Cryptography

A. Antonov, F. Gounelas, J. Kauppila

June 13, 2006

Contents

1 Introduction

1.1 Secure Transmission of Information

Whether we always acknowledge it or not, our daily lives are often very dependent on secure communication of information. Things like credit card payments at Tesco's, online shopping at Amazon, and mobile phone calls to our Mom all require a way to keep the information confidential and correct. So to get a general idea of what cryptography is, it is good to put it into context of general secure transmission of information. There exist three conventional ways to transfer messages securely between two persons:

1. Absolutely secure channel to which no-one else has access to guarantee confidentiality. However, this is usually not achievable in any real-life applications.

2. Steganography tries to hide the fact that a message was even sent in the first place. This is usually implemented by embedding a secret message (for example a piece of text) in a non-suspicious segment of data (for example an image file).

3. Cryptography works by transforming (or encrypting) the input-message into such an obscure form that it is (ideally) only reconstructable by the intended recipient.

We will concentrate on the last of these three ways with the aim of providing a brief account of the *history of cryptography*, a more detailed overview of the different *modern methods* available, a quick look at the *future directions*, and a couple of *case studies* to provide some examples about the applications of cryptography.

2 Overview of Historical Events

There is no doubt that spontaneous desire for the concealment of information has existed ever since the early days of human existence. Then it is not surprising that the first accounts of simple substitution ciphers go back thousands of years. However, over the years much has changed. The never ending battle between the codemaker and the codebreaker has enabled the disciplines of cryptography and cryptanalysis evolve into truly complex fields of mathematical study.

The history of cryptography can be roughly divided into three separate time eras: classical ciphers of the early days, rotor machines of the World Wars, and modern algorithms of the Computer Age. The following will provide an outline of some of the notable events in this race towards the perfect cipher.

2.1 The Classical Era

20th Century BC - First Known Occurence of Cryptography in History

The first accounts of cryptography trace back some 4000 years to the town of Menet Khufu in Egypt. This is the resting place for the tomb of the nobleman Khnumhotep II. The hieroglyphic inscriptions on the tomb were done with a variety of unusual symbols to obscure the meaning of the text.

Figure 1: The hieroglyphic text of Khnumhotep II (http://www.xramp.com).

1st Century BC - First Known Military Substitution Cipher

The *Caesar Shift Cipher* is a simple transposition cipher which works by shifting each letter of the plaintext by three places further down the alphabet. The cipher is named after Julius Caesar, who used it to communicate with his generals. This makes it the first documented substitution cipher used for military purposes. This is a very elementary cipher that offers no real protection anymore.

Figure 2: Illustration of Julius Caesar (http://www.senmerv.com).

9th Century - Discovery of Cryptanalysis by Frequency Analysis

The earliest known description of frequency analysis comes from a 9th century Arabic scientist. The manuscript essentially points out how certain letters of the alphabet occur with different probabilities in our language usage. This was a break-through in cracking simple monoalphabetic substitution ciphers.

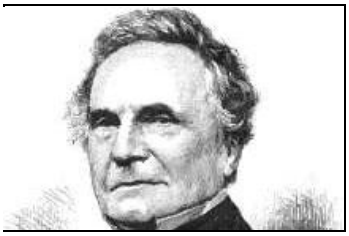
Figure 3: Excerpt of the Arabic manuscript (http://www.simonsingh.net).

16th Century - Vigenère Cipher Emerges as a Polyalphabetic Cipher

Origins of the cipher can be traced to the works of L.Alberti in the 15th century and further works of J.Trithemius and G.Porta. The fully coherent cipher system was published by Blaise de Vigenère in 1586. Essentially it uses a series of different Caesar ciphers based on a mutually known keyword. The choice of shift cipher used for a given letter is determined with the Vigenère square.

Figure 4: Illustration of Blaise de Vigenère (http://www.cqrsoft.com).

1854 - Statistical Analysis for Polyalphabetic Ciphers

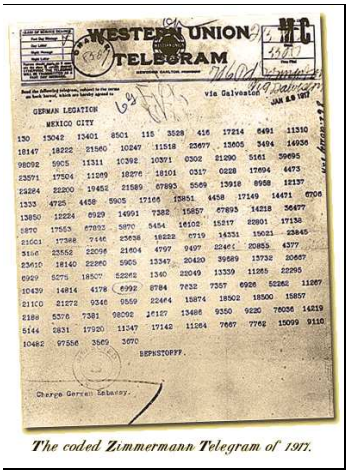


This was originally developed by Charles Babbage but was left unpublished due to the fact it was used by the British to decipher enemy messages in the Crimean War. This was rediscovered and published by F.Kasiski in 1863 and has since been called Kasiski Test.

Figure 5: Illustration of Charles Babbage (http://www-etsi2.ugr.es).

2.2 The World Wars

WWI - Cryptanalysis of the Zimmermann Telegram

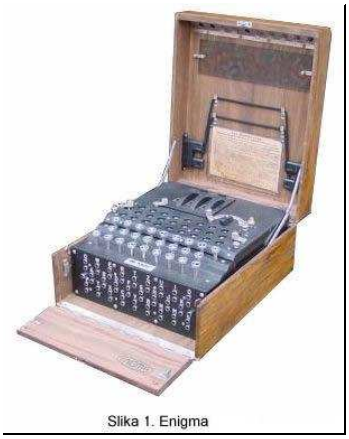


The coded Zimmermann Telegram of 1917.

British cryptanalysis experts deciphered the Zimmermann Telegram which lead the so far neutral US to declare war on Germany. By 1918 Germany had been defeated. Another milestone worth mentioning is the development of the truly secure (but often too impractical) one time letter pad to encrypt top secret information.

Figure 6: The Zimmermann Telegram (http://www.retro-gram.com).

WWII - Cryptography Placed at the Centre of Military Strategy


Slika 1. Enigma

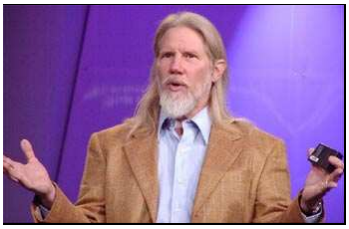
The extent of cryptanalysis research vastly increased in the Second World War. The prior-war works of the polish cryptanalysists helped a great deal in Britain's efforts to crack German Enigma codes. The importance of US deciphering the Japanese top secret diplomatic codes encoded with the Purple machine was also vital. Cryptography was forever placed in the centre of military and political strategy.

Figure 7: The Enigma Machine (http://web.math.hr).

2.3 The Modern Era

1975 - DES cipher was published

1976 - "New Directions in Cryptography" published by Whitfield Diffie



This was the first non-secret publication on the break-through ideas of asymmetric ciphers. The paper descriped the Diffie-Hellman algorithm.

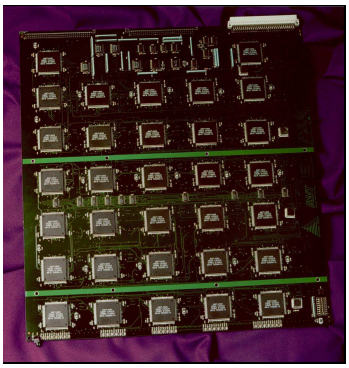
Figure 8: Whitfield Diffie (http://www.at-mix.de).

