Hulaan Mo: Some Notes on Cryptography

Shiela Kathleen L. Borja
Joseph Anthony C. Hermocilla
Mark Froilan B. Tandoc

1 Introduction

Cryptography is an area of study of encryption of plaintext to a so called ciphertext. Cryptographic systems are characterized along three independent dimensions. First is the type of operations used for transforming plaintext to ciphertext. All encryption algorithms are based on two general principles: *substitution* where each element in the plaintext is mapped into another element, and *transposition* where elements in the plaintext are rearranged. Second is the number of keys used. If both sender and receiver use the same key, the system is referred to as *symmetric*, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as *asymmetric*, two-key, or public-key encryption. Last dimension is the way in which the plaintext is processed. A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time [1].

There are two types of attack on an encryption algorithm. *Cryptanalysis* is based on properties of the encryption algorithm and *brute-force* which involves trying all possible keys.

2 Classical Encryption Techniques

2.1 Caesar Cipher

The earliest and simplest known substitution cipher by Julius Caesar [1]. This cipher involves replacing each letter of the alphabet with the letter standing three places (or even more for the general algorithm) further down the alphabet. The alphabet is wrapped around where Z follows A. For example:

plain: COMSCI IS THE BEST
cipher: FRPVFL LV WKH EHVW

If the attacker knew that Caesar cipher was used, brute-force cryptanalysis is easily performed and there are only 25 keys to try.

2.2 Monoalphabetic Cipher

It was shown that Caesar cipher is far from secure. The key space can be dramatically increased if we allow arbitrary substitution for each letter/character. With this, the number of possible keys is increased from 25 to the number of permutations for the given alphabet or 26! possible keys. There is still, however, another possible attack. If the cryptanalyst knew the nature of the plaintext (e.g. noncompressed English text), then he/she can exploit the regularities of the language like relative frequency of the letters in the ciphertext and compare it to the standard frequency distribution for English [1].

2.3 Playfair Cipher

The Playfair algorithm uses a 5 x 5 matrix of letters constructed using a keyword. The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom. Then filling the remainder of the matrix with the remaining letters in alphabetical order. The letters I and J count

as one letter [1]. For example with keyword *monarchy*:

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

The plaintext is encrypted two letters at a time, according the the rules:

- Repeating plaintext letters that are in the same pair are separated with a filler letter, such as x, so that balloon would be treated as ba lx lo on.
- Two plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, withe first element of the row circularly following the last. For example, ar is encrypted as RM.
- Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, mu is encrypted as CM.
- Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, hs becomes BP and ea becomes IM/JM.

The Playfair cipher is a great advancement over monoalphabetic ciphers. However, despite this level of confidence in its security, it is still relatively easy to break because it still leaves much structure of the plaintext language intact.

2.4 Polyalphabetic Cipher

Another way to improve monoalphabetic technique is to use different monoalphabetic substitutions through encryption of the plaintext message. The general name of the technique is polyalphabetic substitution cipher [1].

The best known and one of the simplest is the **Vigenère cipher**. It is essentially multiple caesar ciphers. For every letter in the plaintext, we use one letter at a time of the secret key as the key to apply for the Caesar cipher. For example:

key:	deceptivedeceptive
plaintext:	wearediscoveredsav
ciphertext:	ZICVTWQNGRZGVTWAVZ

When expressed numerically,

key	3	4	2	4	15	19	8	21	4
plain	22	4	0	17	4	3	8	18	2
cipher	25	8	2	21	19	22	16	13	6
key	3	4	2	4	15	19	8	21	4
plain	14	21	4	17	4	3	18	0	21
cipher	17	25	6	21	19	22	0	21	25

The length of the key can be derived by observing for repeating sequences in the cipher text. In the example, the sequence VTW occurred twice and it can be used to determine the length of the key which is 9.

To defend on such cryptanalysis, we can choose to have a very long keyword as long as the plaintext and has no statistical relationship to it. It was introduced by an AT&T engineer Gilbert Vernam in 1918 called **Vernam Cipher** [1].

2.5 One-Time Pad

An Army Signal Corp officer named Joseph Mauborgne, improved the Vernam cipher that is said to be unbreakable. The key should be random and as long as the plaintext such that the key is not repeated.

Moreover, new messages requires new keys of the same length as the new message. This produces random output with no statistical relationship with the plaintext, therefore no way of breaking the code. However, there are practical difficulties. One is the problem of making large quantities of random keys. Another is the problem of key distribution and protection.

2.6 Transposition Techniques

The techniques discussed were all substitution of a ciphertext symbol for a plaintext symbol. A different kind of mapping is by performing some sort of permutation on the plaintext letters. This is referred to as a transposition cipher [1].

The simplest of a transposition cipher is the **rail fence** technique where the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows. For example, the message "*meet me after the toga party*" with a rail fence of depth 2:

```
m e m a t r h t g p r y
 e t e f e t e o a a t
```

will yield the encrypted message:
MEMATRHTGPRYETEFETEOAAT

A more complex scheme is to write the message in a matrix and read the message off, column by column, but permute the order of the columns. The order of the columns would then become the key of the algorithm. For example:

Key:	4	3	1	2	5	6	7
Plaintext:	a	t	t	a	c	k	p
	o	s	t	p	o	n	e
	d	u	n	t	i	l	t
	w	o	a	m	x	y	z

Ciphertext:
TTNAAPTMTSUOAODWCOIXKNLYPETZ

The transposition stage can be done repeatedly to have more complex permutation to improve security.

