The ISO/OSI Security Architecture

> *"Remote exploit + local root exploit ⇒ remote root exploit."*
>
> *— Olde saying handed down through the ages.*

As well as defining their seven-layered model, the ISO/OSI group also defined a range of terminologies forming their **ISO/OSI Security Architecture**.
It includes the requirements:

- *data confidentiality* - protects data as it traverses the network from being disclosed to incorrect parties. Even the presence of particular communication sequences between parties should not be identified.
- *data integrity* - protects the data from modification or removal while in the network,
- *data origin authentication* - validates the sender of the data,
- *data receiver authentication* - validates the receiver of the data,
- *peer-entity authentication* - validates all network components, such as hardware routers and peer software components through which a data stream must travel, and
- *non-repudiation* - creates and verifies evidence that the claimed sender sent the data, that the intended receiver did receive it, and that neither can deny that this occurred.

NOTE: *the core TCP/IP internetworking suite meets **none** of the requirements of the ISO/OSI Security Architecture. Support for additional services is evolving, primarily at the Application Layer, but changes cannot be easily made to lower layers.*

Cryptography's Role in Networking

> *"Cryptography is the science of making the cost of improperly acquiring or altering data greater than the potential value gained. The value of information usually drops with time, and cryptography makes the time required to obtain data in unauthorized ways long enough to decrease its value well below the money spent on obtaining it."*
>
> *— Jalah Feghhi, Digital Certificates*

The need for cryptography to provide security arises with the possibility of stolen hardware, wiretapping, the broadcast mechanisms of LANs and WLANs, and network traffic passing through foreign networks.

We assume that an adversary is able to :

- Copy data from disk storage for remote analysis,
- Passively listen (only) on broadcast channels (such as wired-Ethernet and WiFi),
- Aggressively monitor traffic though intermediate routers or workstations (situated anywhere on a message's path),
- Actively replay, modify or insert their own messages into the message stream.

Cryptography provides solutions to *most* of these problems.

So where should the encryption be performed?

- Users encrypting individual files stored in a standard file-system,
- File-systems encrypting all data before writing it to disk,
- Datalink and Network layers: in switches and routers (e.g. VPNs),
- Session Layer: with end-to-end data conversion (e.g. SSL),
- Application Layer: in programs such as email agents (e.g. PGP).

Basic Cryptographic Terminology

We use an *encryption function* and a *key* to convert the *plaintext* (the input) into the *ciphertext* (the output).

The intended receiver hopes to quickly and correctly reverse the process - something that adversaries should be unable to do.

We assume that the adversary *knows* the encryption function (process) being used, and that the *key* is secret and changed frequently. The *key length* is expressed in bits:

Technology	Key lengths	Possible keys	Key length
ATM PIN	4 decimal digits	10,000	14 bits
old-style Unix passwords	8 characters	$126^8 = 6.3x10^{16}$	56 bits
Unix passwords with MD5	any number of characters	unlimited, though duplicates	128 bits

The following attacks against cryptography are common. Each has the goal of determining the cryptographic key(s), so that existing data may be exposed, or future messages decrypted without the repeated effort:

- *Known plaintext attack* - the cryptanalyst (fancy name for an adversary with a Maths degree) has (or determines) a block of plaintext and its corresponding block of ciphertext. This may seem unlikely, but regularly exchanged encrypted messages have fixed or predictable payloads (e.g. email headers, VPN-session establishment).
- *Chosen plaintext attack* - the cryptanalyst can have their intended victim unknowingly encrypt fixed, known blocks of data.
- *Differential analysis* - a kind of plaintext attack involving many very similar plaintexts being encrypted, and their resulting ciphertexts being compared.

If there is no separate *integrity check* on the encrypted data - an attacker may be able to alter it so that it decrypts to data of the attacker's choice.

Simple Substitution :

Caesar Cipher : `ch = (char)((ch + K) % 26);`

```
PT : abcdefghijklmnopqrstuvwxyz
CT : DEFGHIJKLMNOPQRSTUVWXYZABC
```

Here `K = 3`.

Monoalphabetic Substitution :

Each symbol maps to some other symbol. The *key* is simply a 26 letter string from the alphabet.

```
PT : abcdefghijklmnopqrstuvwxyz
CT : QWERTYUIOPASDFGHJKLZXCVBNM
```

Here there are 26! = 4×10^{26} possible keys.

At 1msec per solution, a brute force approach would take 10^{13} years.

However, in English and many other natural languages, text statistics are available on the most frequently used letters, (26^2) digraphs, (26^3) trigraphs and words. To break simple codes we :

- count frequencies of each letter and match *e's, t's*, etc.
- count digraphs (th, er, on, an, re, he, in, ed, nd, ha, at, en, es, of, or...) and trigraphs (he, and, tha, ent, ion, tio, for, nde, has, nce...)
- match *th, the, in, and,* ...

The movie *Windtalkers* focuses on the use of the natural language ofNavajo Indians as a code in WWII.
Speakers of Klingon claim that their language is, for other reasons, indecipherable (in practice).

The Influence of Computers on Cryptography

The first role of computers was to gather statistics and perform 'brute-force' searches of ciphertext using some heuristics.

Traditionally, convoluted, obscure, and *undisclosed* algorithms requiring long keys were used to encrypt text. More recently the focus has changed to support simple, *open* algorithms, but with complex 'solutions'.

The algorithmic inverses themselves are intended to be so complex that brute-force techniques take millions of years to succeed.

- An *algorithm's strength* is not simply derived from its keys' length, but from its peer evaluation and public review.
- A *weak algorithm* is one whose algorithm and implementation are not available, and whose strength would be compromised if these were made public.

In 1883 Auguste Kerckhoff stated as one of his six axioms of cryptography:

> "*If the **method** of encipherment becomes known to one's adversary, this should not prevent one from continuing to use the cipher as long as the **key** remains unknown.*"

Symmetric Ciphers

The Data Encryption Standard (DES, 1976) is known as a *symmetric cipher*, often a *private key algorithm* - in which the sender and receiver use the same key that must be kept private.

In general, longer keys provide stronger encryption, but it is a mistake to imply the strength of an encryption algorithm in terms of bits alone. Brute force attacks are the most successful.

