

Introduction to Cryptography

by
John Gordon
Concept Labs

email john@ConceptLabs.co.uk

<http://www.ConceptLabs.co.uk>

Contents

1. Prolog	5
2. Background	6
2.1 Four Important Definitions	7
2.1.1 Information	7
2.1.2 Entity	7
2.1.3 Enemy	8
2.1.4 Key	8
2.2 Cryptography	8
2.2.1 Confidentiality	8
2.2.2 Data Integrity	8
2.2.3 Authentication.....	8
2.2.4 Identification.....	9
2.2.5 Non-repudiation.....	10
3. Cryptography	10
3.1 Encryption	10
3.2 Block and Stream Ciphers	11
3.3 Simple Cipher	12
3.4 Assumptions when Assessing Strength	13
3.5 Cryptanalysis	13
3.6 Other ways of breaking ciphers. Effective Key Size.	14
3.7 Proving the Strength of a Cipher	14
3.8 End-To-End-Encryption	16
3.9 Easy, Hard	16
4. Key Management	16
4.1 Cryptographic Key Lifetime	17
4.2 Session, and Session Key	17
4.3 Exchanging Cryptographic Keys	17
4.4 Changing Cryptographic Keys	17
4.5 Storing Cryptographic Keys	18
4.6 Destroying Cryptographic Keys	18

5. Symmetric Block Ciphers	18
5.1 Feistel Architecture	18
5.2 Avalanche Effect	19
6. Modes of Use of Block Ciphers	20
6.1 Cipher Block Chaining - Detection of Message Alteration	20
6.2 Cipher Feedback (CFB)	21
7. Data Origin Authentication, and Data Integrity, MACs.	21
7.1 Message Authenticating Codes	22
7.2 Separate Cryptographic Keys	23
8. Identification - Entity Authentication	23
8.1 Fixed Passwords	23
8.2 Passkeys	23
8.3 Challenge-Response	24
8.3.1 Size of challenge and response fields	24
9. Mathematical Background	25
9.1 Modular Arithmetic	25
9.1.1 Euclidean Algorithm.....	26
9.2 Arithmetic Operations	27
9.3 Implementation of Big-Number Arithmetic	28
9.4 Prime Numbers	28
10. Public Key Cryptography	29
10.1 What is It?	29
10.2 Digital Signatures	30
10.3 Key Certificates, Certification Authority	30
11. The RSA Public Key Cryptosystem	31
11.1 Digital Signatures using RSA	32
12. Bigger Messages - Hash functions	32
12.1 The Secure Hash Algorithm	32
12.1.1 Example	32

13. Digital Signatures using the DSA.....	33
13.1 Signature Generation.....	33
13.2 Signature verification	34
13.3 Example.....	34
14. Elliptic Curve Systems.....	34
14.1 What is an Elliptic Curve?	34
14.2 Operations on points.....	35
14.2.1 Addition	35
14.2.2 Multiplication by a scalar	36
14.3 Cryptographic Functionality using Elliptic Curves.....	36
15. Diffie Hellman Key Exchange Protocol.....	36
15.1 Caveat.....	37
15.2 Elliptic Curve Analog	37
16. Standards	37
17. Glossary	40
18. References.....	51

1. Prolog

In around 100 AD, the *Khama Sutra* of Vatsyayana described a series of 64 life-skills, or *yogas*, for the edification of young, well-to-do ladies in India. While many of these *yogas* were highly erotic, number 44 was the *Science of Writing in Secret Ciphers*.

So cryptography¹ is hardly new.

Indeed, 150 years before this, Julius Caesar was adept at the use of ciphers, and modern textbooks still speak of the *Caesar cipher* he invented, and with which he secretly corresponded with Cicero.

We can go back even further, for Cryptography is as old as writing itself. In 800 BC, Homer in the *Iliad*, made only one reference to writing - the story of Bellerophon - the first man in history or legend to carry an encrypted message. It was to be his doom.

But to find the very earliest known examples, we must go back to around 2000 BC, to the tomb of Khnumhotep II, which was inscribed with mysterious, occult symbols, quite unlike the common, *demotic* hieroglyphs which the Egyptians used for their normal record keeping. These *hieratic* signs were used for secret and magical purposes by the priests.

Cryptanalysis - the science of breaking ciphers - was certainly well established by 1412 when Qualqashandi wrote a 14-volume treatise on the routine breaking Arabic codes.

Cryptography and cryptanalysis have been major forces in history. In sixteenth century England, Mary Queen of Scots was sentenced to death in 1586 and beheaded, on the evidence of the cryptanalyst, Thomas Phillipps. She had been treasonably corresponding in a secret cipher with Philip II of Spain. In 1917, the breaking of the Zimmerman telegram was instrumental in bringing the United States into World War I. The course World War II was dramatically altered by the cryptanalysts at Bletchley Park in Buckinghamshire, who broke German high grade ciphers.

Today, cryptography is part of our everyday lives, being found in such commonplaces as gas meters, cash payment systems, franking machines, vehicle alarms, nautical charts, TV signal scramblers, the internet, and so forth.

This document is an introduction to some of the basic concepts. Because cryptography is a complex subject involving much mathematics, the depth to which we can explore will be restricted, but the topics we touch upon will include some of the most recent innovations.

¹ The word *Cryptography* derives from two Greek words, *kruptw* (“crypto” = hidden - hence *cryptic*, *crypt*) and *grafikoV* (“graphikos” = writing - hence *graphite*, *graphic*).

2. Background

We begin our study by taking a careful look at what we might be trying to achieve by the use of cryptography.

The diversity of tools in the cryptographic armoury is often underestimated. There is much more in the kit bag than just the ability to encrypt. Indeed, there is a widespread and dangerous belief that merely “encrypting everything” - whatever that means - will somehow provide protection “against anything”.

Let us take a look at some counterexamples - situations where merely “encrypting everything” would not provide protection. We will use the banking profession to provide the scenario.

Banks encrypt messages which in effect transfer money from one bank account to another. Nevertheless, even though I might not be able to understand these messages, I would surely gain much advantage on diverting into my own account, cash transfers intended for someone else’s account. I would even be prepared to risk *replacing* or *swapping* a payment into my account with a payment of an *unknown amount* intended for the account of a multi-millionaire.

Also I would become rich if I could divert or stop all messages which debit my account, while allowing the passage of all messages which credit it. It might be possible to distinguish these types of messages merely from their size or from the protocol, without being able to understand the messages themselves.

I could also bring a bank to its knees if I could somehow inject random messages (which even I do not understand) and cause that institution to treat them as cash transfers.

These are trivial examples of attacks which cannot be prevented by “encrypting everything”, and in fact banks use properly designed measures to prevent these attacks. The reason that banks encrypt messages is not because encryption prevents these attacks but because banks are also concerned with the confidentiality of client’s information.

Typically we will want to protect against one or more of these attacks:

- Unauthorised insertion of information (loss of data authentication). To protect against this we need some form of *data origin authentication*.
- Unauthorised modification of information in transit (loss of data integrity). To protect against this we need to protect data integrity, for example by *detection of data manipulation*.
- Unauthorised replay of a recording of an earlier, legitimate data transmission, but in new, unauthorised circumstances. This is another example where we need *data origin authentication*.
- Unauthorised access to information (loss of confidentiality), such as reading out data from a device, perhaps by impersonating a legitimate diagnostic tool. We can protect against impersonation by *entity authentication*. We can protect against loss of confidentiality by *encryption*.

Regarding identification during an exchange of information, it is not sufficient that we merely identify ourselves correctly at the beginning of an exchange. This would not prevent an authorised person from establishing a dialogue, then walking away, leaving some unauthorised person to continue using the link. We may need to authenticate each and every

piece of information being exchanged, not merely the first piece of information or the person who initialises it. One process which achieves this is called *chaining*.

So we need to be aware of a range of protection measures and the attacks which they prevent or detect. To achieve this there is no easy way to avoid taking a look at some security technology. We will keep the treatment here very short and simple compared with standard texts (e.g. [1]).

2.1 Four Important Definitions

A comprehensive glossary is provided in section 17. For the moment we should note the following terms to avoid confusion from the outset.

2.1.1 Information

In the present context, *information* will mean *an ordered collection of (typically binary) symbols, capable of being transmitted and/or stored and/or destroyed, and having some meaning, significance or value*. It will include software, records, data, money, text, parameter settings, diagnostics and so forth. We may sometimes use the term *Data* to mean *Information*.

An important property of information is that, unlike mass and energy, it is *non-conservative*. In other words it can be created and destroyed. This fact is at the root of most security issues concerned with information. We cannot infer that it has not been stolen merely because we still have it.

Money is but a form of information - it exists only in the computer records of financial institutions - and accountancy practices can be viewed merely as an, at best partially successful attempt to endow money with conservative properties. Paper money and coins are but tokens signifying entitlement to have such computer records altered.

There are also legal issues concerned with stealing information. Certainly under English law, *stealing* implies the *intention to deprive the owner of the use* of whatever is stolen. Since stealing information usually means just copying it, it follows that in legal terms, this is not really stealing.

2.1.2 Entity

We will be concerned with protocols. These are dialogues between things called *entities*. An *entity* is a person, terminal, server, tool, modem, random number generator, smartcard, card reader, engine management unit, gas meter, automatic teller machine, etc., taking part in a protocol or providing or modifying information.

The essential point is that while an *entity* may be a person, more usually it is some smart gadget acting as an agent. Entities are said to have *beliefs* about the trustworthiness of other entities with which they exchange information, and these beliefs are based on responses during protocols.

Despite the fact that *entities* are usually not people, nevertheless to illustrate protocols they are commonly given names, usually *Alice* and *Bob* (i.e. rather than A and B). This anthropomorphism pervades the literature, and even the very choice of names follow an unspoken convention, being nearly always Alice, Bob, Carol etc.

2.1.3 Enemy

Enemy is the name we give to an *entity* attacking the security of an IT system to compromise its confidentiality, integrity or availability. Also known as an *attacker*, or *bad guy*. Again, note that *enemy* is not restricted to a person.

2.1.4 Key

In the cryptographic community, *Key* usually means *Cryptographic Key* - a collection of one or more numerical parameters without which it is supposed to be *hard* for an *entity* to recover certain *information* which has been *encrypted*. *Key* also means the index under which parameters are stored in a database.

However, the term *Key* is notoriously liable to lead to misunderstanding unless used with care.

Consider for example the expression *Key Data*. Does this mean the information necessary to reconstruct a key? Is it a synonym for *Cryptographic Key*? Does it mean the symbol-string under which something is indexed in a database? Or does it merely mean *Some Important Information*?

In this document we will try to make the meaning clear from context; and by never using the term *Key* without a some qualifying word, e.g. *Cryptographic Key*, *Door Key*; *Key Certificate*, and by never using *Key* to mean *Significant*, *Important* or *Pivotal*.

2.2 Cryptography

Cryptography is the study of mathematical techniques related to aspects of information security which aims to provide some or all of the services known as *Confidentiality*, *Data Integrity*, *Authentication* and *Non-Repudiation*, which we now briefly introduce.

2.2.1 Confidentiality

Confidentiality is a service used to keep the content of information from all but those *entities* authorised to have it. It is synonymous with *Secrecy*. This is the oldest and most traditional of the services provided by cryptography, and usually operates by *encryption* - the process of converting *plaintext* to *ciphertext* - using a *cryptographic algorithm* and a *cryptographic key*, i.e. making information unintelligible to all entities who do not possess some secret, *cryptographic key*. It is covered in greater depth starting with section 3. The use of Encryption in data transmission systems makes debugging software and systems very difficult, and it is frequently the cause of lost information. It is often mistakenly used when it might be more appropriate, and much simpler to use one of the other services.

2.2.2 Data Integrity

Data Integrity is a service which addresses the unauthorised alteration of data. It does this by detecting data manipulation by unauthorised entities. This is a different process from using Cyclic Redundancy Codes - which only protect against accidental errors, not deliberate alteration.

2.2.3 Authentication

This description owes much to [1]. Authentication is a term which is used in a very broad sense. By itself it has little meaning other than to convey the idea that some means has been provided to guarantee that entities are who they claim to be, or that information has not been

manipulated by unauthorised parties. Authentication is specific to the security objective which one is trying to achieve.

Authentication is one of the most important of all information security objectives. Until the mid 1970s it was generally believed that secrecy and authentication were intrinsically connected - this is related to the misconception mentioned earlier that encrypting everything achieves all goals. Later it was realised that secrecy and authentication were truly separate and independent information security objectives.

It may at first not seem important to separate the two but there are situations where it is not only useful but essential. For example, if a two-party communication between *Alice* and *Bob* is to take place where Alice is in one country and Bob in another, the host countries might not permit secrecy on the channel; one or both countries might want the ability to monitor all communications. Alice and Bob, however, would like to be assured of the identity of each other, and of the integrity and origin of the information they send and receive.

The preceding scenario illustrates several independent aspects of authentication. If Alice and Bob desire assurance of each other's identity, there are two possibilities to consider.

1. Alice and Bob could be communicating with no appreciable time delay. That is, they are both active in the communication in *real time*.
2. Alice or Bob could be exchanging messages with some delay. That is, messages might be routed through various networks, stored, and forwarded at some later time.

In the first instance Alice and Bob would want to verify identities in real time. This might be accomplished by Alice sending Bob some challenge, to which Bob is the only entity which can respond correctly (a *Challenge-Response* protocol - see section 8.3). Bob could perform a similar action to identify Alice. This type of authentication is commonly referred to as *entity authentication* or more simply *identification*.