1977 - RSA cipher published

1991 - Pretty Good Privacy saw daylight
This cryptographic privacy and authentication program was developed by P.Zimmermann. It is offerred as a freeware program. Gnu downloads can be found from here (external link). See more information about PGP at the case-studies page.

1993 - Blowfish algorithm developed by B.Schneier
It was intended as a replacement for the aging DES. The altorithm was placed in the public domain, and thus freely available for anyone's usage.

1998 - DES cracked by brute force



Electronic Frontier Foundation performed a large scale project nick named Deep Crack against the DES symmetric cipher key space. The project amounted to $250,000 in costs, but proved to crack a standard 52-bit DES cryptogram in 56 hours. The system included 1856 custom DES chips and could test over 90 billion keys per second.

Figure 9: The Deep Crack chipboard (http://www.eff.org).

2001 - Rijndael cipher made the official AES algorithm

3 Mathematical Background

3.1 Number Group Cryptography

We here deal with the case where someone chooses a simple number group, such as $\mathbb{Z}_p^*$, to do his cryptography. We will discuss the two main mathematical problems that the secrecy of this type of cryptography is based upon.

3.1.1 The Discrete Logarithm (DLOG) Problem

Definition of the DLOG Problem

Suppose that we have a cyclic group $\mathbb{G}$ and we also know g and g^x, where $g \in \mathbb{G}$ and x is an integer which is less than the order of the group $\mathbb{G}$. The DLOG problem is that of finding x.

We need to stress here the fact that

$$g^x = \underbrace{g \otimes g \ldots \otimes g}_{x \ times}$$

where $\otimes$ is the operation over the group $\mathbb{G}$.

We will later see that this operation $\otimes$, is multiplication modulo a prime number (in the case where we use simple prime-order fields such as in the *Diffie-Hellman* algorithm or *Elgamal* or *RSA*), or elliptic curve addition (in the case of elliptic curve ciphers).

One might consider the same problem over the real numbers $\mathbb{R}$. Here, the solution is very simple as a simple calculator can approximate $x = log_r(r^x)$. Nevertheless, there exists no straight forward method such as this in the case of multiplication in prime order fields (the elliptic curve DLOG problem is covered seperately in the section on Elliptic Curve Cryptography), in which, from a mathematical point of view, we are attempting to solve

$$log_g(g^x) \equiv x(mod\ n) \text{ where } n = |g|$$

There exist various methods that attempt to solve this problem, the most naive of which is simply trying all possible x's, starting from 0 and moving upwards. This method will on average take $\frac{n}{2}$ group operations. On the other hand, the best methods require $O(\sqrt{|\mathbb{G}|})$ where $|\mathbb{G}|$ is the order of the group $\mathbb{G}$ (since the group is cyclic this is also n, the order of the element g). The reader is suggested to resort to [12] for further reading on these methods.

3.1.2 The Integer factorisation (IF) Problem

"The problem of distinguishing prime numbers from composite numbers, and of resolving the latter into their prime factors, is known to be one of the most important and useful in arithmetic"

Gauss wrote in his *Disquisitiones Arithmeticae*, 1801 (as stated in [2]).
It is known from the *Fundamental Theorem of Arithmetic* that every number can be expressed as a product of prime numbers in a unique way. On this note we define the problem as follows

Definition of the IF Problem

Suppose we have an integer N. The Integer factorisation problem is the problem of finding one or more (or all!) prime factors of N.

No solution exists to this problem that will automatically give a result. In modern day cryptography, large numbers are chosen that are the product of two prime numbers p and q, as in this case, the IF problem is harder to solve. The best algorithm is the *general number field sieve* (GNFS) in which the time complexity is $O(exp((\frac{64}{9}n)^{\frac{1}{3}}(log(b))^{\frac{2}{3}}))$ which is super-polynomial, but sub-exponential.
As an example, the German Federal Agency for Information Technology Security (**BSI**) ran a super computer equipped with 80 AMD Operon proccesors for several months before factorising an 193 digit (640-bits) integer into two primes! (See [24])

Even though there exists no method of factorising a large n-bit number in polynomial time, in 1994, Peter Schor established an algorithm that would work on a large quantum computer, that would take only $O((log(n))^3)$ time on $O(log(n))$ space.

3.2 Elliptic Curve Cryptography

Apart from standard groups such as $\mathbb{Z}_p^*$ one can use groups generated by elliptic curves. We by introducing the maths behind elliptic curves.

3.2.1 Mathematical Foundations

An elliptic curve in the context related to cryptography is defined as the following function

Definition of an Elliptic Curve

$\varepsilon : y^2 = x^3 + ax + b$ where $x, a, b \in \mathbb{F}$

In the previous, $\mathbb{F}$ is a field. Having chosen $\mathbb{F} = \mathbb{R}$ and for example $a = -6$ and $b = 6$ we get the following graph

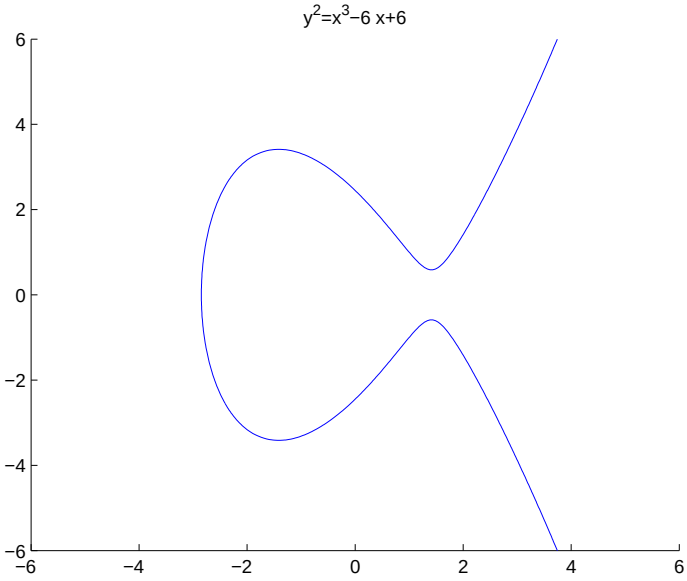


Figure 10: An Elliptic Curve Over $\mathbb{R}$

Nevertheless, the application of elliptic curves in cryptography is only usefull when $\mathbb{F}$ is a field of the form $\mathbb{Z}_p$ when p is a prime number. This relies on the fact that when this condition holds and furthermore $4a^3 + 27b^2 \neq 0$, it can be proven that the elements (x, y) together with the point $\varnothing$ form a group $\mathbb{E}$ under addition. Addition in this group is to be defined geometrically in the following section.
However, as with $\mathbb{Z}_p^*$, the multiplicative group of order $p-2$ where p is prime, we need to stress the meaning of a^k where a is an element of our group and $k \in \mathbb{Z}$, as this will be used in further sections. In $\mathbb{Z}_p^*$ this is defined as

$$a^k = \underbrace{a * \ldots * a}_{k \text{ times}} \tag{1}$$

whereas, in $\mathbb{E}$ it is

$$a^k = \underbrace{a + \ldots + a}_{k \text{ times}} = k * a \tag{2}$$

3.2.2 Addition in $\mathbb{E}$

We describe the case where $\mathbb{F} = \mathbb{R}$. Given points P and Q, it is easily proven that the line through P and Q , call it l_{PQ}, cuts our elliptic curve ε at exactly one more point, which for now we will call $-R$. This is obvious should one consider the fact that l_{PQ} is a linear equation, and the elliptic curve ε is a cubic in x, hence it has 3 roots (Remember: since P and Q are points on the curve and they are solutions of the system between l_{PQ} and ε then this implies that R is a real root and not a complex one, as they go in pairs).

