COMP8411 - Cryptography and Network Security

Christopher Williamson

December 27, 2016

Contents

1 Introduction

Cryptography is 'the art of keeping messages secure' by scrambling messages so that they can't be understood by unauthorised people.

- Plaintext: a message in its original form
- Ciphertext: a message in encrypted form
- Encryption: code a message to hide its meaning
- Decryption: convert an encrypted message back to its original form
- Cryptanalysis: attempts to discover plaintext or key

2 The XOR Operator

1. You can apply XOR in any order: $a \oplus b = b \oplus a$, no matter what values a and b are.
2. Any bit XOR itself is 0: $a \oplus a = 0$.
3. Any bit XOR 0 is that bit again e.g. if a is 0 then $0 \oplus 0 = 0$, if a is 1 then $1 \oplus 0 = 1$.

The following rules imply that $a \oplus b \oplus a = b$. We'll use this property when using XOR for encryption. The first XOR with a ($a \oplus b$) is the encryption and the second one is the decryption.

2.1 One-time pads

This is an encryption scheme that involves a sequence (the 'pad') of random bits that is used as the encryption/decryption key. If the bits are truly random and the pad is only used once, the attacker learns nothing about the plaintext when they see the ciphertext. This property is called *perfect security.*

2.1.1 Reusing the 'one-time' Pad

Suppose an attacker gets two ciphertexts that have been encrypted with the same key. The attacker can then XOR the two ciphertexts, which is also the XOR of the plaintexts.

$$\begin{aligned}
c1 \oplus c2 &= (p1 \oplus k) \oplus (p2 \oplus k) && \text{(definition)}\\
&= p1 \oplus k \oplus p2 \oplus k && \text{(reorder terms)}\\
&= p1 \oplus p2 \oplus k \oplus k && (a \oplus b = b \oplus a)\\
&= p1 \oplus p2 \oplus 0 && (x \oplus x = 0)\\
&= p1 \oplus p2 && (x \oplus 0 = x)
\end{aligned}$$

To extract either $p1$ or $p2$, you'd need to know the other plaintext. The problem is that even the result of the XOR operation on two plaintexts contains quite a bit of information about the plaintexts themselves.

2.1.2 Remaining Problems

One-time pads are rarely used due to the fact that the key is at least as large as all of the information you'd like to transmit. You also need to ensure that the keys are exchanged securely ahead of time with all of the people you want to communicate with.

Since the keys have to consist of truly random data for it's security property to hold, key generation is fairly difficult and time-consuming without specialised hardware.

3 Block Ciphers

A block cipher is an algorithm that allows us to encrypt blocks of a fixed length. It provides an encryption function E that turns plaintext blocks P into ciphertext blocks C, using a secret key k:

$$C = E(k, P)$$

Once we've encrypted the plaintext blocks into ciphertext blocks, they later have to be decrypted to recover the plaintext. This is done using a decryption function D, which takes the ciphertext block C and the key k and returns the original plaintext block P.

$$P = D(k, C)$$

Block ciphers are an example of a symmetric-key encryption scheme, also known as a secret-key encryption scheme. This means that the same secret key is used for both encryption and decryption.

A block cipher is a *keyed permutation*. It's a permutation because it maps every possible block to some other block and it's also a *keyed* permutation because the key determines exactly which blocks map to which. This results in a system where knowing a bunch of (input, output) pairs for a given key shouldn't give you any information about any other pairs under that key.

Below are some design criteria for block ciphers:

