John Ewart
March 6, 2004

Cryptography: Techniques, Algorithms and Methods

Cryptology is really the official term for the study of methods of obfuscating and deciphering information. It is a field fraught with mathematics, creativity, logic, and ultimately a little bit of luck. Cryptography, on the other hand, is an application of cryptology. For the purposes of this paper, we will lump them together and simply refer to the entire field, both the study and the application, as cryptography.

A cryptographic cipher can be anything that obfuscates a particular piece of data from anyone but the intended recipient(s). Mathematically, encryption can be viewed as any function E, which, when applied to a plaintext P, yields a ciphertext C_E. Concurrently, there exists a decryption function D which when applied to C_E yields P. There are two basic classes of encryption algorithms: symmetric and asymmetric.

In a symmetric algorithm, both ends of the communication have a pre-shared key K which, when used as a parameter along with P to function E yields C_{KP}, and when passed to D along with C_{KP}, returns P. This is akin to a lock that has two keys, both of which are identical. As long as nobody except the intended targets get their hands on either copy of the key, communication is secured.

With asymmetric algorithms, on the other hand, each end has two separate keys. One key is a "public" (or encrypting) key (K_E), which is freely shared with anyone who wants it. The other is a "private" (or decrypting) key (K_D), which is never shared with anyone. The algorithm is designed such that any plaintext message P when transformed

by the algorithm using K_E as the key yields C_E such that it can be decrypted by transforming it using K_D and only K_D. There is some very fancy mathematics involved in this, which I will describe further on.

To understand cryptography, one must understand the need for cryptography. As time passes and technology gets better, so does the ability for people to invade our privacy, and thus the need for stronger cryptographic techniques to keep our private information private. For example, as we moved from old crossbar telephone systems to newer ESS (electronic switching systems), our capacity for communications increased, but with it came an enhanced ability to monitor those communications. Such is the same with the Internet. A packet you send from point A to point B will not only be seen by those two points. Rather, it will likely traverse a wandering path of hops until it reaches its destination. If the packet's contents are not protected from an outsider's view in some way, then anyone along the path has access to view the contents of that packet. These types of scenarios are where encryption comes into play.

Unfortunately, we cannot trust everyone to play fair in our world; unscrupulous individuals are always willing to use the weaknesses in technology and humans to their advantage. This fact was recognized as far back as Julius Caesar's time. While not necessarily accredited with the discovery or invention of ciphers, he is often referred to as the first historical figure to make use of cryptography in his communications. In his case, he used a relatively weak, but effective, cipher to obfuscate his messages known as the Caesarian Cipher. In the Caesarian Cipher, one chooses a value k between 1 and 25. This value is used to shift every letter to be written over k positions to produce the end result, or *ciphertext*.

For example, if the sender and recipient agree upon a value of 3 for *k*, then the *plaintext* message "Meet at the river at dawn" would be transformed as follows:

```
P: MEET AT THE RIVER AT DAWN
C: PHHW DW WKH ULYHU DW GDZQ
```

Yielding a string of text that looks like gibberish at first glance. Of course, this is not a very good cipher for reasons I will discuss later on, but in its day, it served its purpose. Without a computer at hand, or advanced mathematical tools, it can take someone a long while to figure out the encoded message. This type of cipher is very much like the puzzles in the New York Times of today. Another example of a polyalphabetic cipher is the Vigenere cipher, which is much more advanced than the Caesarian cipher. A detailed description of this cipher is attached at the end of this paper.

With the advent of mechanical computation devices (and later, electronic computers), the ability of cryptologists to produce stronger encryption methods grew with leaps and bounds. The onset of World War II showed the world its first glimpse of what might be referred to as "information warfare", a war where battles would be won or lost based upon who knew the enemy's presumed "secrets." This need for secret keeping caused the Germans to develop what became known as the Enigma, a complex mechanic-electrical device for encrypting messages, which stumped the British intelligence for quite a long time. This was the first real step towards computer encryption in the form of a small box that had both mechanical and electric parts.