Some popular examples of symmetric ciphers:

- DES: a block based cipher of 64bit blocks in, 64bit blocks out, 56 bit key filled to 64bits (8 odd-parity bits).
- Triple-DES: encrypts the same plaintext with DES three times. Three or two keys are provided, the plaintext is encrypted with the 1st key, decrypted with the second, and finally encrypted with the third (or 1st again).

  A double-DES scheme (with only 2 keys) does not require 2^{2n} brute-force tests but 2^{n+1} tests with a *meet-in-the-middle* attack.
- Ron Rivest's RC2 block cipher employs keys up to 1024 bits, and executes at a speed independent of key length.
- Ron Rivest's RC4 stream cipher (as used in WiFi's WEP encryption) employs keys of 40 to 256 bits, but has the property that if two messages are encrypted with the same RC4 key, their encryptions are related in a known way.

The DES Algorithm

As an example of the first *official* deployment of encryption, we'll consider the **Data Encryption Standard**, DES.

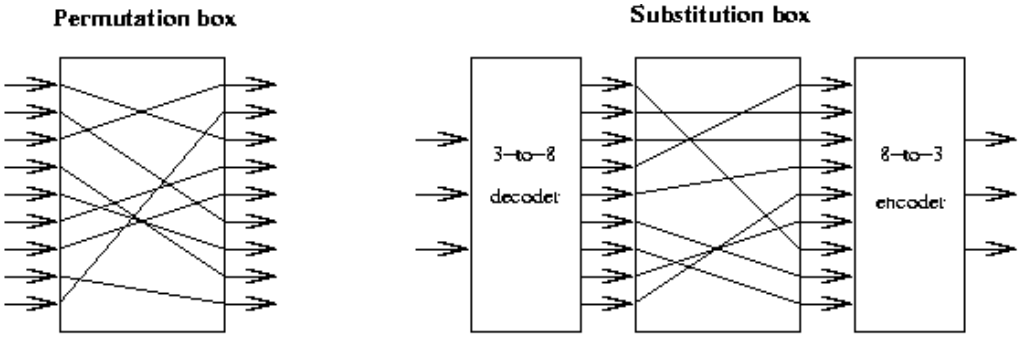
In 1977 the US Government adopted a product cipher from IBM and later that year was defined as as the *official encryption standard* by the US Standards Bureau.

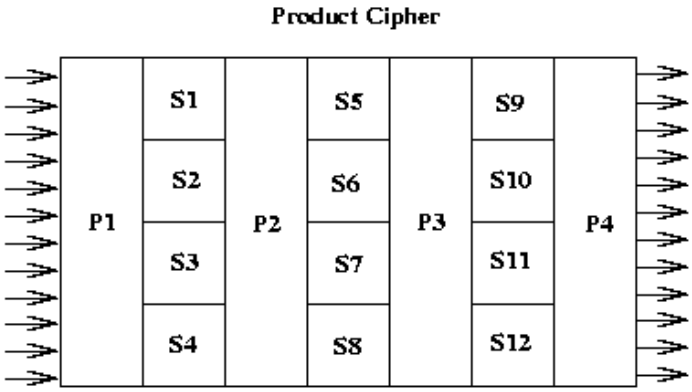
DES software is publicly available (see DES-crypt.c), but is more efficiently implemented in hardware. Users are confident of its security (they will openly say when it is in use - for example, Unix passwords).

- Data is encrypted in 64 bit blocks.
- Ciphertext is output in 64 bit blocks.
- A 56 bit key is used.
- The *same* key is used for both encryption and decryption.

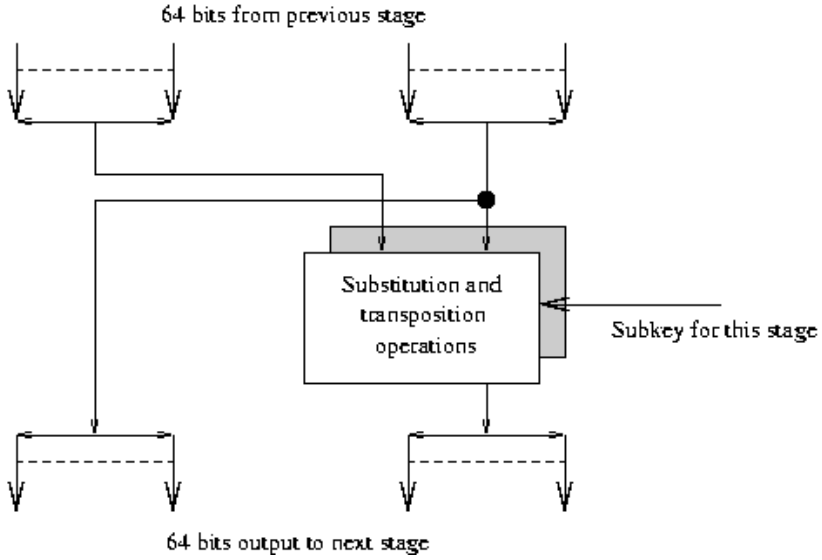
Product and Substitution Cipher Boxes

Each 'box' is parameterized by the pairings of input->output wires, and each input->output mapping in invertible. An algorithm is, then, further parameterized by the arrangement of P- and S-boxes, and is also invertible.



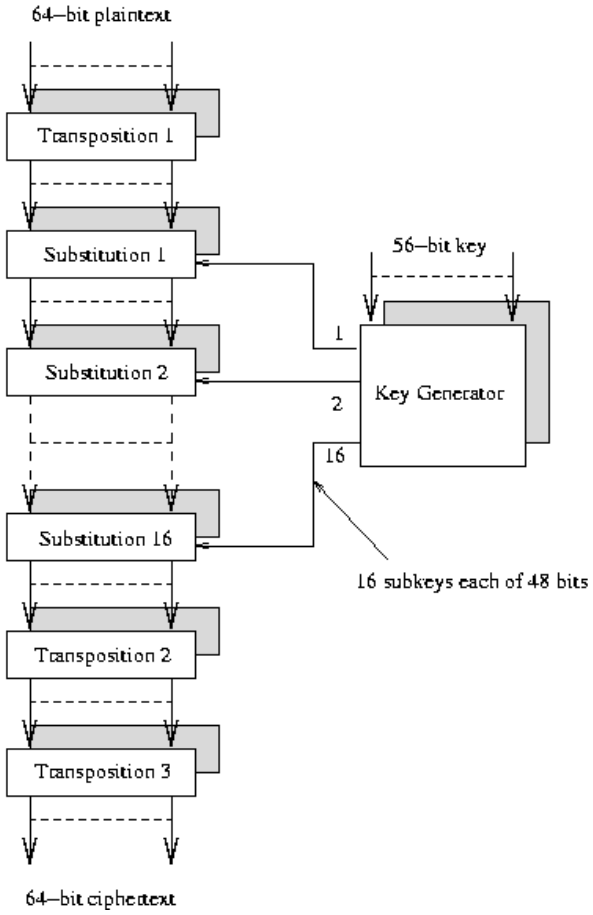


The Substitution Stages



The Steps of the DES Algorithm

The algorithm has 19 steps for encryption; decryption simply performs the steps in the reverse order.