3 Directions of Cryptography

3.1 Symmetric Ciphers

Symmetric encryption is where encryption and decryption are performed using the same key. It is also known as conventional encryption [1]. A symmetric encryption scheme has five parts:

1. **Plaintext:** The original intelligible message or data.
2. **Encryption Algorithm:** Performs various substitutions and transformations on the plaintext.
3. **Secret Key:** It is also an input for the encryption algorithm. The exact substitutions and transformations performed by the algorithm depend on the key.
4. **Ciphertext:** The scrambled message produced as output.
5. **Decryption algorithm:** Essentially the encryption algorithm performed in reverse. It takes the ciphertext and the secret key as input and produces the plaintext as output.

Other terms that are used in the sections are:

1. **Field:** A set where the four fundamental operations preserving the closure property. [1]

2. **GF(2^8):** defined as a finite field containing 2n elements. [1]

3. **prop ratio:** the relative amount of all input pairs that for the given input difference give rise to the output difference. [2]

3.1.1 Block Ciphers

Block cipher allows a block of plaintext to be treated as a whole and used to produce a ciphertext block of equal length. The simplest techniques that can be used to encrypt a block of plaintext are substitution and permutation. Substitution replaces a symbol or groups of symbols with a corresponding ciphertext symbols or groups of symbols. Permutation replaces a sequence of plaintext symbols with a permutation of the same set of symbols. [1]

3.1.2 Feistel Cipher Structure

As discussed by Stallings [1], the inputs required in the encryption algorithm are the plaintext block of length 2w bits and a key K. The plaintext block is then divided into two halves, L_0 and R_0, which pass through n rounds of processing and then combine to create the ciphertext. In each round, substitution is performed on the left-half block , L_0. This is done by applying a round function F to the right-half block and then applying XOR to F's output and the left-half block. The round function has the same general structure for each round but is parameterized by the round subkey K_i. The decryption algorithm for the Feistel cipher is basically the same as the encryption process with the rule: Use the ciphertext as a input to the algorithm, but use subkeys K_i in reverse order. In order to ensure that decryption is the inverse of encryption, the final round of a Feistel cipher switches the ciphertext to (R_0, L_0). Figure 1 shows the Feistel encryption and decryption processes.

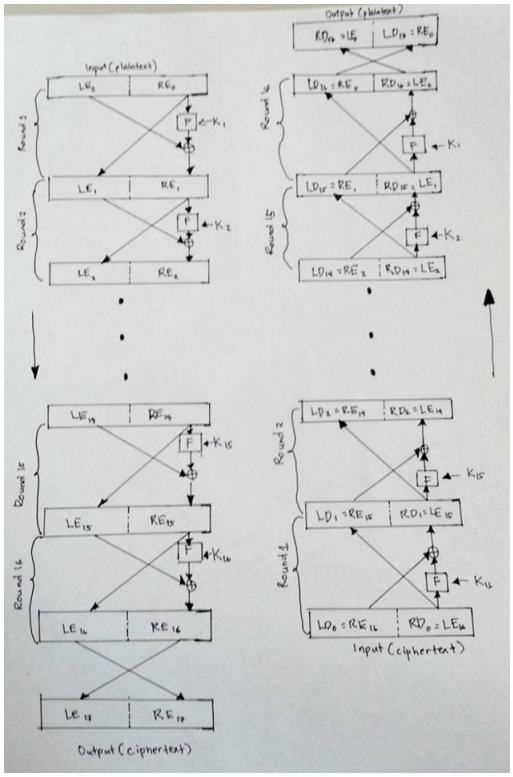


Figure 1: Feistel Encryption and Decryption

3.1.3 The Data Encryption Standard (DES)

The Data Encryption Standard is a block cipher encrypts the data in 64b blocks through a 56b key. It is the most widely used public cryptosystem in the world. It is mainly used for electronic funds transfer, civilian satellite communications security, and UNIX password protection. [3] DES was first proposed in 1975 and was adopted by the National Bureau of Standards(NBS) in 1977. DES was approved as the Federal Information Processing Standard 46. [1] The development of DES can be traced back to the development of LUCIFER, a Feistel block cipher that uses 64b blocks and 128b key. LUCIFER became very promising leading IBM, its developers, to launch a new marketable commercial encryption product which can be implemented on a chip. The result of the new project was a more cryptanalysis-secure version of LUCIFER that uses 56b key. In 1973, NBS accepted proposals for a national cipher standard where IBM submitted the refined version of LUCIFER which was later adopted as the Data Encryption Standard. [1] DES came under fire even before its adaptation as a standard due to its enormous key size reduction and its classified design criteria for its internal structure, S-boxes. The critics were concerned that the new key size is vulnerable to brute-force attacks. Also, they were concerned that the classified design criteria might hide weak points that would enable the National Security Agency(NSA) to decipher messages without the key. [1] Despite the issues surrounding DES, it remained as a strong encryption algorithm until the mid 1990s when a USD250,000 computer built by the Electronic Frontier Foundation (EFF) decoded a DES-encoded message in 56 hours. This was later improved to 22 hours using a combination of 100,000 networked PCs and the EFF machine. [3]