2.2.4 Identification

Identification commonly implies these features:

- It applies to *entities* in *real time*, i.e. both entities are present now. It does not apply to recordings of messages, or to stored documents.
- If Alice successfully authenticates herself to Bob, then this implies that Bob will accept Alice's identity.
- Bob cannot reuse the exchange with Alice to successfully impersonate Alice.
- It is infeasible for (say) Carol to play the role of Alice, causing Bob to accept Carol as Alice.
- These features remain true even if large numbers of observations of honest exchanges between Alice and Bob have been observed.

2.2.5 Data Origin Authentication

In the second instance (exchanging messages with some delay), it is not convenient to challenge and await response, and moreover the communication path may be only in one direction. Different techniques are now required to authenticate the originator of the message. This form of authentication is called *data origin authentication*.

2.2.6 Non-repudiation.

This is a service which prevents an *entity* from denying previous commitments or actions, for example *having received a message*, or *having signed a document*. As a last resort it is used to resolve disputes, but more commonly its existence is a deterrent against aberrant behaviour.

3. Cryptography

3.1 Encryption

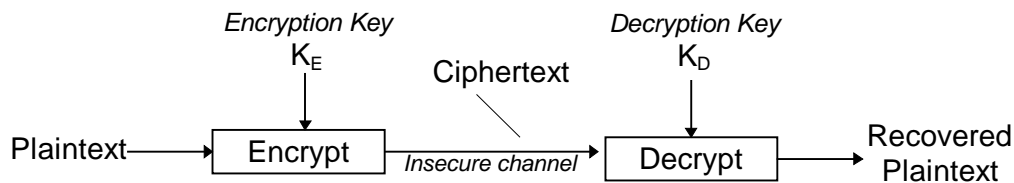


Figure 1 Encryption Paradigm

Many security processes are related to, or derive from encryption. The original purpose of encryption was to protect the confidentiality of information. Now it often forms a building block for other processes.

Regarding Figure 1, a message in its original, useful form is called *Plaintext*. This is then *Encrypted* to form an unintelligible *Ciphertext* or *Cryptogram*. The *Encryption* process (also known as the *Cryptographic Algorithm*) uses a parameter called an *Encryption Key*, without which the encryption cannot be reversed by the legitimate recipient. The ciphertext may now be transmitted over an insecure channel since the *Decryption* algorithm needed to recover the plaintext requires the use of a *Decryption Key*, without which the ciphertext remains unintelligible. This whole set up is called a *cryptosystem*. The term *Cipher* is often used to mean cryptographic algorithm, cryptogram, or cryptosystem.

We need to understand two distinct kinds of *Cryptosystem*. Both can be represented by the diagram shown in Figure 1.

- *Symmetric* (or *conventional*, or *single-key*) cryptosystems, and
- *Asymmetric* (or more usually *Public Key*, or *two-key*) cryptosystems.

With Conventional, or Symmetric cryptography, the Decryption Key is the same as the Encryption Key. A common symmetric cryptographic cipher is DES (see section 5).

By contrast, with Asymmetric, or Public Key cryptography the Encryption Key is quite different from the Decryption Key, and in fact the one may not be inferred from the other except by an infeasible, exhaustive search. A well known Public Key Cryptosystem is the RSA [19] (after its inventors, Ron Rivest, Adi Shamir and Len Adleman)

With a Public Key cryptosystem, if Alice gives Bob her public key, but never reveals her secret key, this enables Bob to send her a secret message which only she can read. This cannot normally be to her disadvantage.

Accordingly it is permissible - indeed normal - to publish the *Public Key*, but to keep the *Secret Key* confidential.

Public Key Cryptography (ref [5]) (often abbreviated to *PKC* - also used to mean *Public Key Cryptosystem*) is one of the most ingenious discoveries in mathematics the twentieth century.

Public key cryptography has the following properties

- It is very slow to compute - typically thousands of times slower than conventional cryptography.
- It possesses a large block structure - typically over 500 bits being the least amount of material which can be encrypted.
- It uses large prime numbers and relies on number theory.

The first two of these are limitations. Accordingly the use of PKC is normally reserved for

- Key Management, e.g. encrypting the cryptographic keys for conventional cryptosystems.
- Authentication applications - where its use is often referred to as a *Digital Signature*.

One of the major problem areas with cryptography is the establishment of cryptographic keys. Public Key Cryptosystems sometimes play a valuable role here.

Attacking cryptosystems is called *Cryptanalysis*, and is carried out by a *Cryptanalyst*, or in popular terms a *Codebreaker*. The study of cryptography and cryptanalysis is called *Cryptology*.

3.2 Block and Stream Ciphers

Conventional or Symmetric Cryptography comes in a variety of forms. One division is between *Block* and *Stream* ciphers. In a block cipher, plaintext is partitioned into *blocks* of a fixed size, say 64 or 128 bits, called the *blocksize*, and the encryption algorithm accepts a plaintext block, x as input and generates a ciphertext block, y as output. Normally x and y would have the same blocksize, but there are exceptions. The encryption algorithm would in fact have two parameters, one is the plaintext, x and the other is the cryptographic key, k . A common notation is to write

$$y = E_k(x)$$

which is read as “ y is the encryption of x under cryptographic key k ”.

The cryptographic key k would also consist of a block of bits, typically 64 or 128 in number, called the *keysize*. Blocksizes can range from 32 bits to 256 bits or even more. Public key cryptosystems typically use much larger block sizes, say several hundred or thousand bits, due to the requirements of number theory.

By contrast a stream cipher consists of a very long sequence of bits which is not partitioned into blocks, but instead the ciphertext is formed by ex-oring its bits with the bits of a *keystream*. If x_i is the i^{th} bit of plaintext, y_i is the i^{th} bit of ciphertext and k_i is the i^{th} bit of keystream, then $y_i = x_i \oplus k_i$, where \oplus means exclusive-or

The keystream sequence is a cryptographic, random or pseudo-random stream of bits.

If the keystream is truly random, and non-repeating, and known only to the sender and recipient, the cipher is said to be a *one time pad*, which is the only truly, provably unbreakable

cipher known. However it is impractical to use because of the difficulty of distributing the keystream.

More usually the keystream is a function of some finite, cryptographic key, K , in other words $k_i = F(K, i)$, where $F(\)$ is some cryptographic function.

3.3 Simple Cipher

We can illustrate some of these points with the following example.

Consider the simple cipher shown in Figure 2 in which each letter of the alphabet is replaced by another letter:

Plaintext	A B C D E F G H I J K L M O N P Q R S T U V W X Y Z
Ciphertext	q w e r t y u i o p a s d f g h j k l z x c v b n m

Figure 2 Monoalphabetic Substitution Cipher

This is called a monoalphabetic substitution cipher.

It has 26 distinct possible blocks, and so we could think of it as having a blocksize of 5 bits with some illegal blocks (because $2^4 < 26 < 2^5$).

In a 26-letter alphabet there are clearly $26!$ ($=26 \times 25 \times 24 \times \dots \times 3 \times 2 \times 1$) different such ciphers, each one corresponding to a different permutation of the letters a to z . Note that $26!$ is approximately 2^{88} , and we could therefore index each permutation by an 88-bit number, so that such a cipher has a cryptographic key size of 88 bits.

However, despite having a very respectable keysize, such a cipher can be broken by most 12-year-olds in about 15 minutes, providing the original message is in some common language such as English, and providing the attacker is given a ciphertext of more than about 30 symbols.

The reason that short messages are hard to break is that short messages do not provide enough information to pin down the key. For example the ciphertext *auv* could correspond to any, three-letter English word in which all the letters are different, such as *AND*, *BUT*, *CAR*, *CAT*, *FIX*. We would need to see more of the ciphertext to pin it down. The minimum amount of ciphertext needed to pin down the plaintext to a single value is called the *unicity distance*.

Nevertheless, even short messages can be sometimes be broken easily, for example

S PX P XPT

which is a simple sentence in English with letter substitution².

Even if spaces are substituted by another symbol (making a 27 letter alphabet) the solution to such simple *monoalphabetic*s is still trivial. The existence of clearly defined letter-frequency statistics makes such ciphers very easy to break.

Notice particularly that the weakness of this particular cipher is in no way due to the smallness of the keysize. Indeed, 88-bits is a very respectable keysize.

Notice too that it may be broken without the attacker being given an example of a matched plaintext and ciphertext. Indeed, for this cipher, being given such a matched pair is in effect being handed the cryptographic key on a plate.

² But what is it!

3.4 Assumptions when Assessing Strength

The assumptions made when assessing the strength of a cipher are usually that the Cryptanalyst:

1. Knows the cipher in as much detail as the designer.
2. Has a large number of matching plaintext-ciphertext pairs.
3. Has powerful computational resources and plenty of time.
4. Is an expert mathematician and computer scientist.

For the simple cipher of section 3.3, assumptions 1 and 2 above are equivalent to *being given* the cryptographic key.

Notice particularly that the security does not depend on the methodology being kept secret - we assume that the enemy knows everything except certain cryptographic keys which are normally changed frequently.

This set of resources constitute what is called a *Known Plaintext Attack*. There are other possible assumptions, for example

- The much less powerful *Ciphertext-Only Attack* which we assumed when we looked at the simple cipher of section 3.3.
- The more powerful *Chosen Plaintext Attack* in which the cryptanalyst can choose the plaintext and have it encrypted.
- The even more powerful *Chosen Ciphertext Attack* in which he can choose ciphertext and have it decrypted.

Remember that, unless otherwise stated, *the assumption is always a Known Plaintext Attack*.

3.5 Cryptanalysis

Cryptanalysis, or *Breaking a cipher*, means any way of recovering the plaintext without being given the cryptographic key. This usually boils down to finding the cryptographic key given the resources of a *Known Plaintext Attack* (section 3.4).

One obvious way to break a cipher, is to search through all possible cryptographic keys until a match is found. In other words given plaintext x and ciphertext y , search through all possible values of cryptographic key k until we find some k satisfying $y = E_k(x)$. Such a method is called an *exhaustive search*.

If k is n bits in length, then an exhaustive search must try 2^n possible values of k , with success occurring on average halfway through the process, so the mean number of searches is 2^{n-1} . If we can carry out one search in T seconds, and we have N search engines all acting in parallel, then the average time to find a solution is $2^{n-1}T/N$.

For example if we have resources such that $T = 1\text{msec}$, and if in addition, we can use one million such machines in parallel to carry out our search, then with a 64-bit cryptographic key the mean time to find a solution is $2^{63} \times 10^{-6} / 10^6 = 9.223 \times 10^6$ seconds or about 3.6 months.. These search facilities are marginally feasible for most organisations in say the year 2000.

Note that every time we increase the keysize by one bit we double the time for exhaustive search.

Clearly there can be little need for a keysize of 128 bits. Nevertheless plenty of ciphers do use such a keysize or even larger.

As a general rule one should be suspicious of symmetric ciphers with very large key sizes (bigger than 200 bits). There can be no good reason for this. It may mean that the algorithm suffers from some dreadful weakness and the designer is attempting to restore confidence by suggesting incredible strength. Remember however that for asymmetric ciphers (PKCs), large key sizes are normal due to the requirements of number theory.

Note that if one finds a value of k such that $y = E_k(x)$, it does not follow that k is necessarily the correct cryptographic key. For example, in Table 1 is shown a simple cipher in which there are 4 possible plaintext messages called $x_1 .. x_4$, and 4 possible ciphertexts $y_1 .. y_4$, and 4 possible keys $k_1 .. k_4$. Clearly the encryption of x_4 to give y_1 would occur with the two cryptographic keys k_2 and k_3 and we would not be able to determine which of these two is the correct one. To be sure we would need to check further plaintext-ciphertext pairs.

Table 1 Simple Cipher

	____ Plaintext x ____			
Key k	x_1	x_2	x_3	x_4
k_1	y_1	y_2	y_3	y_4
k_2	y_2	y_3	y_4	y_1
k_3	y_3	y_4	y_2	y_1
k_4	y_4	y_2	y_1	y_3

3.6 Other ways of breaking ciphers. Effective Key Size.

For many ciphers, an exhaustive key search is not the most efficient way to break them. Most ciphers have some weakness which means that there is some way of finding the cryptographic key which requires less than 2^{n-1} operations on average. The larger the value of n , the greater is this probability.

Suppose there were some way of breaking a cipher which requires on average around 2^{m-1} operations where $m < n$. Then we would say that the *effective key size* of the cipher is m bits, regardless of what the “official” key size is. However to be able to say this we have to know about the weakness and how to exploit it.

There are some important classes of attack, so powerful that most ciphers are susceptible to them to at least some extent. Two such classes are called *Differential Cryptanalysis* [4] and *Linear Cryptanalysis* [14]. Most modern ciphers are designed to provide at least some resistance to these attacks. These attacks are very complex, and beyond the scope of this document, but see the Glossary (section 17) for some notes.

3.7 Proving the Strength of a Cipher

Cryptology is a rapidly evolving science. Every day more algorithms are designed, and more are broken. A common question is *How strong is a given cipher, and can we prove it?*

Proving the strength of a cryptographic algorithm is quite unlike any other type of proof. This is because a single example of a weakness is sufficient to demonstrate that the algorithm is weak, but the inability to find such a weakness does not prove strength, although it may give some comfort.

There have been attempts to *prove* in some mathematical sense the strength of algorithms, but there remains no totally satisfactory methodology. The closest we can come to a proof is in the related discipline of *Computational Complexity*.

Computational Complexity is concerned with determining how hard various problems are. The most important question we can ask turns out to be the hardest to answer, namely

*Is such and such a problem really hard, or is it merely that we don't know how to solve it?*³

In the discipline of Computational Complexity we try to rank the hardness of breaking the algorithm among the hardnesses of reasonably well understood classes of problem. A powerful tool in computational complexity is called *reduction*, and we say that if we can *reduce* problems of type A to problems of type B then type B problems are *at least as hard as* type A problems.