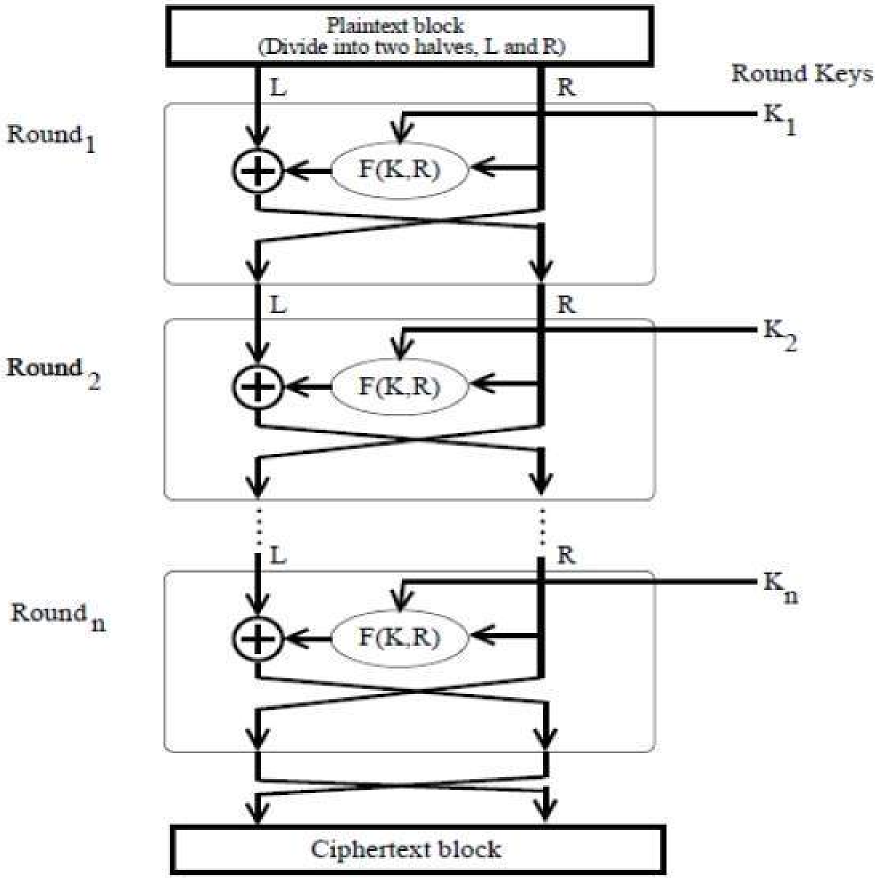
- Completeness - Each bit of the output should depend on every bit of the input and every bit of the key.
- Avalanche effect - Changing one bit in the input should change many bits in the output. Also, changing one key bit should result in the change of many bits in the output.

- Statistical independence - Input and output should appear to be statistically independent.

3.1 Feistel Block Cipher

The operation overview for the Feistel block cipher is as follows:

1. Initial permutation of bits
2. Split in half (left & right)
3. 16 rounds of identical operations, but each round uses a different sub key
4. Inverse initial permutation



The round function f typically uses permutations, substitutions and modular arithmetic. It take an n-bit block and returns an n-bit block. A larger block size, n, means greater security but makes encryption/decryption slower. It is typically 128 or 256 bits. The key size follows the same principles, 128-bits is the norm. Finally, the number of rounds, r is usually over 10.

3.2 Data Encryption Standard (DES)

DES was first published in 1977 as a US Federal standard and is a de facto international standard for banking security. it's a feistel block cipher whose block length is 64 bits and key is 56 bits. The sub-keys, $k1, k2...k16$ are each 48-bits and are generated from the main key. The decryption algorithm is exactly the same as the encryption algorithm except that the keys for each round must be used in reverse order.

3.2.1 DES Round Function

1. Expansion permutation - Right half is expanded (and permuted) to 48 bits. This diffuses the relationship of input bits to output bits.
2. Use of round key - All 48 bits of the round key are XOR-ed with out 48 bits from step 1.
3. Splitting - The result is split into eight lots of six bits each.
4. S-box (substitution) - Each six bit block is used as an index to an S-box to produce a 4 bit result (so we'll get eight 4 bit blocks).
5. P-box (permutation) - The 32 bits of output from step 4 are permuted which gives us the output of our round function.

For the S-box step there are 8 different S-boxes, one for each block. Each of these is a table of four rows and 16 columns. The 6 input bits specify which row and column to use for the index of the table. Bits 1 and 6 select the row, bits 2-5 select the column. The decimal value in that cell is then converted to a 4-bit result.

3.2.2 Triple DES

Triple DES involves the use of two or three DES keys.

EDE2 is triple DES using two keys, therefore the key length is 112 bits:

$$C = E_{K1}(D_{K2}(E_{K1}(M)))$$

EDE3 is triple DES using three keys, therefore the key length is 168 bits:

$$C = E_{K3}(D_{K2}(E_{K1}(M)))$$

3.3 Advanced Encryption Standard (AES)

This is the most common block cipher in current use. It was selected through a public, peer-reviewed competition.