DES Encryption As discussed by Landau [3], input to the algorithm are the plaintext grouped into 64b blocks and a single 56b key. First step in the encryption is to rearrange the 64b plaintext to produce a permuted input. Throughout the whole plaintext, a single 56b key is used to determine the transformation of all the blocks. DES uses a 48b subkey for each of the sixteen identical rounds of mixing. DES begins with an initial permutation P and ends with its inverse, P^{-1}. Selecting the subkeys or key schedule begins with splitting the 56b key into two 28b halves. Each half is rotated one or two bits (one bit in rounds 1, 2, 9 and 16; two bits otherwise) then the two halves are put back together, and then the 48 bits are chosen and put in order as follows:

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

DES is a standard Feistel construction:

$L_i = R_{i-1}$,
$R_i = L - i - 1\, xor\, f(R_{i-1}, K_i)$
where
$f(R_{i-1}, K_i) = P(S(E(R_i - 1)\, xor\, K_i))$
with the operations
$E(expansion), S(S - boxlookup), P(permutation)$

The expansion operation E begins with bit 32 and cycles back to the beginning, using all the bits in order, repeating every fourth and fifth bits. E allows every bit of a DES ciphertext to depend on every bit of the plaintext and every bit of the key. [3]

After performing XOR operation on the round subkey and the expanded right half, the result is then passed through the S-boxes. Each of the S-box takes an input of six bits and has an output of four bits. Each of the output of the eight S-boxes is concatenated producing a 32b output. [3]

P^1, a specific 32b permutation of the S-boxes' output, completes the round function. It carries 1, ..., 32 into the following list:

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

P guarantees that the output from one round of DES affects the input for the multiple S-boxes in the next round. At the end of all the rounds, DES ends with a final exchange of the left and right halves (and then P^{-1}). [3] Figure 2 shows the Data Encryption Standard.

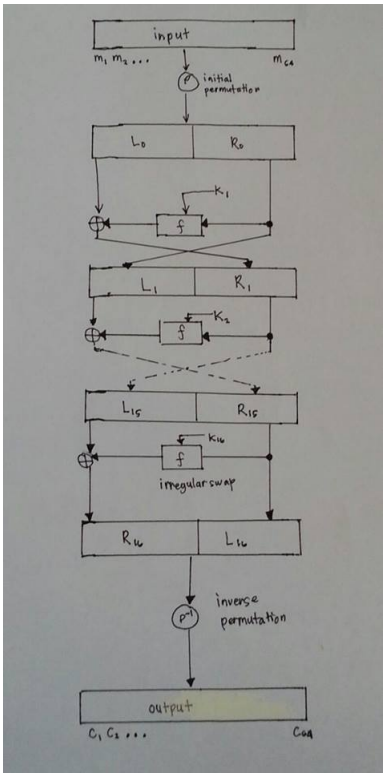


Figure 2: The Data Encryption Standard

DES Decryption Since DES has a Feistel cipher structure, it follows that its decryption algorithm uses the same algorithm as its encryption algorithm, except that the application of the subkey is reversed. [1]

Attacks on DES It was no secret that DES was bombarded with a lot of protests regarding its small key space. A lot of its critics tried to prove that DES was not a very strong encryption algorithm. Discussed here are some of the attacks made on DES.

Differential Cryptanalysis Pioneered by Eli Biham and Adi Shamir, differential cryptanalysis attack requires only examining 2^{47} texts which is lower than the 2^{56} required in the exhaustive search. As described by Landau [3], Biham and Shamir's attack on DES is:

1. Pick an appropriate difference ΔX.
2. Create an appropriate number of plaintext pairs with this ΔX, encrypt with DES, and store the ciphertext pairs.
3. For each pair, from the plaintext ΔX and the ciphertext pair, determine the expected output difference of as many S-boxes in the last round as possible.
4. For each possible key value, count the number of pairs that result with the expected output change using the value in the last DES round.
5. The right key value is the one suggested by all the key pairs.

Linear Cryptanalysis The seemingly simple and unanticipated Linear Cryptanalysis attack was developed by Mitsuru Matsui. Its objective is to find an effective linear equation of the form [3]:
$P[\alpha_1\alpha_2\cdots,\alpha_3] \oplus C[\beta_1\beta_2\cdots,\beta_3] = K[\gamma_1\gamma_2\cdots,\gamma_3]$

Matsui determined the best linear approximate expressions for DES until 20 rounds. He then used a combination of reduced-rounds linear approximation and exhaustive search to get the key. On the average, the attack requires 2^{43} known plaintexts. In 1994, this attack broke a DES-encoded message in 50 days. [1]

3.1.4 Advanced Encryption Standard (AES)

As discussed by Stallings [1], AES was first published in 2001 by the National Standards and Technology (NIST) to replace Data Encryption Standard as the approved standard for commercial purposes. The winning algorithm was first known as the Rijndael Block Cipher proposed by Joan Daemen and Vincent Rijmen. AES is a symmetric block cipher that uses a 128b block size and 128, 192, or 56 bit key size. AES do not follow the Feistel structure but each round consists of four different functions:byte substitution, permutation, arithmetic operations over a final field, and XOR with a key. AES also operates in 8-b bytes and the four fundamental operations–addition, subtraction, multiplication and division are all defined over the finite field GF(2^8). Addition of two bytes is defined as XOR operation. Multiplication of two bytes is defined as the multiplication of polynomials modulo an irreducible polynomial of degree 8, $m(x) = x^8 + x^4 + x^3 + x + 1$.

