

# SANS Institute
# InfoSec Reading Room



## The mathematics behind the security features that the computing industry takes for granted

Throughout this paper I aim to explain mathematical/encryption concepts that are fundamental to security as it was in the past, as it is today and my vision for the future. I shall explain and in some cases actually prove how this theory is applied. Many computing professionals may well have heard of the RSA or the DES algorithm and may even frequently use GUI's (Graphical User Interfaces) to encrypt/decrypt data but may be unaware of the beauty of the mathematics behind it. Without documents like this, the comparative...

October 14, 2002 Author: **Ricky Wald**

**Reference:** GSEC (GIAC Security Essentials Certification)

# The mathematics behind the security features that the computing industry takes for granted.

Please note that the reader is assumed to have a reasonable background in mathematics and computing.

***Introduction:***

Throughout this paper I aim to explain mathematical/encryption concepts that are fundamental to security as it was in the past, as it is today and my vision for the future. I shall explain and in some cases actually prove how this theory is applied. Many computing professionals may well have heard of the RSA or the DES algorithm and may even frequently use GUI's (Graphical User Interfaces) to encrypt/decrypt data but may be unaware of the beauty of the mathematics behind it. Without documents like this, the comparatively unprofitable world of mathematical cryptography will not receive the human investment needed to push the boundaries of this frontier. I believe that whilst writing about this subject I must pay tribute to the heroes who founded it. Most importantly I will concentrate on a real world example and go from (or at least close to) the axioms taken for granted right through to the final application.

***The need for encryption:***

In the examples that follow please read the decoded plaintext because I have purposely written interesting security information as an incentive to follow through with all the decoding.

Suppose that Alice wants to send secret information to Bob over a public/unsecured network as depicted in the following diagram.

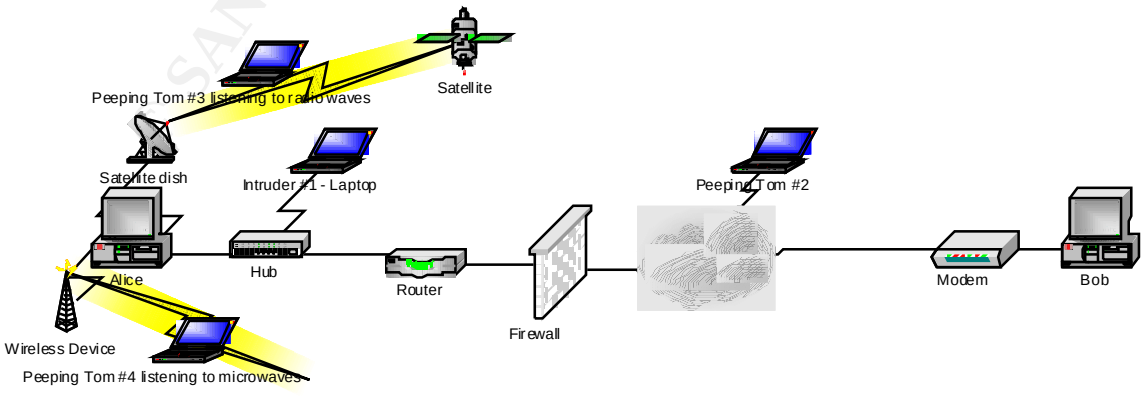



Diagram 1 (Source: Ricky Wald)

Imagine that intruder #1 has broken into Alice's office and has connected to a hub through which Alice's message travels. Suppose that Peeping Tom #2 has simply connected to the Internet and is listening to the data etc sent by Alice to Bob. It is very possible that Peeping Tom #3 is picking up the radio waves (wavelength > 0.1m) being sent by Alice's satellite dish to an orbiting satellite (possibly even with a geostationary setup). Suppose that Alice uses some sort of wireless device and further suppose that Peeping Tom #4 is close enough to listen into the microwaves (0.1m < wavelength < 1mm) caused by Alice's wireless device. Without encrypting the plaintext all the intruders and peeping toms are able to read the sensitive material. My point is that even with firewalls (e.g. the new check point NG firewalls), routers with access control lists (for example standard or extended IP Access Lists using the "access-list", "access-group" and "access-class" commands with recent Cisco routers) and Intrusion Detection systems, privacy of sensitive data/files requires encryption that cannot ever be broken or with an encryption that takes so long to crack, that by the time this occurs the data is no longer sensitive.

In the same way that a building is only as strong as its weakest part, a LAN, MAN or WAN is only as private as its weakest part. The above illustrates how crucial encryption, cryptography etc are despite other security infrastructure, methods and software. In this case a picture speaks a thousand words.

Now that I have convinced readers of the importance of encryption for confidentiality (if not authentication, integrity, non-repudiation and availability with regards to encrypting private keys in PKI using session keys), I will use history as an example to show how previous encryption methods that once were perceived to be secure can be cracked in minutes/hours. These basic examples are intended to scare readers into progressing with today's encryption methods and not assuming them to be unbreakable in the future.

***Breaking a Vigenère Cipher to illustrate how today's usable cipher methods are tomorrows useless cipher methods.***

We translate plaintext into ciphertext using a cipher.

Suppose we use "gsec" as a poly-alphabetic substitution cipher (whose definition will soon become clear). Think of the letters of the alphabet being equivalent to their corresponding numbers between 0 and 25. For the first letter in a plaintext message use the letter "g" as a shift cipher (to code the first letter). For the second letter in a plaintext message use the letter "s" as a shift cipher (to code the second letter in the plaintext message). Similarly do this for "e" and "c". So we have just used the letter "c" to code the fourth letter in the plaintext message. Next use the letter "g" again to code the fifth letter in the plaintext message. Repeat this pattern until the entire plaintext message has been converted to a

ciphertext message. To explain what a shift cipher is, for clarity suppose that the first letter in the plaintext message is the letter "f", which as it is the first letter we will be coding using the letter "g". "f" takes the numeric value 5 (remember we are counting from zero), and "g" takes the numeric value "6". 5+6=11 so the coded letter is "l".