- Step 1. Transposition of plaintext, independent of key.
- Step 19. Inverse of Step 1.
- Step 18. Exchange left 32 bits with right 32 bits.
- Steps 2-17. Use a function of the key for each stage, which we shall call K_i.

```
 Left out := Right in
Right out := XOR(left in, f(Right in, Ki))
                          where f is a 4 step function.
```

What are the steps of this magic function **f**?

1. E := R_1 (which is 32 bits) expanded to 48 bits.
2. D := XOR(E, K_i).
3. Divide D into 8x6 bits;
   feed each of these 6 bits into a different S box each producing 4 bits.
4. Feed these 8x4 bits (= 32 bits) through a P box.

How is the DES key used?

- K_0 := 56 transposition cipher of K.
- Divide K_0 into 2x28 bits.
- ROLeft each part.
- K_i := 56 bit transition of the number formed.

Triple DES

DES is still in use today, in an extended form called triple-DES, or 3DES. Whereas the original DES used only one 56-bit key, 3DES uses up to three 56 bit keys (plus one parity byte) in order to increase the difficulty of breaking the cipher:

```
3DES_encrypt(key1, key2, key3, message) =

     DES_encrypt(key1, DES_decrypt(key2, DES_encrypt(key3, message)))
```

The encrypt-decrypt-encrypt approach is used to make the algorithm compatible with single DES, in the special case when `key1 = key2 = key3`.

DES under Unix in software

A number of routines are provided which are typically used for encrypting passwords and files.

```
    char *crypt(char *key, char *salt);

    setkey(char *key);

    encrypt(char *buf, int edflag);
```

Implementations of Unix supporting remote file-systems and remote-logins often support DES encryption of transfers (such as with `RPC/XDR` authentication) using the user's login password as the key for the encryption.

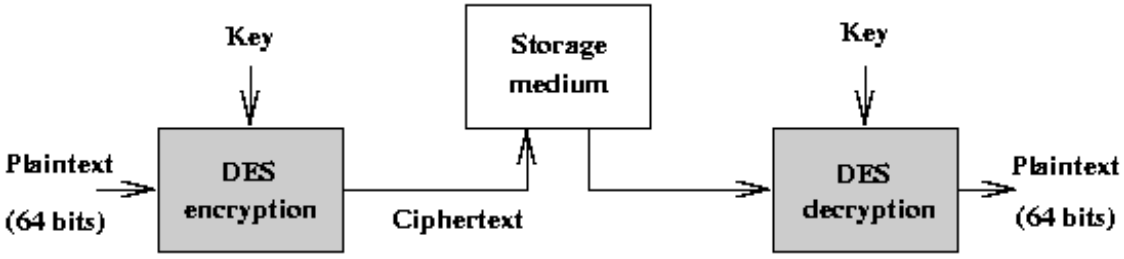
At the Crypto'94 conference, M.Matsui presented a DES-breaking technique termed 'linear-cryptanalysis'. Using 2^{43} known ciphertexts, he was able to determine a *single* DES key in 50 days on a 100MHz desktop machine.

As the cracking process is linear, comparable times on contemporary machines are measured in *hours*.

In 1998 EFF's (then) US$250,000 DES cracking machine contained 1,856 custom chips and could brute force all 2^{56} DES keys in 9 days.

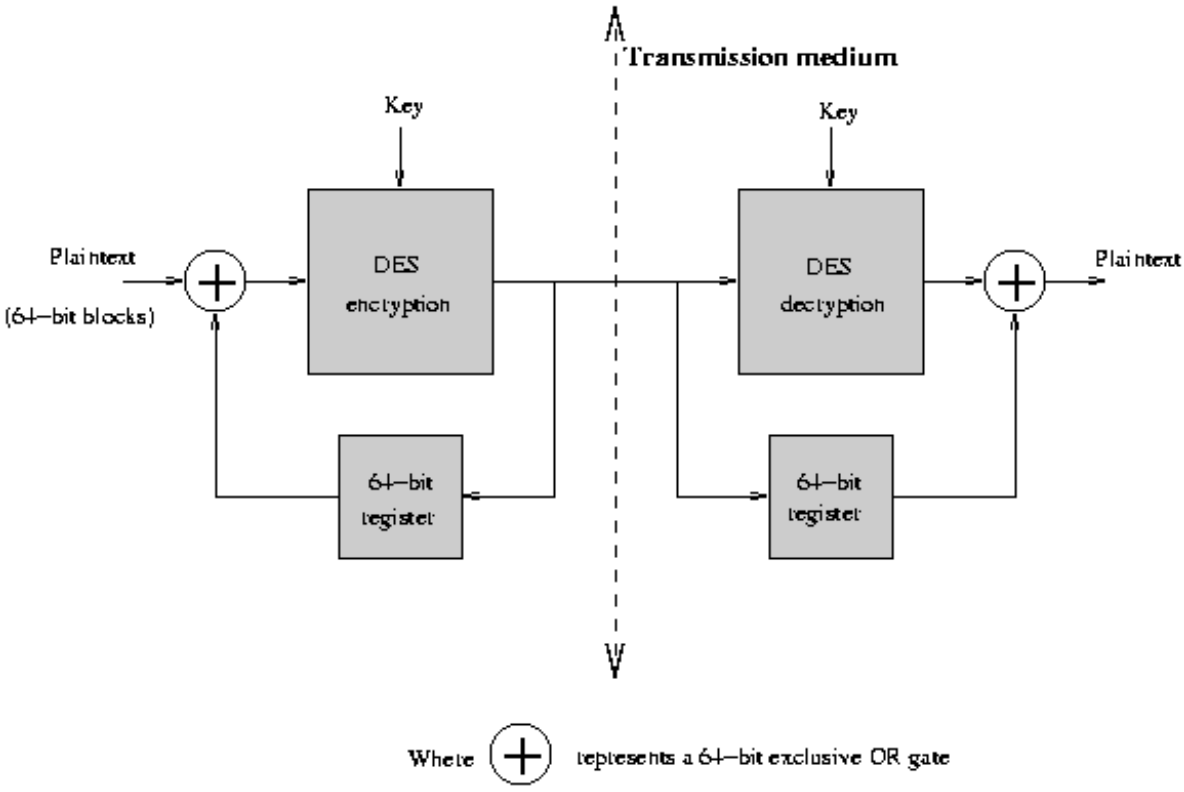
DES Modes - Electronic Code Book (ECB)

