1.2 Methods for Generating a Key Stream

The main naive methods for generating the key stream are:

- periodic bit sequence,
- running-text,
- "true" random sequence.

A better method uses a

- pseudorandom sequence

and leads to really useful procedures. The essential criterion is the quality of the random generator.

Note We sometimes use the term "random generator" for an algorithm that produces pseudorandom sequences (of bits or numbers). The more correct denomination is "pseudorandom generator".

Periodic Bit Sequence

A periodically repeated (longer or shorter) bit sequence serves as key. Technically this is a Bellaso cipher over the alphabet $\mathbb{F}_2$. The classical cryptanalytic methods for periodic polyalphabetic ciphers apply, such as period analysis or probable word.

For an example see XOR in Part I.

Known or probable plaintext easily breaks periodic XOR encryption.

MS Word and Periodic XOR

The following table (generated ad hoc by simple character counts) shows the frequencies of the most frequent bytes in MS Word files.

byte (hexadecimal)	**bits**	**frequency**
`00`	00000000	7–70%
`01`	00000001	0.8–17%
`20` (space)	00100000	0.8–12%
`65` (e)	01100101	1–10%
`FF`	11111111	1–10%

Note that these frequencies relate to the binary files, heavily depend on the type of the document, and may change with every software version. The variation is large, we often find unexpected peaks, and all bytes `00`–`FF` occur. But all this doesn't matter here since we observe *long chains of* `00` *bytes.*

For an MS Word file that is XOR encrypted with a periodically repeated key the ubiquity of zero bits suggests an efficient attack: Split the stream of ciphertext bits into blocks corresponding to the length of the period and add the blocks pairwise. If one of the plaintext blocks essentially consists of 0's, then the sum is readable plaintext. Why? Consider the situation

	…	block 1			…	block 2			…
plaintext:	…	a_1	…	a_s	…	0	…	0	…
key:	…	k_1	…	k_s	…	k_1	…	k_s	…
ciphertext:	…	c_1	…	c_s	…	c'_1	…	c'_s	…

where $c_i = a_i + k_i$ and $c'_i = 0 + k_i = k_i$ for $i = 1, \ldots, s$. Thus the key reveals itself in block 2, however the attacker doesn't recognize this yet. But tentatively paarwise adding all blocks she gets (amongst other things)

$$c_i + c'_i = a_i + k_i + k_i = a_i \quad \text{for } i = 1, \ldots, s,$$

that is, a plaintext block. If she realizes this (for example recognizing typical structures), then she recognizes the key $k_1, \ldots, k_s$.

Should it happen that the sum of two ciphertext blocks is zero then the ciphertext blocks are equal, and so are the corresponding plaintext blocks. The probability is high that both of them are zero. Thus the key could immediately show through. To summarize:

> *XOR encryption with a periodic key stream is quite easily broken for messages with a known structure.*

This is true also for a large period, say 512 bytes = 4096 bits, in spite of the hyperastronomically huge key space of 2^{4096} different possible keys.

Running-Text Encryption

A classical approach to generating an aperiodic key is taking an existing data stream, or file, or text, that has at least the length of the plaintext. In classical cryptography this method was called running-text encryption. We won't repeat the cryptanalytic techniques but summarize:

> *XOR encryption with running-text keys is fairly easily broken.*

True Random Sequence

The extreme choice for a key is a true random sequence of bits as key stream. Then the cipher is called **(binary) one-time pad (OTP)**. In particular no part of the key stream must be repeated at any time. The notation "pad" comes from the idea of a tear-off calendar—each sheet is destroyed after use. This cipher is unbreakable, or "perfectly secure". SHANNON gave a formal proof of this, see Part I, Section 10.

Without mathematical formalism the argument is as follows: The ciphertext divulges no information about the plaintext (except the length). It could result from *any* plaintext of the same length: simply take the (binary) difference of ciphertext and alleged plaintext as key. Consider the ciphertext $c = a + k$ with plaintext a and key k, all represented by bitstreams and added bit by bit as in Figure 1.2. For an arbitrary different plaintext b the formula $c = b + k'$ likewise shows a valid encryption using $k' = b + c$ as key.

This property of the OTP could be used in a scenario of forced decryption (also known as "rubber hose cryptanalysis") to produce an innocuous plaintext, as exemplified in Figure 1.4.

If the one-time pad is perfect—why don't we use it in any case and forget of all other ciphers?

