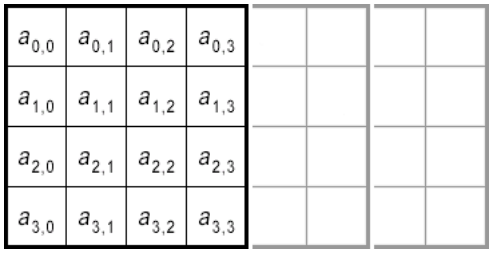
3.2.1 History

After the successful attacks in the 90s, it was time for a new request for proposalsto the NBS. The main requirements for the new standard were to be more secure, more efficient and versatile than 3DES. In 1998 15 algorithms were submitted and after a verification period the algorithm closest to the requirements, "Rijndael", was chosen and released as standard in 2000.

3.2.2 The Algorithm

Because AES just adds some restrictions on the usage of Rijndael, the following sections will describe Rijndael itself and not AES.

Rijndael is a block cipher with possible key and block sizes of 128, 192, 256 bits each. Depending on this combination 10, 12 or 14 rounds are specified. Internally the cipher operates on the data in form of a matrix, called the state.



The pseudocode follows:

```
Rijndael(State, CipherKey) {
  KeyExpansion(CipherKey, RoundKey);
  AddRoundKey(State, RoundKey[0]);
  for i=1..Nr-1 do Round(State, RoundKey[i]);
  FinalRound(State, RoundKey[Nr]);
}
```

Again, the following sections describe the single operations.

3.2.3 Key Schedule

For Rijndael 1+Nr (number of rounds specified) roundkeys are used, each of them equal in size to the block size. The key schedule operates on 4-byte entities with the cipher key in the first positions of this word array. Iteratively it computes the next values until the required length of the array

is reached, typecasting it to an array of roundkeys. It uses rotations, some constants and ByteSub which is also used in the cipher itself.

3.2.4 The Round Transformation

The Round Transformation looks like this:

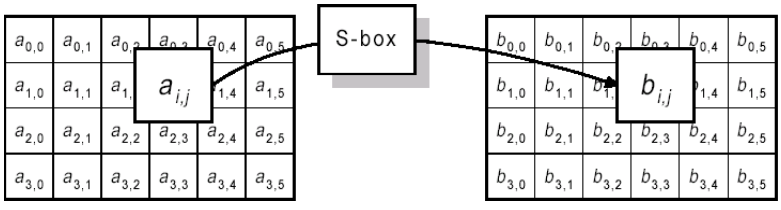
```
Round(State, RoundKey) {
  ByteSub(State);
  ShiftRow(State);
  MixColumn(State);
  AddRoundKey(State, RoundKey);
}
```

As in DES the final round is modified for decryption purposes.

```
FinalRound(State, RoundKey) {
  ByteSub(State);
  ShiftRow(State);
  AddRoundKey(State, RoundKey);
}
```

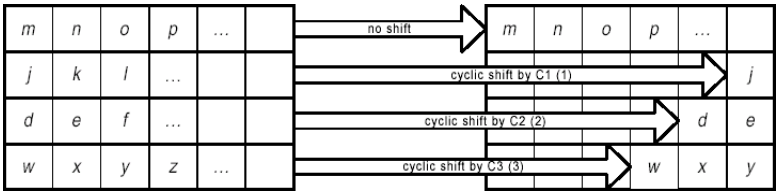
3.2.5 ByteSub

ByteSub is a nonlinear byte substitution operating on all bytes of the state independently. It is compounded of two operations. First the 8 bits are interpreted as a member of $GF(2^8)$ taking the multiplicative inverse of it. 0 is mapped on itself. The result is then interpreted as a vector of GF(2), and applying an affine transformation results in the output.