Traditionally, each block of ciphertext is *independent* of other blocks and is most frequently used for the coding of data on some storage medium (such as a disk or transmitted via a network).



DES Modes - Cipher Block Chaining (CBC)

The notion of *chaining* ensures that each block is *dependent* on earlier blocks:



Now, an intruder can neither insert nor delete any block without detection. Cipher block chaining is hence typically used in networking applications.

Exchanging Encryption Keys

Despite centuries of evolution of symmetric key cryptography, the fundamental problem of *secure key distribution* remains:

> *"How can two people (or machines) encrypt and decrypt messages using a key if they are not sure that the key itself is secure?"*

Diffie-Merkle-Hellman Key exchange

In 1976 Diffie and Hellman, from Stanford University, wrote the paper *Multi-User Cryptographic Techniques*, proposing a method of exchanging keys.

The Diffie-Hellman key exchange technique enables two *active* participants (who may never have met) to agree on a new, temporary, *session key* with which they will exchange a message.

Moreover, anyone eavesdropping on their agreement discussion, will not be able to further eavesdrop on the message exchange.

A simple (physical) analogy of how keys can be exchanged:

- A wants to send a key to B.
- A puts the key in a secure box and locks it with A's padlock.
- B does not have the key to A's padlock, so instead,
- B receives the box and adds B's own padlock to the box and returns it to A.
- A removes A's padlock with A's own key and sends the box back to B.
- B can now remove B's own padlock and remove the key which is now shared by A and B.

The work of Diffie and Hellman was revolutionary in the way we think about cryptography. Previously it was 'intuitively obvious' that the key needed to encode and decode a message needed to be the same (or trivially related).

Public Key Cryptography

Using *public key encryption* we use *two* keys rather than just one.

- The *public* key, E, may be openly published.
- The *private* key, D, is known only by the intended recipient.

The plan is to choose keys such that even knowing the public key does not reveal the private key:

- A and B openly publish their public keys (viewed as algorithms) E_A and E_B.
- A sends $E_B(Plaintext_{message})$ to B.
- B calculates $D_B(E_B(Plaintext_{message})) = Plaintext_{message}$.
- B can then reply with $E_A(Plaintext_{reply})$ for A to read.

The MIT/RSA Algorithm

In 1978, Ron Rivest, Adi Shamir, and Leonard Adleman, all of MIT, published the imaginatively named RSA algorithm for the generation of encryption/decryption functions from number theory.

The difficulty arises when chosing the algorithms E_A and D_A such that they are inverses of one another and yet difficult to crack.

Key length	Factorization times	With 10^7x1GHz machines
429-bits (RSA-129)	4,600 MIPS-years	14.5 secs
512-bits	420,000 MIPS-years	22 minutes
700-bits	4.2×10^9 MIPS-years	153 days
1024-bits	2.8×10^{15} MIPS-years	280,000 years

- We choose two very large prime numbers, *p* and *q*, each over 100 digits.
- We define E_A to be the pair (e,n) where n = pxq
  (for p, q being 100 digit primes, n will typically at least 200 decimal digits).
- We define D_A to be the pair (d,n)
  where (e $\times$ d) **mod** ((p-1) $\times$ (q-1)) = 1

We then use:

- Encryption function : C := P_e **mod** n
- Decryption function : P := C_d **mod** n

Asymmetric ciphers

RSA is an example of an *asymmetric cipher*, employing different keys for encryption and decryption. The relationship between keys simplifies an attack.

RSA has become ubiquitous. It is commonly used in applications such as Microsoft Internet Explorer and Firefox for implementing security (SSL), within mail clients for signing and encrypting emails (S/MIME), within payment systems, and for encrypting traffic between VPN gateways.

Keys for asymmetric ciphers need to be longer than keys for symmetric ciphers to achieve similar resistance to brute-force attacks:

Symmetric Key Length	***Asymmetric Key Length***
56 bits	384 bits
64 bits	512 bits
80 bits	768 bits
112 bits	1792 bits
128 bits	2304 bits

The 17-year patent on RSA was due to expire on September 20, 2000, but was released into the public domain two weeks early on September 6, 2000.

> *"Because the RSA algorithm remains one of the most widely used methods of implementing public key cryptography, the expiration of the RSA patent is good news for software companies, e-commerce, and users of private or secure communications. The fundamental patents on public key cryptography have all now expired, and we should expect an explosion of new and powerful implementations of this technology. I expect there will be expiration parties from Palo Alto to Perth as technology companies roll out new applications of this important technology!"*
>
> *— Pat Finn, "Handbook of Intellectual Property Claims and Remedies"*

Strong Encryption is not enough - the need for Digital Signatures

The push for eCommerce has demonstrated the need, not for greater bandwidth, nor (strictly) for greater end-to-end security, but for *authentication* and *authorization* of the end players.

Digital signatures were first discussed by Diffie and Hellman in their 1976 *'New Directions in Cryptography'*, but eCommerce is only recently demonstrating their worth to a wider audience (and their patent has expired!).

Unlike traditional signatures, a digital signature cannot be a constant; it must be a function of the document that it signs.

A digital signature prevents two types of fraud -

- the forging of a signature by the receiver (or any third party), and
- the repudiation of the transmission of a message by the sender.