- Block size: 128 bits (called a state)
- Key sizes: 128, 192 or 256 bits

AES consists of several independent steps. At a high level, AES is a substitution-permutation network involving r rounds.

Key length	Rounds
128	10
192	12
256	14

A single round transformation consists of the following steps:

- Byte substitution.
- Shift rows.
- Mix columns.
- Round key addition.

Note that in practice, the round key addition step if done before the rounds start, and in the last round, the mix columns step is not performed.

3.3.1 Byte Substitution

The SubBytes transformation is a simple table lookup. There is only one S-box (substitution box) for the while cipher, a 16x16 matrix of byte values that contains a permutation of all possible 256 8-bit values. Each individual byte of **state** is mapped into a new byte in the following way:

- Leftmost 4 bits of the byte are used as a row value; rightmost 4-bits are used as a column value.
- These row and column values serve as indexed into the S-box to select a unique 8-bit output value

3.3.2 Shift Rows

For this step we first put the bytes into a 4x4 grid. The first 4 bytes on the top row, 5-8 on the second etc. The following permutations then occur:

- 1st row: not altered
- 2nd row: a 1-byte circular left shift

- 3rd row: a 2-byte circular left shift
- 4th row: a 3-byte circular left shift

The decryption process uses shifts to the right. This step permutes bytes between the columns.

3.3.3 Mix Columns

Mix columns multiplies each column of the state with a fixed polynomial. Shift rows and mix columns represent the diffusion properties of AES.

3.3.4 Add Round Key

In this step, each byte of the state is combined with the round key using XOR i.e. the 128 bits of state are bit-wise XORed with the 128 bits of the round key.

3.3.5 Pseudocode (AES-128)

```
addRoundKey(S, K[0]);

for (i = 1; i <=9; i++) {
    subBytes(S);
    shiftRows(S);
    mixColumns(S);
    addRoundKey(S, K[i]);
}

subBytes(S);
shiftRows(S);
addRoundKey(S, K[10]);
```

3.4 DES vs AES

DES	AES
Substitution-permutation, iterated cipher, feistel structure	Substitution-permutation, iterated cipher
64-bit block size, 56-bit key size	128-bit block size, 128-bit (192, 256) key sizes
8 different S-boxes	1 S-box
Design optimised for hardware implementations	design optimised for byte-orientated implementations.
Closed (secret) design process	Open design and evaluation process

3.5 Other Symmetrical Cryptosystems

Algorithms	Mode (block length in bits)	Key length (bits)
DES	Block cipher (64)	56
Triple DES	Block cipher (64)	168 ($= 3 * 56$) (112 effective)
Rijndael	Block cipher (128, 192 or 256)	128, 192 or 256
Blowfish	Block cipher (64)	variable up to 448
IDEA	Block cipher (64)	128
RC5	Block cipher (32, 64, 128)	Variable up to 2040

3.6 Modes of Operation - Encrypting Large Messages

If a message is longer than its block size, the block cipher can be used in a variety of ways to encrypt the plaintext. There are a number of modes of operations, below are three of them:

- ECB - Electronic Code Book mode
- CBC - Cipher Block Chaining mode
- CTR - Counter mode

These modes of operation have been standardised internationally and are applicable to any block ciphers.

3.6.1 ECB Mode

In this mode, each block is encrypted independently using the same key. The last block should be padded if necessary. Usually the last byte indicates the number of padding bytes added; this allows the receiver to remove the padding. The blocks are then encrypted independently of each other. Reordering the ciphertext blocks will result in correspondingly reordered plaintext blocks.

3.6.2 CBC Mode

In this mode the plaintext is divided into blocks and the last block is padded if necessary. Ciphertext block C_j depends on plaintext block M_j **and all preceding plaintext blocks**:

$$C_n = E_k(M_n \oplus C_{n-1})$$
$$C_0 = IV \text{ (Initialisation vector)}$$

3.6.3 CTR Mode

A counter, equal to the plaintext block size is used. The counter value must be different for each plaintext block. Typically the counter is initialised to some value, and then incremented by 1 for each subsequent block (modulo 2^n, where n is the block length).

