Stream ciphers

Block ciphers, like Playfair and Hill ciphers, encrypt plaintext of a fixed length – digraphs for the Playfair cipher and *n*-graphs for *n*-dimensional Hill ciphers. If the length of the plaintext message is not an integral multiple of the length of a block, the plaintext message must be padded.

Stream ciphers can encrypt plaintext messages of variable length. The one-time pad can be thought of as an example – each message uses a portion of the key with length equal to the length of the plaintext message. (Then that portion of the key is never re-eused.)

The ideas that resulted in modern stream ciphers originated with another AT&T Bell Labs engineer, Gilbert Vernam (1890 – 1960). In 1917, Vernam developed a scheme to encrypt teletype transmissions. Unlike Morse code, which uses symbols of different lengths to substitute for letters of the alphabet, teletype transmission used what we would today call a 5-bit code for the letters of the alphabet and certain keyboard commands. Its use was similar to the way that the 8-bit ASCII code is used today in computing.

A = XX•••	B = X••XX	C = •XXX•	D = X••X•	E = X••••
F = X•XX•	G = •X•XX	H = ••X•X	I = •XX••	J = XX•X•
K = XXXX•	L = •X••X	M = ••XXX	N = ••XX•	O = •••XX
P = •XX•X	Q = XXX•X	R = •X•X•	S = X•X••	T = ••••X
U = XXX••	V = •XXXX	W = XX••X	X = X•XXX	Y = X•X•X
Z = X•••X				

Like today's keyboards, two functions were associated to each key – a "letter" function and a "figures" function.

letters	figures
A	-
B	?
C	:
D	who are you ?
E	3
F	%
G	&
H	local currency
I	8
J	ring bell
K	(
L	)
M	.
N	,
O	9
0	P
Q	1
R	4
S	‘
T	5
U	7
V	=
W	2
X	/
Y	6
Z	+

The 26 letters of the alphabet and the corresponding "figures" used 26 of the 32 possible 5-bit strings.

••••• was not used.

The remaining five 5-bit strings were used for commands.

••X••	space
•••X•	carriage return
•X•••	line feed
XX•XX	shift to figures
XXXXX	shift to letters

This code is called the International Telegraph Alphabet number 2 (ITA2) and is still in use in TDDs and some amateur radio applications. It was based upon a 5-bit teletype code developed by the French telegraph engineer Émile Baudot (1845 – 1903) around 1874. (The rate of symbol transmission called the baud is named after Baudot.) Around 1901, Baudot's code (ITA1) was modified by Donald Murray (1865 – 1945), a New Zealand sheepfarmer. A later modification resulted in ITA2.

The idea of using 5-character strings to represent the letters of the alphabet seems to be due to Sir Francis Bacon (1561 – 1626). Bacon used the first two letters of his name for the characters in the strings.

a	AAAAA	i/j	ABAAA	r	BAAAA
b	AAAAB	k	ABAAB	s	BAAAB
c	AAABA	l	ABABA	t	BAABA
d	AAABB	m	ABABB	u/v	BAABB
e	AABAA	n	ABBAA	w	BABAA
f	AABAB	o	ABBAB	x	BABAB
g	AABBA	p	ABBBA	y	BABBA
h	AABBB	q	ABBBB	z	BABBB

When Vernam developed his encryption scheme during the early Twentieth Century, he was not thinking about 5-bit strings of 1's and 0's; he was thinking about cross x or dot •, mark or no mark, low voltage or high voltage, etc., but today it is easiest to think of 1's or 0's.

Thinking this way, Vernam's encryption scheme can be easily described. The key was a random string of 0's and 1's. The plaintext message was converted to a string of 0's and 1's using ITA2. The two strings were XORed to generate the ciphertext. Pattern (plaintext) plus ($\oplus$) random (*key*) yields random (CIPHERTEXT).

Exercise 3.2. Encrypt the message nku using a Vernam cipher with keystring 011111110010010.

The operation XOR is equivalent to bitwise addition modulo 2.

$\oplus$	0	1
0	0	1
1	1	0

0 is the identity for XOR; i.e., XORing a bit with 0 does not change the bit. $0 \oplus 0 = 0$ and $1 \oplus 0 = 0 \oplus 1 = 0$. Notice that, for XOR, each bit is its own inverse; i.e., a bit XORed with itself yields 0. $0 \oplus 0 = 0$ and $1 \oplus 1 = 0$.