AES Encryption As cited by Stallings [1], the input for the encryption and decryption algorithms is a single 128b block which is depicted as a 4x4 matrix of bytes and is copied on a state array modified at each state of encryption and decryption. After the final stage, the state is copied to the output matrix. The key used is also depicted as a square matrix which is then expanded into an array of key schedule words, four bytes each. The ordering of the bytes within the matrix is by column.

The number of rounds(N) in the cipher depends on the length of the key: 10 rounds for 16-B keys, 12 rounds for 24-B, and 14 rounds for 32-B keys. For the first N-1 rounds, there are four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey. *SubBytes* uses an S-box to perform a byte-by-byte substitution of the block. *ShiftRows* is a simple permutation. *MixColumns* is a substitution that makes use of arithmetic over GF(2^8). *AddRoundKey* is a simple bitwise XOR of the current block with a portion of the expanded key. The final round has three transformation functions. There is also an initial single transformation (AddRoundKey) before the first round (Round 0). The cipher begins and ends with an AddRoundKey stage since only this stage makes use of the key.

All of the stages are reversible. For the SubBytes, ShiftRows, and MixColumns, an inverse function is used in the decryption algorithm. In the AddRoundKey stage, the inverse is determined by performing an XOR operation between the same round key to the block, using the result that $A \oplus B \oplus B = A$.

Figures 3 and 4 shows an AES Encryption Round and the AES encryption and decryption respectively.

AES Decryption The decryption algorithm uses the expanded key in reverse order but is not identical to the encryption algorithm. [1]

AES vs Known Attacks As discussed by Daemen and Rijmen [2], here are some of the known attacks during the writing of the Rijndael Block Cipher Proposal.

Symmetry properties and weak keys of the DES type The cipher and its inverse have different components, eliminating the the possibility of weak or semi-weak keys of DES type. Equivalent keys are prevented by the non-linearity of the key expansion.

Differential Cryptanalysis Differential Cryptanalysis Attacks are possible if the prop ratio is significantly larger than 2^{1-n} where n is the block length. AES has no 4-round differential trails with a prop ratio above 2^{-150}.

Linear Cryptanalysis Linear Cryptanalysis Attacks are possible if there are linear trails with correlation coefficients greater than $2^{n/2}$. AES has no 4-round linear trails with correlation greater than 2^{-75}.

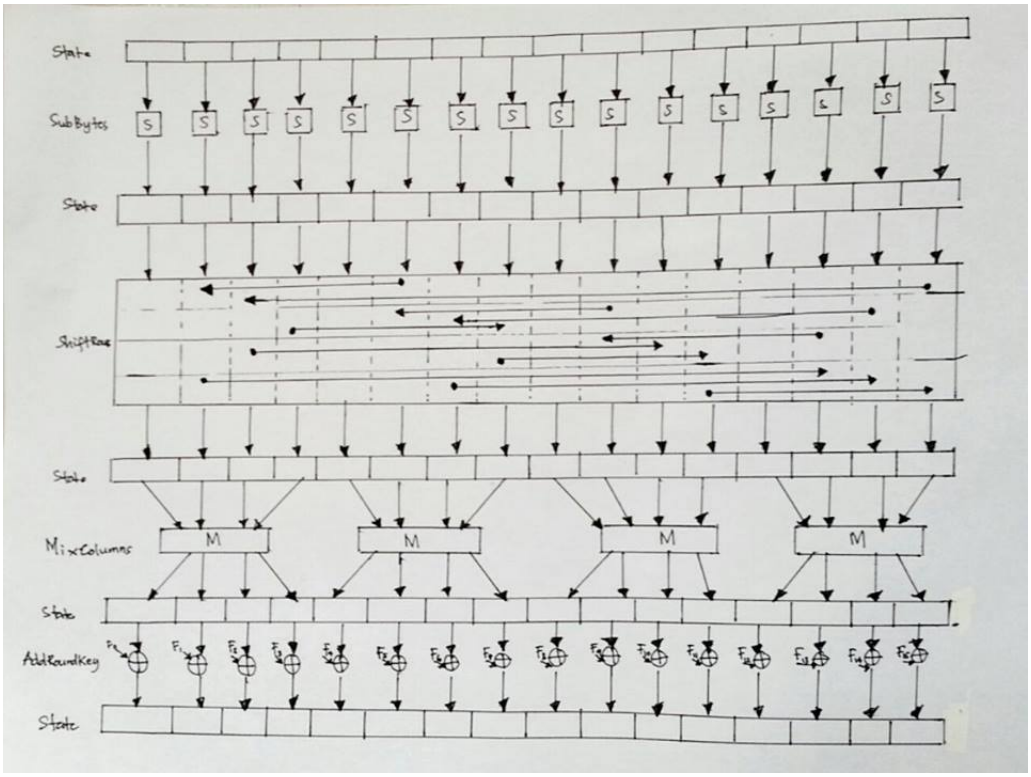


Figure 3: AES Encryption Round

Truncated Differentials Truncated Differentials are possible when the number of differential trails for certain sets of input difference patterns and output difference patterns is very large. Since AES meets the condition for truncated differentials to occur, all transformations operating on bytes were investigated. For six or more rounds, no attacks faster than exhaustive key search were found.