Take the example plaintext "giac stands for global information assurance certification", again using "gsec" as a poly-alphabetic substitution cipher. Lets code the first nine letters to clarify how this works.

| Plaintext | G | I | A | C | S | T | A | N | D |
|---|---|---|---|---|---|---|---|---|---|
| "key" | G | S | E | C | G | S | E | C | G |
| Ciphertext | M | A | E | E | Y | L | E | P | J |

The first row contains nine letters from the plaintext message that we wish to encrypt. In the second row: Under each plaintext letter I have written the corresponding letter from the poly-alphabetic substitution cipher "gsec" that will be used to shift the plaintext letter. The third row contains the ciphertext.

Now that we have gone through the Vigenère method let us examine a piece of ciphertext created using this method and together LETS CRACK IT. I have not been so nice as to use the same "key". Do not even assume the one I will use is the same length as "giac".

Below is the ciphertext:
Jucbr titnq twpdyzv c wgtzqzpfkgcqfz qb cmmp k aigrdmgzqmcjaqb ryiv mye kqxrrqp nyki vinzkcvjp l fsezbcv avzvsdzkcdc r xwljzk mow rvf srj xtstrbg ucp kqelkmtzyib cbc jbqbcu wp k qditd arzf dfv xwijzk mow jpqeju jg ovgwtdystg isk bjo niqxkrv sgi qywwvb ….

Shortly lets try to determine the size of the key by using properties of the English language. For one moment step back from what we are doing. We are as stated going to use properties of the English language to find out about the method of encryption which will allow us to crack the code. This illustrates the value of this paper. Very often to hack, crack or break codes the menace to the security specialist will use some property to learn about the actual information that the menace is after, or to learn about the encryption method used which will in turn enable them to find the actual information that they are looking for. The thing to consider is that when security professionals for example develop an encryption method they have not yet identified the properties that can be used against them. Mr Vigenère had not considered the properties that we are using when he developed his cipher method. Similarly the people who developed and the people that use todays encryption and security tools may be lulled into thinking that what they developed/use is secure and not breakable just because they have not thought of the correct ways to use associated properties. Unfortunately I am also making the point (with my example) to would be menaces that they should look for properties that they can exploit. For a further example consider that it is

possible to identify the make of routers by their response times. You have been warned and the example below should back this up.

In the English language, there are sequences of two letters that occur more often than other sequences of two letters. These two letter patterns that occur with a high probability are called bigrams. Similarly for sequences of three letters the corresponding high probability patterns are called trigrams. The most common bigrams in decreasing order are:
TH, HE, IN, ER, AN, RE, ED, ON, ES, ST, EN, AT, TO, NT, HA (source http://www.cs.bris.ac.uk/~cooper/Cryptography/crypto.html). Think about what happens when we use our cipher on the plaintext. For a moment let us wrongly assume the key is "xyz" for clarity of the point I am about to make. Consider the two letter bigram "re". Every occurrence of "re" will be converted by the key, but with the "r" being shifted with a different letter of the key "xyz". By probability every so often the "r" will be converted by the "x" and thus the "e" will be converted by the "y" (of the key "xyz"). Hence in the ciphertext there will be a two letter pattern corresponding to "r","e" converted by "x" and "y" respectively, which will be distances apart that must be multiples of the size of the key "xyz". Hence if we look for two letter patterns in the ciphertext then find the greatest common divisor of the distances between these ciphertext patterns, we should be left with only a few options for the size of the key. Now disregard any reference to the fictional key "xyz".

| Pattern | Separation |
|---|---|
| Wp | 135 |
| Zv | 76 |
| Qx | 134 |
| Bc | 52, 5 |

In the table above: For bigrams (repeating two letter patterns) that I have noticed in the ciphertext I have written the seperation (measured in characters) between occurrences of these patterns.

[Notational comment: gcd(x,y) means the greatest common divisor of x and y]
Now gcd(135,5)=5, gcd(135,76)=1, gcd(134,52)=2, gcd(134,135)=1, … without loss of generality we can stop here.
Because it is unlikely that the password is 1 or 2 in length (because that would not be particularly secure) let us assume it is of size 5. If we were wrong in our assumption we could always return to this point.

What we have found is that every fifth letter was coded using the same letter from the key. Thus if we split our ciphertext up into the appropriate five groups we will have five groups of ciphertext each of which has been created by using a simple shift cipher which is easily solved. This demonstrates a technique used by hackers, crackers and code breakers, where they turn the problem they are solving into a simpler one without actually solving the problem. I studied mathematics at university and one of the problem solving techniques that my

piers and I used was not to try and solve the actual problem but to get a result that can be used to solve the problem. As simple as this may sound many people in practice, when faced with a new problem do not employ this method. Do not underestimate it and do use this method of attack to find security flaws before they are found by black hats, hackers, crackers or even the occasional grey hat.