Two categories of digital signature are identified:

- True signatures, signed by the sender, verified by the receiver.
- Arbitrated signature may only be sent and verified through a *trusted third party*. The recipient is unable to verify the sender's signature directly, but is assured of its validity through the mediation of the arbitrator.

Message Digests - basic building blocks

A message digest is a 16-, 20-, 32-byte 'fingerprint' of a message.

Message digests are central to digital signatures. When a message is signed, its contents are first hashed to give a message digest. The digest is then encrypted with the sender's secret key, giving a proof of the sender's identity.

A good digest must have the properties:

- An absence of collisions. Unlike simpler file checksums, which quickly demonstrate file or data integrity, it must be *hard* to find two messages with the same digest.
- Must not be invertible. Digests are deterministic *many-to-one* functions.
- A uniform distribution of results. A change in just one input bit should affect at least half the output bits.

Simple changes to even a single byte (even a single bit) should result in dramatic changes to the digest:

- *The winner is Sydney*
  `2f8eff80630eb401b0038d8df420719b`
- *The winner is Sydey*
  `f2b91cf6f8ad805a127182e8a46d450f`

Some popular message digests:

- MD2 and MD5: developed by RSA, producing 16-byte hashes. Research in 1994 found weaknesses in collision frequencies.
- RIPEMD-160: The European standard producing 20-byte hashes.
- SHA-1, SHA-2 and SHA-256 are specified by the US government for their DSA, outputting 20-byte hashes.

From 2004: MD5 To Be Considered Harmful Someday.

(Old) performance of the basic building blocks

Message digest algorithms (on 233MHz Pentium-II, but speed scales *linearly*):

Algorithms	Calculation (KB/sec)
MD5	36,250
SHA-1	20,428

Symmetric key algorithms (233MHz Pentium-II):

Algorithms	Setup (ms)	Encryption (KB/sec)	Decryption (KB/sec)
DES (56 bit)	6.3	4,386	4,557
Triple-DES (112 bit)	22	1,596	1,620
RC4 (128 bits)	29.8	27,325	28,132
RC5 (128 bit)	352	4,576	4,691

Asymmetric key algorithms (233MHz Pentium-II):

	512 bits (KB/s)	1024 bits (KB/s)	2048 bits (KB/s)
RSA encryption	10.5	4.23	0.436
RSA decryption	5.28	2.87	1.4

The command-line *openssl* program provides speeds on contemporary machines:

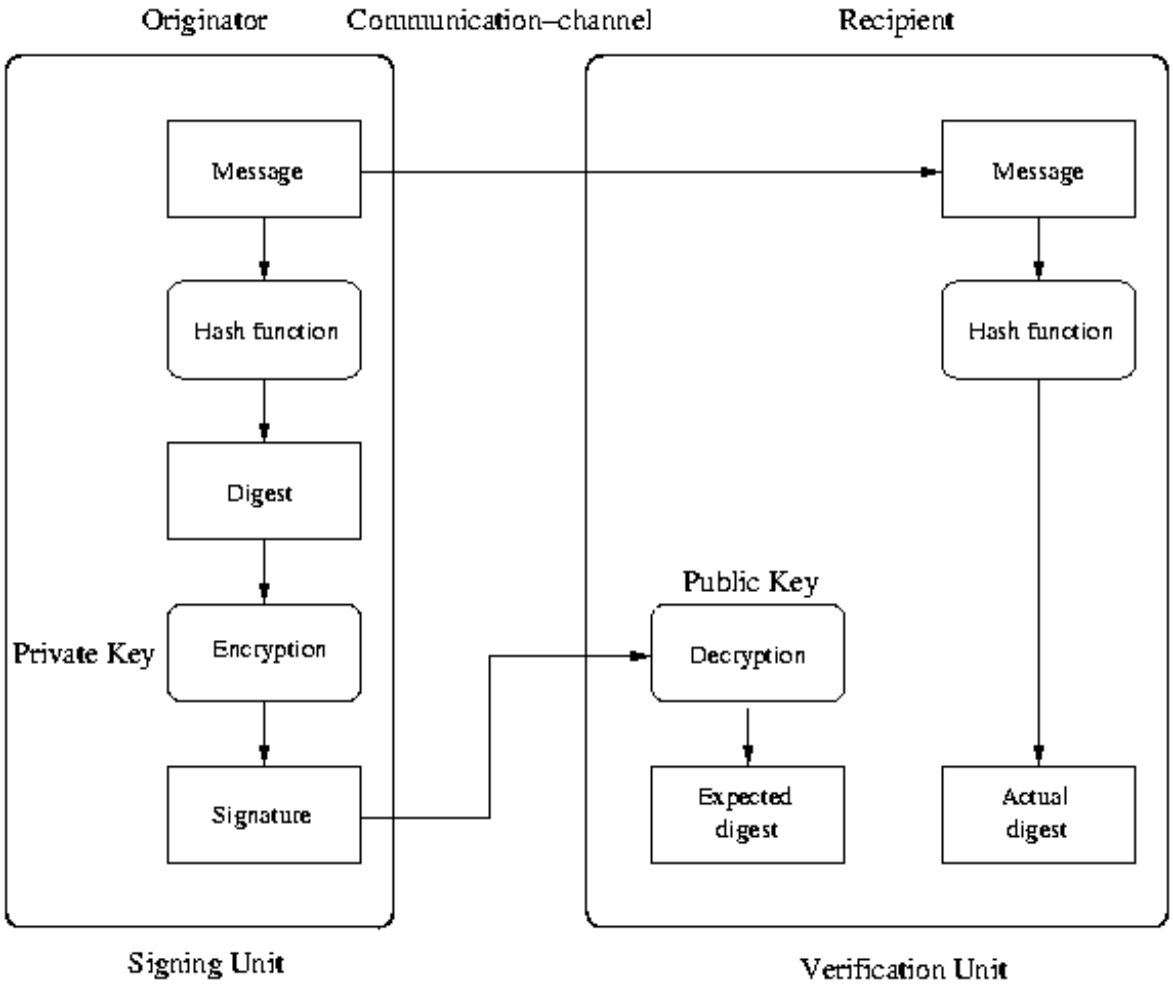
```
linux>  openssl
OpenSSL> speed md5
To get the most accurate results, try to run this
program when this computer is idle.
Doing md5 for 3s on 16 size blocks: 1722968 md5's in 2.80s
Doing md5 for 3s on 64 size blocks: 1469874 md5's in 2.80s
 ....
type    16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
md5     9833.40k    33603.65k    92756.49k   167532.74k   217651.97k
```

For large amounts of data, we first encrypt the data with a symmetric algorithm and then encrypt the symmetric key with an asymmetric algorithm.
Hybrid protocols are used in most current cryptographic architectures, such as SSL.