DEA (Digital Encryption Algorithm) was one of the first algorithms designed for computers in the early 70s. In 1976, DEA was adopted as the federal standard for all non-classified government communications and then became known as DES (Digital

Encryption Standard). DES is a 64-bit block cipher, which means that a 64-bit block of plaintext "goes in one end" of the algorithm and a 64-bit block of ciphertext "comes out the other end." DES is a symmetric cipher and uses a 56-bit key with 8 bits of parity to ensure the key is transmitted properly. DES combines 16 rounds of substitution followed by permutation based on the key used. Its repetitive nature made it well suited for computational application in the 70s, as well as for specialized chips (which exist today) to do DES encryption.

Algorithmically, DES is quite simple; it uses algebraic functions that can be carried out quickly by a CPU. The initial and final permutations have no effect on the security of the DES algorithm, they are thought to be there to make it easier to handle the data on the CPUs DES originally targeted. The DES function itself is the complicated portion of the algorithm, and a description and example is attached at the end of the paper. DES, as an encryption method, suffers from one small flaw, being designed in the 70s, we can now use today's computers to crack a simple 56-bit DES encrypted block in a relatively timely fashion (meaning months, not eons). This was proven by the EFF (Electronic Frontier Foundation) in 1998 in the form of a custom-built machine. The cost of the hardware was $220,000, and was able to crack a 56-bit DES key in an average of 4.5 days. This threat can be side skirted, however, by using what is known as 3DES (or triple-DES), in which the block is encrypted using one key, decrypted using a second -- but incorrect -- key and then encrypted once more using the original key. This brings the order of complexity up from 2^{56} to 2^{112}, making it a much tougher nut to crack.

The leap from symmetric encryption functions such as DES to asymmetric models such as DSA (digital signature algorithm) was a significant one. Pioneered by

Whitfield Diffie and Martin Hellman, their discoveries in the area of public-key cryptosystems helped to move cryptography into the next era. Their idea is genius and simple at the same time. By using public keys, we can eliminate the need for pre-shared secrets, reducing the margin of human error and allowing us to use much stronger encryption algorithms because we can use much larger keys. Additionally, this makes implementation of this encryption much more flexible, as software can securely handle the negotiation of secrets to provide the encryption of data.

Looking forward, we have quantum cryptography upon the horizon, which leverages natural phenomena found in electrons to provide secrecy and immunity from prying eyes. Think -- if the act of observing the transmission alters the transmission, it can be mighty tricky to intercept the data en-route. As technology moves forward, so does our need for stronger encryption to keep our private data private. This means newer and better methods of encryption to keep up with newer and better eavesdropping capabilities.

References:

Bletchley Park. Station X.
Unknown revision date. Bletchley Park. Visited 01 March, 2004.
<http://www.bletchleypark.org.uk/>

Burton, David.
Elementary Number Theory.
McGraw Hill, 2002

Rivest, Ronald L. Ronald L. Rivest : Cryptography and Security.
Unknown revision date. Visited 26 February, 2004.
<http://theory.lcs.mit.edu/~rivest/crypto-security.html>

Sale, Tony. The Enigma Cipher Machine.
Unknown revision date. Bletchley Park. Visited 03 March, 2004
<http://www.codesandciphers.org.uk/enigma/>

Schneier, Bruce.
Applied Cryptography: Protocols, algorithms, and Source Code in C.
Wiley, 1995.

Singh, Simon.
The Code Book.
Doubleday, 1999.

The Vigenere Cipher

The Vigenere cipher is another simple example of a “polyalphabetic” cipher. This cipher uses a simple device called the “Vigenere Tableau”, which dictates letter substitution for a given combination of plaintext and keyword letters. For one to encode text using this cipher, first they must choose a pre-shared keyword, such as "FOOBAR", and write it above the plaintext message character for character until running out of plaintext as such:

```
   FOOB AR FOO BARFO OB ARFO
P: MEET AT THE RIVER AT DAWN
```

The ciphertext letter is looked up in the tableau by looking at the column containing the keyword’s letter, and the containing the plaintext letter. Thus the first "M" of the message is encrypted as "R.” Following this, the first "E" becomes an "A,” and so on, until we result in the fully encrypted ciphertext as:

```
   FOOB AR FOO BARFO OB ARFO
P: MEET AT THE RIVER AT DAWN
C: RAAU AK YVS SIMJF OU DRBB
```

To decrypt the Vigenere cipher, you simply run this process in reverse.