I will do the first of the five groups and to save space leave it up to the reader to reiterate my method for the other four groups.
Below is every other fifth letter starting from the first letter in our ciphertext:
Jttztffmigjyerkzpzvzrzrjrpkijudrvzjugskivy (42 letters)
Rearranged this is:
d(1)e(1)ff(2)gg(2)iii(3)jjjjj(5)kkk(3)m(1)pp(2)rrrrr(5)s(1)ttt(3)uu(2)vvv(3)yy(2)zzzzzz (6)

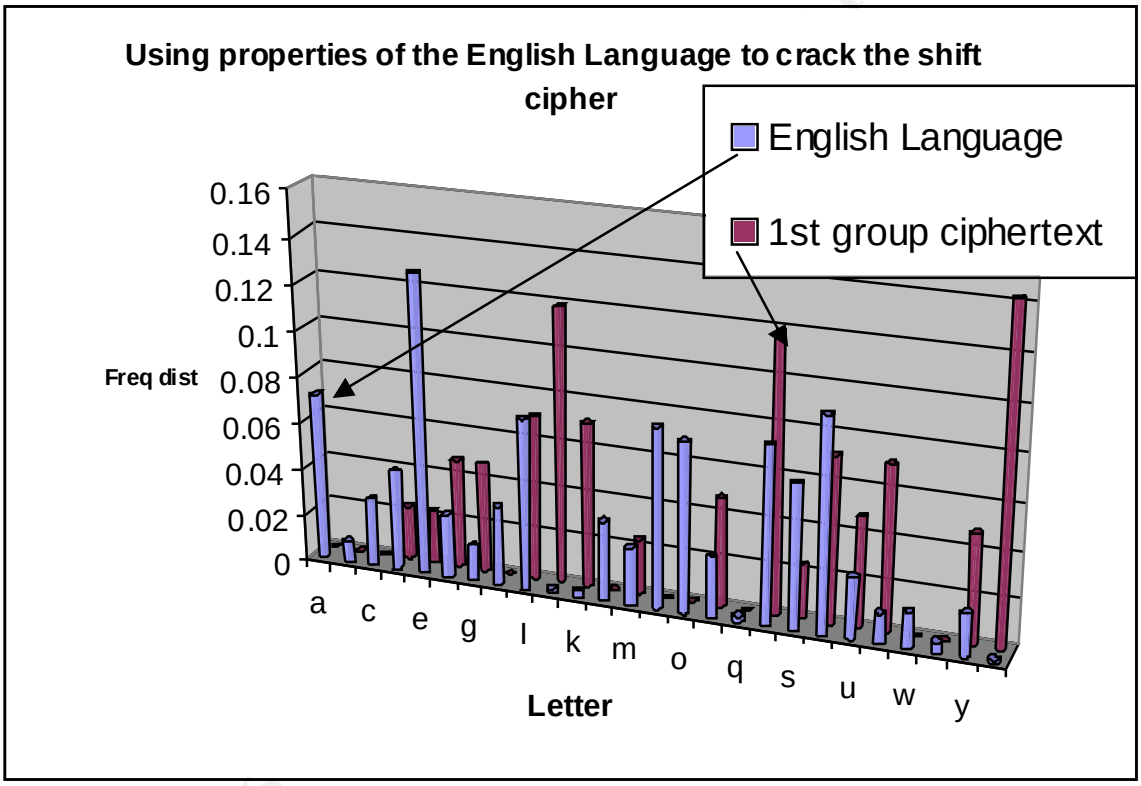
Now lets compare the frequency distribution of the English Language against that of the actual ciphertext for the first group, i.e every fifth letter of the original ciphertext starting with the first letter.

| | Frequency distribution | |
|---|---|---|
| Letter | English Language | 1st group ciphertext |
| a | 0.073 | 0.000 |
| b | 0.009 | 0.000 |
| c | 0.030 | 0.000 |
| d | 0.044 | 0.024 |
| e | 0.130 | 0.024 |
| f | 0.028 | 0.048 |
| g | 0.016 | 0.048 |
| h | 0.035 | 0.000 |
| I | 0.074 | 0.071 |
| j | 0.002 | 0.119 |
| k | 0.003 | 0.071 |
| l | 0.035 | 0.000 |
| m | 0.025 | 0.024 |
| n | 0.078 | 0.000 |
| o | 0.074 | 0.000 |
| p | 0.027 | 0.048 |
| q | 0.003 | 0.000 |
| r | 0.077 | 0.119 |
| s | 0.063 | 0.024 |
| t | 0.093 | 0.071 |
| u | 0.027 | 0.048 |
| v | 0.013 | 0.071 |
| w | 0.016 | 0.000 |
| x | 0.005 | 0.000 |
| y | 0.019 | 0.048 |
| z | 0.001 | 0.143 |

| Total | 1 | 1 |
|---|---|---|

Table 2 (source: The values of the frequency distribution for the English language were from www.trincoll.edu/depts/cpsc/cryptography/caesar.html, but the rest of the data and the table are from Ricky Wald).

In the above table (in red) you can see the frequency distribution for the group of ciphertext that we are examining. To explain how we get this, consider the letter "z". This occurs 6 times out of the 42 characters in its group of ciphertext. To get the frequency distribution, simply divide 6 (the number of occurrences) by 42 (the total number of characters) to result in 0.143=6/42.




Graph 3 (source: Ricky Wald)

All we must do is look at graph 3 and work out how far right we would need to shift the English Language frequency distributions to (as best as possible) match up with the pattern of the frequency distributions of the first group of the ciphertext. Not only should we try to match up individual letters but we should also consider groups of letters, for example the frequency distribution increases as we go from letters "a" to "e" in the English Language. Notice that if we shifted a (eng) to r, then we would have shifted: a (eng) to r, e (eng) to v, t (eng) to k … By carrying on seeing how this would shift each letter, we should see that this provides an excellent match and in fact a better match than other possible shifts. We could mathematically verify this by: For each shift, sum the squares of the difference between the frequency distribution of the English language and the

corresponding frequency distribution of the appropriate ciphertext. We should find that the correct shift results in the lowest value for this sum of squares calculation.