Digital signature generation

Digital certificates are often confused with *digital signatures*.

Like a message digest, a digital signature is a 'summary' of the original message, but also provides an assurance that the original creator of the signature has the private key matching the public key used to generate the signature.



But *who* holds the public and private keys?

What if the public key had been replaced with another?

Digital certificates

Digital certificates have been loosely described as *the driver's license for the Internet*.

A digital certificate provides a *binding* between an entity's public key, and one or more attributes to its identity.

- An *entity* may be a person, a executing piece of software, or a device such as a router or a smart-card.
- A *certification authority* (CA) attests to the authenticity of the entity's public key by digitally signing a message with its own private key.
- The 'quality' of the certificate depends on the detail of information provided to the CA (more later).
- Either, public *and* private keys may be issued by the CA, or the CA may*challenge* the entity's public key.

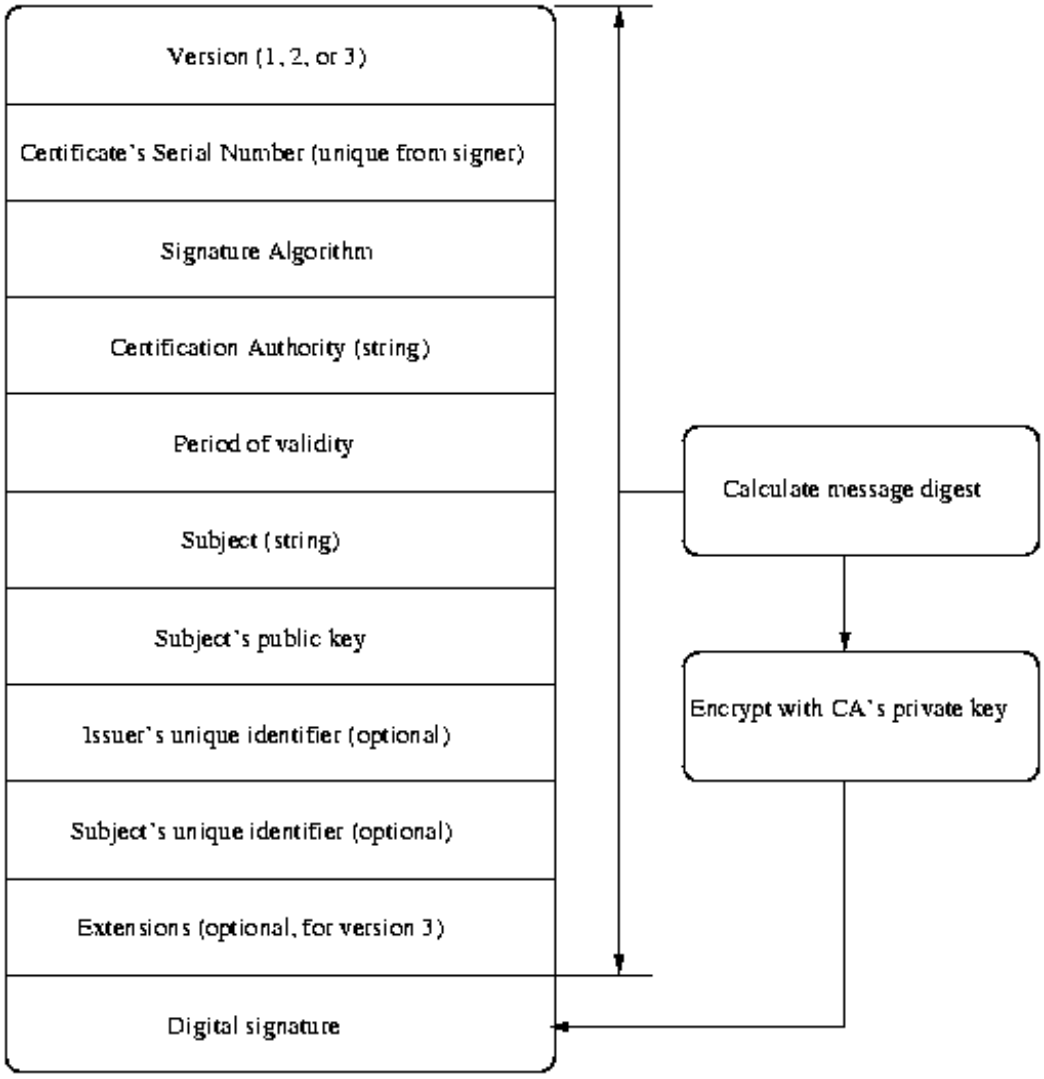
The successful use of digital certificates appears within a large community - little is gained by issuing one's own.

Digital certificate encoding

Today, certificates are defined by the ISO X.509 protocol and appears as an *application/x-x509-user-cert* MIME type.

The data is encoded using Abstract Syntax Notation (ASN.1), encoding and transmitted in ASCII using base64 encoding.

(18bit data -> 24bit representation).