The Square Attack The Square attack is an attack dedicated to the Square. It takes advantage of the byte-oriented structure of Square Ciphers. AES inherits many properties form Square cipher therefor AES is also vulnerable to this type of attack.

The Square attack is found to be faster than exhaustive key search but experiments also showed that there were no extensions to 7 rounds faster than exhaustive key search.

Interpolation Attacks Interpolation attacks are feasible if the cipher components have a compact algebraic expression and can be mixed to provide expressions with manageable complexity. AES prevents these attacks (for more than a few rounds) through its complicated expression of the S-box in GF(2^8) together with the effect of diffusion layer.

Weak Keys as in IDEA These weak keys typically occur in ciphers with non-linear operations depending on the actual key value. In AES, weak keys are prevented by applying the keys using EXOR and putting all non-linearity in the S-box.

Related-key attacks Related-key attacks can be achieved through using different keys with a chosen relation in different cipher operations. This attack is prevented by AES through its high diffusion and non-linearity.

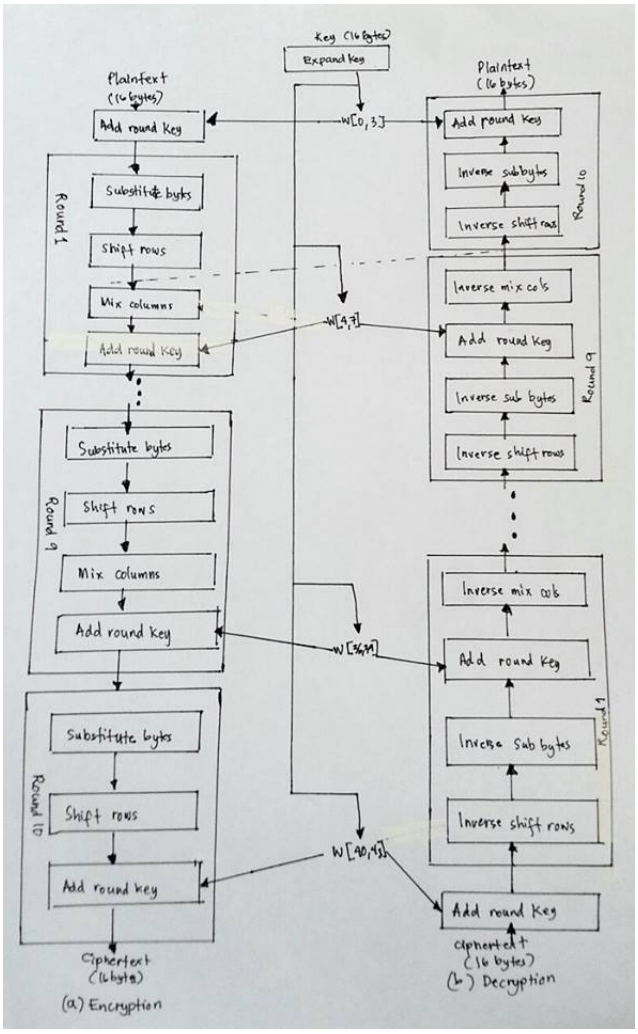


Figure 4: AES Encryption and Decryption

Limitations of AES As discussed by Daemen and Rijmen [2], here are some of the known limitations of the AES.

- Since the inverse cipher takes moire code and cycles, the inverse cipher is less likely to be implemented on a smart card than the cipher itself.
- The cipher and its inverse has different codes and/or tables.
- The inverse cipher cannot fully re-use the circuitry implementing the cipher.

3.1.5 Stream Ciphers

Stream ciphers typically encrypts plaintext one byte at a time, although may differ depending on the design. In the structure of the stream ciphers, a key is input to a pseudo-random bit generator that produces a stream of 8-bit random numbers. This stream of random numbers is called a **keystream** which is combined one byte at a time with the plaintext using the bit-wise exclusive-OR (XOR) operation [1]. For example:

	11001100	plaintext
encrypt	$\oplus$ 01101100	key stream
	10100000	ciphertext

	10100000	ciphertext
decrypt	$\oplus$ 01101100	key stream
	11001100	plaintext

A stream cipher can be as secure as a block cipher of comparable length if used with a properly designed pseudo-random number generator.

3.1.6 RC4

RC4 is a stream cipher designed by Ron Rivest of RSA Security in 1987. It is used in the Secure Sockets Layer/Transport Layer Security (SSL/TLS) standards that have been defined for communication between Web browsers and servers. It is also used in the Wired Equivalent Privacy (WEP) protocol and WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard [1].

A variable-length key of from 1 to 256 bytes is used to initialize a 256-byte state table [4]. The state table is used for subsequent generation of pseudo-random bits and then to generate a pseudo-random stream which is XORed with the plaintext to give the ciphertext.