Definition of Addition in $\mathbb{E}$

If P and $Q \in \mathbb{E}$ then $P + Q = R$ where $R \in \mathbb{E}$ is as defined previously.

An example of addition is given in the figure below.

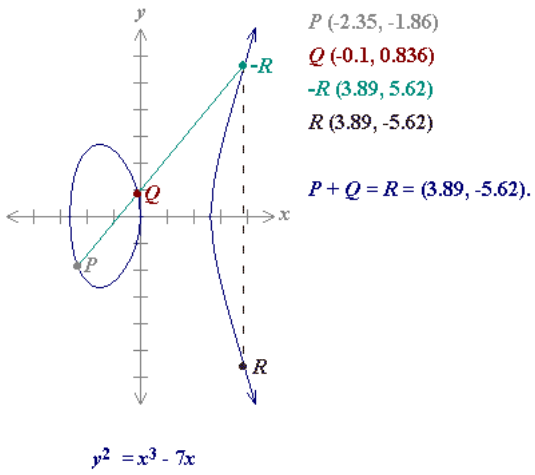


Figure 11: Addition in the Elliptic Curve Group $\mathbb{E}$ over $\mathbb{R}$
Image courtesy of the certicom.com tutorial, see [1] ¿]

3.2.3 The Elliptic Curve Discrete Logarithm (ECDLOG) Problem

This is in some ways similar to the normal DLOG problem, but proves itself more complex to solve. We can define this problem as follows

Definition of the ECDLOG Problem

The ECDLOG problem involves finding $x \in \mathbb{Z}$ if A and $B = xA$ are given, where $A, B \in \mathbb{E}$, are points of an elliptic curve.

Solving this problem is not as simple as one might think. We will spare the reader from explaining in detail complex brute force methods (i.e. trying different x's chosen randomly) or in the case of *supersingular elliptic curves* (i.e curves of the form $y^2 = x^3 + ax$ with some further rescrictions about a) the method of imbedding the group $\mathbb{E}$ into the finite multiplicative group of a field $\mathbb{F}_{q^k}$ and then solving the DLOG problem on $\mathbb{F}_{q^k}$ (See [4] for further reading).
Nevertheless we provide an example of an elliptic curve which makes one's life hard when trying to solve the ECDLOG problem on it.

According to the National Institution of Standards and Technology (**NIST P-192**) A choice of:

$p = 6277101735386680763835789423207666416083908700390324961279$

in the following elliptic curve

$y^2 = x^3 - 3x + 2455155546008943817740293915197451784769108058161191238065$ over $\mathbb{Z}_p$

respresents a good safe choice. To brute force a key based on this elliptic curve one would need to perform between $3 * 10^{57}$ and $6 * 10^{57}$ elliptic curve additions! (Numbers courtesy of http://www.certicom.com)

4 Symmetric-key Cryptography

Since the beggining of cryptographic schemes and up until the creation of public-key cryptography in 1976, any form of cipher, was a symmetric-key one. They are based on the fact that there is one key, or method, with which plaintext or ciphertext is encrypted or decrypted. It has been since the beginning obvious though, that this key must not be publicly known. Therefor, maintaining the secrecy of the key gave birth to the *Key distribution Problem* described in a glossary.

Considering an encrypting function E_e and also a decrypting function D_d where e and d are keys, symmetric-key cryptography requires that one can computer e from d with little or no computation. Typically in the past though, we have had cases where $e = d$ such as the *Vigenere Cipher*. Alternatively, the *Caesar Cipher* had $e = 3 = -d$. Consider the following figure, in which c is the ciphertext and m is the plaintext. Adversary plays the role of Eve the evesdropper.

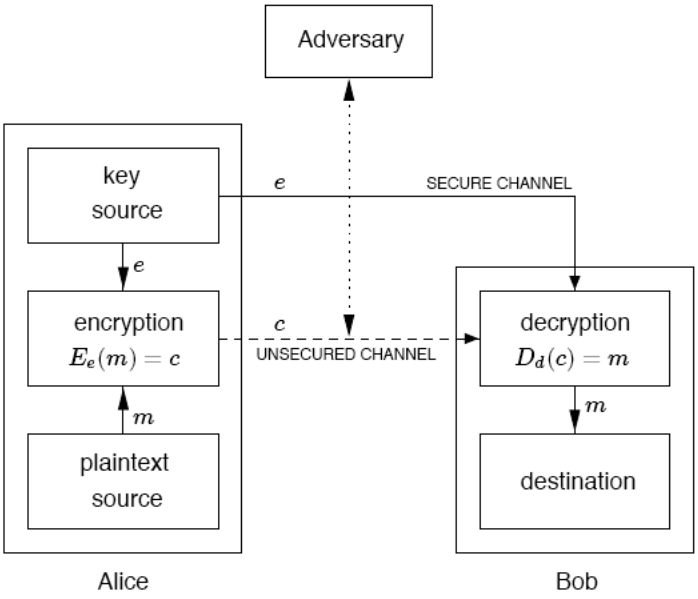


Figure 12: A two party encrypted communication scheme
Courtesy of the Handbook of Applied Cryptography [6]

Symmetric-key ciphers are split into categories, such as **Permutation Ciphers**, where the ciphertext can be a letter permutation of the plaintext. Also, **Transposition Ciphers** and **Substitution Ciphers** where the ciphertext characters represent a shift, or a substritution of the plaintext characters. There exist other, but also combinations of the above.

Since the birth of public-key cryptography though, the distribution of the key is almost no issue at all (see intro on public-key cryptography for details), and so nowadays, the main focus of symmetric cryptography is efficiency and difficulty to be cracked, should one not have the key in his possesion.

In the following sections we review some of the main symmetric-key cryptography schemes: the Data Encryption Standard (**DES**), the Advanced

Encryption Standard (**AES**), and **Blowfish**.

4.1 The Data Encryption Standard

The **Data Encryption Standard (DES)** represents the first modern symmetric cipher. Developed in the US in the late 1970's, it is today hardly usable on its own, since with current computers it can be cracked by brute force. Nevertheless, some still use *triple-DES*, which as implied by the name, is a three-key *DES* encryption. The *DES* algorithm is based on the *Feistel Network*.