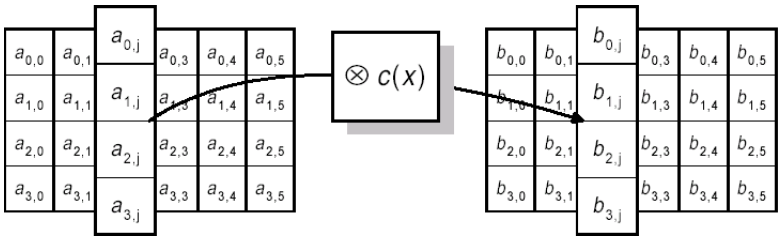
3.2.6 ShiftRow

This operation simply rotates the rows of the state by a given offset depending on the block size chosen, producing a pattern similar to a brick wall.



3.2.7 MixColumn

MixColumn takes a whole column as input, interpreted as polynomials over $GF(2^8)$. The output is computed by a multiplication modulo $(x^4 + 1)$ with a fixed c(x). This operation can be represented as a matrix multiplication.



3.2.8 AddRoundKey

This is very simple: Just XOR the equal sized state and roundkey.

3.2.9 Decryption

The modification of the last round and the following facts make it possible to represent the decryption in the same pattern as encryption:

- ShiftRow and ByteSub commute
- MixColumn and its inverse are linear respective $\bigoplus$:
  $InvMixColumn(State \bigoplus Key) = IMC(State) \bigoplus IMC(Key)$.

Straightforward	**Transformed**
AddRoundKey(RoundKey[2]);	AddRoundKey(RoundKey[2]);
InvShiftRow	
InvByteSub	InvByteSub
	InvShiftRow
AddRoundKey(RoundKey[1]);	InvMixColumn
InvMixColumn	AddRoundKey(IRoundKey[1]);
InvShiftRow	
InvByteSub	InvByteSub
	InvShiftRow
AddRoundKey(RoundKey[0]);	AddRoundKey(RoundKey[0]);

The only blemish in this scheme is IRoundKey[1], but this can be resolved by altering the key schedule.

3.2.10 Security of Rijndael

Rijndael was designed to resist all known attacks like linear or differential attacks, and it does, so AES is certainly more secure than DES and 3DES. However, Rijndael is a very algebraic cipher and it did not take long until first algebraic attacks on AES were published. They use the fact that AES can be written as an overdefined system of multivariate quadratic equations. Then solving such a system in less time than searching the whole keyspace would be an alternative to break AES. But still nobody was able to prove or disprove this attack scheme (equations, XSL algorithm, T' method) to be successful. Thus AES may share the fate of all ciphers which are 'only' coputationally secure, and time will bring reasonable attacks better than brute force.

3.3 The One Time Pad

Although the One Time Pad is not a typical steam cipher it is the most prominent one, because it enables 'absolute security'.

3.3.1 The Algorithm

The algorithm is very simple. The output is computed by $(PlainTest \bigoplus Key)$. The key has to be as long as the message itself and has to be sufficiently random. One could 'prove' the absolute security of this cipher as follows: Assuming the attacker has the ciphertext and infinite time, he could do a

bruteforce attack on that huge key. He will thus sure get the correct plaintext, but he will also decrypt the ciphertext to all other possible texts of this length, reasonable ones and absurd ones.

3.3.2 Security

The nice feature of absolute security has its price. First, the key is as long as the message and has to reach the receiver or in other words, there are very high key management costs. More important is the randomness of the key. The security of this cipher depends on how random the key is, so taking a pseudo random number generator with initial seed, OFB or CFB will fail, because they belong to the class of deterministic machines, making it impossible to produce really random data. Furtheron the key data must not be used twice (see note on OFB in the comparsion), increasing the overhead of key management quadratically if n parties want to communicate. Further notes on key data supply and n-way communication are given in [3].

Bibliography

[1] Eckert, Claudia: IT-Sicherheit

[2] Menezes, A.: Handbook of Applied Cryptography freely available here: www.cacr.math.uwaterloo.ca/hac/.

[3] Notes on One Time Pad: www.contestcen.com/crypt005.htm

[4] Rijndael home page: www.esat.kuleuven.ac.be/ rijmen/rijndael/

[5] NIST AES page: csrc.nist.gov/CryptoToolkit/aes/