A reader may not be convinced that the two patterns when shifted along by r do in fact match up. My response to that is simply that the ciphertext was short and they should consider the following version of the central limit theorem,

**Theorem:** Suppose that $\{ X_j \}$ are independent, identically distributed random variables with mean • and variance $•^2 < •$. Let $S_n = X_1 + X_2 + ... + X_n$

Then,

$$Z_n := \frac{S_n - n\mu}{\sigma\sqrt{n}} \xrightarrow{Convergernce\_in\_distribution} Z \sim N(0,1) \quad \text{as } n \to \infty$$

Note that N(0,1) is the standard normal distribution with mean 0 and standard deviation 1.

Outline Proof: Find the characteristic function of $Z_n$ by expanding the exponential function in a taylor series, then look at the limit of this as $n \to \infty$. Then use uniqueness theorems about characteristic functions. • A full proof can be found in most degree level probability books.

Basically what this theorem means is that the larger the length of the plaintext/ciphertext the more accurately the ciphertext will (when shifted) match the distribution of the English language.

We have cracked the first letter of the poly-alphabetic substitution cipher, (we found it to be "r"). I leave it up to the reader to repeat this for the other four letters to find that the poly-alphabetic substitution cipher is in fact "RICKY". With this password we can easily convert the ciphertext back to the original plaintext message:

"Smart cards contain a microprocessor (or even a cryptoprocessor) that can contain data. Typically a digital certificate, a public key and its private key counterpart are stored on a smart card. The public key should be exportable but the private key should…"

The rest of the plaintext message is:

"be kept securely on the card. There is even a device called the si shell which consists of a wafer of silicon covering the chip, so that if I villain tries to use light/an electron microscope to discover the contents of the chip, then the chip will be destroyed by this layer of silicon."

(Note that I have added punctuation into the plaintext message above.)

***A currently used encryption method (DES – Data Encryption Standard) that today is thought by many to be secure enough to be widely used in industry.***

I want to explain this encryption method for the knowledge itself, for the method used and mainly because the other explanations that I have found are confusing or inaccurate or simply because they do not explain the topic in a computational mathematics setting.

Below I shall first explain the tools that we are going to need, then I shall put all these pieces together to explain how to use DES to go from a **plaintext** message to the **ciphertext**. We will be using a 64 bit **key (denoted K)**. The tools that I am referring to are definitions, methods that can be explained before the meat of the algorithm (analogous to lemmas) and notation. [Please see the wider picture and apply this method of first writing down the "tools" before trying to understand quantitative security concepts. Again this is something that is not difficult to comprehend but is often not followed by experienced practitioners, even though is makes reading the actual explanation infinitely easier to follow and it makes it easier to reuse the "tools" elsewhere.]

***Text, its halves and the initial permutation***

Plaintext = initial 64 bit block of plaintext that we intend to encrypt

Below is the table for the IP, (whose use will be explained at the very end).

| IP: Initial Permutation | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Bit** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1** | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| **9** | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| **17** | 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| **25** | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |
| **33** | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| **41** | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| **49** | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| **57** | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

Table 4 (Source: http://www.tropsoft.com/strongenc/des.htm)

For the duration of this paragraph let us use the notation:
A is a string of text that has not undergone any permutations.

B is the string of text found by performing the permutation IP on A, i.e. B = IP(A) For clarity consider the 7th bit in A. By looking at table 4 we see that this is the 64th entry in the table. Thus what was the 7th bit in A will be permuted to the 64th bit of B. Similarly Consider the 60th bit in A. By looking at table 4 we see that this is the 9th (because exactly the eight numbers 58, 50, 42, 34, 26, 18, 10, 2 precede the number 60 in the table) entry in the table. Thus what was the 60th bit in A will be permuted to the 9th bit of B. Similarly for all the 64 bits that make up A.

This table has its corresponding inverse which is denoted $IP^{-1}$

We will be splitting our plaintext message in half, labelling these halves $L_i$ and $R_i$ respectively. We will be swapping these halves in an iterative method for $i = 1,2,3,...,16$.

***Notation*** (that will be used in the following explanation):

$L_0$ (32 bit block) = left half of the initial 64 bit block of the plaintext after it has been initially permuted.

$R_0$ (32 bit block) = right half of the initial 64 bit block of the plaintext after it has been initially permuted.

***Rounds/Iterations***

$i^{th} round = i^{th}$ iteration as described just below.

$L_i := R_{i-1}$ for $1 \le i \le 16$

$R_i := L_{i-1} \oplus f(R_{i-1}, K_i)$ for $1 \le i \le 16$ (Both the function f and the subkeys $K_i$ are described below)

Further Notation

$R_i$ ':= $R_i$ once it has been expanded during a particular application of the function f.

$R_i$ '':= $R_i$ ' once it has been XORed during a particular application of the function f.

$R_i$ ''':= $R_i$ '' once it has been S-Boxed during a particular application of the function f.

This symbol "$\oplus$" denotes the XOR operation. There is no standard notation for this meaning. At least that is what several professors have told me. To clarify its meaning, consider the example $A \oplus B$

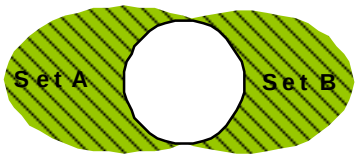



Diagram 5 (source: Ricky Wald.)