DES uses 56-bit keys, which allegedly was the result of IBM being coerced to downgrade their original 128-bit keys, so that the NSA could read information encrypted with *DES*. *DES* is a block cipher and uses input and output blocks of 64-bits.

4.1.1 The Algorithm

The algorithm for encryption and decryption is described below.

The DES Algorithm

1. Apply the permutation IP (see below) to the 64-bit plaintext or ciphertext m. Split $m = (L_0, R_0)$. Henceforth the L_i, R_i are 32-bits long.

2. for $i = 1$ until 16
   $L_i = R_{i-1}$
   $R_i = L_{i-1}\ XOR\ f(R_{i-1}, k_i)$
   end

3. Finally, the block (L_{16}, R_{16}) is passed through IP^{-1} and hence the result is $c = IP^{-1}(L_{16}, R_{16})$.

The f function first expands R_{i-1} from 32 to 48 bits by applying the expansion permutation E (see below) and then performs the XOR operation between the expanded R_{i-1} and k_i, which is a 48-bit substring of our 56-bit key k. This k_i is generated by the key scheduling algorithm, which splits k into two 28-bit pieces and applies rotations to generate a 48-bit substring. Continuing from the XOR operation, our function f then splits this result into eight 6-bit blocks and permutes each of those using 8 known permutations called *S-boxes* (see below). The *S-boxes* have as input a 6-bit block and as output – a 4-bit block. Hence, the result of the f function is $8 * 4 = 32$ bits.

4.1.2 Remarks on the algorithm

- The permutations E, IP and the 8 *S-boxes* are publicly know permutations which can be found at [21]

- The main part of the algorithm is when the key k_i is XOR'ed (in step 2) with part of the ciphertext. This provides the essential crypto-security.
- Other points to notice are the different permutations which make it difficult for an attacker to reverse the process without knowing the key.

4.1.3 Security of DES

The main problem with the **DES** encryption cipher is that it uses a key of short length. This makes it prone to brute force attacks – attacks where the attacker tries every possible key in the 56-bit range. During the 80's and 90's special chip-boards were developed at very high prices that could crack a *DES* encrypted ciphertext in approximately a day. Nowadays, it would not take long for someone to do this on a home computer.

4.2 Blowfish

Blowfish is the product of *Bruce Schneier*, a major cryptographer of our time. It was developed in 1993 as a potential replacement for *DES*. Unfortunately, this never happened, as *Twofish* and *AES* are considered better replacements due to the fact that they use bigger blocks. Nevertheless, it is unpatented and fully in the public domain. It has been applied in many secure software packages that are used today.

Blowfish is a symmetric block cipher that uses 64-bit blocks. Its keys can be of length between 32 and 448 bits. Like *DES*, *Blowfish* is based on the *Feistel Network* scheme, and, also like *DES*, it uses 16 rounds.

4.2.1 Algorithm

The algorithm for encryption and decryption is outlined below.

The Blowfish Algorithm

1. Initialise the P_i $\forall i = 1..18$ which are generated by the key in a specific (known) way.
2. Split the text m into two 32-bit halves, (m_1, m_2).
3. for $i = 1$ to 16
   $m_1 = m_1$ XOR P_i
   $m_2 = f(m_1)$ XOR m_2
   $swap(m_1, m_2)$
   end for

   $swap(m_1, m_2)$
   $m_2 = m_2$ XOR P_{17}
   $m_1 = m_1$ XOR P_{18}
   $c = (m_1, m_2)$

The f function is defined as follows:
Split $m1$ into four 8-bit pieces (a, b, c, d), then

$$f = ((((S1a + S2b) \bmod 2^{32}) \text{ XOR } S3c) + S4d) \bmod 2^{32}$$

where the S_i are the *S-boxes*, generated by the key k, in contrast to *DES* where the *S-boxes* are known.

4.2.2 Remarks on the algorithm

- The block m is 64-bits long
- The key k has size $L \in [32, 448]$
- The swap function simply swaps the values of m_1 and m_2
- Notice how similar it is to the *DES* algorithm. This structure is derived from the *Feistel Network* scheme.

4.2.3 Security of Blowfish

No official way of cryptanalysing Blowfish exists. Despite this, Chloe in the TV series *24* mentions in one episode of season 4 that the NSA has in its hands a proprietary algorithm that cracks Blowfish in no time at all! Although this is highly unlikely, it should not be ruled out (remember, the NSA does spend alot of time on stuff like this!).

Nevertheless, *John Kelsey* managed to crack 3-round *Blowfish*, but could not extend his method it to the neccessary 16 rounds.

4.3 The Advanced Encryption Standard

In 2000 the *Rijndael* cipher was chosen as the new **Advanced Encryption Standard (AES)** after a competition. Another notable candidate for *AES* was *Twofish*, an evolved version of *Blowfish*.

Interestingly, *AES* using 192 or 256-bit keys was approved by the NSA for TOP SECRET U.S. Government information. This is the first time that a publicly available cipher has been approved for such use.

The *Rijndael* cipher was developed by two Belgian cryptographers, *Daemen* and *Rijmen*. It uses keys of either 128, 192, or 256 bits. This is a major improvement on *DES* which uses 56-bit keys. *Rijndael* is based on a *substitution-permutation network* in contrast to *DES* being a *Feistel network*.

4.3.1 Algorithm

The algorithm for encryption and decryption is outlined below.

The AES Algorithm

1. $InputBlock$ is split up into bytes depending on its size L (see below), ex for $L = 128$ into 16 bytes, $m_0, m_1, ..., m_n$ and the same is done to $InputKey$ which is of the same size, $k_0, k_1, ..., k_n$.

2. According to the key size, a specific number of rounds of the following function are performed (for example, when $L = 128$ we have 10 rounds).
   $Round(S, RoundKey)$ Where S is initialy $InputBlock$ and $RoundKey$ is derived from $InputKey$ via *key scheduling*.

3. $Round(S, RoundKey) =$
   $SubBytes(S)$;
   $ShiftRows(S)$;
   $MixColumns(S)$;
   $AddRoundKey(S, RoundKey)$;

We include a brief description of the operations each of these internal functions perform

Subbytes(S) Performs $y_i = Ax^{-1} + b$ where x is every byte of S, A is an *S-box* and b is known dependant on the size of A (and subsequently on the size of m_i).

ShiftRows(S) This is the simplest operation of the four which simply shifts the elements of the matrix whose elements are the bytes of S by a given number of positions.
For example, with $L = 128$, $ShiftRows(S)$ would look like this

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

MixColumns(S) Works by manipulating the columns of the state S (remember they look like the above). It forms a polynomial with coefficients the byte values of the columns. This polynomial is then multiplied with a fixed polynomial $c(x)$ modulo $x^4 - 1$. Modulo arithmetic ensures the result to be a polynomial of degree 3. The resulting polynomials's coefficients will be the column of the new state S'. We work through all the columns of S in a similar fashion to form S'.

AddRoundKey(S, RoundKey) This function performs the XOR operation between the elements of S and the elements of $RoundKey$.