This type of reasoning is often completely misunderstood. A common, and completely fallacious type of “proof” is the following:

If we could solve problem A then we could break the cipher. However problem A is believed to be hard. Therefore the cipher is probably strong.

This puts the reduction exactly the wrong way round⁴. In fact to prove the strength of a cipher we would need to be able to show the following:

If we had an easy way to break this cipher then we could adapt the method to solve problems of type A. However problem A is believed to be hard. Therefore the ability to break this cipher would be equivalent to the ability to solve the hard problem A. So we are forced to believe either that the cipher is strong or that problem A is not really hard.

With this formulation it at once becomes clear why it is so hard to prove the strength of a cipher. To do so we need to find some other problem which is universally believed hard, and a way of adapting an imaginary and unspecified way of breaking the cipher so that the other problem would become easy with the aid of the unspecified method. If we can find such a hard problem and such an adaptation then we have found either (a) a way of breaking the cipher, or (b) a way of solving a problem which everyone believes is hard. Either way we have some kind of success. Very few cryptographic algorithms have been proved strong this way. Finally, even if such a proof can be carried out, it is still only a proof about *beliefs*.

³ To see how difficult it is to answer such questions consider if this is a hard problem:

Multiply MMMLCCCDXXVII by MMLCDXXXIX

The Romans thought it was impossible. The answer in Roman numerals is MMM...MMMLCCCCXXXXII where the run contains 10,427 letter Ms.

⁴ To see why this is a fallacy consider the analogous statement:

If I had wings I could get to Zanzibar. However I don't have wings. Therefore I can't get to Zanzibar.

3.8 End-To-End-Encryption

For systems where data needs to pass through several entities between sender and recipient, it is best to use *End-To-End-Encryption*, rather than *Link-by-Link-Encryption*. With the former, data is encrypted once for all. With the latter the data is decrypted inside each entity and re-encrypted, possibly using a different method, for the next link. *End-To-End-Encryption* is considered vastly more secure, but requires a common message format standard for all the intermediate links, and a common cryptosystem for the original sender and the final recipient.

3.9 Easy, Hard

Up till now we have used the terms *easy* and *hard*, somewhat loosely. In *complexity theory*, these terms have rigorous meanings which we need not go into. We should note however that *hardness* refers to how quickly a problem grows as we make the size of the numbers bigger.

By the *size* of a number we mean the *number of bits in it, not its magnitude*. For example the *size* of 29, which is 11101 in binary, is 5-bits. But its *magnitude* is of course 29. When we represent messages as binary numbers, it is the size of the message which is important from a communications standpoint, not the magnitude.

Here are some examples. The problem of multiplying two, n -bit integers grows in proportion to n^2 . As we will show in section 9.2 the time needed for adding two, n -bit numbers grows in proportion to n . The resources needed for modular multiplications such as $y = x^e \bmod m$, grow as n^3 if x , y and e are all n -bits in size. The time to determine if an n -bit number p , is prime grows as n^4 .

In all these examples the resources grow in proportion to some power of n , e.g. n^3 . These problems are therefore said to be *polynomially hard*. The term comes from the fact that n^3 is a polynomial in n - albeit a rather trivial one.

By contrast the resources needed to perform these next examples grow faster than any power of n . The time to factorise an n -bit number grows roughly as $e^{\sqrt{n \ln(2) \ln(n \ln(2))}}$ and the amount of memory needed to achieve this grows as $e^{\sqrt{n \ln(2) \ln(n \ln(2))}/2}$.

The time needed to search through all possible cryptographic keys of length n -bits grows as 2^n .

Problems for which the resources needed grow faster than any power of n are said to be *intractable*. So factoring is intractable.

If the reverse of a particular intractable problem is easy, then the problem is said to be *NP-Hard*. For example factorisation is NP-hard since factoring m is hard, but if someone tells us its prime factors, p and q then we can easily check that $m = pq$, by multiplication, which is the reverse of factorisation, and is easy.

4. Key Management

The original purpose of cryptography was to provide for the secure transmission of confidential information. In reality encryption does not remove the problem entirely, but it reduces the problem to the secure transmission of the secret cryptographic key. For when the recipient has a confidential copy of the latter, the sender can use a cryptosystem to send the message itself. The weak spot is thus moved to the key management system.

In general the problem of ensuring that secret, cryptographic keys are possessed by precisely those entities entitled to do so, and by no others, is nontrivial.

Key Management includes the processes for

- Generating cryptographic keys in a secure and random manner
- Exchanging cryptographic keys in a secure manner
- Changing cryptographic keys at timely intervals
- Storing cryptographic keys in a secure manner
- Destroying cryptographic keys after they have expired.

Again, note that many security systems have as their weakest point their cryptographic key management procedures.

Notice that many so-called secure transmission systems have been proposed for which the cryptographic key is fixed for all time. This no better than not having a key. Such a set-up is not really a security system at all and would not be considered worthy of study.

Many standards have been proposed for key management. See section 16.

4.1 Cryptographic Key Lifetime

Cryptographic Keys have a *lifetime*. After a certain time they are best discarded and fresh ones used. This limits the damage when cryptographic keys become compromised. More importantly, there are known methods of finding cryptographic keys given enough matching plaintext-ciphertext pairs. Some authorities ensure that they change cryptographic keys before sufficient such material has been generated.

4.2 Session, and Session Key

A common procedure is to agree a cryptographic key valid for a *session*. A session is a single period of communication, normally distinguished by opening a channel, establishing contact, identifying the other entity, establishing a cryptographic key, sending and receiving data, and finally closing the channel. Such a cryptographic key is called a *Session Key*.

4.3 Exchanging Cryptographic Keys

A core problem in cryptographic key management is getting the right cryptographic keys to the right entities. One method is by the use of a trusted third party using a complex protocol. Another is by the use of public key cryptography (sections 7 and 11), or a related procedure such as the Diffie-Hellman Key Exchange Algorithm (section 15).

4.4 Changing Cryptographic Keys

Failure to change cryptographic keys when their lifetime has expired introduces an alarming weakness into any cryptosystem. Human nature being what it is, people will not change cryptographic keys if they can get away without. Therefore it is sensible to make the process automatic.

4.5 Storing Cryptographic Keys

In many systems it is not practical to exchange cryptographic keys at the time a message is sent, and they must be exchanged some time in advance. In some applications, considerable numbers of keys must be exchanged long periods in advance of when they will be used. In these circumstances cryptographic key storage becomes a serious issue. One obvious way of breaking such systems is to capture stored cryptographic keys. Therefore their secure storage becomes a vital part of the system security.

4.6 Destroying Cryptographic Keys

Slightly less obvious is the need to destroy cryptographic keys when they have expired. Several issues are involved here.

Even when a cryptographic key is no longer used for encryption, it may still be needed for decryption, since some messages may be stored in encrypted form as a protection against being disclosed to unauthorised entities. Indeed, storing information in encrypted form is an excellent protection against disclosure. Therefore, ensuring that a cryptographic key really is no longer required is non trivial.

The actual destruction of a cryptographic key is also non trivial. For example, overwriting a bit in RAM memory does not normally erase all record of it. Most RAM chips can be induced to disclose earlier data by clever juggling of the supply voltage. Magnetic disks record in tracks, and the edges of these tracks are not always erased entirely. Also, magnetic hysteresis effects can be exploited to recover earlier data.

To truly erase data in such a way that it cannot be recovered by any of these tricks requires the successive overwriting of each data bit by an alternating series of 0s and 1s.

Finally, on most computer systems, *deleting* files does not even attempt to *erase* the data. It merely sets the *delete flag* - indicating that the disk tracks are eligible for re-use. Some computer systems intended for security applications contain utilities for *shredding* files (or *wiping* or various other terms), indicating that the tracks are overwritten such that each bit is set, first to 0, then to 1, alternately, many times.

5. Symmetric Block Ciphers

The most well known Symmetric Block Cipher (or conventional cipher) is the DES - the Data Encryption Standard - FIPS-46 (section 16).

The DES algorithm is an *Iterated Block Cipher* using a *Feistel* [7] architecture.

5.1 Feistel Architecture

The idea of the Feistel architecture is as follows. The plaintext is initially partitioned into two equal halves, (L_0, R_0) . There then follow n rounds, or passes through a process, with round i having (L_i, R_i) as input and (L_{i+1}, R_{i+1}) as output,

where $L_{i+1} = R_i$

and $R_{i+1} = L_i \oplus f(R_i, K_i)$

These relationships give (L_{i+1}, R_{i+1}) in terms of (L_i, R_i) , where $f()$ is a complicated function, and K_i is a subset of the key bits.

The main feature of the Feistel architecture is the fact that it is unconditionally reversible, without there being any constraints of reversibility on the function $f()$. This follows since we can manipulate the previous relationships to give

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

which clearly give (L_i, R_i) in terms of (L_{i+1}, R_{i+1})

This in turns is a result of the self inverse nature of the ex-or relationship, $a = b \oplus c$.

For if $a = b \oplus c$ then all these relationships are also true

$$b = a \oplus c, \quad c = a \oplus b, \quad a \oplus b \oplus c = 0$$

In the case of DES, the blocksize is 64 bits and the keysize is 56 bits. The algorithm was developed in the early 1970s and is still in use today. It has stood the test of time in the sense that there has never been discovered any cryptanalytic method which is significantly faster than merely searching through all 2^{56} keys, although some very sophisticated methods of doing the latter through the concerted efforts of tens of thousands of participants using the internet have been applied.

Other well known block ciphers include IDEA [12], FEAL [15], SAFER [13], RC5 [18] and TEA [23].

The American NIST organisation (National Institute for Science and Technology) recently commissioned a public competition for a new Advanced Encryption Standard, and the winner was a Belgian algorithm called *Rijndael* (pronounced “rain doll”) [17].

5.2 Avalanche Effect

One of the most important properties which any block encryption algorithm must possess is the *avalanche effect*. By this is meant that if the plaintext is changed by as little as 1 bit, the ciphertext changes a lot. Specifically, if we average over all plaintexts and over all keys, then such a change in plaintext should evoke a change in any ciphertext bit with 50% probability, independently of other ciphertext bits.

Similar ciphertext avalanche effects should obtain if we make the change to the key, and similar plaintext avalanche effects should obtain if we make the change to the ciphertext.

5.3 Nonlinearity

Another vital property for any cipher algorithm is nonlinearity. Any linearity would lead to a simple cryptanalysis by merely solving simultaneous, linear equations. Even a slight linearity can be exploited.

5.4 Triple DES

Sometime the need arises to increase the keysize of a block cipher. A common recourse is to encrypt the plaintext, then the ciphertext, and so on until it is felt that enough key bits have been used. A formal way of doing this is exemplified in the so-called Triple DES, which, perversely uses two keys, not three, and as the name suggests, was originally applied to the DES algorithm.

If the two keys are called A and B , and $E_k(x)$ is the encryption of x under key k , and $D_k(x)$ is the corresponding decryption, then triple DES is the operation

$$y = E_A(D_B(E_A(x))).$$

In other words we first encrypt using key A , then decrypt using key B , and finally encrypt using key A again.

The reason for this is that if $A = B$ then Triple DES is identical to normal DES.

6. Modes of Use of Block Ciphers

Block ciphers are also used as “building blocks” for various processes. These various processes are often called Modes of Use. Important ones are Electronic Codebook (ECB), Cipher Block Chaining (CBC) and Cipher Feedback (CFB).

Electronic Codebook merely means using the block cipher as a lookup device to encrypt blocks. The remaining two modes are more interesting.

6.1 Cipher Block Chaining - Detection of Message Alteration

Chaining, or more correctly *Cipher Block Chaining* (CBC), is a way of detecting alteration of long messages. It is used during a session to protect against a bad guy who seizes the channel after the two principals have identified themselves. It extends any authentication at the beginning of a message right through the message.

It operates by linking together all the ciphertext blocks of a long message in such a way that each of them is a complex function, not just of its own plaintext, but of all its predecessors.

This is how it works. If $E_k(x)$ is the encryption of block x under cryptographic key k , then if a long message consists of the blocks x_0, x_1, \dots, x_{n-1} , then the ciphertext consists of the blocks y_0, y_1, \dots, y_{n-1} , where

$$y_0 = E_k(x_0 \oplus u)$$

$$y_1 = E_k(x_1 \oplus y_0)$$

$$y_2 = E_k(x_2 \oplus y_1)$$

.....

$$y_{n-1} = E_k(x_{n-1} \oplus y_{n-2})$$

and where \oplus means *exclusive-or*. In these relationships, u is an *initialisation vector*. It is normally some random but non-secret value which the sender and recipient agree beforehand.

To see why we need u consider what would happen if two long messages differed only in the n^{th} block. If u were zero, or any fixed value, the ciphertexts would be identical up to y_{n-1} and different thereafter, thus giving valuable information to an attacker. By using a previously-unused value for u , all blocks would be different in the two cases, regardless of which, if any blocks of plaintext were different, and the attacker would lose this clue.

Decryption is by reversing the process as follows.

$$x_0 = D_k(y_0)$$

$$x_1 = D_k(y_1) \oplus y_0$$

$$x_2 = D_k(y_2) \oplus y_1$$

.....

$$x_{n-1} = D_k(y_{n-1}) \oplus y_{n-2}$$

where $D_k(y)$ is the decryption of block y under cryptographic key k .

If an error occurs in a block of the ciphertext, the recovered plaintext is corrupted in the corresponding block and also in its successor block, but thereafter all blocks are recovered correctly.

If however the recovered, and incorrect plaintext is re-encrypted using CBC, then all ciphertext blocks from the first altered plaintext block forwards will be different. In particular the last ciphertext block will be changed if any of its predecessor plaintexts is changed. This is the basis for a Message Authentication Code, or MAC which is dealt with at greater length in section 7.