$A \oplus B$ is all the shaded area. I.e. A or B, but not the intersection. Alternatively this operation can be thought of by looking at the resulting truth table:

| x | y | $x \oplus y$ |
|---|---|---|
| True 1 | True 1 | False 0 |
| True 1 | False 0 | True 1 |
| False 0 | True 1 | True 1 |
| False 0 | False 0 | False 0 |

What this truth table is telling us, is that given two binary digits, the XOR of them is only 1 (or true) when exactly one (not both) of the two numbers is 1 (or true).

To clarify its meaning in this encryption algorithm lets XOR the number 11001010… with the subkey 10100001

| number | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 … |
|---|---|---|---|---|---|---|---|---|
| Subkey used for XORing | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 … |

The resulting number is: **0 1 1 0 1 0 1 1**

***Key and the 16 subkeys derived from it*** (whose use will be explained later)

The encryption key (K) will be used to generate "subkeys" $K_i$ that will be used in the rounds/iterations.

K := Key (64 bits) = {56 bits determining how the transformations of the text is to be carried out} $\cup$ {8 bits used for parity}

Rotate by one left, means to cycle each digit along anti-clockwise by one place. For example consider the sequence 2,3,4,5. Rotating this by one results in: 3,4,5,2. Doing another rotation by one left, will result in: 4,5,2,3.

$K_i = i^{th} subkey =$ made by:

- Take the Key (64 bits)
- Apply a permutation called the Permuted Choice 1 (PC-1) to the (64 bit) Key
- Remove 8 partition bits leaving only 56 bits of K
- Split these 56 bits in half leaving two sets of 28 bits
- Rotate each half one or two bits (depending on $i$)
- Stick the two 28 bit halves back together
- You now have 56 bits
- Apply a permutation called the Permuted Choice 2 (PC-2) to these 56 bits
- Choose 48 of these 56 bits using a defined method
- You now have the required subkey $K_i$ (48 bits)

***Iterative function***

f = f($R_{i-1}, K_i$): $\{R_i \mid i \in [0,15]\}$ x $\{K_i \mid i \in [1,16]\} \rightarrow$ {a subset of the initial plaintext}

= the function used to determine the iterative sequence $R_i$

In the text box below we describe the function f and describe how to get from the input variables $R_{i-1}, K_i$ to the value of the function f($R_{i-1}, K_i$).

***A) Expansion of the 32 bit right half of text to 48 bits***

Expansion = expansion permutation done on the right half $R_i$ $(0 \le i \le 15)$ of the text, such that the size of the half is increased from 32 bits to 48 bits. [Notice 48 bits is the same size as the subkeys $K_i$.]

Figure 6 (source: http://www.cs.bris.ac.uk/~cooper/Cryptography/crypto.html)

For the next paragraph I am expecting readers to constantly be referring to figure 6 above.

Look at the fourth bit in the 32 bit number, and consider every fourth bit after that. Every time one of these fourth bits is used to make the 48 bit number, the pattern changes slightly. The 5th (or corresponding fifth multiple) from the 32 bit number is used for the 6th (or corresponding sixth multiple) bit in the 48 bit number. This 5th (or fifth multiple) bit will be used a second time for the 8th bit in the 48 bit number. Thus we are using the fifth bit twice to increase the size of the 32 bit number. The 4th (or fourth multiple) bit in the 32 bit number that was previously used for the 5th (or fifth multiple) bit in the 48 bit number, will now be used a second time for the 7th (or seventh multiple) bit in the 48 bit number. Thus we have reused both the 4th and 5th bits (as will be every fourth and fifth multiples) of the 32 bit number to "pad" this (32 bit) number to create a 48 bit number.

With the R notation as defined above: We have converted $R_i$ to $R_i$'.

***B) XORing the expanded half*** $R_i$***'***

On the expanded half $R_i$' $(0 \le i \le 15)$ of the text, whose length is now 48 bits, do the XOR operation with the 48 bit subkey $K_i$ (where $i$ is the number of the round).

We have converted $R_i$' to $R_i$''.

There are 8 of these

***C) S-Boxing the expanded, XORed half*** $R_i$***''***

$S_n$-Boxes (by notation are the eight occurrences of these S-Boxes). Think of these as a black box into which you insert 6 bits of the 48 bit expanded & XORed half of the text $R_i$'' $(0 \le i \le 15)$. The box returns a 4 bit number.