For further information consult [5] and [15] .

4.3.2 Remarks on the algorithm

- $InputBlock$ and $InputKey$ are of the same size L and can be 128,192, or 256 bits long.
- In each round, the state S is updated by the four internal functions.
- Note that all operations in the *Round* function are reversible. To decrypt text one clearly needs to follow the sequence:
  $AddRoundKey(S, RoundKey)^{-1}$;
  $MixColumns(S)^{-1}$;
  $ShiftRows(S)^{-1}$;
  $SubBytes(S)^{-1}$;
- Most of these operations are trivial computationaly. This is another advantage of the Rijndael cipher – encrypting and decrypting text while knowing the key can be done fast.

4.3.3 Security of AES

AES can be attacked using the **Timing Analysis Attack**. This occurs when Malice (the malicious Alice) runs the *SubBytes* method on different data and observes the time it takes for each execution. In doing this, Malice can potentially decide whether the operation was done on bit 1 or 0! This attack method can be prevented by using the *"table-lookup"* method, which relies on keeping 256 (2^8) pairs of bytes (x, x^{-1}), thus making looking up a value when encrypting or decrypting very easy. In this way all conversions take approximately the same time, giving no useful information to Malice [5] .

4.4 Modes of Operating on Block Ciphers

When dealing with block ciphers, one needs to consider whether each block will be encrypted on its own, or whether the encryption of one block will depend on the encryption of the rest in some way. Here we present the most popular block cipher operation modes.

4.4.1 Notation

P_i**:** a block of plaintext

C_i**:** a block of ciphertext

$E(P_i)$**:** a function that encrypts block P_i

$D(C_i)$**:** a function that decrypts block C_i

IV**:** the initial vector, a vector that is used in different modes and is not secret; it is distributed with the ciphertext and should be random for better results

$LSB_n(k)$**:** gives the n least significant bits of k (i.e., the rightmost n bits)

$MSB_n(k)$**:** gives the n most significant bits of k (i.e., the leftmost n bits)

$\|$ *operator***:** the padding operator, e.g., $01 \| 00 = 0100$

4.4.2 Electronic Code Book (ECB)

The simplest way of applying a cipher to a set of blocks is to apply the cipher to each block seperately. In other words:

ECB Algorithm

- ECB Encryption $C_i = E(P_i) \qquad i = 1, \ldots, m$
- ECB Decryption $P_i = D(C_i) \qquad i = 1, \ldots, m$

However, ECB is hardly the safest mode. This is well illustrated in the following image sequence.

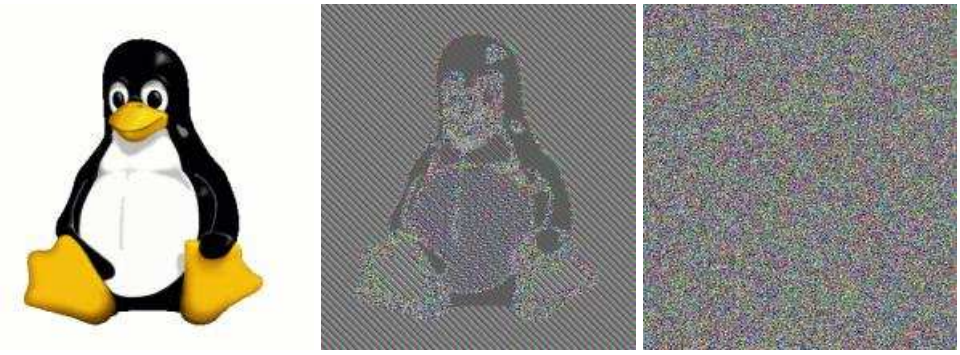


Figure 13: Plaintext image, ciphertext image with ECB, ciphertext image with another mode

Images courtesy of wikipedia and Larry Ewing and The GIMP

4.4.3 Cipher Block Chaining (CBC)

In CBC, each block depends on the encryption of the previous block. We set C_0 to be an initial vector (IV) that will be distributed along with the ciphertext. This IV does not need to be secret, but it should be randomly chosen. The algorithm is as follows:

CBC Algorithm

- CBC Encryption
  $C_0 = IV$
  $C_i = E(P_i \text{ XOR } C_{i-1}), \qquad i = 1, \ldots, m$
- CBC Decryption
  $C_0 = IV$
  $P_i = D(C_i) \text{ XOR } C_{i-1}, \qquad i = 1, \ldots, m$

Hence, when Alice needs to encrypt blocks P_i, she chooses an IV, applies the algorithm and then sends the C_i and IV (the IV need not be encrypted in any way) to Bob, who can apply CBC Decryption.

A drawback of CBC was observed by *Lars Ramklide Knudsen* who showed that

$$C_i = C_j \implies C_{i-1}(XOR)C_{j-1} = P_i(XOR)P_j$$

Attacks using this equation can be limited if a random IV is chosen to limit the values for which $C_i = C_j$, $i \neq j$.

4.4.4 Cipher Feedback Mode (CFB)

In CFB, each ciphertext C_i with $i \in (0, \ldots, m)$ is used to encrypt the next block P_{i+1}. The way it is used though is quite different from CBC. Assume IV is of length n and that we have m plaintext messages P_i of size s. Notice how E is used in both the encryption and the decryption process. The algorithm is as follows

CFB Algorithm

- CFB Encryption
  $I_1 = IV$
  for i=1 to i=m
    if $i \geq 2$
      $I_i = LSB_{n-s}(I_{i-1}) \parallel C_{i-1}$
    end if
    $O_i = E(I_i)$
    $C_i = P_i$ XOR $MSB_s(O_i)$
  end for

- CFB Decryption
  $I_1 = IV$
  for i=1 to i=m
    if $i \geq 2$
      $I_i = LSB_{n-s}(I_{i-1}) \parallel C_{i-1}$
    end if
    $O_i = E(I_i)$
    $P_i = C_i$ XOR $MSB_s(O_i)$
  end for

4.4.5 Other Modes of Operation

Information about other modes can be found by consulting the bibliography (See [5]). Other modes of interest include Output Feedback Mode (**OFB**), Counter Mode (**CTR**).

5 Public-key Cryptography

Until the invention of public-key cryptography in the 1970s, one fundamental concept underlay all of cryptography – the parties that wished to communicate in private had to agree on a shared secret key. The key could be used to both encrypt and decrypt the message. The security of information transferred in this way depended as much on keeping the encryption algorithm secret as on keeping the key secret. Consequently, one of the main cryptographic problems was secure key distribution.

Public-key (asymmetric) cryptography avoids this problem by using separate keys for encryption and decryption. Each party has a key pair: a public key e and a corresponding private key d. The public key is published and thus viewable by anyone. Alice's public key, for example, can be used by anyone to encrypt a message m to her. Thus the encrypted message $c = E_e(m)$ where E_e is the encryption transformation with key e. The message can be decrypted using Alice's private key only, which is not known to anyone else: $m = D_d(c)$ where D_d is the corresponding decryption transformation.