6.2 Cipher Feedback (CFB)

L -bit Cipher Feedback is a way of generating a self-synchronising stream cipher. This is how it works.

The transmitter maintains a register whose contents, v are the last n bits of transmitted cipher, where n is the blocksize. The contents of this register are encrypted using the block cipher to produce a block w . Then L bits of w are XORed with the next L bits of plaintext to form the next L bits of ciphertext. Finally the oldest L bits of the v are shifted out and the latest L bits of ciphertext are shifted into the register which held v . The whole process is repeated for every batch of L bits of plaintext. The value of L can be anywhere in the range 1 to n , but 1 and 8 are common values. If $L = n$ the process is a variation on the chaining idea.

The receiving end maintains an exact copy of the register v and can therefore decrypt the incoming ciphertext stream. In the event of a transmission error, the fault will clear itself after n bits. This is said to be an n -bit error extension effect. This same effect can be used to initialise the receiver, i.e. merely by ignoring the first n bits of output.

With a conventional stream cipher there is no self-synchronisation and if the sender and receiver get out of step the message is lost. CFB thus provides self synchronisation at the expense of error-extension.

7. Data Origin Authentication, and Data Integrity, MACs.

We now pass onto other issues which are of considerable economic importance, and which employ the notion of Message Authentication Codes or MACs.

MACs are weaker than digital signatures since they do not give third party authentication, and are based on conventional cryptography rather than public key. But they are widely used and our treatment would be incomplete without a look at them.

The twin requirements to give assurance that a message was generated by a given entity (Data Origin Authentication), and to give assurance that it has not subsequently been altered (Data Integrity) cannot be separated. For if a message has been altered then this can be viewed as a change of origin. And if an origin cannot be determined then we have no basis from which to discuss alteration.

A common misconception is to suppose that encryption, *per se*, can provide authentication. The misconception being that if a message decrypts to make sense, then it must have originated from an entity which knows the secret key. However this is very unreliable. For example if the message is a number (e.g. a order-number, setting, serial number, whatever) about which the recipient has little or no *a priori* knowledge, then he cannot check that it meets with his expectations, and hence he cannot validate it. There are more subtle shortcomings too.

Data Origin Authentication can only really be provided by any of the following methods.

1. Using a Digital Signature (sections 10.2 and 11)
2. Appending a secret authenticating message and then encrypting the entire message using some form of error extension such as Cipher Block Chaining (section 3.3).
3. Appending a Message Authenticating Code (MAC)

The first two are discussed in the sections indicated. Here we introduce the concept of Message Authenticating Codes.

7.1 Message Authenticating Codes

Message Authenticating Codes (MACs) are extra fields attached to messages to prove the authenticity of the message. They rely on the use of a common, secret, cryptographic key being shared both by the message author, and by the intended recipient of the message. The MAC is a cryptographically secure function of both the message and the cryptographic key.

The idea is that the author and the legitimate recipient are the only two entities who can create or check the MAC. An incorrect MAC is evidence that the message was not authored as claimed, or that the message has been corrupted.

However, since the recipient could create the MAC, the author has no way to refute authorship if the legitimate recipient forges a MAC. The ability to repudiate a forged message can only be provided by a Digital Signature (sections 10.2 and 11) which is a more powerful function than a MAC, but harder to compute.

Neither MACs nor Digital Signatures, *per se*, provide evidence of time of authorship, but both can be augmented to do so by including a date/time stamp in the message.

MACs should not be confused with Cyclic Redundancy Codes (CRCs) which are also attached to messages to detect corruption. CRCs are a linear function of every bit of the message and are used to detect only certain, random errors. These cannot be used to detect malicious errors since CRCs are not cryptographic functions, and they have no cryptographic keys, so anyone can easily recalculate a CRC after the message is altered.

A MAC is typically calculated as a Cipher Block Chaining operation (section 6.1) on a message, with say 32-bits of the final block constituting the MAC.

7.2 Separate Cryptographic Keys

Cryptographic keys for MACs should be different from those used for encryption, since it is considered bad practice to use the same cryptographic key for more than one purpose.

8. Identification - Entity Authentication

The purpose of Identification is to allow one entity - the *verifier* to gain assurance that the identity of another entity - the *claimant* - is as declared, thereby preventing *impersonation*, or *masquerade*.

A major difference between entity authentication and message authentication is that the former involves actual communication in real-time between verifier and claimant, and so establishes timeliness guarantees.

An effective Identification system has these properties.

1. If Alice authenticates herself to Bob then Bob will accept this.
2. Bob cannot reuse an identification exchange with Alice and successfully impersonate Alice in an exchange with Carol.
3. It is infeasible for a third party, say Carol to impersonate Alice to the satisfaction of Bob.

Identification must make use of one of these mechanisms

1. Something known to both parties. For example a PIN or fixed password.
2. Something possessed by the claimant. For example a passport, a smart card, a passkey, a *transponder* (section 8.3).
3. Something inherent in the claimant. For example a fingerprint, retinal pattern, speech pattern.

8.1 Fixed Passwords

Fixed Passwords are considered a weak authentication mechanism. They may be attacked by replay, observation, searching and so forth.

Fixed passwords should not be stored on the machine used by the verifier, but rather the result of a one-way function of the password. This is prevent an attacker from learning passwords by perusing the file of stored passwords.

So let x be a password and $f(x)$ be a one-way function of x - a function such that, given x it is easy to find $f(x)$, but given y , it is hard to find x such that $y = f(x)$. Then for passwords x_0, x_1, x_2 the system should store $f(x_0), f(x_1), f(x_2)$ etc, and when a user enters his password x , comparison should be made between $f(x)$ and $f(x_0), f(x_1), f(x_2)$ etc.

8.2 Passkeys

A passkey with a PIN is a smart card which conducts a complex protocol with a verifier, and a simple protocol (knowledge of a PIN) with the claimant. This is considered stronger than a fixed password system since it has elements both of something known and something possessed.

8.3 Challenge-Response

Challenge-response is a much more powerful identification mechanism. The idea is that the verifier sets a challenge to the claimant, to which only the true claimant could give the correct response. Frequently the claimant will make his response using a gadget called a *transponder*. Challenges are chosen at random, so the chances that an earlier one will be re-used are small. This is to increase the difficulty faced by an attacker who makes recordings of challenge-response pairs.

An easy way to make a challenge-response system is for the claimant and verifier to share a common, cryptographic key and for the response to be a cryptographic function of the challenge. The transponder contains the function and the cryptographic key.

Vehicle ignition keys often contain a transponder buried in the plastic. The vehicle challenges the ignition key to determine its legitimacy, and if found wanting, the engine is immobilised.

8.3.1 Size of challenge and response fields

When calculating the parameter size for a challenge-response system it should not be assumed that the attacker will try random responses. There is an attack, called a *Dictionary Attack*, against a challenge-response system which is much more cost-effective. However it requires the attacker to have prior access to a legitimate transponder (say in an ignition key).

In this attack, the attacker borrows a transponder and thereby has an opportunity to make n_o observations of challenge-response pairs from which he builds a table, or *Dictionary*, and puts his table into special hardware.

Later he uses special hardware to attack the challenge response mechanism. All he does is invoke challenges till one of them turns out to be in his table.

The question we now ask is this. What is the number, n_c of challenges he must face on average, before encountering one already in his table? The answer turns out to be surprisingly small.

If the number of possible, distinct challenges is $N (=2^{\text{bits in challenge}})$ and if he faces n_c challenges, then the mean number of challenges which turn out to be in his table is $n = n_o n_c / N$. Assuming he stops at the first hit we know that $n = 1$, so $n_c = N / n_o$. So if we fix n_c , we get $n_o = N / n_c$

How long does this take? Suppose making an observation takes T_o seconds and facing a challenge takes T_c seconds, and suppose that the attacker is prepared for example to spend three hours making an exposed attack, but longer building his table since this can be done at a safe location.

Then for say a 32-bit challenge, if T_c is 1 second, we get $n_c = 10800$ (the number of challenges in three hours). It is common to restrict the maximum rate of responses to say 1 per second, to prevent a fast, automated attack.

$n_o = N / n_c = 2^{32} / 10800$, which if $T_c = 1$ second will take about 4 days 15 hours. So the attacker will need make measurements for about 5 days, and then expect to find a hit within three hours.

Note that this is a vastly different estimate of the security than using the assumption that the attacker will merely try random responses, for which the mean time to succeed is of course $T = NT_c = 2^{32}$ seconds, or about 136 years, if the response is a 32-bit number too. Note that for this latter attack the critical number is the size of the response rather than the size of the

challenge as in the dictionary attack above. These two numbers just happen to be the same size in this instance.

But the attacker can do even better than this. Suppose he considers it optimal to minimise the total time, $T = n_o T_o + n_c T_c$ (equivalent to being indifferent to whether he spends time building tables or facing attacks) then

$$\begin{aligned} T &= n_o T_o + n_c T_c \\ &= n_o T_o + NT_c / n_o \end{aligned}$$

and to find the minimum we put $\frac{dT}{dn_o} = T_o - NT_c / n_o^2 = 0$

giving the optimal number of observations as $n_o = \sqrt{NT_c / T_o}$

and the mean challenges to succeed as $n_c = \sqrt{NT_o / T_c}$

So for example, if the time to make an observation on the transponder, T_o , and the time between challenges, T_c are both one second, then the optimum number of observations is \sqrt{N} . For a 32-bit challenge, $n_o = n_c = 2^{16}$, and the observation phase and the attack phase both take about 18 hours each.

9. Mathematical Background

We shortly wish to look at public key cryptosystems in a little more detail, but before proceeding we need to deal with some mathematics which may not be familiar to the non specialist.

9.1 Modular Arithmetic

Many processes in modern cryptography use what is known as *modular arithmetic*. Without an understanding of modular arithmetic, when we look at public key cryptosystems we will grind to a halt, so we dispose of it now.

It is best explained by example.

If we write $y = ax$ we mean that y is to be calculated by multiplying a by x .

But if we write $y = ax \bmod m$ we mean that y is to be calculated by multiplying a by x and then we divide the product ax by m and set y equal to the remainder.

So for example $9 = 5 \times 7 \bmod 13$ because $5 \times 7 = 35 = 2 \times 13 + 9$.

In this notation, m is called the *modulus*.

There is a similar notation with a slightly different meaning, namely $x \equiv y \pmod{m}$, pronounced “ y is congruent to x modulo m ”. This notation means that $x = y + km$ where k is a positive or negative integer or zero. In other words any difference between x and y is confined to a multiple of m .

So $10 \equiv 3 \pmod{7}$, and $17 \equiv 3 \pmod{7}$, and $22 \equiv 15 \equiv 1 \pmod{7}$.

It is easy to verify that any addition, subtraction or multiplication result which is valid using normal arithmetic is also valid if we replace all the numbers with their remainders after dividing by some modulus, m .

In modular arithmetic, division is performed by multiplying by the *modular inverse* of an element.

For example the modular inverse of $7 \pmod{19}$ is the number x , such that $7x = 1 \pmod{19}$. It turns out that $x = 11$. This is because $7 \times 11 = 4 \times 19 + 1 = 1 \pmod{19}$.

We can use the notation $x^{-1} \pmod{m}$ to mean the modular inverse of x with respect to the modulus m . In other words $x^{-1}x = 1 \pmod{m}$.

With this notation $y/x \pmod{m}$ is evaluated as $x^{-1}y \pmod{m}$. So for example $10/7 \pmod{19}$ is $10 \times 3^{-1} \pmod{19} = 10 \times 11 = 15$, because $10 \times 11 = 110 = 5 \times 19 + 15$.

Note that $y/x \pmod{m}$ is not the quotient, y/x expressed modulo m . That would be $[y/x] \pmod{m}$, where $[v]$ means the integer part of the fraction, v .

For a number x to have a modular inverse with respect to a modulus m , it is necessary that $(x, m) = 1$, where the notation (x, m) means the *highest common factor of x and m* , in other the largest integer which exactly divides into both x and m .

9.1.1 Euclidean Algorithm

There is a simple method of finding both (x, m) and $x^{-1} \pmod{m}$. It is called the *Extended Euclidean Algorithm*, and was known as long ago as 300 BC.

A modern form of this algorithm is as follows. The description about to be given is enormously simpler than that given in most texts.

We initialise a row of numbers with the integers $L=m$ and $R=x$. This would be 19, 7 in our example.

Next we find the quotient q , when we divide m by x and write it underneath x , then subtract xq from m . Thus

$$\begin{array}{r} m, \quad x, \quad m-xq \\ \quad \quad \quad q \end{array}$$

Our example now looks like this

$$\begin{array}{r} 19, \quad 7, \quad 5, \\ \quad \quad \quad 2 \end{array}$$

Then we shift one place to the right and start over. So now L becomes 7 and R becomes 5. We continue in this way till R becomes zero.

$$\begin{array}{r} 19, \quad 7, \quad 5, \quad 2, \quad 1, \quad 0 \quad L/R \text{ row} \\ \quad \quad \quad 2, \quad 1, \quad 2, \quad 2, \quad \quad \quad q \text{ row} \end{array}$$

The last-but-one integer before the 0 is (x, m) , in this case 1.

Now we find the inverse of x . We do this by initialising a third row (the U/V row) with the values $U=0, V=1$ then using the same quotients q , we repeatedly write $U - qV$ under each next column, then move one place to the right. The number appearing under the column with 1 at the top is the required inverse:

$$\begin{array}{r} 19, \quad 7, \quad 5, \quad 2, \quad 1, \quad 0 \quad L/R \text{ row} \\ \quad \quad \quad 2, \quad 1, \quad 2, \quad 2 \quad \quad \quad q \text{ row} \\ \quad \quad \quad 0, \quad 1, \quad -2, \quad 3, \quad -8 \quad U/V \text{ row} \end{array}$$

Thus -8 is the required inverse. Of course $-8 \bmod 19$ is just 11, so 11 is the result we are after, but -8 is a perfectly valid way to express it if we prefer.

