a demonstration of different encryption methods on the same sample message

Canonical name	ADemonstrationOfDifferentEncryptionMethodsOnTheSameSampleMessage
Date of creation	2013-03-22 15:49:10
Last modified on	2013-03-22 15:49:10
Owner	PrimeFan (13766)
Last modified by	PrimeFan (13766)
Numerical id	13
Author	PrimeFan (13766)
Entry type	Example
Classification	msc 94A60

Most of the examples given here will use the sample message

$$PROCEEDASPLANNED$$

which Bob will send to Alice.

1 Substitution cyphers

Perhaps the simplest method of encryption is the additive cypher, in which a fixed value is added to each value of the plaintext. For the first example, Bob will add 13 to the ASCII value of each character of our sample message (thus 13 is the private key). The encoded message is then:

$$]\_\backslash PRRQ-N`-]YN[[RQ$$

which looks like gibberish but is vulnerable to frequency analysis. The character R occurs three times, more than any other character, and it probably stands for the letter E, the most statistically frequent letter in English. (If modular arithmetic is used to keep the values within a certain range, then these cyphers are called rotation or shift cyphers).

One way of defeating frequency analysis is by adding a different value to each character depending on the position, according to a code word. For example, if we add the values of the word code to our sample message, then subtract 128 from any values above 128 (to avoid issues with characters not supported by PlanetMath) the encoded message would look something like this:

$$3A3((4((ASCIIENQ)\$B(ASCIIEOT)5/023(3$$

If this was a longer message, it might become very easy to determine the length of the code word and find trigrams that by some lucky coincidence get encoded the same each time.

Other operations besides addition can be used: bitwise exclusive OR, for example. If we XOR our sample message with the word code repeated enough times to match the length of the message, the encoded message looks like:

$$3=+\&\&*E\backslash" < D5/.*+\&+$$

This is of course still vulnerable to frequency analysis.

It is quite possible to pile on different substitution cyphers, which would theoretically make the encryption harder to break. The fatal flaw of this method is that all senders and receivers need to know what the codes being used are and what are their key. If any one of the users was captured, or an interceptor got his hands on the key, the whole system would be compromised. If, in our example, Bob carelessly revealed the encryption algorithm to an interceptor, they send Alice the message QUIT, GO HOME properly encrypted and Alice would have nothing, save perhaps her intuition, to indicate that Bob was not the one who sent the message.

2 Permutation cyphers

Instead of substituting each character of the message for another, one can simply scramble the message according to a pattern agreed upon by the sender and recipient. Suppose Alice and Bob agree to send each other messages scrambled thus: ABCDE becomes DECAB (then DECAB is the private key). To send Alice the sample message, Bob breaks it up into five-letter groups and scrambles each group according to the pattern. Just to throw off frequency analysis, Bob takes the opportunity to pad the last group of five with two extra P's.

$$CEOPRASEDANLPPPDNE$$

Of course it's entirely possible to use both a substitution cypher and a permutation cypher.

$$PR\backslash]\_N` - RQN[Y-]]]Q[R$$

3 Public key cryptography

To remedy these problems, public key cryptography was invented. In such systems, each sender recipient has both private and public keys. The public keys may be freely divulged. If the interceptor obtained Bob's private key, the system would be only partially compromised; Alice and Carol could continue to use the system with their own private and public keys just as before.

3.1 Rivest, Shamir and Adlemann

Perhaps the best known public key system is RSA. Even though our sample message is relatively short, it requires large numbers to properly encrypt using RSA. A toy example might be helpful to elucidate the principle before trying it out with larger numbers. Alice chooses two primes, 2 and 5. Their product is 10, which has 4 coprimes below it. Alice then chooses a number that is coprime to 4, in this example, 3. She publishes her public key, which consists of the public modulus, 10, and the public exponent, 3. For her private key, Alice chooses a number whose product with the public exponent will leave a remainder of 1 when divided by the coprime count of the public modulus. Alice chooses 11, which multiplied by 3 is 33, and that divided by 4 leaves a remainder of 1. The number 11 is Alice's private key, which she does not publish.

In this toy example, Bob wants to send Alice a single bell character, which has ASCII code point 7. Bob raises 7 to the power of the public exponent, 3, which gives 343. Next, he divides 343 by the public modulus, 10, and this leaves a remainder of 3. Bob sends Alice the 3. To decrypt the 3, Alice raises it to the power of her private key, 11, which gives 177147. Divide by 10, that leaves as remainder 7, which is the bell character Bob intended to send.

To show that this toy example is not at all secure, suppose Isaac intercepts the 3 Bob sent Alice. Isaac looks up Alice's public key and on sight factors 10 and calculates the coprime count without the aid of pencil and paper. He doesn't need to know Alice's private key, any number that multiplied by the public exponent is one more than a multiple of the coprime count of the public modulus will do just fine. Isaac tries 15. Raising 3 to the 15th power gives 14348907, and dividing that by 10 leaves 7, Bob's bell character.

Now we may try a slightly more secure example. Alice chooses two primes, 26451800000000083787 and 26551900000000000001459. Their product is 702345548420000222508997706200000122245233, which has 70234554842000022250631606440000
coprimes below it. Alice then chooses a number that is coprime to 70234554842000022250631606440
in this example, 47. She publishes her public key, consisting of the public modulus, 702345548420000222508997706200000122245233, and the public exponent, 47. For her private key, Alice chooses a number whose product with the public exponent will leave a remainder of 1 when divided by the coprime count of the public modulus. Alice chooses 33 as her private key (since dividing 702345548420000222506316064400000122159988 by 1551 has a remainder of 1).

To send Alice the sample message, Bob concatenates the binary (or octal or hexadecimal) into a single number, which works out to 6996991386928867305761549997581651468 in base 10, raises it to the 47th power and divides it by the public modulus, 702345548420000222508997706200000122245233. This is 553523338318669957148919347287695608 which converted to base 16 and interpreted in ASCII gives the encrypted message that might look something like

$$A/c3ya\div AED$$

This is invulnerable to frequency analysis. Only one character occurs twice, and most of the other characters occur just once.

To decode this message, Alice raises 5535233383186699571489193472876956086355524 to the power of her private key, 33, and divides that by the public modulus, leaving a remainder of

The only way to decrypt the message without having Alice's private key is to factor the integers of the public key. 702345548420000222508997706200000122245233 looks intimidating to factor by hand, but a program like Mathematica can factor it in about five seconds. In real life, much larger primes and products are necessary.

To decode the message, Alice raises the number of the message to the power of her private key, modulated by 59701.

Of course, if the interceptor knows RSA is being employed here, they might try to factor the numbers in Alice's public key. If these numbers were very large, accomplishing this might take so long that by the time the number was factored Alice had already accomplished her plan and changed her private and public keys.