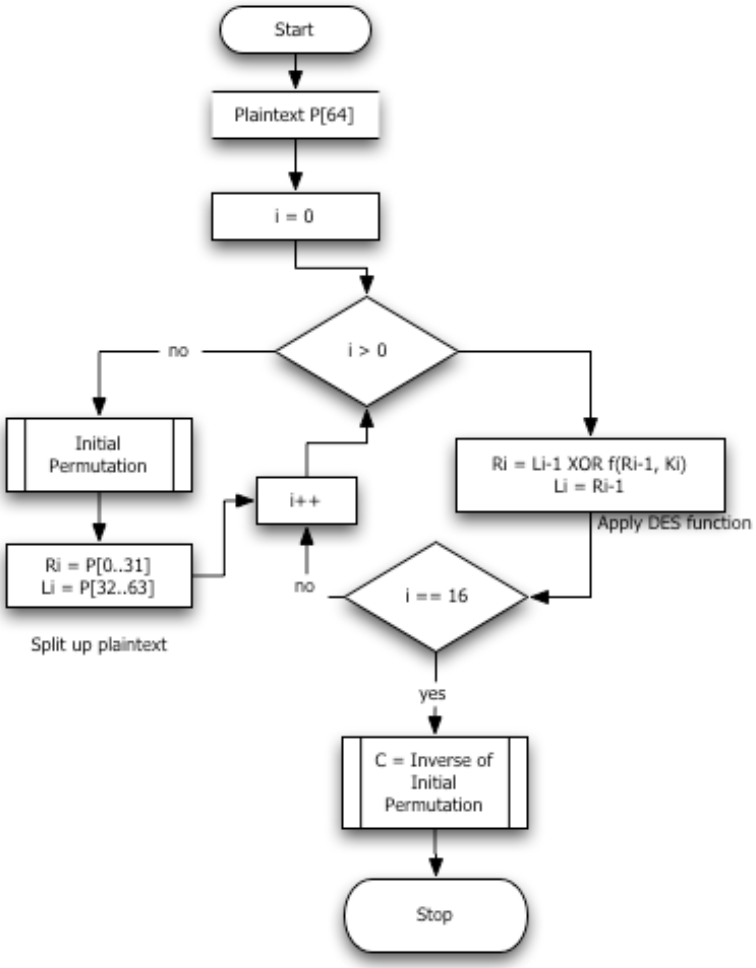
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z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

The Vigenere Tableau

DES Algorithm

The DES algorithm relies upon two basic mathematic functions, substitution and permutation. This makes it a relatively quick algorithm to use, as well as easy to implement both in software and hardware. To begin with, we need a data block of 64 bits, this is because DES is a "64-bit block cipher", meaning it works on data chunks that are 64 bits wide. If we have a block of data that is not evenly divisible by 64, it will typically be padded with 0s to make it so. In addition to our 64-bit data block, DES requires a 64-bit key *K*. The output is a 64-bit block of ciphertext that can be decrypted by applying DES in reverse using the same original key *K*.

DES works in 16 rounds, each round involves swapping left and right halves and applying the DES function to the new right half. Below is a simple flowchart of DES.



A simplified DES flowchart

At the heart of this is the actual DES function (f on the chart above), which is where all the real work happens.

Example:

```
M: 0000000100100011010001010110011111000100110101011110011011110111 1
K: 0001001100110111001011001010110000001000000100010010100001000000
```

1. **Create 16 sub keys of K each with 48-bit length**
   Our 64 bit key *K* is permuted according to Table 1 giving us K_P, note how there are bits missing, this is because every 8th bit is used as parity, so we drop them in the permuted key. We take K_P and split it into two 28-bit halves, A_0 and B_0. Now, we create 16 variations of A_0 and B_0 according to Table 2. Each cycle, we perform a certain number of left shifts, as taken from the table. For example, taking K and permuting it, we get:

   ```
   KP: 10000000010011100010001101010010001100000011000011101100
   ```

   Splitting in half yields:

   ```
   A0: 10000000010011100010001101 01
   B0: 00100011000000110000111011 00
   ```

   Following the rotation schedule for the first 2 iterations gives us:

   ```
   A1: 00000000100111000100011010 11
   B1: 01000110000001100001110110 00

   A2: 00000001001110001000110101 10
   B2: 10001100000011000011101100 00
   ```

   And, finally, once we've followed through with all 16 rotations, we concatenate A_n and B_n and permute based on Table 3 to yield K_n as such:

   ```
   K1: 011100000010001011001010110010001011010100 10010
   K2: 001000010011101010110100100010011011000001 01001
   K3: 100101000011110001000000001000100101111001 00000
   K4: 010000110110011001110000010110000000100001 10110
   K5: 110011001101010100000100100001010100100100 11000
   K6: 010000101000001101001011010000010011001010 10001
   K7: 001010001101000000100011101100111000000000 01100
   K8: 101000010000100111001010000000000000111110 00110
   K9: 000000110001001011011010000111000011000100 00000
   K10: 001111000101100001000100101000000110000011001 01
   K11: 000000100110100101001000011000101010101100 00010
   K12: 000010010110010100110101101101000000010000 11011
   K13: 110001000000110100001001000011110001001010 00010
   K14: 010000111010001010100001010101001110000011 00000
   K15: 100110001001110010000010001000001000110010 01100
   K16: 100000000000101000001010010000100001010110 11000
   ```

   Once we have the subkeys, we move on to the next step, encoding the 64-bit block of data.

2. Encoding the 64-bit block

Using Table 4, we rearrange the bits of the original message M to get M_{IP}. This initial permutation has no effect on the outcome of the algorithm; it is said that it was implemented initially to make the DES algorithm easier to implement on smaller-word machines such as those back in the 70s. Once we have M_{IP}, we split it in half to give us 2 32-bit blocks, a 'left' block L_0 and a 'right' block R_0.

Now we take these two halves and perform the DES operation on them 16 times. Each time, we swap the left and the right halves, making the left value the right hand result of the previous iteration, and the right value is the sum of the DES function applied to the previous right hand side and the current key K_n combined with the previous iteration's left hand side with XOR addition. Like such:

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} + DES(R_{n-1}, K_n)$$

The hard part of this is understanding the DES function, which is a brilliant and obscure function that I will try my best to explain.

1. Expand our 32-bit string into a 48-bit string. This is accomplished by looking at Table 5 and permuting the old string into a new one. In this permutation, you will notice we duplicate certain bits, this makes the input block the same length as our keys which are 48-bits wide.
2. XOR the new input block with our key K_n
3. Now we take the XORed result and group it into 8 groups of 6-bit chunks, $C_{i,}$.
4. For each of the chunks, Ci, we need to create a 4 bit block as our output (our original input block is really 64 bits, so we need 2 32-bit blocks, right now we have 48), so we create this 4-bit value by looking up the appropriate numbers in the matching S box (See Table 6) using the following algorithm:
    a. For $1 <= n <= 8$, take block C_n and S-Box S_n
    b. Take bits 0 and 15 of the 16 bits and let this represent a 2-bit base-2 number from 0 to 3, use this as the row index in S_n. Take the middle 4 bits and let this represent a 4-bit base-2 number between 0 and 15; use this as the column index in S_n. Take this value in 4-bit binary as your 4-bit block in place of C_n.

3. Finishing up.

Once you've completed the 16 iterations in step 2, apply the permutation bit-order in Table 6, this functions as the inverse of the initial permutation. This will yield an encrypted 64-bit ciphertext block.

Tables

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Table 1 -- K_P Bit-order

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
# SHL	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 2 – Key schedule (Left shift count)

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Table 3 – K_n Bit-order

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Table 4—Initial Permutation bit-order

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Table 5 – Shuffle Permutation

0	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Table 6 – Inverse IP bit-order

Table 7 – “S Boxes”

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S1

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S2

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7

1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S3

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S4

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S5

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S6

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S7

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

S8

Table 5 S-Boxes