Clearly we do not have to wait till the L/R row is complete before starting the U/V row. They can both be run at the same time. If we only want to find (m, x) and not $x^{-1} \bmod m$, we merely omit the U/V row calculation.

9.2 Arithmetic Operations

We have already considered the topic of complexity in section 3.9. When we come to look at Public Key Cryptography (section 10) we will need a modest grasp of the nitty gritty of big-number arithmetic, and this is a convenient place to take a closer look.

Any very large integer, x needs to be stored as an array of smaller integers. The size of such an array is proportional to n , the number of bits in x . The time to add two such large integers is also proportional to the size of this array, and hence to n . We write this as $O(n)$, pronounced “of the order of n ”.

Since multiplication can be written as a doubly nested loop, the time to multiply two large integers is proportional to the product of their sizes, say $n_1 n_2$, or n^2 if they are both the same size. This is written as $O(n^2)$ (“of the order of n -squared”).

Less obvious is the complexity of modular exponentiation.

An expression of the form $y = x^e \bmod m$ is said to be a *modular exponentiation*.

Bearing in mind that all three variables, x , e and m are likely to be big numbers, containing hundreds of bits, a common initial reaction is incredulity that it is possible to calculate a result like $y = x^e \bmod m$ without performing something like $e-1$ multiplications and a similar number of divisions.

But by using what is known as the *method of repeated squares* it is possible to determine the result of such a calculation using at most $2n$ multiplications and divisions, where n is the number of bits in e .

To see how this works, we will *write exponents in binary*, and x^{abc} will mean x raised to the power abc where a , b and c are the binary digits of the exponent. Here are the recipes for various operations:

To get	From	Do this
$x^{10} \bmod m$	$x \bmod m$	square $x \bmod m$
$x^{100} \bmod m$	$x^{10} \bmod m$	square $x^{10} \bmod m$
$x^{101} \bmod m$	$x^{100} \bmod m$	multiply x^{100} by $x \bmod m$
.....
$x^{abc\dots xyz0} \bmod m$	$x^{abc\dots xyz} \bmod m$	square $x^{abc\dots xyz} \bmod m$
$x^{abc\dots xyz1} \bmod m$	$x^{abc\dots xyz} \bmod m$	multiply $x^{abc\dots xyz}$ by $x \bmod m$

After perusing this table it is clear that finding $y = x^e \bmod m$ by the method of repeated squares is simply:

```

Put  $y = 1$ 
starting with most significant bit of  $e$  down to the least significant
{
  put  $y = y^2 \bmod m$ 
  if the  $e$ -bit is 1 then put  $y = xy \bmod m$ 
}

```

Figure 3 Method of Repeated Squares

It is easy to see that for every bit of e we perform a modular multiplication, and if the bit is 1 we perform a second modular multiplication.

A modular multiplication $ab \bmod m$ requires $O(n^2)$ operations if a , b and m are all n -bits long. So if all 3 operands, x , e and m are n -bits long, a modular exponentiation needs at most $2n \times O(n^2)$ operations.

In other words the complexity of modular exponentiation is $O(n^3)$, if all operands are n -bits long.

9.3 Implementation of Big-Number Arithmetic

One of the first problems encountered when experimenting with modern cryptography is the need to deal with the arithmetic of large numbers in an environment in which most computer languages only define numbers of up to say 32 or 64 bits.

The solution is to define a new class of numbers and the arithmetic operations needed - in other words to write an arithmetic unit.

There are good and bad ways to go about this. In the author's humble opinion, a good way is an object-oriented approach using operator overloading.

The language C++ permits the definition of new classes of object. So it is possible to define a class representing a massive integer - a *mint* say - together with the operations needed to read, write, test and manipulate such objects. So a *mint* might be defined as an array of words, which we might call *digits*, together with a parameter to denote the number of digits actually used, and another to denote the sign. We would then need routines to create and destroy *mints*, to convert *mints* to and from strings, and to convert them to and from other types of number, and to add, subtract, multiply, divide and compare *mints*. We would also need routines to find modular inverses and to perform modular exponentiation.

In C++ we have the luxury of overloading the operators. So we can define expressions like $x * y$ where x and y are *mints*. We can give meaning to tests such as "if ($x > y$)" and so forth. The existence of such an arithmetic unit enormously simplifies the problem of checking the correctness of code to carry out sophisticated coding functions.

9.4 Prime Numbers

Public Key Cryptosystems typically employ number theory [21] involving large, prime numbers.

A prime number, or more simply a *prime*, is an integer which cannot be exactly divided by any other integer, except trivially itself and 1.

The first few primes are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37. Primes become increasingly sparse as their size increases - the mean distance between primes of length n -bits is about

$0.69n$, but there are an infinite number of primes, and there are always primes larger than any given integer, no matter how large.

An integer which is not prime is said to be a *composite*. Every composite is the product of a *unique* combination of primes, called its *prime factors*. For example $551 = 19 \times 29$. There are no other primes which multiply to give 551. This uniqueness was known to the ancient Greeks. The process of determining the prime factors of a composite is called *factorisation*. The difficulty of factorisation grows very rapidly as the size of the composite (measured by the number of bits needed to write it) increases.

For example, what are the factors of 21? We immediately say 3 times 7. But how do we know? Because we have taken the trouble to memorise certain “multiplication tables”. What then are the prime factors of $m = 1101107603148934264935119$? The answer is 1064160690193 times 1034719298783 , but finding these values is a problem whose difficulty increases very rapidly as the number of bits in m increases. In fact this difficulty increases roughly as $O(e^{\sqrt{n \ln(2) \ln(n \ln(2))}})$ where n is the number of bits in the integer to be factored.

Our arithmetic unit will also need routines to test for primality. Fortunately it is not necessary to be able to factorise large numbers to determine if a number is prime. There are simpler methods which require just a few modular exponentiations.

Finally we note the term *mutually prime*. Two integers x, y are said to be mutually prime if $(x, y) = 1$, where (x, y) means the highest common factor of x , and y .

10. Public Key Cryptography

10.1 What is It?

Public Key Cryptography is cryptography, just as in Figure 1, but in which the cryptographic key K_E - used to encrypt, is different from K_D - the one used to decrypt, one of them being secret, the other public knowledge.

Originally attributed to Whitfield Diffie and Martyn Hellman in 1976 (ref [5]) it was recently revealed that the technique was actually discovered years earlier by J H Ellis and C Cocks, at the British Intelligence organisation GCHQ in Cheltenham, England (ref [6]), but of course this work was classified, and remained so until December 1997.

The idea is that to encrypt plaintext x into ciphertext y using the public, encryption key, e we use some formula

$$y = F_e(x)$$

and the recover x using the secret, decryption key, d the legitimate recipient uses another formula

$$x = G_d(y)$$

The system is arranged so that d cannot feasibly be computed from e .

Without doubt, the most important PKC is that proposed in 1977 by Ron Rivest, Adi Shamir and Len Adleman, and known as the RSA Cryptosystem [19]. This is described in detail in section 11.

Note that in some PKCs, and notably in RSA (section 11), functions $F_e(x)$ and $G_d(y)$ may be identical apart from the parameters e and d .

10.2 Digital Signatures

By interchanging the roles of plaintext and ciphertext in a Public Key Cryptosystem it is often possible to produce what is commonly called a *digital signature*. This topic is dealt with in much greater detail when we discuss a particular example of public key cryptosystem, the RSA algorithm in section 11, but a simplified explanation is given here.

Suppose Alice wants Bob to sign a message, y . She sends y to Bob and asks him to send her $x = G_d(y)$, and then she checks that $y = F_e(x)$. If this turns out to be the case, then she knows that whoever sent x must have known Bob's secret key d . In a sense Bob has *signed* the message, y by showing that he knows d . Note that for this to work, Bob must take great care not to lose or disclose d , and Alice must believe this.

One important point to note is that in the event of a dispute between Alice and Bob, Alice can take the message y and signature x to an impartial third party (in effect a judge) who can check for himself that $y = F_e(x)$ without the need to disclose Bob's secret, e . This is called *Third Party Authentication*.

There are lots extra tricks to handle long messages using short signatures and so forth, but these are all dealt with in section 11.

10.3 Key Certificates, Certification Authority

The idea of digital signatures is a very powerful tool. Let us see an example of one of the things we can do with it.

Alice wants to do business with Bob, but she has never met Bob and does not have a copy of his public key.

She will not believe anyone who gives her a public key and says it is Bob's, since if it is not Bob's and she sends confidential information using it, then this information will be read by whoever knows the corresponding secret key, and that person is not Bob.

So what can she do? Well suppose she trusts Carol and has a copy of Carol's public key E_C , and suppose that Carol has a copy of Bob's public key. Then she can ask Carol to send her a copy of Bob's public key and sign a message saying in effect

*I, Carol, hereby declare that Bob's public key,
 E_B , is given by the sequence 10743...50449*

Signed

Carol.

Where **Carol** is Carol's signature for the preceding message. This sort of message-signature pair is called a *Key Certificate*.

We can do even better than that. There is no reason why Bob should not have a copy of this certificate, and when Alice wants Bob's public key, he gives her, not his public key but his Key Certificate.

This leads on to the idea of a trusted agency, or *Certification Authority*. There can be one or more agencies who perform the role of Carol. Such an agency performs checks about the identity and *bone fides* of various entities who wish to partake in the scheme, and issues each of them with a Key Certificate. The participating entities can use these certificates to authenticate themselves to each other without the need to contact the trusted agency each time.

Notice that they are using *Third Party Authentication* (section 10.2) since they do not trust each other.

Moreover these certificates are not limited merely to specifying public keys. They can specify which privileges each entity may exercise, such as *reading* data, *writing* data, *using* data, and so forth.

Certificates must contain an expiry date as part of the signed data, since it is difficult to revoke already issued certificates.

Note that there is nothing confidential about a key certificate, and there is nothing lost, and everything to gain by giving it wide publicity.

11. The RSA Public Key Cryptosystem

The RSA public key cryptosystem was first described in the open literature in 1976 by Rivest, Shamir and Adleman, then all at MIT (ref [19]). However it has recently been disclosed that it was in fact devised years earlier at GCHQ in England by Ellis and Cocks [6]. This recently came to light when that organisation, concerned with the need to justify its budget, declassified some important papers. Meantime the secrecy maintained by GCHQ enabled Rivest *et al.* to make a fortune in royalties.

This is how it works. It makes use of very large *prime numbers*.

The RSA cryptosystem depends on three facts:

- It is easy to find large primes, say p and q .
- It is easy to multiply p and q to form the composite, $m = pq$.
- It is hard to factorise the product m .

In other words we can easily obtain large primes p and q , and we can easily multiply them together, and we can publish the result, m . Yet nobody can find p and q from knowledge of m .

Another easy calculation is called *modular exponentiation*. It means calculations of the form

$$y = x^e \bmod m$$

where x , y and e may be very large numbers. This was discussed in section 7.

Finally there is an important result in number theory called the *Euler-Fermat Theorem* (it is not supposed to be obvious) which says that if p and q are primes, and e and d are chosen so that

$$e \text{ is mutually prime to } p-1 \text{ and } q-1.$$

and

$$ed = 1 \bmod (p-1)(q-1)$$

and if we choose x and form

$$y = x^e \bmod m$$

then x may be recovered from y using the formula

$$x = y^d \bmod m.$$

Now x could be a plaintext message, considered for the purpose of the calculation to be a large number, and y could be the corresponding ciphertext. The pair of numbers, e and m would constitute the public key K_E , and the pair, d and m would be the secret key K_D .

If someone knows e and m , they cannot feasibly find d since they will need to solve $ed = 1 \pmod{(p-1)(q-1)}$, and to do that they need to know p and q , and to find these they need to factorise $m = pq$. Which is hard. Note that this is not a proof of hardness, and indeed it cannot be since it presents the facts exactly the wrong way round. See section 3.7.

11.1 Digital Signatures using RSA

We can interchange the roles of plaintext and ciphertext to produce what is usually called a *Digital Signature*. It is easiest to understand with an example. Suppose that Alice has Bob's public key $K_E = (e, m)$. Suppose that Alice wants Bob to sign a short message y of a few hundred bits. We emphasise that y is not secret or encrypted. It is just a message. Bob generates the integer x given by $x = y^d \pmod m$ and sends it to Alice. Alice now checks that $y = x^e \pmod m$, which she can easily do since she knows (e, m) . But only someone knowing (d, m) could have generated x . In other words x is Bob's signature to the document y .

12. Bigger Messages - Hash functions

Now suppose that the message M is thousands or millions of bits long. It would be too tedious to break it into manageable chunks and sign each one. So we use a neat trick called a *Cryptographic Hash Function* or where there is no ambiguity, merely *Hash Function* or even just *Hash*.

A Cryptographic Hash Function is a function $y = H(M)$, where M is a message of arbitrary and unlimited length, and y has a fixed, predetermined length, say n bits, such that knowing y , it is infeasible to find any message M_1 such that $y = H(M_1)$, and also infeasible to generate any two messages M_1 and M_2 such that $H(M_1) = H(M_2)$.

So any alteration to a message M_1 , which changes it to M_2 will mean that $H(M_1) \neq H(M_2)$, and so $H(\cdot)$ can be used to detect tampering with M_1 .

Note that $H(M)$ is not unique since M is large and $H(M)$ is probably only a few hundred bits long. Therefore there must be lots of M s sharing the same value of $H(M)$. But it is hard to find them.