The last property makes decryption easy. Any string XORed with itself is a string of 0's.

$$\begin{array}{r} 011111110010010 \\ \oplus\ 011111110010010 \\ \hline 000000000000000 \end{array}$$

Therefore, ciphertext message can be decrypted by XORing (again) with the key.

$$\begin{aligned}
&\text{CIPHERTEXT} \oplus key \\
&= (\text{plaintext} \oplus key) \oplus key \\
&= \text{plaintext} \oplus (key \oplus key) \\
&= \text{plaintext} \oplus 00\ldots0 \\
&= \text{plaintext}
\end{aligned}$$

Exercise 3.3. Decrypt the message that was encrypted in exercise 3.2.

Exercise 3.4. Decrypt the following message that was encrypted using a Vernam cipher with keystring 1100001000111101111111111110101.

101111100010100110010011110010

Vernam's cipher was not a one-time pad. It was implemented using relays, and the keystring was punched on a (long) paper tape. Even though the tape was long, it would eventually repeat and create depth. An AT&T engineer Lyman Morehouse developed an idea that created long keystrings by combining two shorter strings of different lengths. The idea can be exhibited using two small strings. Let the first string be of length 6

101110

and the second of length 7

1111011

Create the keystring by XORing these two strings.

```
10111010111010111010111010111010111010111010101110
11110111111011111101111110111111011111101111111011
01001101000001000111000100000101100101010101010011
```

Because of the difference in lengths and because the lengths are relatively prime, combining these strings, which have periods 6 and 7, results in a string having period 42. A string of 999 bits combined with a string of 1000 bits would result in a keystring of 999000 bits. But, it would still not be a one-time pad.

The need for the key to be both "endless and senseless" was recognized by Joseph Mauborgne (1881 – 1971) an officer in the United States Army Signal Corps.

The United States – Soviet Union hotline that was constructed during the 1960's used a one-time pad with the keystring on a paper tape. The voice encryption device SIGSALY

that was developed by Bell Labs for the United States Army during World War II used a one-time pad with the key consisting of random noise recorded on a record. Concern about the poor encryption systems that they used during World War I caused the Soviets to implement a one-time pad, hand cipher for their spies during World War II and after. Unfortunately for them, the need for many keys caused them to re-use keys. This was noticed by American cryptanalysts who were able to break many messages. The breaking of these messages was called the VENONA project and led to the arrests of the Soviet atomic spies.

From the 1920's until the 1970's encryption machines dominated cryptography. The purpose of these machines was to implement ciphers with long keys and, therefore, protect against attacks using the period.

Known plaintext attack on depth of two

A known plaintext attack can succeed against a stream cipher with a depth of two. If two messages are encrypted with the same keystream, XORing the two ciphertexts will remove the keys and result in the XOR of the plaintexts.

$$\begin{aligned}
&\text{CIPHERTEXT1} \oplus \text{CIPHERTEXT2} \\
&= (\text{plaintext1} \oplus \textit{key}) \oplus (\text{plaintext2} \oplus \textit{key}) \\
&= (\text{plaintext1} \oplus \text{plaintext2}) \oplus (\textit{key} \oplus \textit{key}) \\
&= (\text{plaintext1} \oplus \text{plaintext2}) \oplus 00\ldots0 \\
&= \text{plaintext1} \oplus \text{plaintext2}
\end{aligned}$$

If a crib, a portion of the plaintext, is known, an attack can be made.

Exercise 3.5. Here are two ciphertext messages that were encrypted using a Vernam cipher and the same keystring.

CIPHERTEXT1

```
11111111101111001000001110001111010000110101001000
11000010010110010010010000001100111101110100101011
10000010011010101111001101001 1
```

CIPHERTEXT2

```
11000110000010010011111010001101010111101011010011 0
11011100000010010110001101011100111111011000111010 0
10010
```

XOR the two ciphertexts to obtain the XOR of the plaintexts.

It is known that the word `jayhawk` appears in one of the plaintext messages. Drag the crib through the XOR of the plaintexts. For example, assume that the crib begins one of the plaintext messages

```
Crib                110101100010101001011100011001111110
XOR of plaintexts   001110011011010110111111010010101111
                    111011111001111111110001011110010001
                    q     k     v     k     h     u     z
```

If the crib is in the correct location, plaintext should result.

$$\begin{aligned} &\text{plaintext1} \oplus (\text{plaintext1} \oplus \text{plaintext2}) \\ &= (\text{plaintext1} \oplus \text{plaintext1}) \oplus \text{plaintext2} \\ &= \text{plaintext2} \end{aligned}$$

This crib is not in the correct location; shift it one place to the right and try again. Etc.