The algorithm can be broken into two stages: initialization and operation. In the initialization stage, the state table **S** is populated using **K** as a seed. Once the state table is initialized, it continues to be modified in a systematic pattern as data is encrypted. The initialization process is summarized in the pseudocode below:

$j = 0$
for $i = 0 \rightarrow 255$ **do**
    $S[i] = i$
end for
$j = 0$
for $i = 0 \rightarrow 255$ **do**
    $j = (j + S[i] + K[i]) mod 256$
    $swap(S[i], S[j])$
end for

Stream generation involves cycling through all the elements of the state table, and for each S[i], swapping S[i] with another byte in **S** according to a scheme dictated by the current configuration of **S**. After S[255], the process continues starting over again at S[0].

$i, j = 0$
while *true* **do**
    $i = (i + 1) mod 256$
    $j = (j + S[i]) mod 256$
    $swap(S[i], S[j])$
    $t = (S[i] + S[j]) mod 256$
    $k = S[t]$
end while

To encrypt, XOR the value k with the next byte of plaintext. To decrypt, XOR the value k with the next byte of ciphertext.

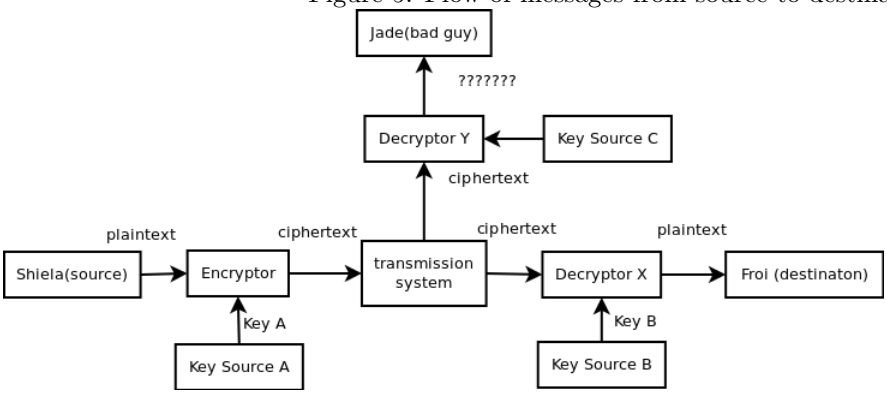
According to Mousa and Hamad [4], the speed of encryption or decryption time for RC4 is directly related to the encryption key length and to the data file size.

4 Asymmetric Ciphers

Cryptographic systems rely on keys for encryption and decryption. Traditionally, a single key is required to encrypt and to decrypt. In order for the recipient of the encrypted message to be decrypted by the recipient, the key must also be transmitted. However, sending the key over the channel (normal channel) where the actual message will be sent is insecure. The key must be transmitted on a different and secure channel (key channel) [5]. This secure channel where the key should be transmitted cannot be used for

normal transmission because it is costly and sometimes difficult for users to access and use [5]. This begs the question whether it is possible to send encrypted messages in such a way that the key can also be transmitted over the normal (insecure) channel and still achieve secure communication. In this section, we focus on solving this problem by describing the relevant and important work on asymmetric ciphers. Figure 5 describes the flow in asymmetric cryptography.

Figure 5: Flow of messages from source to destination.



4.1 Merkle (1978)

Secure communication, as described by Merkle [5], allows two parties to communicate in a private manner even though a third party tries its best to learn what is being communicated. We refer to the two parties as Froi and Shiela, and the third party as Jade (Figure 5). Since the key channel is important, the following describes the characteristics of the channel in relation to Jade.

1. All attempts by Jade to change the messages on the key channel are detectable.
2. Jade will not be able to know the actual content of any message passing on the key channel.

The approach by Merkle relaxes the second condition: It is not necessary for Jade not to know what is being sent in the key channel, he can even know everything passing on it. The challenge then is how to securely distribute the key satisfying the conditions above. If Froi and Shiela have agreed upon a key, and the work needed by Jade to find the key is much higher than the effort by Froi and Shiela needed to generate the key , then it is a solution. The effort by Jade should be exponentially higher compared to the effort by Froi or Shiela for a method to be considered a solution.

Merkle's method uses the concept of puzzles [5]. A puzzle is a cryptogram that is meant to be solved. Any encryption function can be used to generate a puzzle. To allow the puzzle to be solved, the key size (N) used in the encryption function is restricted. The difficulty of solving a puzzle can be controlled by adjusting the size of N. A very large size (in bits) of N will make it very difficult to solve the puzzle. In addition, in order to be able to solve the puzzle, some redundancy is needed. Redundancy is introduced by encrypting, along with the original message, some constant known to Froi, Shiela, and Jade. The absence of the constant when a puzzle is decrypted would mean that a wrong key has been used.

Let us consider the scenario when Shiela wishes to send a message to Froi. First, they both agree on the value of N to use. Shiela then generates N puzzles and transmits these N puzzles to Froi using the key channel. Each puzzle generated will have a puzzle ID and puzzle key. The puzzle ID uniquely identifies each puzzle. The puzzle key on the other hand will be used in future communications that will happen once this puzzle has been solved.