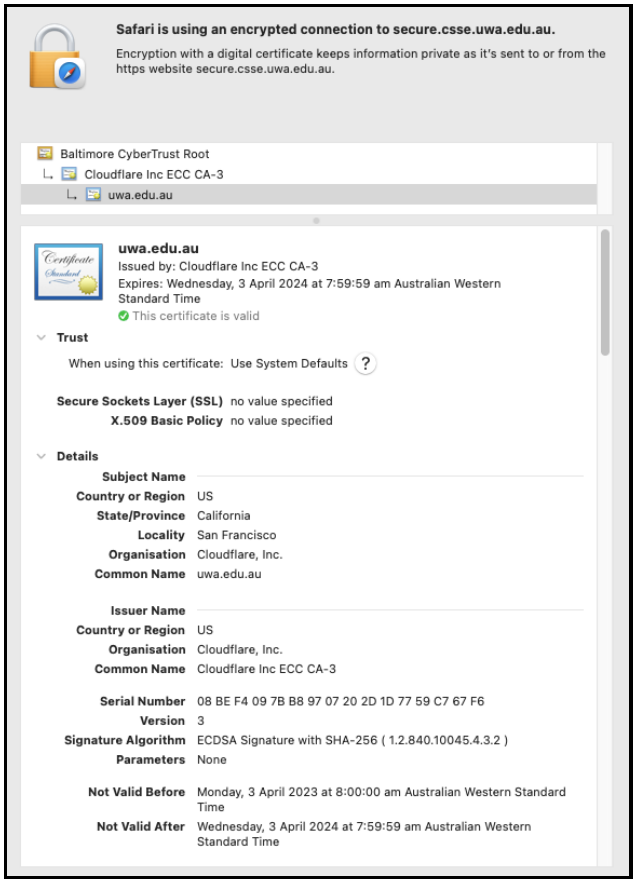


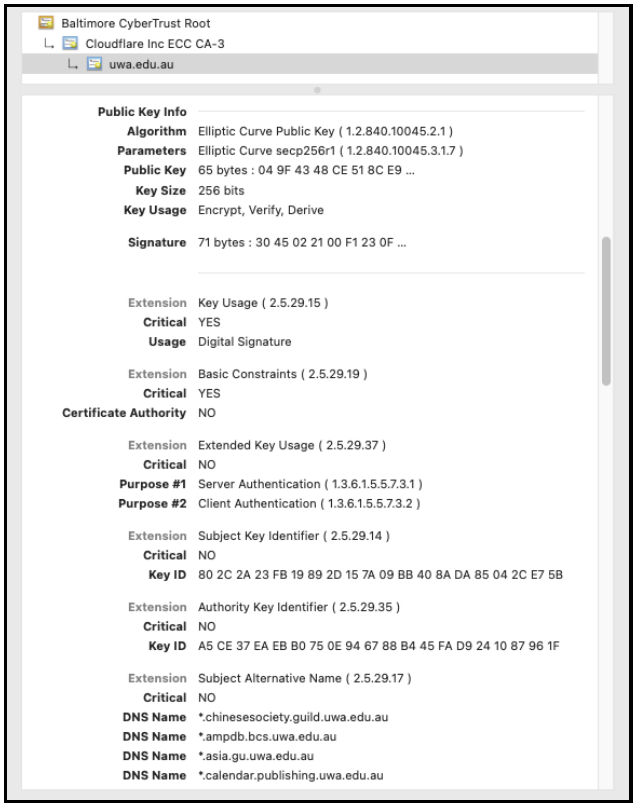
Early debate centred on whether the certificate itself needed to be encrypted (now not).

Browser support for digital certificates

Digital certificates are managed by all common browsers: Firefox, Safari, Opera, Netscape Navigator, Microsoft Internet Explorer ...

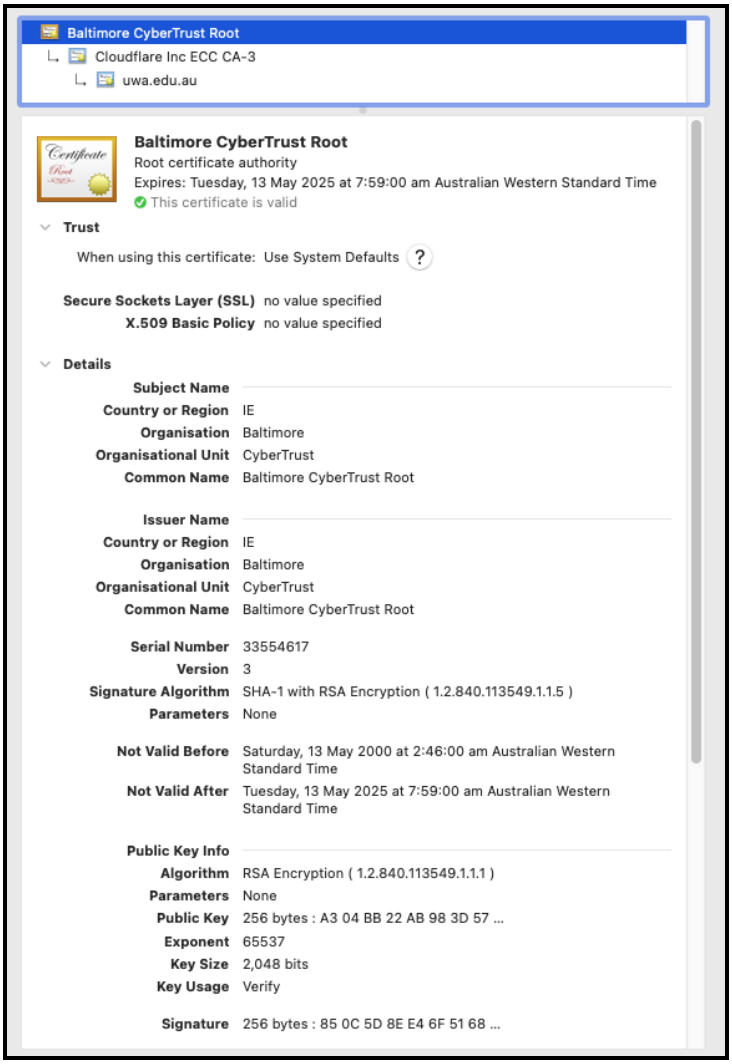
If visiting a site with the *secure* Hypertext Transport Protocol, as with *https://secure.csse.uwa.edu.au/* we can view digital certificate information via the 'padlocked' icon.





Unfortunately, there are often few CAs from Australia in most common browsers.

Browser support for digital certificates



The browser will display the digital certificate from the current page - here showing:

- The subject of the certificate,
- The issuer (CA) of the certificate,
- The serial number of the certificate,
- The period of validity of the certificate, and
- The message digest of certificate.

If the issuer of a site's digital certificate is already known by the browser (either 'hard-wired' or manually added), the issuer's certificate may be viewed *and verified*.