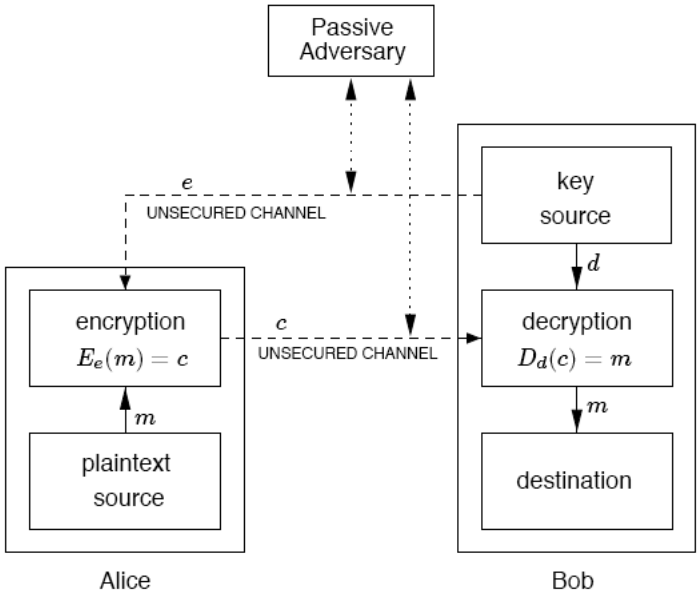


Figure 14: Public-key cryptography
Courtesy of the Handbook of Applied Cryptography [6]

The security of a public-key scheme relies on the property that given the encryption key e, it is computationally unfeasible to deduce the decryption key d. However, the key asymmetry does not automatically solve the key-distribution problem since the encryption key may not be authentic. The public encryption key is transmitted over an insecure channel, and so it is possible for a third party to intercept and replace it with another key, thus performing the so-called impersonation attack. This can be solved by signing one's public key using one's private key. This works since the encryption and decryption transformations satisfy $D_d(E_e(m)) = E_e(D_d(m)) = m$.

The main advantage of public-key encryption systems lies in the fact that ensuring the authenticity of public keys is easier than ensuring secret keys are

distributed securely. However, such encryption is also usually much slower than a comparable symmetric-key scheme and consequently is primarily used for the safe transport of a secret key used to encrypt the bulk of the data using a symmetric algorithm.

In the following sections we review some of the main public-key cryptography schemes: the Diffie-Hellman key exchange, RSA, and ElGamal.

5.1 Diffie-Hellman key exchange

The Diffie-Hellman key exchange protocol was the first cryptographic algorithm to implement the notion of public-key cryptography. The scheme was first published by Whitfield Diffie and Martin Hellman in 1976 and relies on the difficulty of solving the Discrete Logarithm problem (DLOG) in a finite abelian group of large prime order. The Diffie-Hellman key exchange is a non-authenticated protocol since it does not provide authentication of the parties involved. Nevertheless, it is used in TLS (Transport Layer Security), a protocol which provides secure communication on the internet.

5.1.1 The Algorithm

Suppose Alice and Bob want to agree on a shared secret key.

They first establish the domain parameters (shared information): they pick a finite abelian group G_p, where p is prime, such that $p-1$ is divisible by another smaller prime q, at least 160 bits long. Then they choose a generator g for G_q.

Alice picks a non-zero integer a from $\mathbb{Z}_{p-1}$, computes $g_a = g^a \bmod p$ and sends g_a to Bob.

Bob picks a non-zero integer b from $\mathbb{Z}_{p-1}$, computes $g_b = g^b \bmod p$ and sends g_a to Alice.

Alice then computes $k = g_b^a \bmod p$, and Bob computes $g_a^b \bmod p$, which also equals k since $g^{ab} = g^{ba} = k$. And so a shared key has been established.

5.1.2 Elliptic Curve version

This is a modification of the Diffie-Hellman key exchange, which uses an elliptic curve group $\mathbb{E}$ as the finite abelian group instead of $\mathbb{Z}_p^*$.

First, Alice and Bob meet and choose a field $\mathbb{F}_q$ (where $q = p^r$ where p is prime and r can be any possitive integer). They then decide on a good elliptic curve (for example, NIST P-192 given in the section below). They now have to choose a random point B, called a base, which is an element of $\mathbb{E}$. B is not kept secret.

Alice then chooses a large integer $a < q$ which she keeps secret. She publishes aB. Similarly, Bob chooses an integer b and publishes bB.

Alice now needs to calculate abB, and Bob calculates baB. Since $abB = baB = k$, a shared key has been established.

5.1.3 Security of Diffie-Hellman

As described, the Diffie-Hellman protocol is vulnerable to a man-in-the-middle attack since there is no authentication of the parties. Thus an eavesdropper may establish two separate keys with Alice and Bob and then pre-

tend to be either of them to the other. However, once that is amended, the protocol is quite secure since to find the key, the eavesdropper would have to solve the DLOG problem which is still considered hard to do.

The elliptic curve version of the protocol is much more secure than the standard one. A 160-bit key in the elliptic version provides as much security as a 1024-bit key in the normal version. The same methods that are used to cryptanalyse the standard version apply; however, they are less efficient here.

5.2 RSA

RSA was the first fully developed public-key encryption algorithm. It is easier to understand and implement compared with other public-key algorithms and hence is also the most popular. It was published in 1977 by Ron Rivest, Adi Shamir and Len Adleman at MIT whose last initials gave the algorithm its name. The security of the algorithm relies on the difficulty of factorising large integers quickly (the integers can be anything between 100 and 200 digits long or even more).

RSA is quite common in practice; however, it is not usually used to encrypt entire messages as it is too slow. In hardware, RSA is about 1000 times slower than DES (a symmetric cipher formerly used as an encryption standard). It is more often used to encrypt the session key for a symmetric cipher, which is then used to encrypt the message, or to digitally sign a message or file. The algorithm is currently used in the GNU Privacy Guard software and a range of proprietary solutions offered by the RSA security company.

5.2.1 The Algorithm

Suppose Bertha wants to send a message to Arthur. She must first know Arthur's public key.

Arthur picks two large primes p and q which he keeps secret and computes $N = pq$. Arthur then chooses an encryption exponent e (not secret), such that e and $(p-1)(q-1)$ are coprime, that is, $\gcd(e, (p-1)(q-1)) = 1$. Arthur publishes his public key as the tuple (N, e). He also chooses d, such that $ed = 1 \bmod (p-1)(q-1)$, using the extended Euclidean algorithm. His private key is the triple (d, p, q).

To encrypt her message, Bertha looks up Arthur's public key and encodes her message m, as a non-zero element of $\mathbb{Z}_N$. She can encrypt the message by computing $c = m^e \bmod N$.

Arthur decrypts the message by computing $c^d \bmod N$.

5.2.2 Security of RSA

The security of the algorithm is assumed to rely solely on the difficulty of the integer factorisation problem; however, it has never been mathematically proven that m cannot be calculated without factorising N. Nevertheless, RSA has been cryptanalysed for quite a some time now without finding such way to sidestep the integer factorisation. It seems that RSA will remain secure until a fast method of factoring large integers is discovered. In

recent years new mathematical techniques have been developed that reduce the time needed to factor integers significantly, but they still stop short of being a feasible threat. Further, quantum-computer algorithms have been proposed which could factor integers in polynomial time. However, quantum computers are still only a distant possibility.

