OTP and AES: A historical transition between two systems of cryptography

Valdemar Thanner
Kantonsschule Zug
Supervised by Mr. Bernhard Keller
Linguistic supervision by Ms. Margrit Oetiker

October 13, 2016

Contents

Chapter 1

OTP: The One Time Pad

1.1 What is a "One Time Pad"?

When speaking about OTP, it is important to distinguish between its two meanings: On the one hand, it is a technique used to encrypt information. This technique requires one single key, used both to encrypt and decrypt the information. This key is also referred to as a one time pad; therefore, it is important to distinguish between the one time pad (a cryptographical technique) and a one time pad (a key which is used to encrypt and decrypt information).

The One Time Pad is largely derived from the Vernam cipher, which is named after Gilbert Vernam. The Vernam cipher utilized a perforated tape (one of the earliest types of data storage) as the secret key[1]. Each bit of data was stored in the form of a hole punched into the perforated tape.

Figure 1.1: Perforated tape, utilized to store bits as punched holes

However, this system had a vulnerability which the One-Time Pad solved: In Vernam's original method, the perforated tape was not exchanged after it had completed one cycle; instead, it was looped around continuously, often being used to encrypt multiple different messages.

This made the entire system vulnerable. The re-usage of the key meant that the resulting ciphertext suffered from a so-called known-plaintext vulnerability [2]. This means that, if a plaintext and its corresponding ciphertext are captured, the key utilized to generate the ciphertext can be derived from them. This is not an issue if the key is exchanged each time a new message is encrypted. However, if the key of any Vernam cipher machine was compromised through a known-plaintext attack, any further intercepted ciphertexts could be decrypted.

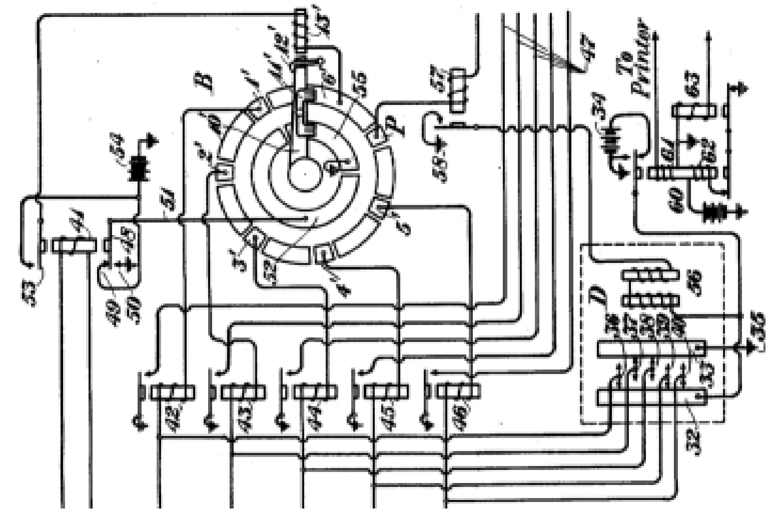


Figure 1.2: A technical diagram from Vernam's famous "Secret signaling system" patent of 1919.

1.2 Method used

In the following section, the utilized method will be clarified through usage of an example. In this example, the message *"cryptography"* will first be encrypted by its sender, sent to its intended recipient, and finally decoded by the recipient.

1.2.1 Generation of the random key

In order to encrypt the plaintext, a key must first be generated. This key will be utilized to encrypt the plaintext through the usage of modular addition, turning it into the ciphertext.

This key must fulfil some crucial criteria [3]. Foremost, the length of the key (the amount of characters contained within it) must be equivalent to or greater than the length of the plaintext; otherwise, it is not possible to perform any encryption (using the OTP). Secondly, the key must be generated randomly. This is mainly due to the fact that a randomly generated key makes frequency analysis[4], the form of cryptanalysis most commonly used to break classical ciphers, impossible.

The key consists of numbers. Usually, when the plaintext is made up of Latin letters, the numbers range between 0 and 25. The key can be converted into Latin letters through the same method applied to the plaintext outlined in the following chapter, however, this is not necessary, although the key is often transported in the form of text.

As the message being encrypted in this example has 12 characters, the key must also posses at least 12 characters. For the sake of this example, the key *"sytruifgnihm"* will be utilized.