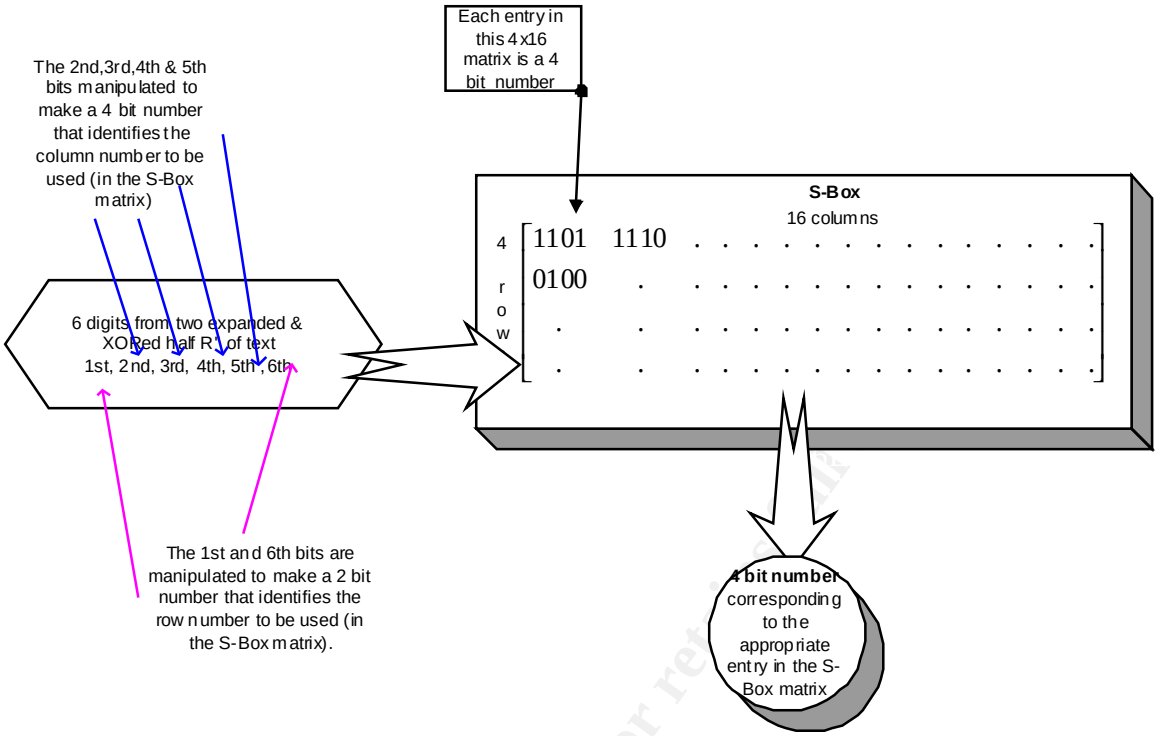



Diagram 7 (Source: Ricky Wald.)

There are eight of these boxes and each takes as input 6 bits of the 48(=6*8) bit expanded & XORed half $R_i$'' of the text.

First 6 bits of the expanded & XORed half $R_i$'' of the text $\rightarrow S_1$

Second 6 bits of the expanded & XORed half $R_i$'' of the text $\rightarrow S_2$

$\vdots$

Last 6 bits of the expanded & XORed half $R_i$'' of the text $\rightarrow S_8$

Consider 6 bits from the expanded & XORed half $R_i$'' of the text. The first and sixth bits can be used to make a two bit number which can be used to reference the row number in the 4x16 S-Box matrix. The remaining 4 of the 6 bits (from the expanded & XORed half $R_i$'' of the text) can be used to make a 4 bit number that references the column number in the 4x16 S-Box matrix. Thus as you can see in the diagram above we can use this 6 bit number to identify a specific entry in the S-Box matrix. As you can see, each element of the 4x16 S-Box matrix is a 4 bit number (e.g. 1100, 1010, …).

These eight S-Boxes each return 4 bit numbers, that are concatenated to result in an 32(=8*4) bit number.

Whence we have a 32 bit expanded & XORed & S-Boxed half of the text $R_i$''' ($0 \le i \le 15$).

***D) Transposition matrix***

To this 32 bit expanded & XORed & S-Boxed half of the text $R_i$''' ($0 \le i \le 15$), we do one further manipulation using a transposition matrix.

The result of doing this final transposition to $R_i$''' leaves us with f($R_{i-1}, K_i$).

***Putting all of our tools together.***

We have all the tools (notation, methods and definitions) and we have described each of the steps as fairly separate entities, so each step should be clear and easily separated from the process. Now we must fit all these parts together.

Consider a piece of plaintext, and without loss of generality (because we can pad the plaintext until it is a multiple size of 64 bits) suppose that it is at least 64 bits in length. What we shall do is to split the plaintext into 64 bit sections and then concatenate the 64 bit ciphertext sections at the end (when we are done). We have no reduced to problem to looking at encrypting a piece of ciphertext that is 64 bits in length.

-1) Split the 64 bit ciphertext into two initial halves then swap the halves.
0) Next perform the initial permutation (IP) on the two swapped halves. Now label the two 32 bit halves $L_0$ and $R_0$ respectively. $L_0$ and $R_0$ are called the permuted input.

1) Round/Iteration 1 – Use the iterative formulae

$$L_i := R_{i-1} \qquad i = 1$$
$$R_i := L_{i-1} \oplus f(R_{i-1}, K_i) \qquad i = 1$$

2) Round/Iteration 2 – do the same as round 1 except use $i = 2$
3) Round/Iteration 3 – do the same as round 1 except use $i = 3$
⋮
16) Round/Iteration 16 – do the same as round 1 except use $i = 16$

17) Perform the inverse to the initial permutation $IP^{-1}$. We now have the required ciphertext.

***Decryption***

One of the most beautiful things about the DES algorithm is that its inverse is itself, although the subkeys $K_i$ must be applied in the opposite order. The DES algorithm is almost a sort of identity algorithm. The implications of this are beyond the scope of my paper but I will simply hint that this enables the use of

hardware that can more simply encrypt / decrypt data leveraging the DES algorithm.

### *Attempts at cracking the Data Encryption Standard.*

Below I only include a subset of the known attacks against DES.

In 1972 a team at IBM developed DES, which was formally chosen as a US federal standard in 1997.

**DES Challenge I**
In 1997, the RSA offered $10k as a prize for cracking DES. Computers all over the world utilized their processing power in one of the largest distributed attacks that the world has ever known. Both the 56 bit key (64 bits including parity bits) and the original plaintext were obtained within 96 days. Personally I am not intellectually impressed because this used a brute force attack.

**DES Challenge II-2**
Only a year later (1998) DES was cracked in under three days by the Electronic Frontier Foundation (EFF), using a purpose built machine aptly called the DES Cracker. The financial investment required to build this machine was approximately $250k, (not modified to reflect inflation since 1998). Again because this was a brute force attack, it is the computational power, rather than the technique that is particularly striking. The processor used was able to attempt 88 billion keys per second, each of length 56 bits (64 bits including parity bits).