Each block can be decrypted independently of the other, therefore it can be run in parallel and there is also no error propagation.

3.7 Block Ciphers vs Stream Ciphers

While block ciphers encrypt blocks of characters, stream ciphers encrypt individual characters or bit streams.

Stream ciphers:

- Are usually faster than block ciphers in hardware; mostly used for continuous communications and/or real-time applications.
- Requires less memory space, so cheaper for resource restrained devices such as embedded sensors.
- Have limited or no error propagation, so advantageous when transmission errors are probable.
- Can be build out of block ciphers, e.g. by using CTR modes.

4 Public Key Cryptography

Up to this point, all cryptographic schemes are based on shared secret-keys (symmetric cryptography). One major problem with symmetric cryptography is that a separate key is needed for each pair of users. Therefore, an n-user system requires $n*(n-1)/2$ keys - the n^2 problem. Generating and distributing these keys is also a challenging problem and maintaining security for already distributed keys is challenging. Finally, as two parties share the same key, non repudiation can not be achieved. It was in 1976 that Diffie and Hellman first presented the concept of public key cryptography.

- Based on mathematically hard problems rather than substitution/transposition.
- A pair of keys is used, public and private. Pair are related mathematically.
- Encryption generated by one key can only be decrypted by the other key in that pair.
- Examples: RSA, DSS, DH (Diffie Hellman).

4.1 Achieving Confidentiality

Our scenario is that Alice wants to send a message to Bob and ensure that is has confidentiality protection.

1. Alice encrypts the message with Bob's **public** key
2. Bob decrypts Alice's message with his **private** key

This method ensures that only Bob can decrypt the ciphertext (as he is the only one in possession of his private key). Usually this is only applied to short messages, e.g. for secure transmission of symmetrical keys.

4.2 Achieving Authenticity

Our scenario here is that Alice wants to send a message to Bob and Bob wants to be assured that the message has come from Alice.

1. Alice signs the hash value of the plaintext with her **private** key
2. Bob verifies the hash value using Alice's **public** key

Usually the signature is signed in the hash value of the plaintext, and a timestamp should also be included.

4.3 Modulo Operator

$$a \equiv b \bmod n$$

Means that there exists an integer number k such that a can be represented as:

$$a \equiv k \cdot n + b$$

Given integers, a, b, and $n \neq 0$. a is congruent to b mod n iff $a - b \equiv k \cdot n$ for some integer k.

Examples:

- $9 \bmod 5 = 4$
- $20 \bmod 9 = 2$
- $17 = 2 \bmod 5$ since $17 - 2 = 3 \cdot 5$

Below are some properties of modular arithmetic:

- $a \equiv a \bmod n$
- $a \equiv b \bmod n \leftrightarrow b \equiv a \bmod n$
- $a \equiv b \bmod n$ & $b \equiv c \bmod n \rightarrow a \equiv c \bmod n$

- $(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$
- $(a \cdot b) \bmod n = ((a \bmod n) \cdot (b \bmod n)) \bmod n$
- $a \cdot (b + c) \bmod n = (a \cdot b + a \cdot c) \bmod n$
- $a \cdot x \bmod n = 1$ where x is an integer and called the multiplicative inverse of a

Below is some code that finds the multiplicative inverse of a under n:

```
def multiplicative_inverse(a, n):
  x = 1

  while x < n:
    if (a*x) % n == 1:
      return x
    else:
      x += 1
```

a and b are said to be relatively prime if only 1 can divide each of them e.g. 8 and 15.

4.4 RSA Algorithm

The RSA algorithm was invented by Ron **R**ivest, Ali **S**hamir and Leonard **A**dleman. It has withstood years of cryptanalysis and remains by far the most popular and well trusted scheme.

The algorithm consists of two numbers, the modulus, n, and the public exponent, e. The modulus is a product of two very large prime numbers (100 to 400 digits), represented by the letters p and q. They need to be kept secret.

It is a block cipher in which the plaintext and ciphertext are integers between 0 and $n - 1$ for some n.