1.2.2 Modular addition of the key and plaintext

Next, the ciphertext is created through modular addition of the key and the plaintext. This can be applied not only to a message consisting of alphabetical characters, but also to any sequence of bits. If the plaintext consists of a message made up of alphabetical characters, the plaintext and the key are added using arithmetic referred to as *"addition modulo 26"*. The correct mathematical notation for modular arithmetic is $(a + b) \, mod \, c$, where c is referred to as the *modulus*, which is the value that cannot be passed nor reached in modular addition. In order to perform modular addition, the variables a and b are first added, after which they are divided by the modulus c, up to an integer. The resulting remainder r is the final result of the operation.

Before the modular addition of the plaintext and the key can begin, each character (of the plaintext as well as of the secret key, in the case that the key was generated as a string of Latin letters instead of as a sequence of numbers) must be converted to a number, corresponding to it's position in the Latin alphabet:

c	r	y	p	t	o	g	r	a	p	h	y
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
2	17	24	15	19	14	6	17	0	15	7	24

As the key was also generated in the form of characters, it too must be converted to a sequence of numbers:

s	y	t	r	u	i	f	q	n	i	h	m
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
18	24	19	17	20	8	5	16	13	8	7	12

Afterwards, the key and the plaintext are added together utilizing modular addition. Of course, all operations below are performed in *mod 26*. Finally, the resulting numbers are converted to the character to which they correspond in the Latin alphabet. The resulting sequence of characters is referred to as the ciphertext.

2+18	17+24	24+19	15+17	19+20	14+8	6+5	17+16	0+13	15+8	7+7	24+12
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
20	15	17	6	13	22	11	7	13	23	14	10
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
u	p	r	g	n	w	l	h	n	x	o	k

The modular addition of the plaintext and the key can also, however, be performed bitwise. This simply means that, instead of adding a message consisting of (up to) 26 different letters with a key (also consisting of up to 26 different letters), we simply add the bits of the plaintext (a computer file consisting of bits), which can assume the value of either 1 or 2, with the bits of the key. The bits are added in modulo 2. This process is referred to as XOR, one of the basic bitwise operations which can be performed directly by a computers central processor.

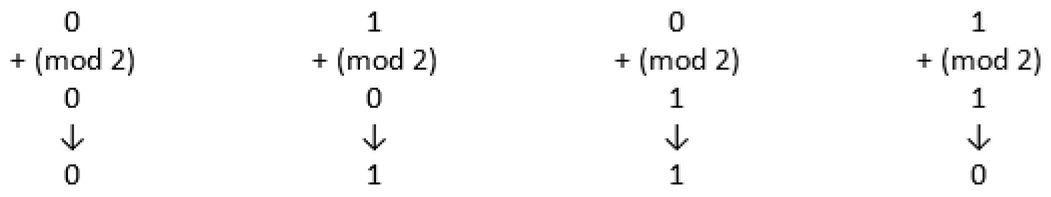
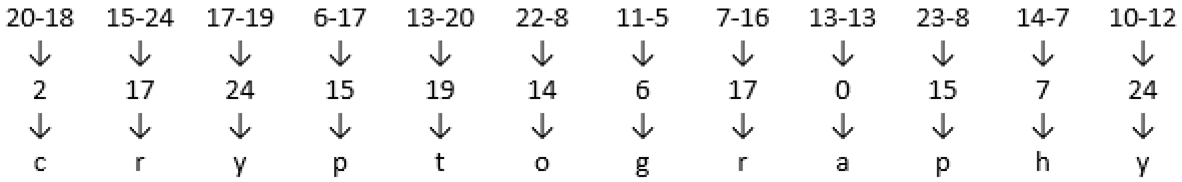


Figure 1.3: The formula used by an XOR gate.

1.2.3 Decoding of the ciphertext using the key

Assuming the message has been delivered to the intended recipient, who must hold a copy of the secret key (which, as OTP is a symmetrical cryptographical method, is identical to the one utilized to create the ciphertext), the recipient can now decode the ciphertext in order to view the plaintext.

Since the ciphertext was created through the modular addition of the plaintext and the key, the recipient can utilize modular subtraction in order to view the plaintext. In order to do this, the recipient must subtract the key from the ciphertext in modulo 26, and convert the resulting numbers to Latin characters. However, it is now also necessary for the numbers not to become negative. Fortunately, this is also made possible by modular arithmetic, as the values simply loop back around from 0 as well. Once again, all below operations are in *mod 26*.



Now, the message's journey is complete, having passed from plaintext into ciphertext, being transported to its intended recipient in the form of ciphertext, and finally being decoded and read by its recipient.