**DES Challenge III**
In 1999 the EFF worked in partnership with distributed.net to crack DES in 22 hours and 15 minutes.

I am not aware of DES being cracked by methodologies similar to those that were used in the Vigenère Cipher above, where properties were used apposed to simply trying every possible combination.

Due to these attempts/successes at cracking DES, an extension to this has been developed called triple DES, which works as follows: The plaintext is encrypted using the DES algorithm with a 56 bit key. This ciphertext is then decrypted with the DES algorithm but using a different 56 bit key. The resulting text is then further encrypted using a third 56 bit key. Thus the total key size used with triple DES is 168 (=56x3) bits (or 192 bits including parity bits). A longer key size means that a brute force attack requires more computations and thus more time.

Whether password cracking with dictionary, hybrid or brute force attacks, readers may wish to research the following common tools: Crack, L0phtCrack and John the Ripper.

### *Future Algorithms, Standards and developments.*

The National Institute for Standards and Technology (NIST) has also recognized the need to develop future encryption algorithms. In January 1997, NIST invited the submission of Algorithms that NIST could help become encryption standards. I have stressed that we must progress with encryption, but I must also make the point that standardization is required for these advances to be put into practice. Standards enforce the practicality and usage of developments in computational mathematics, particularly in areas related to security. The NIST was searching for what would be called the Advanced Encryption Standard (AES).

Security professionals, researchers and other academics submitted algorithms for consideration. Enforced by NIST, an algorithm must support key sizes of 128, 192 and 256 bits. NIST chose fifteen algorithms that they would research, analyze and compare to determine which is most suitable. Throughout the whole process NIST encouraged people involved in the field to crack/attack each of the methods. In 1999, NIST had short listed the candidates for AES to only a third of the original number. These five algorithms were; Twofish, MARS, RC6, Rijndael and Serpent. In October 2000, NIST after considering the input from the cryptographic community backed Rijndael (pronounced Rhine-doll) to be the AES.

In 2001, NIST drafted and refined a Federal Information Processing Standard (FIPS) for AES. Despite taking 4 years to go from tender to standard for AES, it is encouraging to see an active involvement and progression.

The specific details of the Rijindael encryption algorithm can be found at, http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf. This document is mainly broken down into mathematical preliminaries, design rational, specification, implementation aspects, performance figures, motivation for design choices, strength against known attacks, expected strength, security goals, advantages and limitations and extensions.
The techniques and mathematical approach of the DES algorithm that we went through earlier will have prepared us surprisingly well for comprehending AES and other algorithms. They share many similarities. In this “advanced” block cipher we find both transformations and polynomials. Some of the tools that we covered in DES, do reoccur again, for example the XOR operation, transition matrices and S-Box concepts. This confirms the value of explaining the tools first so that they can be identified as distinct parts, in order for them to be reused.

### *Conclusion*

We have shown how fundamental encryption is to the current world of security. We followed on from this to demonstrate how previously secure (i.e. thought to be secure) cipher methods can be cracked. We used a counter example (the

Vigenère Cipher) to prove this. Furthermore we covered a methodology for solving security problems in general and the concept of using properties to crack codes/ciphertext.

We then went through the currently used Data Encryption System. Following this we examined some historic attacks on DES. Although people have claimed that DES is not secure and even dedicated years to cracking this **without a brute force attack**, it is trusted by the majority and is widely used in todays industry. We also covered NIST and FIPS standards relating to AES to indicate their profound impact on cryptographic evolution.

I hope that I have accomplished two aims. Firstly to encourage security professionals to invest more resources in developing the more mathematical side of computing. Secondly to reiterate to readers a few of the fundamental security problem solving techniques. Research todays ciphers, do not assume them to never be broken and push the boundaries of this most practical, intellectually stimulating and paramount subject.

---

Diagram References:
Table 4 and Figure 5 only.
All others were created by Ricky Wald.

***References:***


http://www.cs.bris.ac.uk/~cooper/Cryptography/crypto.html (Author: Cooper, Oli. Title: "Cryptography", Last Updated: 22nd February 2001, Date Accessed: 11/9/02)

www.ecn.ab.ca/~jsavard/crypto/co040201.htm (Author: Savard, John J. G. Title: "Details of the Data Encryption Standard", Copyright (c) 1998 1999, Date Accessed: 11/9/02)

http://www.wanet.com.au:8080/~diamond/pbcrypto/view.php?algorithm=dos-des (Author: Napel, Harm ten. Title: "DES cipher for PBDOS", 1993 / 2001, Date Accessed: 11/9/02)

www.trincoll.edu/depts/cpsc/cryptography/caesar.html (Contact: Morelli, Ralph. Author: Trinity College Department of Computer Science, Title: "Cryptography", Last Updated: Sunday July 14 2002, Date Accessed: 11/9/02)

http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf (Author: Daley, William M. & Kammer, Raymond G. Title: FIPS PUB 46-3, Reaffirmed: 1999 October 25, Date Accessed: 11/9/02)

http://findvpn.com/articles/encryption.cfm (Authors: from findVPN at the WHIR. Title: "What is VPN encryption", Copyright 2000/2001/2002, Date Accessed: 11/9/02)