Version 3 of X.509 introduced *extension fields* - the association of additional information with a certificate. Each extension has:

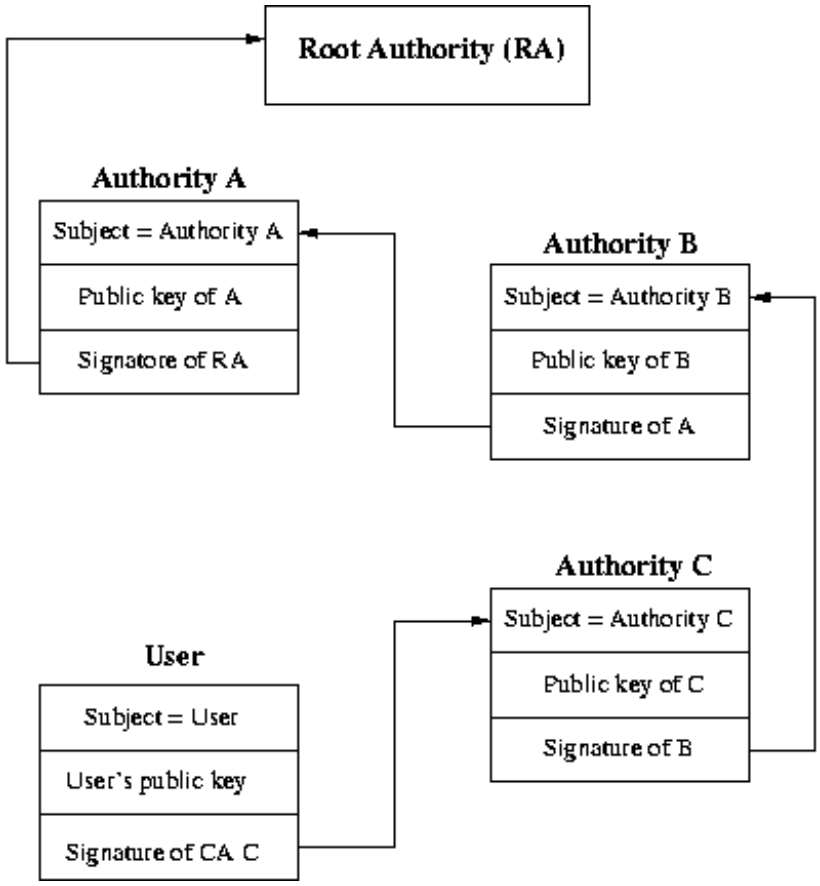
- an extension type providing semantics and typing of the extension (e.g. a string),
- an extension value", such as an email or IP address, and
- a criticality indicator indicating if the whole certificate should be ignored if an extension is not recognized.

Standard extensions (?) now describe the 'strength' and purpose of the certificate - digital signature, non-repudiation, key encipherment, data encipherment, certificate signing, etc.

Certificate Path validation

CAs are organized in hierarchies - each parent CA signs a certificate vouching for a subordinate CA's public key.

When validating a chain of certificates, the *certificate path*, the path is followed until the top of the chain is reached (when?).

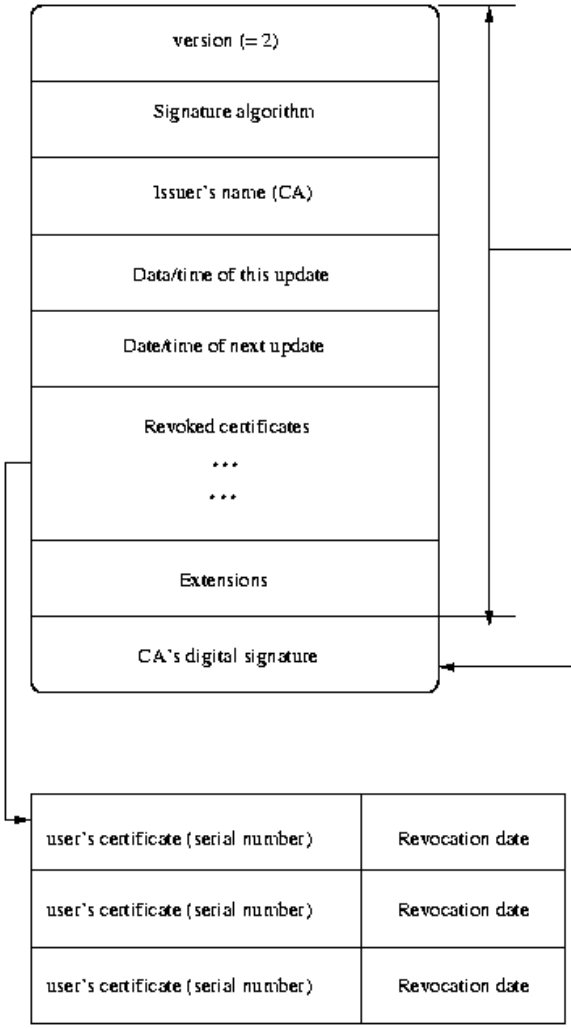


There is no automated way of verifying the top of a certificate chain other than verifying that it is one of a list of directly known (and implicitly trusted) certificates (such as in a browser).

Several companies, such as VeriSign, Thwaite, Baltimore, AT&T, and a growing band of government departments have positioned themselves 'at the top'.

Certificate Revocation Lists

A *certificate revocation list* (CRL) allows clients and servers to check whether the entity they are dealing with has a valid certificate.



Trust breaks down, and CRLs are required, when:

- a subject's private key is exposed,
- a CA's private key is exposed, and
- the relationship between the subject and CA changes (e.g. the subject is no longer employed by the CA, or stops paying money to the CA).

Certificate revocation plays a crucial part in the authentication process:

- Obtain the subject's digital certificate and verify its validity.
- Extract the serial number of the certificate.
- Fetch the current CRL from the CA.
- Verify the CRL's digital signature, and record its publication time and when the next CRL is to be published.
- Examine the CRL to determine if the intended certificate been revoked or suspended (based on the certificate serial number).
- Alert the user if the certificate is revoked.

Limitations of Certificate Revocation

In a large public key infrastructure community, CRLs are both large and must be downloaded frequently.

Applications can be significantly slowed by the need to retrieve the latest CRL from a heavily taxed directory server (or other distribution point).

There exists a compromise between always being up-to-date, versus the risk of false certificate acceptance.