When Froi receives the N puzzles, he selects a puzzle at random and attempts to solve the puzzle, with the amount of effort required, as defined by the size of the key space specified by Shiela. After solving a puzzle, Froi sends the puzzle ID back to Shiela using the key channel. The puzzle key, associated with the puzzle ID sent by Froi, is then used for future communications, this time over the normal channel. At this point Jade knows the puzzle ID, since it was sent using the key channel, but not the puzzle key. If Jade

wants to know the key, then he must solve puzzles randomly and check the puzzle ID if it matches the one sent by Froi back to Shiela. This will take Jade a long time to solve. To put it formally, Jade will require $O(N^2)$ effort to determine the key whereas Froi will only need, on the average, $O(N)$. The function below generates the puzzles sent by Shiela to Froi. The encryption function is arbitrary.

```
1  void generate_puzzle()
2  {
3     bit_string id, key, c, random_key, puzzle, k1, k2;
4     int i;
5
6     k1 = rand(MAXINT);
7     k2 = rand(MAXINT);
8     c = rand(MAXINT);
9     send(c);
10    for (i=0; i<N; i++)
11    {
12       id = encryption_function(k1,i);
13       key = encryption_function(k2, i);
14       random_key = rand(c*N);
15       puzzle = encryption_function(random_key,id,key,c);
16       send(puzzle);
17    }
18 }
```

The code below is executed on Froi's side.

```
1  void get_id()
2  {
3     bit_string id, key, c, selected_puzzle_id, the_puzzle, current_puzzle,
4                temp_constant;
5     int i;
6
7     selected_puzzle_id = rand(N);
8     receive(c);
9     for (i=0; i<N; i++)
10    {
11       receive(currrent_puzzle);
12       if (i == selected_puzzle_id)
13          the_puzzle = current_puzzle;
14    }
15    for (i=0;i<(c*N);i++)
16    {
17       id = get_id(finverse(i, the_puzzle));
18       key = get_key(finverse(i, the_puzzle));
19       temp_constant = get_constant(finverse(i, the_puzzle));
20       if (temp_constant == c)
21          send(id);
22    }
23 }
```

Once Shiela receives the the puzzle ID from Froi, then the following code will be executed. key will be used for subsequent communications between the two.

```
1  void continue_transmission()
2  {
3     receive(ID);
4     key = encryption_function(k2, ID);
```

5 }

The approach by Merkle requires an effort of $O(N^2)$ from Jade to get the key. However, in todays available computing resources, this can be easily broken. The possibility of exponential methods will be more attractive. Also the amount of information sent during the initial setup of the communication is large because N puzzles, consequently N keys, are sent initially.

4.2 Diffie-Helman (1976)

The work by Diffie and Helman [6] proposed a method such that only one "key" needs to be exchanged and in addition the time required from Jade to perform cryptanalysis is exponential. In addition, it allows authentication because its use allows it to be tied to a public file of user information. Shiela can authenticate Froi and vice versa.

Diffie and Helman differentiate *public key cryptosystems* and *public key distribution systems*. We let K be the finite key space from which keys K can be obtained and M be the finite message space where messages M are derived. A *public key cryptosystem* is a pair of families of algorithms E_k andD_k which represent invertible transformations [6].

$$E_k : \{M\} \rightarrow \{M\}$$

$$D_k : \{M\} \rightarrow \{M\}$$

such that

1. for every key K, E_k is the inverse of D_k,
2. for every K and M, the algorithms E_k and D_k are easy to compute,
3. for almost every K, each easily computed algorithm equivalent to D_k is computationally infeasible to derive from E_k,
4. for every K, it is feasible to compute inverse pairs E_k and D_k from K.

Property 3 allows E_k to be made public without compromising D_k. Key distribution in this system is simplified. Users generate two keys, an enciphering key E and a deciphering key D. E can be made public but D is kept privately by the user. Any entity who would like to send messages to a user can use the publicly available E to encrypt messages but only the user can decrypt the message using D. In their paper, Diffie and Helman gave an example public key cryptosystem by multiplying a binary n-vector message m with an invertible binary n x n matrix E. However, this approach is not practical.

Merkle's [5] work was classified by Diffie and Helman as *public key distribution system* and highlighted its limitations specifically its high transmission overhead again because of sending N puzzles initially. The proposed system is similar to the public key cryptosystem described above, but unlike Merkle's technique, Diffie and Helman approach allows the authentication of users by making the public file read-only [6].

The technique proposed is dependent on the difficulty of computing $logs\, mod\, q$ where q is a prime number representing the number of elements of a finite field. Users generate independent random numbers X_i from the set of integers $\{1, 2, ..., q-1\}$. The users keep these numbers but the computed value

$$Y_i = \alpha^{X_i} mod\, q$$

is placed publicly together with the user information(such as name and email).

Consider for example Shiela and Froi would like to talk to each other privately. They are going to use the key $K_{Shiela,Froi}$below after they generate $X_{Shiela}$and X_{Froi} and published Y_{Shiela} and Y_{Froi}.

$$K_{Shiela,Froi} = \alpha^{X_{Shiela}X_{Froi}} mod\, q$$

Shiela will be able to obtain $K_{Shiela,Froi}$ by using the public file Y_{Froi} and then computing

$$K_{Shiela,Froi} = Y_{Froi}^{X_{Shiela}} mod\, q$$

$$= (\alpha^{X_{Froi}})^{X_{Shiela}} mod\, q$$

$$= \alpha^{X_{Froi}X_{Shiela}} = \alpha^{X_{Shiela}X_{Froi}} mod\, q$$

Froi will be able to obtain the key in the same manner.

$$K_{Sheila,Froi} = Y_{Shiela}^{X_{Froi}} mod\, q$$

Jade might be able to compute$K_{Shiela,Froi}$ from Y_{Shiela} and Y_{Froi} by computing

$$K_{Shiela,Froi} = Y_{Shiela}^{(\log_\alpha Y_{Froi})} mod\, q$$

However, if Jade is to perform this computation, it will take him a long time to do so. This system takes advantage of the fact that $logs\, mod\, q$ are expensive to compute.