http://www.cs.bris.ac.uk/Teaching/Resources/COMS30124/Lectures/Part_I.pdf (Author: Smart, Nigel from the University of Bristol. Title: "The Science of Cryptography", Date Published: January 30 2002, Date Accessed: 11/9/02)
http://www.tropsoft.com/strongenc/des.htm (Authors: Tropical Software, Title: "DES Encryption overview", Date Published: 2001, Date Accessed: 11/9/02)
http://www.signalguard.com/encryption/triple-des.htm (Authors:SignalGuard employees, Title: The DES algorithm, Date Published: 2001-2001, Date Accessed: 13/10/02)
http://www.interhack.net/projects/deschall/what.html (Authors: Curtin, Mattew. Title: "DES, We cracked the code/DESCHALL", Date Last Modified: April 24th 1998, Date Accessed: 13/10/02)
http://csrc.nist.gov/encryption/aes/index2.html#overview (Authors: NIST employees. Title: Advanced Encryption Standard development effort, Date Last Modified: Feb 28th 2001, Date Accessed: 13/10/02)
http://www.nist.gov/public_affairs/releases/g00-176.htm (Authors: Bulman, Philip. Title: "Commerce Department Announces Winner of Global Information Security Competition". Date Published: Oct. 2nd 2000, Date Accessed: 13/10/02)
http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=462 (Authors: RSA Data Security, Inc. Title: "RSA Code-Breaking Contest Again Won by Distributed.Net and Electronic Frontier Foundation (EFF)". Date Published: Copyright: 2002, Date Accessed: 13/10/02)

***Acknowledgements***
I would like to thank my father, mother and twin brother for being understanding and helpful whilst I was spending enormous amounts of time researching, coming up with my own examples and writing this paper. I would also like to thank Martin Koistinen for his wealth of knowledge that he always clearly explains. I will take this opportunity to thank Alan Buglass and Jacqui Chau for their continued guidance in the computing industry.

Last Updated: August 23rd, 2017

# Upcoming SANS Training

**Click Here for a full list of all Upcoming SANS Events by Location**

| | | | |
|---|---|---|---|
| SANS Tampa - Clearwater 2017 | Clearwater, FLUS | Sep 05, 2017 - Sep 10, 2017 | Live Event |
| SANS Network Security 2017 | Las Vegas, NVUS | Sep 10, 2017 - Sep 17, 2017 | Live Event |
| SANS Dublin 2017 | Dublin, IE | Sep 11, 2017 - Sep 16, 2017 | Live Event |
| SANS Baltimore Fall 2017 | Baltimore, MDUS | Sep 25, 2017 - Sep 30, 2017 | Live Event |
| SANS Copenhagen 2017 | Copenhagen, DK | Sep 25, 2017 - Sep 30, 2017 | Live Event |
| SANS London September 2017 | London, GB | Sep 25, 2017 - Sep 30, 2017 | Live Event |
| Data Breach Summit & Training | Chicago, ILUS | Sep 25, 2017 - Oct 02, 2017 | Live Event |
| Rocky Mountain Fall 2017 | Denver, COUS | Sep 25, 2017 - Sep 30, 2017 | Live Event |
| SANS SEC504 at Cyber Security Week 2017 | The Hague, NL | Sep 25, 2017 - Sep 30, 2017 | Live Event |
| SANS Oslo Autumn 2017 | Oslo, NO | Oct 02, 2017 - Oct 07, 2017 | Live Event |
| SANS DFIR Prague 2017 | Prague, CZ | Oct 02, 2017 - Oct 08, 2017 | Live Event |
| SANS Phoenix-Mesa 2017 | Mesa, AZUS | Oct 09, 2017 - Oct 14, 2017 | Live Event |
| SANS October Singapore 2017 | Singapore, SG | Oct 09, 2017 - Oct 28, 2017 | Live Event |
| SANS AUD507 (GSNA) @ Canberra 2017 | Canberra, AU | Oct 09, 2017 - Oct 14, 2017 | Live Event |
| Secure DevOps Summit & Training | Denver, COUS | Oct 10, 2017 - Oct 17, 2017 | Live Event |
| SANS Tysons Corner Fall 2017 | McLean, VAUS | Oct 14, 2017 - Oct 21, 2017 | Live Event |
| SANS Tokyo Autumn 2017 | Tokyo, JP | Oct 16, 2017 - Oct 28, 2017 | Live Event |
| SANS Brussels Autumn 2017 | Brussels, BE | Oct 16, 2017 - Oct 21, 2017 | Live Event |
| SANS Berlin 2017 | Berlin, DE | Oct 23, 2017 - Oct 28, 2017 | Live Event |
| SANS San Diego 2017 | San Diego, CAUS | Oct 30, 2017 - Nov 04, 2017 | Live Event |
| SANS Seattle 2017 | Seattle, WAUS | Oct 30, 2017 - Nov 04, 2017 | Live Event |
| SANS Gulf Region 2017 | Dubai, AE | Nov 04, 2017 - Nov 16, 2017 | Live Event |
| SANS Miami 2017 | Miami, FLUS | Nov 06, 2017 - Nov 11, 2017 | Live Event |
| SANS Amsterdam 2017 | Amsterdam, NL | Nov 06, 2017 - Nov 11, 2017 | Live Event |
| SANS Milan November 2017 | Milan, IT | Nov 06, 2017 - Nov 11, 2017 | Live Event |
| Pen Test Hackfest Summit & Training 2017 | Bethesda, MDUS | Nov 13, 2017 - Nov 20, 2017 | Live Event |
| SANS Paris November 2017 | Paris, FR | Nov 13, 2017 - Nov 18, 2017 | Live Event |
| SANS Sydney 2017 | Sydney, AU | Nov 13, 2017 - Nov 25, 2017 | Live Event |
| SANS San Francisco Fall 2017 | OnlineCAUS | Sep 05, 2017 - Sep 10, 2017 | Live Event |
| SANS OnDemand | Books & MP3s OnlyUS | Anytime | Self Paced |