On a more practical note, there are some attacks that work against the implementation of RSA, such as chosen ciphertext attacks. In chosen ciphertext attacks, the attacker can choose to have any piece of ciphertext decrypted with the unknown key. For instance, in one such attack the eavesdropper Eve persuades Arthur to sign a specially constructed number y with her private key. $y = r^e c \bmod N$ where c is the encrypted message Eve wants read, and r is any number less than N. When Arthur signs y with her private key d, he send back to Eve $u = y^d \bmod N = (r^e c)^d \bmod N$. Since Eve knows r, she can also compute its inverse mod N. Premultiplying the number Arthur sent back by that inverse, she gets $r^{-1}u \bmod N = r^{-1}(r^e c)^d \bmod N = (r^{-1}r)c^d \bmod N = c^d \bmod N = m$.

Another attack is possible if the RSA algorithm is implemented with a low value for e. This significantly speeds up signature verification and encryption but can be insecure. If $e(e+1)/2$ linearly dependent messages are encrypted using different public keys but the same value of e, it is possible to crack the system. This problem can be solved by padding out messages with random numbers to ensure that $m^e \bmod N \neq m^e$.

Other attacks may be possible if RSA is implemented by giving everyone the same value for N or if d is less than one-quarter the size of N and $e < N$.

5.3 ElGamal

The ElGamal cryptographic algorithm, like the Diffie-Hellman key exchange, is based on the difficulty of solving the Discrete Logarithm problem (DLOG) in a finite abelian group of large prime order. It was invented by the Egyptian-American cryptographer Taher Elgamal and first published in the paper "A public-key cryptosystem and a signature scheme based on discrete logarithms" (1985).

In contrast to RSA, ElGamal is a probabilistic encryption algorithm, which means that when encrypting the same message several times, different ciphertexts will be output. Although this is usually considered a property of public-key encryption schemes, symmetric-key encryption algorithms can have a similar property (e.g., block ciphers used in CBC mode).

A variation of the algorithm (the ElGamal signature scheme) has been used as the basis for the Digital Signature Algorithm (DSA), adopted as a digital signature standard by the American National Institute of Standards and Technology (NIST). The ElGamal algorithm is currently used in GNU Privacy Guard software and recent versions of PGP.

5.3.1 The Algorithm

Suppose Anita wants to send a secret message to Buckley.

They first agree on some shared information: Buckley chooses a finite abelian group G_p, where p is prime, such that $p - 1$ is divisible by another smaller prime q, of size at least 160 bits. Buckley finds a generator of G_q,

which he labels g. He then calculates $h = g^x \bmod p$ and shares hir public key triple (p, q, h) with Anita. Buckley's private key is x, and is chosen to be an element of G_{p-1}. (Anita creates her public key using the same information, except that she chooses her own secret key, say y, and thus has a different value for h.)

Next, Anita needs to encode her message m as a non-zero element of G_p. (If the message is too long to be represented as one element of G_p, it is split into parts, and each is encrypted separately.) Having done that, Anita makes up a random session key k and uses Buckley's public key to compute $c_1 = g^k$ and $c_2 = mh^k$. The encrypted message is then (c_1, c_2), which Anita can now send to Buckley.

Buckley can decrypt the message by computing c_2/c_1^x using his private key. This works since

$$\frac{c_2}{c_1^x} = \frac{mh^k}{g^{xk}} = \frac{mg^{xk}}{g^{xk}} = m.$$

5.3.2 Elliptic Curve version

Anita and Buckley choose a finite field $\mathbb{F}_q$ and an elliptic curve ε over $\mathbb{F}_q$ whose points together with $\varnothing$, the point at infinity, form a group under addition. Then they both choose private keys a and b and a base point $B \in \mathbb{E}$. Finally, they publish aB and bB.

Suppose that Buckley needs to send a message m to Anita. He chooses a random integer k and sends the tuple $T = (kB, m + k(aB))$ (notice how Anita's public key aB is used here).

Anita now knows T and a, her private key. She then multiplies the first element of T with a and subtracts this from the second element of T to obtain the message. In other words, she computes $m + k(aB) - a(kB)$, which equals m.

5.3.3 Security of ElGamal

For an appropriately chosen G_q, the cipher is believed to be secure against a chosen plaintext attack. However, as described, it is susceptible to a chosen ciphertext attack, since given a message m, encoded as (c_1, c_2), we can easily calculate $(c_1, \kappa c_2)$, which is the message κm.

There are further implementation weaknesses. In practice, in order to increase efficiency, the generator g used may be of order much less than p. If this is the case, and the message m being encrypted is not in the group generated by g, $< g >$, an interception attack is possible. This is because under such circumstances ElGamal is transformed from a probabilistic to a deterministic encryption scheme.

6 Case Studies

6.1 Secure Shell (SSH)

Secure Shell is a common network tool that enableds a user to establish a secure link from a local machine to a remote server. It uses asymmetric public-key methods for authentication of the user and symmetric cryptography for the transmission of data.

Like many good things today, SSH was pioneered in Finland. The first version was originally developed by Tatu Ylönen in 1995. At the time he worked as a research associate at the Helsinki University of Technology.

The original implementation was released as a freeware in July 1995. In just under six months the SSH user base had grown to 20,000 users in fifty different countries. Towards the end of the year, Ylönen went to establish a firm of the same name (which distributes these days a proprietary version of SSH). SSH1 is already slightly outdated in term of security levels and has since been replaced by the much more secure SSH2. Some algorithms that SSH2 can use are AES, Arcfour, Blowfish, DES, 3-DES, IDEA, and RSA.

Nowadays there exists numerous different SSH-softwares in the market. To make all these different vendors work together, the Internet Engineering Task Force (IETF) has created a standard for SSH to which everyone has agreed to conform to. The SSH-client program is now a standard tool in any Linux distribution. The binary can be called with *ssh* command. For Windows operating systems a popular freeware program is the PuTTY.

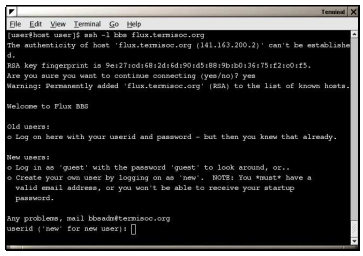


Figure 15: Example of a Secure Shell login session.
http://www.fluxbbs.org/images/linux-ssh.eps

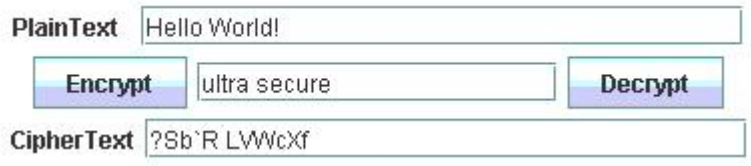
6.2 Vigenere applet

We have written a small applet that simulates an algorithm similar to the well know **Vigenere Cipher**. Originally, the vigenere cipher took a message plaintext with lower case characters only, that is the 26 characters of the

english alphabet. It also took a key word which was used to encrypt the plaintext. At the time, mid sixteenth century, this cipher was considered quite hard to break. Nevertheless it has been broken many times throughout history and clearly today presents no security at all infront of computers.