1.3 Perfect secrecy: Information-theoretical security

An important characteristic of OTP is that the ciphertext convey no information at all in regards to the plaintext; this means that it is not possible to garner any information about the key nor the plaintext if one is in possession of the ciphertext [5] [6].

From this, two other important characteristics of a cryptographical system with perfect secrecy can be derived. Firstly, perfect message indistinguishability. This means that, even if you are provided with multiple different plaintexts and one ciphertext, it is impossible to assign any one plaintext to the provided ciphertext. Secondly, if the key is the same length as the plaintext (as is the case in OTP), then there exists a key for every possible encryption from plaintext to ciphertext. This means that any plaintext can be converted to any ciphertext, making it impossible to determine the key without access to both the plaintext and the ciphertext. This is referred to as perfect key ambiguity.

Although many believe that the ciphertext does provide information about the plaintext, namely its length, this can easily be solved through padding, where characters of no importance to the plaintext are added to it before encryption [2].

1.3.1 Mathematical proof

In 1949, Shannon proved the perfect secrecy of OTP by using probability theory [7]. To be precise, he proved that, in order to achieve perfect secrecy, the key must possess at least the same length as the plaintext; one of the key characteristics of OTP. If this is the case, then an attacker cannot determine the plaintext given the ciphertext, even with access to infinite computational capacity; a "brute-force attack" is impossible.

A simplified method of presenting this proof is as follows:

If the attacker does indeed possess infinite computational capacity, he can generate every possible key which could have been used to generate the ciphertext, and he can then generate every possible plaintext (each of which will correspond to the given ciphertext and one of the generated keys).

It follows that, if the ciphertext was m bits long, there would be 2^m possible keys (or, in the case that the plaintext and ciphertext consist of Latin characters, 26^m possible keys). This is because a bit can possess the value of either 0 or 1, meaning that each individual bit of the key has two possible values. As The entire key possesses m bits, each cell must be multiplied by the following cell in order to obtain the total number of possible keys. In the case that the key consists of Latin characters, however, each cell possess 26 possible values (ranging from 0 to 25); therefore, the total number of possible keys would be 26^m.

If the possible amount of keys that the attacker can generate is equal to 2^m, and the attacker can generate one possible plaintext for each key, that means that the amount of plaintexts which the attacker can generate is also equal to 2^m (or, once again, in the case of Latin characters, equal to 26^m plaintexts).

This amount, 2^m, is equal to the amount of possible bit sequences of the length m; therefore, the attacker cannot possibly eliminate any one possible plaintext of the length m, meaning that he cannot garner any information from the ciphertext on its own. This fulfils the requirements of perfect secrecy.

1.3.2 Why can only OTP achieve perfect secrecy?

From the above proof, it can be concluded that, in order for the elimination of any possible plaintext to be impossible (rendering a brute-force attack impossible), the key must possess at least the same length as the plaintext. Also, the individual bits of the must not be dependant on one another. These conditions are only fulfilled by OTP.

1.4 Issues with OTP

1.4.1 True randomness in generating the key

One of the most important aspects of the utilized key is that it must be completely random (see 1.2.1). However, generating a random sequence of numbers or letters is no trivial task. In fact, most random number generation functions of even modern programming languages are not adequately random for use in cryptography. This is especially difficult for computers, considering that a computer is, in essence, designed to follow a set of predictable instructions as quickly as possible.

Therefore, in order to generate a sufficiently random key, it is preferable to utilize hardware random number generators, for example by utilizing a true random noise source [3]. One such example is the key generator built by mils electronic, where a set of parallel ring oscillators are sampled. Because the oscillation speed of each ring is influenced by random factors such as local variations in temperature and voltage.