Once the crib is in place extend each plaintext message forward and backward until both plaintexts have been obtained. Also determine the keystring.

Modern stream ciphers

Modern stream ciphers operate much the same as Vernam's original cipher. They are not one-time pads; their keystrings are pseudorandom. Because the XOR operation and the methods used to generate keystrings are not complex operations, stream ciphers are typically faster than block ciphers. Stream ciphers are often used in situations (for example, wireless communications) in which the length of the plaintext message is not known beforehand.

References

David Kahn, *The Codebreakers*, Scribner, 1996.

Wikipedia is also an excellent reference for cryptological topics.

RC4

Symmetric-key block ciphers are probably the most important cryptographic tools. They are used to provide privacy, and they are often used as foundations of other cryptographic processes. But, typically a cipher suite includes symmetric-key stream ciphers as well as block ciphers. Stream ciphers are Vernam ciphers. They are not one-time pads; the keystream is pseudorandom. Therefore, stream ciphers can have serious security issues.

A common way to generate keystreams is by using linear feedback shift registers (LFSR). LFSRs are typically attacked – just like the Vigenère cipher was attacked -- by attacking the period.

Stream ciphers are typically faster in software and less complex in hardware than block ciphers. Because they operate on individual bits, buffering for large blocks is not required, it is not necessary to pad blocks, and error propagation is less likely. Stream ciphers are often used in wireless protocols where block size cannot be determined prior to transmission.

Stream ciphers come in two flavors: synchronous and self-synchronizing. Synchronous stream ciphers generate their keystreams independent of the plaintext and ciphertext. If the sender and receiver fall out of synchronization (e.g., by losing a ciphertext character during transmission), gibberish results. Self-synchronous stream ciphers generate keystreams based upon some of the previously transmitted ciphertext characters. If *n* ciphertext letters are used to generate the keystream and even if a ciphertext letter is lost during transmission, the keystream can resynchronize after *n* correct characters. Self-synchronizing stream ciphers are similar to the CFB mode of block ciphers.

RC4 is an example of a modern symmetric-key stream cipher. It was developed in 1987 by Ron Rivest, one of the developers of the public-key cipher RSA. RC4 is a trademark. RC2, RC5, and RC6 are symmetric-key block ciphers. RC4 does not generate its keystream by using a LFSR.

For RC4, stream combinations are done on byte-length strings of plaintext. 256 bytes of memory are required for the state array.

Exercise 4.4. For a simplified RC4 example we will do combinations on nibbles (half-bytes, 4-bit strings). 16 nibbles of memory will be required for the state array.

The state array *S* is initialized by

```
for i from 0 to 15
     S[i] = i
```

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(i)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

For RC4, *S* is initialized by 0, 1, …, 255.

To begin the key scheduling algorithm, RC4 uses some subset of 0, 1, …, 255 repeated as many times as necessary to fill positions 0 through 255 of the key array. In this exercise, we will use 2, 4, 5, 11, and 14 to fill positions 0, 1, …, 15 of the key array.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$K(i)$	2	4	5	11	14	2	4	5	11	14	2	4	5	11	14	2

The key scheduling algorithm randomizes *S*.

```
j := 0
for i := 0 to 15
     j := (i  + S(i) + K(i)) mod 16
     swap(S(i), S(j))
```

After the randomizing of *S* is complete, a pseudorandom number generation algorithm (PRGA) selects the nibbles for the keystream.

```
          i := (i  + 1) mod 16
          j := (j + S(i)) mod 16
          swap (S(i), S(j))
     t := (S(i) + S(j)) mod 16
k := S(t)
```

k (thought of as a nibble) is XORed with a nibble of the plaintext message.

Use the initializations above to generate a keystream and XOR that keystream with the plaintext message:

01000001011011000110010101111000010010110111010101101000011011 00

References

Alfred Menezes, Paul van Oorschot, and Scott Vanstone, *Handbook of Applied Cryptograph, Fifth Edition*, CSC Press, 2001. Sample chapters of this excellent reference may be downloaded from www.cacr.math.uwaterloo.ca/hac/

Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition*, Wiley, 1995.

Richard Spillman, *Classical and Contemporary Cryptology*, Prentice Hall, 2005. This text has a nice discussion of RC4, and the text includes a nice piece of software, called CAP, that can be used to explore RC4 and many other ciphers.