Our algorithm takes a plaintext string and a key string and recursively adds the ASCII value of the i-th character of the key to the i-th character of the plaintext (modulo so that the ciphertext is still in the ASCII table!). If the algorithm runs out of key string characters, it recurses from the beggining of the key. For example the string "abc" with the key "bc" becomes "bdd" in the vigenere cipher, whereas, in a similar manner, it becomes "EGG" in our applet, as we have included more values of the ASCII table (such as upper-case, numbers and special chars).

Here is an image of the applet in action



You can find the applet at

www.doc.ic.ac.uk/~jik04/crypto

under *casestudies*, but also the actual code for the applet at *www.doc.ic.ac.uk/~jik04/crypto/vigenere_applet/vg.java*.

7 Glossary

Asymmetric-Key Cryptography Is basically public-key cryptography.

Block Cipher This is a chiper that splits the plaintext into blocks (of say 32-bits for example) and then encrypts each block by itself, or uses some algorithm that makes the encryption of block i dependant on blocks *i-1* or something equivalent (See implementation of *ECB*, *CBC*, *OFB*, *CFB*, *CTR*).

Cipher This is the algorithm the encrypts/decrypts plaintext/ciphertext.

CipherText This is the result of encrypting plaintext.

Discrete Logarithm Problem Given an element of a group, find the power n of the generator g such that g to power n is our element.

Elliptic Curve Cryptography Is based on the group $\mathbb{E}$ rather than $\mathbb{Z}_p^*$, and proves to provide a harder to solved *Discrete logarithm Problem*.

Feistel Network A cipher developed by IBM. Many ciphers were based on this scheme. Notably, so was the *Data Encryption Standard* (*DES*).

Integer Factorisation Problem Given a large number calculate its prime factors.

Key Is the heart of the encryption algorithm as it provides the neccessary level of secrecy. Keys can be any length, but nowadays it is recommended that they are over 128-bits so that they are harder to crack.

Key Distribution Problem This is related to *symmetric ciphers*. In this, one must give the key to the other person in secret, as the key is the only thing that lies between the encrypted and the decrypted text.

Key Scheduling Algorithm Is used in ciphers where the encryption/decryption is done in rounds. It simply splits the key in such a way so that it is usable in the rounds.

Monoalphabetic substitution cipher Here the key relies on one alphabetic character, an example is the *Ceasar cipher*.

One time letter pad (OTLP) Is a cipher developed in 1917 that uses a randomly generated key which is the same size as the plaintext. This method is unbreakable if the key is truly random, but the *Key Distribution Problem* still remains.

Plaintext The readable unencrypted text.

Public Key Cryptography The user here has two keys, one of which he keeps private and the other he makes available to his friends or anyone. For the encryption of plaintext, one uses the public key. Whereas for the initial user to decrypt ciphertext which was encrypted using his public key, he has to use his private key. Public key cryptography solves the *Key Distribution Problem*.

Polyalphabetic substitution cipher Here the key relies on more than one characters, an example is the *Vigenere cipher*.

Symmetric-Key Cryptography Here, the encryption key is the decryption key (or they are related in some way). This implies that one needs to be carefull when distributing the key (See *Key Distribution Problem.*).

References

[1] Certicom Inc. *elliptic curve cryptography tutorial*. http://www.certicom.com/index.php?action=ecc_tutorial,home.

[2] Daniel J. Bernstein. Integer Factorization, 2006. http://cr.yp.to/2006-aws/notes-20060309.pdf.

[3] Whitfield Diffie and Martin E. Hellman. *New Directions in Cryptography. IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. http://citeseer.ist.psu.edu/diffie76new.html.

[4] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer, second edition, 1994.

[5] Wenbo Mao. *Modern Cryptography: Theory & Practise*. Hewlett-Packard with Prentice Hall, 2004.

[6] Vanstone Menezes, Oorschot. *The Handbook of Applied Cryptography*. CRC Press, 2001. htp://www.cacr.math.uwaterloo.ca/hac/.

[7] Certicom Research. *Standards for Efficient Cryptography (SEC): Elliptic Curve Cryptography*, 2000.

[8] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley & Sons, Inc, second edition, 1996.

[9] Carlos Serrao. Using Java and Linux to crack the DES challenge, 1999. http://linuxgazette.net/issue46/serrao.html.

[10] Simon Singh. *The Code Book*. Fourth Estate, 1999.

[11] Nigel Smart. *Smart Cryptography*. MacGraw-Hill, 2003.

[12] Chris Studholme. The Discrete Logarithm Problem, 2002. www.cs.toronto.edu/ cvs/dlog/.

[13] Wikipedia. S-box — Wikipedia, The Free Encyclopedia, 2004. http://en.wikipedia.org/w/index.php?title=S-box&oldid=16042734.

[14] Wikipedia. Quantum key distribution — Wikipedia, The Free Encyclopedia, 2005. http://en.wikipedia.org/w/index.php?title=Quantum_key_distribution&oldid=30299911.

[15] Wikipedia. Advanced Encryption Standard — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=56470433.

[16] Wikipedia. Block cipher — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Block_cipher&oldid=52800775.

[17] Wikipedia. Block cipher modes of operation — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Block_cipher_modes_of_operation&oldid=56691604.

[18] Wikipedia. Blowfish (cipher) — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Blowfish_%28cipher%29&oldid=54787101.

[19] Wikipedia. Cryptographic hash function — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Cryptographic_hash_function&oldid=50010187.

[20] Wikipedia. Data Encryption Standard — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Data_Encryption_Standard&oldid=54509269.

[21] Wikipedia. DES supplementary material — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=DES_supplementary_material&oldid=55478387.

[22] Wikipedia. Diffie-Hellman key exchange — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Diffie-Hellman_key_exchange&oldid=56209472.

[23] Wikipedia. ElGamal encryption — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=ElGamal_encryption&oldid=55209306.

[24] Wikipedia. Integer factorization — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Integer_factorization&oldid=57804432.

[25] Wikipedia. Key schedule — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Key_schedule&oldid=55314054.

[26] Wikipedia. RSA — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=RSA&oldid=56122079.

[27] Wikipedia. Symmetric-key algorithm — Wikipedia, The Free Encyclopedia, 2006. http://en.wikipedia.org/w/index.php?title=Symmetric-key_algorithm&oldid=53857741.

[28] V.V. Yaschenko. *Cryptography: An Introduction.* AMS: American Mathematical Society, 2002.

[29] Edward Yin. Curve Selection in Elliptic Curve Cryptography, 2005.

[30] James Ford & Neal E. Young. Quantum Cryptography Tutorial, 1996. www.cs.dartmouth.edu/ jford/crypto.html.