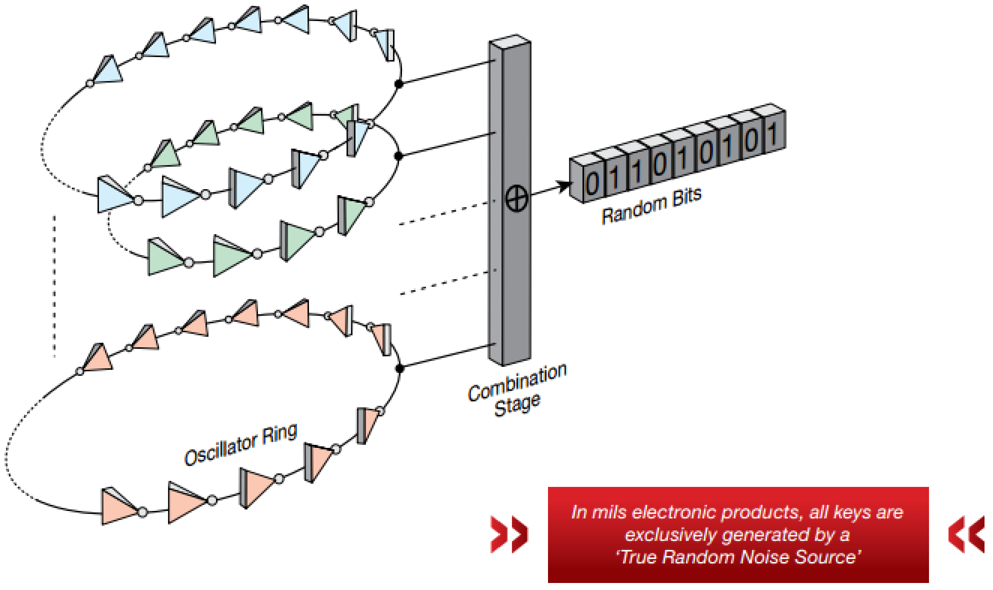

Combination Stage
Random Bits
0 1 1 0 1 0 1 0 1
Oscillator Ring
In mils electronic products, all keys are exclusively generated by a 'True Random Noise Source'

Figure 1.4: A hardware random number generator with sufficient randomness to be utilized in the generation of one-time pads.

1.4.2 Secure distribution of the key itself

The main issue in regards to OTP is that, while it posses perfect security, therefore solving the issue of safely transmitting the ciphertext, the problem instead becomes how to securely transmit the key to the intended recipient of the encrypted message.

Therefore, in order to ensure that the plaintext cannot be accessed by any party other than its intended recipient, the complete security of the transmittal of the key must also be ensured.

However, if the communicants already possess a method by which the key can be transmitted with infallible security, then it stands to reason that the plaintext itself could just as well be submitted through the same channel, considering that it has the same number of bytes (or characters) as the key. So, the problem of securely transmitting the plaintext has simply been transferred to the problem of securely transmitting the key.

One method of mitigating this issue is to first send one very long key, which can then be used for multiple messages in the future. Using this method, only one secure transmission has to be made in order to ensure the security of multiple transmissions.

1.4.3 Secure disposal of a utilized key

After a key has been fully utilized, it has to be disposed of securely, in order to prevent any information from being compromised in the future, which is possible should the key continue to exist. Therefore, it is necessary to completely eradicate the entirety of the key. If the key was stored on a digital media, such as a USB drive or HDD, the only completely secure method of disposal is considered to be complete incineration.

Chapter 2

AES: The Advanced Encryption Standard

2.1 AES viewed from a high level

Viewed from a high level, the AES algorithm consists of one XOR step, followed by 10 rounds of cryptographical measures, with each round as well as the initial XOR step using a specific subkey, generated using the main key. By passing through these steps, each block of plaintext is converted into a block of Ciphertext. In the following chapter, each individual aspect of the complete AES algorithm will be more closely examined.

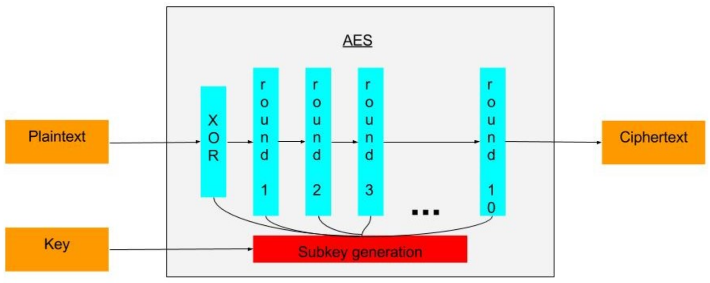


Figure 2.1: A high-level description of AES.

2.2 Method used

Each round possesses 4 steps, explained in detail below. However, before those operations can be understood, some basic terminology must first be clarified.