Having determined $y = H(M)$ Bob now merely signs y . So if for example he is using the RSA, he calculates $x = y^d \pmod m$. Now x is the digital signature of M . Anyone - say Alice - who knows (e, m) can check the signature. All she does is calculate $H(M)$ and $y = x^e \pmod m$, and check that the value of y she calculates is the same as $H(M)$.

12.1 The Secure Hash Algorithm

The (American) National Institute of Science and Technology has defined a standard hash algorithm called the Secure Hash Algorithm, FIPS 180-1 [8]. This generates a 160-bit hash from a message consisting of any number of bits from 1 to 2^{64} . This particular hash function has been subjected to a great number of attacks without any suggestion of weakness. In addition it is unencumbered by intellectual property issues. It should be seriously considered when any such function is needed.

We will use the notation $\text{SHA}(x)$ to denote the string generated from the message x using the Secure Hash Algorithm.

12.1.1 Example

$x =$ "Introduction to Cryptography by John Gordon" (43 bytes)

hashes to the 20 bytes, written in hex as:

SHA(x) = 0x8D161E5A BF03A3B1 2A39D276 DCB2C71E C5B57CC3

13. Digital Signatures using the DSA

Because of export restrictions relating to strong, cryptographic systems, and also due to patent issues relating to RSA, other digital signature algorithms have been proposed. Notable among these is the US standard known as the Digital Signature Algorithm, FIPS 186 [9], commonly referred to as the DSA.

The DSA makes use of the following parameters:

1. p = a prime modulus, where $2^{L-1} < p < 2^L$
for $512 \leq L \leq 1024$ and L a multiple of 64
2. q = a prime divisor of $p - 1$, where $2^{159} < q < 2^{160}$
3. $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p - 1$ such that
 $h^{(p-1)/q} \bmod p > 1$ (g has order $q \bmod p$)
4. x = a randomly or pseudo-randomly generated integer with $0 < x < q$
5. $y = g^x \bmod p$
6. k = a randomly or pseudo-randomly generated integer with $0 < k < q$

The integers p , q , and g can be public and can be common to a group of users. A user's private and public keys are x and y , respectively. They are normally fixed for a period of time. Parameters x and k are used for signature generation only, and must be kept secret. Parameter k must be regenerated for each signature.

13.1 Signature Generation

The signature of a message M is the pair of numbers r and s computed according to the equations below:

$$r = (g^k \bmod p) \bmod q \text{ and}$$

$$s = (k^{-1}(\text{SHA}(M) + xr)) \bmod q.$$

In the above, k^{-1} is the multiplicative inverse of k , mod q ; i.e., $(k^{-1} \times k) \bmod q = 1$ and $0 < k^{-1} < q$.

We note that the use of the SHA is mandated, rather than any other hash algorithm.

The signature is transmitted along with the message to the verifier.

13.2 Signature verification

Prior to verifying the signature in a signed message, p , q and g plus the sender's public key and identity are made available to the verifier in an authenticated manner.

Let M' , r' and s' be the received versions of M , r , and s , respectively, and let y be the public key of the signatory. To verifier first checks to see that $0 < r' < q$ and $0 < s' < q$; if either condition is violated the signature shall be rejected. If these two conditions are satisfied, the verifier computes

$$w = (s')^{-1} \bmod q$$

$$u1 = ((\text{SHA}(M')w) \bmod q$$

$$u2 = ((r')w) \bmod q$$

$$v = (((g)^{u1} (y)^{u2}) \bmod p) \bmod q.$$

If $v = r'$, then the signature is verified and the verifier can have high confidence that the received message was sent by the party holding the secret key x corresponding to y .

If v does not equal r' , then the message may have been modified, the message may have been incorrectly signed by the signatory, or the message may have been signed by an impostor. The message should be considered invalid.

13.3 Example

“Introduction to Cryptography by John Gordon” has the digital signature (in hex)

$$r = 0x29FB93B3\ 5B8E2951\ CA35B861\ 8122E8F7\ 5B11125A$$

$$s = 0x9FB84ED5\ 1E19093F\ FADC6E05\ A487EAB8\ 31858ABA$$

14. Elliptic Curve Systems

New classes of cryptosystem based on the theory of Elliptic Curves are emerging [10]. These systems are too new to have established any track record, but offer potential benefits in terms of speed of computation and cryptographic strength, if early results stand the test of time.

This is a truly vast and complex subject and the treatment here is necessarily superficial. The draft standard document [10] contains a great deal of background information. More general material on Elliptic Curves can be found in [21].

14.1 What is an Elliptic Curve?

There are several kinds of defining equations for elliptic curves, but the most common is the *Weierstrass* equation.

— If p is a large prime, the *Weierstrass* equation is

$$y^2 = x^3 + ax + b \bmod p$$

where a and b are integers for which $4a^3 + 27b^2 \neq 0 \bmod p$.

The constants, a and b , and the prime, p effectively define a particular elliptic curve.

Given a *Weierstrass* equation, the elliptic curve, E consists of the points (x, y) which satisfy $y^2 = x^3 + ax + b \pmod p$, along with an additional element called the *point at infinity* (denoted O). The points other than O are called *finite* points. The number of points on E (including O) is called the *order* of E and is denoted by $\#E (GF (q))$.

Example: Let E be the curve

$$y^2 = x^3 + 10x + 5 \pmod{13}$$

Then the points on E are

$\{O, (1,4), (1,9), (3,6), (3,7), (8,5), (8,8), (10,0), (11,4), (11,9)\}$.

Thus the order of E is $\#E (GF (13)) = 10$.

Whereas in the case of say RSA we deal with large integers, in the case of elliptic curves we deal with points, where the point P means the pair of integers (x, y) , satisfying

$$y^2 = x^3 + ax + b \pmod p.$$

The x and y in $P=(x, y)$ are themselves of course, large integers.

14.2 Operations on points

It is possible to define operations on points. The main operations are Addition and Multiplication by a scalar.

14.2.1 Addition

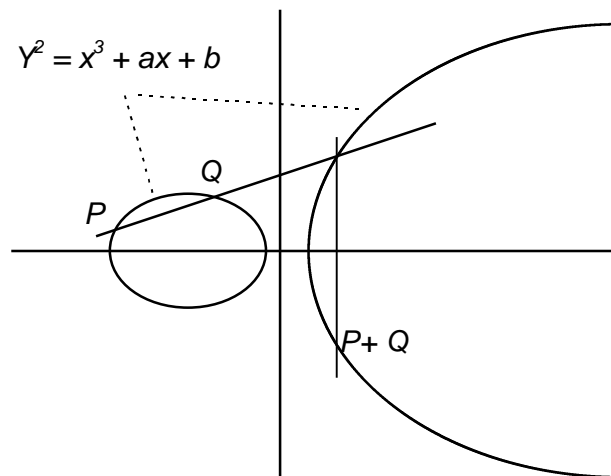


Figure 4 Elliptic Curve (analogy)

Algebraically, the result is rather complicated, but we can give an attractive, geometric interpretation. If we regard the elliptic curve as a sort of plane, geometric type “curve”, with “points” being actual positions on the curve, then the addition of the two points, P and Q , is analogous to joining them with a straight line, and finding where this line again crosses the elliptic curve. The reflection of this point in the x axis is then said to be the *sum*, $P + Q$ of the points. Of course this analogy cannot be made too exact since only integers are permitted, and the arithmetic is modular, so neither the curve nor the process cannot be drawn, except in a loose, abstract sense.

14.2.2 Multiplication by a scalar

Repeatedly adding k copies of the same point P together generates the scalar product kP . There is a fast algorithm for this, analogous to the repeated square algorithm to form modular exponentiation.

(Note that some authorities use the term *multiplication* for what we have called *addition* here, and these same authorities would use *scalar exponentiation* for what we have called multiplication by a scalar.)

As was suggested for the implementation of the basic arithmetic unit (section 9.3), one way to handle elliptic curve functions is to define a class called `EllipticCurvePoint`, with operations such as addition and multiplication, and to overload the addition operator and multiplication operators.

14.3 Cryptographic Functionality using Elliptic Curves

There are analogues to many number-theoretic cryptosystems using elliptic curves in place of large integers. For example there are elliptic curve variants of the Digital Signatures Algorithm.

The reason for interest in elliptic curve cryptography is that the integers (x, y) in the point P do not need to be as big as the integers in say RSA to provide similar levels of security. In other words there are modest improvements in storage requirements and speed, in exchange for greater complexity of operations, and at least at the present, a lack of a track record regarding security.

15. Diffie Hellman Key Exchange Protocol

When the Diffie-Hellman Key Exchange Protocol was first described in 1976 [5] it caused a sensation since it appeared to do the impossible.

It enables two entities which have never met, and which have no prior secret agreement, to agree upon some secret parameter, say k , during a publicly observed dialogue, at the end of which none of the observers would know even one bit of k .

The obvious secret to agree on is a cryptographic key, and that is the main application of the protocol - agreeing on cryptographic keys over an insecure channel.

This is how it works. Given a large prime p , and an integer b , it is easy to calculate $y = b^x \bmod p$. But given y , it is hard to find the corresponding x . The function $y = b^x \bmod p$ is therefore said to be *one way*.

This is how Alice and Bob agree on a secret cryptographic key, k .

- Alice and Bob agree to use certain values of a and p . These are not secret. They might be fixed for all time.
- Alice generates an arbitrary but secret number, a and sends $A = b^a \bmod p$ to Bob.
- Bob generates an arbitrary but secret number, b and sends $B = b^b \bmod p$ to Alice.
- Alice calculates $k_a = B^a \bmod p$, which she can do because she knows a and Bob sent her B .
- Bob calculates $k_b = A^b \bmod p$, which he can do because he knows b and Alice sent him A .

Now it is easy to see that

$$k_a = (b^b)^a \bmod p = (b^a)^b \bmod p = k_b = (\text{say}) k.$$

and k is the cryptographic key which Alice and Bob have agreed upon.

Now so far we have not specified values for b and p . It turns out that to maximise the hardness of finding x from y in the expression $y = b^x \bmod p$, we should

- Choose p to be a *perfect prime*, which means that $(p-1)/2$ is prime as well.
- Choose b to be a *primitive root mod p*, in other words an integer such that all the integers of the form $b^i \bmod p$ are distinct for $i = 0, 1, 2, \dots, p-2$.

It is not particularly easy to generate these values, but it only needs to be done once⁵.

It is a reasonable way to proceed for each entity to generate its secret a once for all, and to publish $A = b^a \bmod p$, which we might call its *partial key parameter*. Such a parameter could be specified on a key certificate (section 10.3) along with other data.

15.1 Caveat

There is an important caveat with this protocol. While it is easy for two entities who have never met, to use this protocol to agree on a secret during an observed session, the problem remains that they do not know the identity of the other entity. It might be an enemy.

So when they use this protocol they should use digital signatures to sign the values of A and B which they send. This means that the entities must previously have obtained copies of each other's public keys.

15.2 Elliptic Curve Analog

There is a proposed elliptic curve key exchange procedure [10] analogous to the Diffie Hellman method.

16. Standards

There are a number of data security standards worth drawing attention to. It is beyond the scope of this treatment to go into details.

ISO 7498-2

(X.800) Specifies the security architecture for the OSI basic reference model.

ISO 8372

Specifies modes of operation - ECB, CBC, CFB and OFB for a general 64-bit block cipher.

ISO 8730

Wholesale banking standard for MACs. Equivalent to ANSI X9.9

ISO 8731

Deals with certain aspects of MACs.

ISO 8732

⁵ The author can supply software to do this.

Standard for cryptographic key management in wholesale banking. Derived from ANSI X9.17.

ISO 9564

This was the basis for ANSI X9.8. Specifies management of PINs.

ISO 9807

Message authentication in retail banking. Similar to ANSI X9.19.

ISO 10126

Equivalent of X9.23. Confidentiality protection of parts of financial messages.

ISO 10202

Deals with Chipcards for financial transactions. ISO 10202-7 deals with cryptographic key management.

ISO 11131

Deals with sign-on authentication. Analogous to ANSI X9.26, but not DES specific.

ISO 11166

Deals with cryptographic key management for banking. Uses symmetric algorithms. Developed from ISO 8732.

ISO 11568

Deals with cryptographic key management for retail banking. More generalised than X9.24.

ISO/IEC 9594-8

Same as X.509. Defines simple authentication techniques based on passwords, and so-called strong methods which do not reveal secret values.

ISO/IEC 9796

Specifies a generic mechanism for digital signature schemes using message recovery.

ISO/IEC 9797

Defines a message authentication code (MAC) based on CBC mode of operation of a block cipher.

ISO/IEC 9798

Specifies entity authentication mechanisms based on symmetric encryption (9798-2) and public key signature algorithms (9798-3), a MAC (9798-4).

ISO/IEC 9979

Specifies procedures for registering encryption algorithms in an official ISO register.

ISO/IEC 10116

Extends ISO 8372 to any n-bit block algorithm.

ISO/IEC 10118

10118-1 specifies common definitions and general requirements.

10118-2 specifies two generic constructions based based on n-bit block ciphers.

ISO/IEC 10181

X.810 -X.816. Concerned with security frameworks.

ISO/IEC 11770

Deals with certain cryptographic key management and cryptographic key establishment mechanisms.

ISO/IEC 13888

Deals with certain non-repudiation services. Non-repudiation of origin, of delivery and acting as third party.

ISO/IEC 14888

Deals with some aspects of digital signatures.

ANSI X3.92

Specifies the DES algorithm. See glossary section 17.

ANSI X3.106

Specifies modes of operation of DES.

ANSI X9.8

Deals with PIN management.