- The key management is unwieldy: Key agreement becomes a severe problem since the key is as long as the plaintext and awkwardly to memorize. Thus the communication partners have to agree on the key stream prior to transmitting the message, and store it. Agreeing on a key only just in time needs a secure communication channel—but if there was one why not use it to transmit the plaintext in clear?

- The key management is inappropriate for mass application or multiparty communication because of its complexity that grows with each additional participant.

- The problem of message integrity requires an extended solution for OTP like for any XOR cipher.

There is another, practical, problem when encrypting on a computer: How to get random sequences? "True random" bits arise from physical events like radioactive decay, or thermal noise on an optical sensor. The apparently deterministic machine "computer" can also generate true random bits, for instance by special chips that produce usable noise. Moreover many events are unpredictable, such as the exact mouse movements of the user, or arriving network packets that, although not completey random, contain random ingredients that may be extracted. On Unix systems these random bits are provided by `/dev/random`.

However these random bits, no matter how "true", are not that useful for encryption by OTP. The problem is on the side of the receiver who cannot reproduce the key. Thus the key stream must be transmitted independently.

There are other, useful, cryptographic applications of "true" random bits: Generating keys for arbitrary encryption algorithms that are unpredictable for the attacker. Many cryptographic protocols rely on "nonces" that have no meaning except for being random, for example the initialization vectors of the block cipher modes of operation, or the "challenge" for strong authentication ("challenge-response protocol").

```
Plain bits and text:
01010100 01101000 01101001 01110011 00100000 01101101   This m
01100101 01110011 01110011 01100001 01100111 01100101   essage
00100000 01101001 01110011 00100000 01101000 01100001    is ha
01111010 01100001 01110010 01100100 01101111 01110101   zardou
01110011 00101110                                        s.

Key bits:
11001000 11010110 00110011 11000000 00111011 10001110
00001000 11101111 01001001 11100101 10111100 10111001
00010010 11000110 01110011 11010111 11000100 01100000
11100110 00010111 01101010 10111011 00010101 11011000
11110000 01000010

Cipher bits:
10011100 10111110 01011010 10110011 00011011 11100011
01101101 10011100 00111010 10000100 11011011 11011100
00110010 10101111 00000000 11110111 10101100 00000001
10011100 01110110 00011000 11011111 01111010 10101101
10000011 01101100

Pseudokey bits:
11001000 11010110 00110011 11000000 00111011 10001110
00001000 11101111 01001001 11100101 10111100 10111001
00010010 11000110 01110011 11010111 11000101 01101111
11110010 00011001 01111011 10101010 00010101 11011000
11110000 01000010

Pseudodecrypted bits and text:
01010100 01101000 01101001 01110011 00100000 01101101   This m
01100101 01110011 01110011 01100001 01100111 01100101   essage
00100000 01101001 01110011 00100000 01101001 01101110    is in
01101110 01101111 01100011 01110101 01101111 01110101   nocuou
01110011 00101110                                        s.
```

Figure 1.4: XOR encryption of a hazardous message, and an alleged alternative plaintext

Pseudorandom Sequence

For XOR encryption—as approximation to the OTP—algorithmically generated bit sequences are much more practicable. But the attacker should have no means to distinguish them from true random sequences. This is the essence of the concept of pseudorandomness, and generating pseudorandom sequences is of fundamental cryptologic relevance.

> *XOR encryption with a pseudorandom key stream spoils the perfect security of the one-time pad. But if the pseudorandom sequence is cryptographically strong (Chapter 4) the attacker has no chance to exploit this fact.*

To be useful for cryptographic purposes the pseudorandom key stream must depend on parameters the attacker has no access to and that represent (parts of) the cryptographic key. Such parameters might be, see Figure 1.5 that extends the basic model of a pseudorandom generator:

- the initial value of the state,
- parameters the transition algorithm depends on.

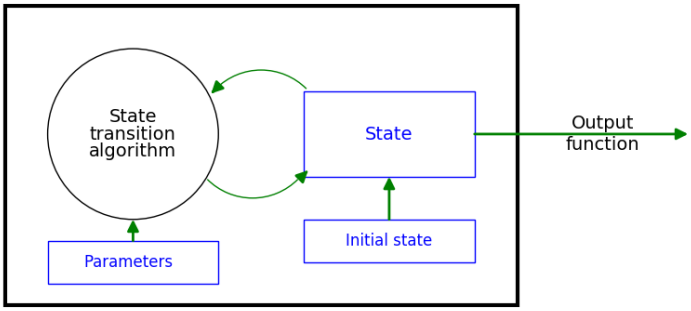

Figure 1.5: Secret parameters for a pseudorandom generator