4.3 Rivest-Shamir-Adleman (1978)

The RSA algorithm by Rivest, Shamir, and Andleman was inspired by the work of Diffie and Helman. Despite the breakthrough in Diffie and Helman's work in public key cryptosystems, they did not present any practical implementation that can be used in actual systems. The creators of RSA took the work further by presenting a practical and efficient implementation. Given an encryption procedure **E** ,a decryption procedure **D**, and a message **P**, a public key cryptosystem has the following properties [7]:

1. Decrypting an encrypted **P** results to **P**. **D**(**E**(**P**)) = **P.**
2. **D** and **E** are easy to compute.
3. Publicly revealing **E** does not mean that it will be easy to compute **D** from **E**. It should be difficult or inefficient to compute **D** from **E**.
4. If **P** is decrypted and then encrypted, **P** is the result. **E**(**D**(**P**)) = **P**.

The encryption and decryption functions rely on a key such that the security of the functions or procedures rests on the security of the key. If **E** satisfies properties 1-3 is referred to as "trap-door one way function". If it also satisfies property 4 then it is referred to as "trap-door one-way permutation". In public key cryptosystems the usual "setup" time is simply the time it takes to make the encryption function public.[1] If Shiela wants to encrypt a message **P**, for Froi, with the key **(e, n)**, she must first represent **P** as an integer between **0** and **n-1**, where **e** and **n** are positive integers. **P** is then encrypted to generate the ciphertext **C** by raising **P** to the e^{th} power modulo **n**. The decryption process will raise the ciphertext to another power **d** modulo **n**. The mathematical formulation is shown below.

$$C \equiv E(P) \equiv P^e (mod\, n)$$

$$P \equiv D(C) \equiv C^d (mod\, n)$$

The the length of the original message is not increased during the encryption process. The pairs **(e,n)** and **(d,n)** are the encryption keys and decryption keys respectively. The encryption key is made public and the decryption key is private to the user. In the above example, **(e,n)** is the public key of Froi, he can decrypt the message using **(d,n)** which is in his possession. The security of the approach is based on the security of the keys, thus the keys must be selected well. **n** is computed as a product of two large random primes **p** and **q**, **n = p * q**. Although **n** will be made public, **p** and **q** will not, which hides the way **d** can

[1] In their paper, it is the encryption function that is made public. Other researchers talk about keys being made public, not the functions. Essentially, however, there is a one-to-one correspondence between the encryption function and the key.

be derived from **e**. **d** is a random integer such that $gcd(d, (p-1)*(q-1)) = 1$. **e** is computed from **p**, **q**, and **d** such that

$$e * d \equiv 1(mod(p-1)*(q-1))$$

This requirement guarantees that properties 1 and 4 above are satisfied, which means that **E** and **D** are inverse permutations. Although Diffie and Helman [6] also used exponentiation and modulo in determining a common key to be used, their approach is not based on a "trap-door one-way permutation" which means that property 4 is satisfied.

The creators of RSA presented an efficient implementation of the of their approach. Note that the basic operations in the encryption and decryption process are exponentiation and division (to compute the remainder). The technique is called "exponentiation by repeated squaring and multiplication." The technique presented will need at most $2 * \log_2(e)$ multiplications and $2 * \log_2(e)$ divisions. The encryption and decryption algorithms are shown below.

```
1  long E(long P,int e, int n)
2  {
3      long C=1;
4      int i;
5
6      for (i=k;k==0;k--)
7      {
8          C = (C * C) % n;
9          if (bit(i,e) == 1) /* bit(i,e) returns the ith bit of e*/
10             C = (C * P) % n;
11     }
12     return C;
13 }
14
15 long D(long C,int d, int n)
16 {
17     long P=1;
18     int i;
19
20     for (i=k;k==0;k--)
21     {
22         P = (P * P) % n;
23         if (bit(i,d) == 1) /* bit(i,d) returns the ith bit of d*/
24             P = (P * C) % n;
25     }
26     return P;
27 }
```

It can be seen that the RSA encryption and decryption algorithms are straightforward to implement. The challenge however is in the selection of the keys which is dependent of several parameters (e, d, n). In their paper, the creators of RSA also presented several approaches how to generate these parameters to guarantee the security.

References

[1] William Stallings. *Cryptography and Network Security: Principles and Practice.* Pearson Education, Inc, 5 edition, 2011.

[2] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 03 1999.

[3] Susan Landau. Standing the test of time: The data encryption standard. *Notices of the AMS*, 47(3):341–349, 2000.

[4] Allam Mousa and Ahmad Hamad. Evaluation of the rc4 algorithm for data encryption. *International Journal of Computer Science and Applications*, 3(2):44–56, 2006.

[5] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, 1978.

[6] Whitfield Diffie and Martin Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[7] Ronald L. Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[8] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *Information Theory, IEEE Transactions on*, 31(4):469–472, July 1985.

[9] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.