ANSI X9.9

Deals with message authentication for wholesale banking using DES.

ANSI X9.17

Deals with cryptographic key management for wholesale banking. Was the basis for ISO 8732.

ANSI X9.19

Specifies a DES-based MAC algorithm for retail banking.

ANSI X9.23

Deals with message encryption for wholesale banking.

ANSI X9.24

Deals with cryptographic key management for retail banking.

ANSI X9.26

Deals with sign-on authentication for wholesale banking.

ANSI X9.28

Deals with multi-centre cryptographic key management for wholesale banking.

ANSI X9.30-1

Deals with the digital signature algorithm (DSA).

ANSI X9.30-2

Deals with a secure hash algorithm (SHA) for DSA.

ANSI X9.31-1

The RSA signature algorithm.

ANSI X9.32-2

Deals with hashing algorithms for use with RSA.

ANSI X9.42

Cryptographic key management using the Diffie Hellman algorithm.

ANSI X9.45

Deals with attribute certificates.

ANSI X9.52

Deals with triple-DES modes of operation.

ANSI X9.55

Deals with certificate extensions and CRLs.

ANSI X9.57

Deals with certificate management.

FIPS 46

This is the original DES standard.

FIPS 74

Guidelines for implementing and using DES.

FIPS 81

DES modes of operation.

FIPS 112

Deals with password usage.

FIPS 113

Deals with data authentication using a CBC-MAC.

FIPS 140-1

Deals with design and implementation of cryptographic modules for protecting US government unclassified information.

FIPS 171

Specifies a subset of the key distribution techniques of ANSI X9.17 for use by US federal government departments.

FIPS 180 and 180-1

Specifies the hash algorithm for the Digital Signature Standard FIPS 186. FIPS 180-1 is an improved version.

FIPS 185

The Escrowed Encryption Standard.

FIPS 186

The Digital Signature Standard.

17. Glossary

Many of the terms defined here are used in this document. Many terms not used here are also included as an aid to exploring other cryptographic literature.

Access Control	Restricting access to resources to privileged <i>entities</i> .
Algorithm	In the present context <i>Algorithm</i> means <i>Cryptographic Algorithm</i> (below) unless otherwise stated. Out of context, <i>Algorithm</i> is to computer science as <i>Recipe</i> is to cookery. It is a very general term meaning an unambiguous specification for performing some manipulation of data, including specifying the nature and format of the data.
Alice, Bob	When describing security <i>protocols</i> , for historical reasons the two <i>entities</i> are usually called <i>Alice</i> and <i>Bob</i> (i.e. rather than merely A and B). If necessary, other characters are introduced, often with names beginning C, D, E, ... or with letters suggesting certain properties, for example <i>Tom</i> for <i>trusted</i> .
Asymmetric Key Cryptosystem	See <i>Public Key Cryptosystem</i> .
Authentication	The verification of the claimed identity of an <i>entity</i> (<i>Entity Authentication</i>) and/or of information generated or guaranteed by that <i>entity</i> (<i>Data Origin Authentication</i>). The latter is stronger than Integrity Checking which is implied by data origin authentication since the origin of data is altered when it is manipulated. <i>Data Origin Authentication</i> is also known as <i>Message Authentication</i> .
Bad Guy	See <i>Enemy</i> .
Belief	A state into which an <i>entity</i> may be set or not set, which affects its behaviour in <i>protocols</i> . A belief corresponds roughly to a conclusion or axiom held regarding the trustworthiness of another <i>entity</i> , the validity of a <i>certificate</i> , or similar. <i>Entities</i> may conduct <i>protocols</i> with other <i>entities</i> to arrive at beliefs, or they may check the validity of <i>certificates</i> using <i>digital signatures</i> , or the beliefs may simply be built in axioms (e.g. that <i>Alice</i> is assumed to be trustworthy).
Block Cipher	A <i>cryptographic algorithm</i> in which data can be processed in independent blocks. The <i>Block Size</i> - the number of bits in the block - is the smallest unit in which it is possible to <i>encrypt</i> . See also <i>Stream Cipher</i> .
CBC	Cipher Block Chaining. See <i>Chaining</i> .
Certificate	Electronic document issued by a trusted <i>entity</i> endorsing some <i>information</i> . It may involve a <i>Digital Signature</i> .
Certification	Endorsement of <i>information</i> by trusted <i>entity</i> . It may involve the issue of a <i>certificate</i> .
Certification Authority	A trusted party which issues <i>Key Certificates</i> or other digitally signed documents.
CFB	See Cipher Feedback.

Chaining	Chaining, or <i>Cipher Block Chaining</i> (CBC) is a way of detecting alteration of long messages. It operates by linking together all the ciphertext blocks of a long message in such a way that if any of them is altered in transit, or if any blocks are inserted or deleted, then all blocks from that point onwards are damaged, and plaintext cannot be recovered from them, thereby bringing to light the modification.
Challenge-Response	A <i>protocol</i> for verifying the identity of an <i>entity</i> called the <i>Claimant</i> to the satisfaction of an <i>entity</i> called the <i>Verifier</i> . It consists of the <i>verifier</i> issuing a question (the <i>Challenge</i>) and checking the reply (the <i>Response</i>) from the <i>claimant</i> . The idea is that only the legitimate <i>claimant</i> can provide the correct response because he/she alone has been given some identifying piece of secret information, such as a <i>cryptographic key</i> . The protocol is often conducted using a gadget called a <i>transponder</i> . For example, vehicle ignition keys sometimes contain a transponder in the head.
Cipher	A general-purpose word used to mean a <i>cryptographic algorithm</i> , a <i>cryptosystem</i> or a piece of <i>ciphertext</i> . Sometimes spelt <i>cypher</i> .
Cipher Block Chaining	See Chaining.
Cipher Feedback	CFB is a method of stream cipher in which each symbol of ciphertext is formed by XORing it with a complex function of several earlier ciphertext symbols and a cryptographic key. It is self-synchronising and is often used in telecommunications links.
Ciphertext	A protected form of a message in which it is unintelligible without being converted back to <i>plaintext</i> . Also known as <i>Cryptogram</i> . See <i>decrypt</i> .
Claimant	See <i>Challenge-Response</i> .
Coin-flips	Data bits which, regardless of how they were actually generated, are statistically indistinguishable from the outcome of flipping a fair coin.
Composite	An integer which is not <i>Prime</i> is said to be <i>Composite</i> .
Confidentiality	A service used to keep the content of information from all but those authorised to receive it. It is synonymous with <i>Secrecy</i> and <i>Privacy</i> . It is provided by <i>Encryption</i> .
Conventional Cryptosystem	<i>Symmetric-Key Cryptosystem</i> .
Cryptanalysis	The science, skill and practice of attempting to, or succeeding in violating the security of a cryptosystem, typically by trying to find the cryptographic key from matched plaintext-ciphertext pairs, or less commonly of trying to recover plaintext from ciphertext without knowing the cryptographic key. It can also sometimes mean the violation of a <i>protocol</i> . A <i>cryptanalyst</i> is one who practices

	cryptanalysis. Hence <i>Cryptanalytical</i> , <i>Cryptanalytically</i> . When restricted to ciphers, Cryptanalysis is popularly called <i>codebreaking</i> . Cryptanalysts are one type of <i>Enemy</i> or <i>bad guy</i> .
Cryptogram	See <i>Ciphertext</i> .
Cryptography	The science and practice of keeping messages secure. Hence <i>Cryptographic</i> .
Cryptology	The study of <i>cryptography</i> and <i>cryptanalysis</i> and all related subjects. Hence <i>cryptologic</i> . The term is very general and all-embracing.
Cryptographic Algorithm	A mathematical function used to change plaintext into ciphertext (<i>Encryption</i>) or <i>vice versa</i> (<i>Decryption</i>). The function normally needs an extra parameter called the <i>cryptographic key</i> .
Cryptographic Key	A parameter used to encrypt plaintext into ciphertext, or to convert ciphertext into recovered plaintext. Unavailability of the appropriate cryptographic key is supposed to make it impossible for an <i>entity</i> to recover the plaintext from ciphertext.
Cryptosystem	A set-up consisting of a matched <i>encryption</i> algorithm, <i>decryption</i> algorithm and <i>key management system</i> .
Data	See <i>Information</i> .
Data Integrity	The property whereby data has not been altered in an unauthorised manner since it was created, transmitted or stored by an authorised source. If the data is a message then we can speak of <i>Message Integrity</i> .
Data Origin Authentication	See <i>Authentication</i> .
Decrypt	The process of converting ciphertext into recovered plaintext using a cryptographic algorithm and a cryptographic key.
Decryption Algorithm	A cryptographic algorithm used to decrypt.
DES	FIPS 46 - the original <i>Data Encryption Algorithm</i> - the name given to a well known symmetric, cryptographic block algorithm published in 1971. It is a well documented and well understood algorithm which has become a <i>de facto</i> standard in the financial community. It has an <i>iterated block structure</i> with what is known as a <i>Feistel</i> architecture. It uses 16 rounds, a 56 bit key size and a 64 bit block size.
Differential Cryptanalysis	A method of cryptanalysis developed by Biham and Shamir [4] which exploits the relationship between differences of plaintext pairs and the differences their corresponding ciphertexts.
Digital Signature	Data appended to a message that allows a recipient of the message to prove the source and integrity of the message. It

may involve the use of *Number-Theoretic* processes and *Hash Functions*. The *RSA* cryptosystem may be used to provide digital signatures.

ECB

See *Electronic Codebook*, and *Modes of Operation*.

Effective Key Size

If a cipher can be broken using around 2^m or fewer computational steps, where m is no greater than the key size, then m is the effective key size (in bits). The *Effective Key Size* may be much smaller than the actual key size, and it will never be larger.

Electronic

Symbols are described as *Electronic* to mean “not-printed on paper” but instead represented by the memory state of computing machines. Electronic symbols are the *raison d’etre* of IT-based systems, and are able to be stored, transmitted, received and destroyed. *Information* is often assumed to be Electronic in IT security contexts. Electronic Mail (email) is an example.

Electronic Codebook

The simplest *Mode of Operation* of a block cipher in which the input is plaintext and the output is ciphertext. Abbreviated to ECB.

Encrypt

The process of converting plaintext into ciphertext using a cryptographic algorithm and a cryptographic key.

Encryption Algorithm

A *cryptographic algorithm* used to *encrypt*. Sometimes referred to simply as *Algorithm* in cryptographic texts.

End-To-End Encryption

Encrypting information once-for-all so that its passage through several interfaces need not be accompanied by decryption/encryption phases at each interface, but instead requiring decryption only when it reaches its final destination. This is in contrast to *Link Encryption*, or *Link-By-Link Encryption*, in which the data is decrypted and re-encrypted at each interface, with consequent increase of risk.

Enemy

An *entity* whose aims are to subvert the security of a cryptosystem, or other electronic security system, perhaps by *cryptanalysis*. Also known as *bad guy*. By contrast, the designers and manufacturers of cryptosystems and their legitimate customers, terminals, smart cards and agents (i.e. all associated *entities*) are sometimes called the *good guys*.

Entity

A person, engine management unit, terminal, server, tool, random number generator, smartcard, card reader etc., taking part in a *protocol* or providing or modifying *information*.

Entity Authentication

See *Authentication*.

Exhaustive

A search, test or attack is exhaustive if it covers all possibilities, e.g. searching through all possible cryptographic keys. This is contrasted with *statistical*

where only a (hopefully representative) subset of all possibilities is examined.

Feistel Architecture

Type of block cipher in which the plaintext is initially partitioned into two equal halves, (L_0, R_0) . There then follow n rounds, with round i having (L_i, R_i) as input and (L_{i+1}, R_{i+1}) as output, and where

$$L_{i+1} = R_i, \quad \text{and} \quad R_{i+1} = L_i \oplus f(R_i, K_i)$$

where $f()$ is a complicated function, and K_i is a subset of the key bits. The main feature of the Feistel architecture is the fact that it is unconditionally reversible, without there being any constraints of reversibility on the function $f()$.

Good Guy, Bad Guy

See *Enemy*.

Hash Function

A Cryptographic Hash Function $y = H(M)$ where M is a message is a function $H(\cdot)$ which takes a message, M of arbitrary and unlimited length, and generates a block, y of fixed, predetermined length, say n bits, in such a way that knowing y , it is infeasible to find any message M_1 such that $y = H(M_1)$, and also infeasible to generate any two messages M_1 and M_2 such that $H(M_1) = H(M_2)$. It is used as part of a *Digital Signature*.

Honest

An entity is said to be *honest* if it uses protocols correctly, and for the purpose for which they were intended, and it supplies only correct information when asked. A protocol is said to be honest if it has been correctly carried out by honest entities. A coin is said to be honest if the probability of flipping a head is exactly one half, statistically independently of all other flips.

Identification

An alternative name for *entity authentication*. Identification has these properties.

- If Alice successfully authenticates herself to Bob, then Bob will accept Alice's identity.
- Bob cannot reuse the exchange with Alice to successfully impersonate Alice.
- It is infeasible for (say) Carol to play the role of Alice, causing Bob to accept Carol as Alice.
- These features remain true even if large numbers of observations of exchanges between Alice and Bob have been observed.

Impersonation

See *Masquerade*.

Information

An ordered collection of (typically binary) symbols, capable of being transmitted and/or stored, and having some meaning, significance or value. Sometimes, but not invariably the term *Data* is used as a synonym.

An important property of information is that, unlike mass and energy, it is *non-conservative*. In other words it can be created and destroyed. This fact is at the root of most security issues concerned with information. Money is but a form of information - it exists only in the records of financial institutions - and accountancy practices can be viewed merely as an, at best partially successful attempt to endow money with conservative properties.