The "state" is the matrix containing the information which is manipulated throughout the encryption process. As such, the state begins its journey through AES as the block of plaintext which is to be encrypted, and ends it as a block of ciphertext. The state is represented in a 4x4 column-major order matrix; this means that, in a block containing 16 bytes, the standard size of an AES plaintext block as well as the standard size of an AES key $(16bytes * 8 = 128bits)$.

$$\begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}$$

It is important to note that, because the state is a column-major order matrix, the bytes are counted from "top to bottom"; that is, in the direction of the columns, and not in the direction of the rows. This shows how the bytes from the plaintext are ordered into the state. However, in order to clarify the manipulations performed throughout the 10 rounds of AES, it is more beneficial to note the starting position each byte takes in the subscript instead of noting its chronological order in the plaintext block.

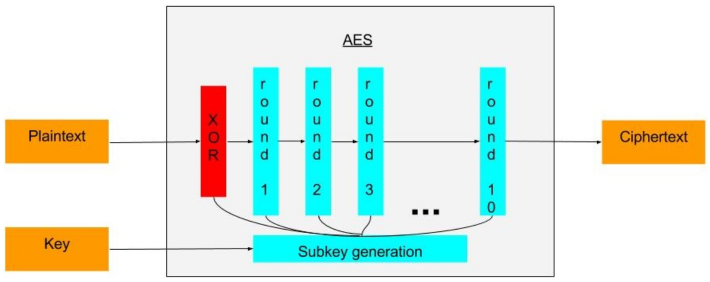
$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

The subkey possesses the same size as the state, and is made into a matrix in the exact same fashion; this is, of course, necessary for the successful modular addition of the bytes contained within the state and those contained within the subkey using the XOR operation.

2.2.1 Subkey generation (KEYEXPANSIONS)

As can be seen in the high-level explanation, AES requires 11 128-bit subkeys (also referred to as round keys); one for the initial XOR operation, and then one for each of the 10 rounds. These subkeys are derived from the main key using the "Rijndael key schedule" [8].

2.2.2 Initial XOR operation

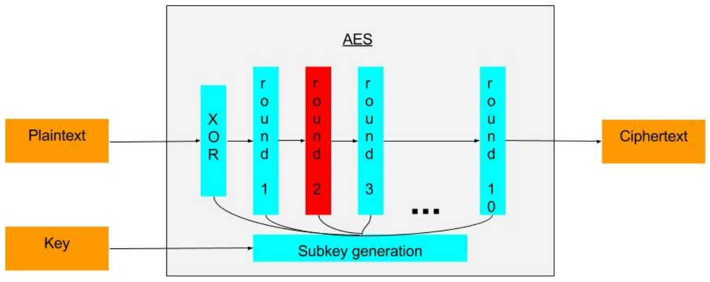


This step is identical to the XOR operation already described in the chapter regarding AES; once again, a simple XOR gate is used to add the state with the initial XOR operations own subkey (which, crucially, is not the main key itself, but is, in fact, derived from the main key during the process of subkey generation).

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} + \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix}$$

Here, each byte of the state is combined with the byte of the subkey which possesses the corresponding position using the XOR operation. For example, $a_{2,1}$ would be combined with $k_{2,1}$. The resulting byte is then placed at the matching coordinate (in this example, once again the coordinate 2,1) of the resulting matrix. The completed matrix resulting from this operation is the new state. Next, this resulting state proceeds into the first of ten so-called "rounds".

2.2.3 Rounds



Substitution using lookup table (SUBBYTES)

The first substitution is relatively simple to perform; this is because it is a bytewise operation. This means that the data being manipulated is one single byte. In this case, the manipulation is a substitution of each byte with another byte, the substitution being determined by a lookup table, in which each individual byte is assigned another byte with which it must be swapped. The reason that a lookup table is practical is that there are only $2^8 = 256$ different possible bytes.

This swapping of bytes is performed using a Rijndael S-box as the lookup table[**SBox**]. Firstly, it is very important to note that this substitution is performed in a Galois field (also referred to as a finite field)[9], specifically $GF(2^8)$. This means that the field only contains 2^8 elements; this makes sense, as there are only 2^8 possible bytes.

The subbytes step can be described as

Transposition (SHIFTROWS)

The next operation, transposition, operates upon the rows of the state by shifting each byte by a certain offset. The offset starts with 0, and is then increased by 1 for each row downwards from the top row.

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,0} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,0} \end{pmatrix}$$

Substitution using formula (MIXCOLUMNS)

Round-closing XOR operation (ADDROUNDKEY)

2.3 Fulfilment of Shannon's properties

2.3.1 Diffusion

2.3.2 Confusion

2.4 Problems with AES

2.4.1 Overconfidence and over-reliance

2.4.2 Keeping of a shared secret