Information Technology

The technology concerned with the transmission, storage, processing, printing and security of information.
Abbreviation IT.

Insertion

Insertion means the introduction of unauthorised data into messages, possibly during a session which is authenticated at the beginning. Insertion destroys the *integrity* of the message. *Chaining* blocks gives some protection against insertion.

Integrity

A message exhibits *Integrity* if it has not suffered unauthorised modification. The term *Data Integrity* is the name of a service which provides various degrees of assurance that data has not been altered without authority.

Intractable

A class of problem is said to be intractable if the resources needed to carry it out grow faster than any power of n , where n is the number of bits needed to specify an instance.

IT

Information Technology.

Key

Cryptographic Key (unless otherwise stated).

Key Authority

An entity authorised by its client entities to issue *key certificates* and in effect trusted by these client entities.

Key Certificate

A digital document bearing the *digital signature* of a trusted authority, stating in effect the numerical value of the public key of a given entity.

Key Lifetime

The maximum permissible amount of plaintext encrypted, or the maximum permissible amount of time, between changing cryptographic keys, as a damage-limitation measure.

Key Management

A set of defined processes for generating, storing, distributing, changing and destroying *cryptographic keys*.

Key Size

The number of bits in a cryptographic key. If this number is n then the number of keys is 2^n , also said to be the *size of the keyspace*. See also *Effective Key Size*.

Known Plaintext Attack

A cryptanalytic attack in which the cryptanalyst has a large amount of matching plaintext-ciphertext, and knows the cipher, and has powerful computational resources.

Linear Cryptanalysis

A method of breaking a suitably susceptible cipher, first proposed by Matsui [14], which involves finding linear Boolean functions of certain key bits, plaintext bits and

ciphertext bits, such that these functions are slightly asymmetric in their probability of taking on the values 0 and 1 for random y, x, k satisfying $y = E_k(x)$.

Lifetime	See <i>Key Lifetime</i> .
Link Encryption	The opposite of <i>End-To-End Encryption</i> .
Masquerade	An attack in which a hostile <i>entity</i> impersonates a legitimate one in protocols. Also known as <i>Impersonation</i> .
MAC	See <i>Message Authentication Code</i> .
Message Authentication	Synonym for <i>Data Origin Authentication</i> .
Message Authentication Code	A MAC is a code to provide <i>Data Origin Authentication</i> by means of a <i>Symmetric Key Cryptosystem</i> . The sender of a message can prove message authenticity to the receiver if both trust each other. It cannot provide <i>Third-Party Authentication</i> . Only Public Key Cryptography can do this.
Message Integrity	See <i>Data Integrity</i> .
Mod	Short for “modulo”. $A \text{ mod } B$ is the remainder when integer A is divided by integer B . For example if we count “modulo 5” we get the sequence 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, and so on, indefinitely.
Modes of Operation	Ways of using a block encryption algorithm. See CBC, CFB, OFB, ECB.
Modular Exponentiation	Calculations of the form $y = x^e \text{ mod } M$, where x, y and e may be very large numbers. Modular Exponentiation may be carried out without difficulty on very large numbers, and is the basis of many <i>Public Key Cryptosystems</i> .
Monoalphabetic	A <i>Monoalphabetic Substitution Cipher</i> (or simply a <i>Monoalphabetic</i>) replaces each symbol of an alphabet with another. This assignment constitutes the cryptographic key, and while this is fixed, so is the assignment. So if “ U ” replaces say “ M ” in one place, it replaces it everywhere as long as the same cryptographic key is in force. Simple monoalphabetic are notoriously weak ciphers.
Non-Repudiation	Preventing the denial by an <i>entity</i> of a previous commitment or action.
Number Theory	A branch of mathematics concerned with the special properties of integers (whole numbers like 0, 1, 2, 3, ... and their negatives). Its significance in the present context is that many cryptographic algorithms (see for example <i>Public Key Cryptosystem</i>) operate by interpreting long strings of digits as very large numbers and process them using number theory. Cryptosystems based on number theory often make use of very large prime numbers.
OFB	See <i>Output Feedback</i> .

Output Feedback	A method of generating a Keystream from a block cipher. Successive outputs from a block encryption algorithm are fed back into the input. Abbreviated to OFB.
One Time Pad	A stream cipher in which the keystream consists of a sequence of truly random bits which does not repeat, and which is known only to the sender and the intended recipient. It is the only provably unbreakable cipher. However it is logistically very difficult to use since it merely trades the problem of secret communication for the equally difficult problem of exchanging the enormous cryptographic keys implied by a one time pad.
One Way Function	A mathematical function, $y = f(x)$, such that, given x , it is easy to find y , but given y , finding x is <i>intractable</i> .
Plaintext	A message in its original or unprotected form. When <i>ciphertext</i> is correctly decrypted the result is <i>recovered plaintext</i> .
Round	A single pass (i.e. one of several passes) through a repetitive procedure during the execution of a cryptographic algorithm. The <i>DES</i> algorithm contains 16 rounds.
Partial Key Parameter	A number of the form $A = b^a \text{ mod } p$, where a is a secret, random number, p is a perfect prime, and b is a primitive root of p . It is used in the <i>Diffie-Hellman Key Exchange protocol</i> .
Perfect Prime	A prime p such that $(p-1)/2$ is also prime.
Prime	A <i>Prime Integer</i> (or <i>Prime</i>) is an integer with no divisors except trivially, itself and 1. There are infinitely many primes, the first few being 2, 3, 5, 7, 11, 13, 17, 19. Some <i>Cryptosystems</i> (e.g. <i>RSA</i>) make use of large primes, and of <i>Number Theory</i> . An integer which is not prime is said to be <i>composite</i> .
Protocol	A cryptographic protocol is a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more <i>entities</i> to achieve a specific security objective.
Public Key Cryptosystem	(Abbreviation <i>PKC</i> .) Also known as <i>Asymmetric</i> , or <i>Two-Key Cryptosystem</i> . A cryptosystem in which there are 2 keys and 2 algorithms - one key and algorithm to encrypt, the other pair to decrypt - and in which disclosure of the encryption key does not provide enough information to discover the decryption key. The encryption (or <i>public</i>) key is commonly published. The decryption (or <i>private</i> or <i>secret</i>) key is kept secret. The most celebrated PKC is the <i>RSA</i> algorithm (<i>RSA</i> = Rivest, Shamir and Adleman) which makes use of <i>Number Theory</i> . PKCs can be used to provide <i>Third-Party Authentication</i> .

Random Number	Can mean a number chosen by a process known to generate numbers at random, such as tossing a coin, throwing a dice, noting the time of emission of radioactive particles, measuring the noise from a zener diode, etc. Can mean <i>number of a type which passes some statistical test of randomness</i> . There is problem with the latter definition in that genuinely randomly chosen numbers will sometimes fail such tests. Commonly misused term. Often really means <i>unpredictable</i> number, or even <i>number not previously used</i> . So-called Random Numbers are sometimes generated by <i>Random Number Algorithms</i> . Of course such numbers are entirely predictable. However they may pass tests of randomness. All in all, there are so many opportunities for confusion that the term <i>Random Number</i> is best avoided except when speaking loosely.
Random Number Algorithm	An algorithm used to generate numbers which pass some test of randomness. Such an algorithm is normally initialised from a <i>seed</i> or initial number. The so-called random numbers generated by such an algorithm are of course totally predictable.
Related Key Attack	A cryptanalytic attack on a cipher which can only succeed if the cryptographic algorithm selects cryptographic key bits in a regular or cyclic manner from <i>round</i> to <i>round</i> . The threat of this attack is not reduced by increasing the number of <i>rounds</i> . See ref [3].
Round, Rounds	Some cryptographic algorithms are organised into routines, or <i>rounds</i> , which are repeated, perhaps with slight variations, a number of times, called the <i>number of rounds</i> .
RSA	See <i>Public Key Cryptosystem</i> .
Sbox	A substitution table used in some cryptographic algorithms, usually as a convenient way to represent a mathematical function with certain desired properties, such as nonlinearity.
Security	In an Information Technology context Security means <i>Availability, Integrity and Confidentiality</i> .
Seed	A number used to initialise a <i>Random Number Algorithm</i> .
Session	A session is a single period of communication, normally distinguished by opening a channel, establishing contact, identifying the other entity, exchanging cryptographic keys, sending and receiving data, and closing the channel.
Session Key	Cryptographic key exchanged for use during one session. It is not essential to use a fresh cryptographic key for each session, but it is a fairly common practice.
Stream Cipher	A cryptographic method whereby the data is processed as a continuous stream of bits or bytes rather than as blocks. Usually each bit of ciphertext is formed by exoring a

plaintext bit with a keystream bit. The protection is thus provided by the keystream. See also *Block Cipher* and *One Time Pad*.

Strong Cipher

A cipher for which the *Effective Key Size* is essentially the actual key size, and where that key size is sufficiently large. A cipher for which there is no known method of breaking it which requires substantially less resources than searching through all possible cryptographic keys.

Substitution Table

See Sbox.

Symmetric Key Cryptosystem

A cryptosystem in which the same key is used to encrypt and decrypt. In other words not a *Public Key Cryptosystem*.

Third-Party Authentication

Normal *Message Authentication Codes* provide authentication only to the recipient, and only where the sender and recipient trust each other. When they do not trust each other, the ability to prove authenticity of a message to an impartial third party without disclosing secret information (such as a secret cryptographic key) is called *Third Party Authentication*. It requires the use of *Public Key Cryptography*.

Timestamp

Verb meaning to attach an authenticated record of the time of creation or existence of information. Also noun meaning the record itself.

Timestamping

Recording the time of creation or existence of information.

Transponder

A gadget used by the claimant to assist in a *challenge-response* protocol.

Two-Key Cryptosystem

See *Public Key Cryptosystem*.

Unicity Distance

The minimum length (in bits) of ciphertext needed on average, when applying a ciphertext only attack, such that only one plaintext could possibly correspond with the given ciphertext.

Validation

A means to provide timeliness of authorisation to use or manipulate information or resources.

Verifier

See *Challenge-Response*.

Weak Cipher

A cipher is said to be weak if its *Effective Key Size* is significantly smaller than its actual key size - in other words if there exists a known method of breaking it which needs substantially less resources (processing speed, memory or time) than searching through all possible cryptographic keys. Also a cipher with a small true key size.

Weight

The weight of a byte, number or vector is the number of 1s in it when it is written in binary.

18. References

- [1] Alfred J Menezes, Paul C van Oorschot, Scot A Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
- [2] American Bankers Association, *Financial Institution Message Authentication* (Wholesale) , X9.9, ANSI, 1986.
- [3] E Biham, *New Types of Cryptanalytic Attack using Related Keys*, Journal of Cryptology, vol.7 No. 4, 1994, pp229-246
- [4] E Biham and A Shimir, *Differential cryptanalysis of DES-like cryptosystem*, Journal of Cryptology, 4 (1991), 3-72.
- [5] W. Diffie and M E Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, 22, (1976), 74-84
- [6] J H Ellis, *The Story of Non-Secret Encryption*, available from website <http://www.cesg.gov.uk> from December 17th 1997.
- [7] H Feistel, W A Notz and J L Smith, *Some cryptographic techniques for machine-to-machine communications*, Proc. IEEE, 63, (1975), 1545-1554.
- [8] FIPS PUB 180-1. The Secure Hash Algorithm. Federal Information Processing Standard 180-1, 1995.
- [9] FIPS PUB 186. *The Digital Signature Algorithm*. Federal Information Processing Standard 186, 1994.
- [10] IEEE p1363 Draft Standard. *Standard Specifications For Public Key Cryptography* Institute of Electrical and Electronics Engineers, Inc. 1998.
- [11] D Knuth, *The art of Computer Programming, vol 2., Seminumerical Algorithms*, Addison-Wesley, 1981
- [12] X Lai, *On the design and security of block ciphers*, ETH Series in Information Processing, J L Massey ed. Vol 1, Hartung-Gorre Berlag Konstan\, Technische Hochschule (Zurich) 1992.
- [13] J L Massey, *SAFER K-64 A byte oriented block-ciphering algorithm*, R Anderson, ed. Fast Software Encryption, Cambridge Security Workshop (LNCS 809), 1-17, Springer-Verlag, 1994.
- [14] M Matsui, *Linear cryptanalysis method for DES cipher*, Advances in Cryptology, Eurocrypt '93, 386-397 1993.
- [15] S Miyaguchi, *The FEAL cipher family*, Advances in Cryptology - Crypto '90 (LNCS 537), 627-638, 1991.
- [16] J H Moore, *Protocol failures in Cryptosystems*, Contemporary Cryptology, The science of Information Integrity, Ed. G J Simmons, 541-558, IEEE Press 1992.

- [17] See the Rijndael home page <http://www.esat.kuleuven.ac.be/~rijmen/rijndael/>
- [18] R L Rivest, *The RC5 encryption algorithm*, B Preneel, ed. Fast Software Encryption, Second International Workshop (LNCS 1008), 86-96, Springer-Verlag, 1995.
- [19] R L Rivest, A Shamir and L Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 21 (1978), 120-126.
- [20] G J Simmons, *Proof of Soundness of Cryptographic Protocols*, Journal of Cryptology, Vol. 7, No. 2, 1994.
- [21] H E Rose, *A Course in Number Theory, Second Addition*, Oxford Science Publications, 1994.
- [22] P W Shor, *Algorithms for Quantum Computation: Discrete Log and Factoring*, Proc.of the 35th Symposium of Foundations of Computer Science, 124-134, 1994.
- [23] D J Wheeler and R M Needham, *TEA, a tiny encryption algorithm*, B Preneel, ed. Fast Software Encryption, Second International Workshop (PCNS 1008), 363-266, Springer-Verlag, 1995.