4.4.1 Key Generation

1. Select two large primes, p and q
2. Calculate $n = p \cdot q$ and $\phi(n) = (p - 1) \cdot (q - 1)$
3. Select an integer, e which is relatively prime to $\phi(n)$ and $1 < e < \phi(n)$
4. Calculate a value d such that $d = e^{-1} \bmod \phi(n)$ or $(d \cdot e) \bmod \phi(n) = 1$
5. Public key $= e, n$
6. Private key $= d, n$

You can see that e is **publicly** chosen and d is **private** and calculated.

4.4.2 Encryption

For the plaintext, we represent it as an integer M in the range $0 < M < n$. The ciphertext is then:

$$C = M^e \bmod n$$

4.4.3 Decryption

To get the plaintext from the ciphertext we use the private key, d, n:

$$M = C^d \bmod n$$

4.4.4 Worked Example

1. Select $p = 7$ and $q = 17$
2. Calculate $n = p \cdot q = 119$
3. Calculate $\phi(n) = (p - 1) \cdot (q - 1) = 96$
4. Select $e = 5$, which is relatively prime and less than $\phi(n)$
5. Calculate $d = 77$ such that $de \bmod \phi(n) = 1$
6. Let $M = 19$, then $C = 19^5 \bmod 119 = 66$

PKCS#1 standard defines the use of the RSA algorithm. It defines the key generation, encryption, decryption, digital signatures, verification, public key format, padding, and several other issues with RSA: `https://en.wikipedia.org/wiki/PKCS_1`

The security of RSA relies on the difficulty of finding d given e, n. p and q should differ in length by only a few digits, and both should be on the order of 100 - 200 digits or even larger.

- n with 150 digits could be factored in about 1 year
- Factoring n with 200 digits could take about 1000 years

4.5 Hybrid Cryptosystems

Public key ciphers are much slower than symmetrical key ciphers, 1000 times slower in hardware, and 100 times slower in software, than DES. Symmetric ciphers also have key management problems and can not provide non-repudiation service without the involvement of a trusted third party.

Due to these issues, we usually combine them to get the strengths of both - this leads to the hybrid cryptosystem:

- Public cipher for symmetric key establishment/transportation and/or digital signature generation.
- Symmetric cipher for bulk encryption.

The technique is called digital enveloping. A symmetrical algorithm with a random session key is used to encrypt the message and then a public-key algorithm is used to encrypt the random session key.

5 Cryptographic Checksums

Conventional symmetric encryption: $a \rightarrow B : E_k[M]$ provides confidentiality, as ONLY A and B share K. It also provides a certain degree of origin authentication as it could only come from A. It does not provide signature as receiver B could also generate the encryption and sender A could deny sending the message (repudiation of origin).

Public key encryption: $A \rightarrow B : E_{KUb}[M]$ provides confidentiality but not origin authentication as everyone knows Bs private key.

Digital signing is applied in the following manner:

$$A \rightarrow B : M \| E_{KR_a}[h(M)]$$

Which means that we have the plaintext, M, with the encrypted hash value of M attached to the end of it. The encryption is done with As private key. This method provides origin authentication and non-repudiation as:

- Only A has KR_a, so signed item must have come from A.
- Any party can use KU_a to verify the item.
- Provided KU_a is trust-worthy, and the signature is dated.

But it does not provide confidentiality.

Some of the cryptographic operations mentioned above could only provide message authentication and integrity protections provided that the message has some structures or is recognisable. We therefore need some redundancy for the receiver to verify the message - this check-value is a **cryptographic checksum**.

A cryptographic checksum can be used to protect:

- Content authentication (origin + integrity)
- Non-repudiation
- Anti-replay

5.1 Digest Function

Given a message, M, of arbitrary length, a Message Digest function, H, produces a fixed-size output, h (called a message digest, checksum, hash value, or fingerprint, of M). h should be a function of all the bits of M

5.1.1 Requirements

A message digest function should take into account the following properties:

- **Compression**: H can be applied to a block of data of any size, but produces a fixed-length output.
- **One-way property**: $H(x)$ is easy to compute for any given x. For any given h, it is hard to compute x such that $H(x) = h$.
- **Weak collision resistance**: Given x, it is hard to find $y \neq x$ such that $H(y) = H(x)$.
- **Strong collision resistance**: It is hard to find two different messages, $x \neq y$, such that $H(y) = H(x)$.

If H is strong collision resistant, it is also weak collision resistant.

If the weak collision resistance property is not met signature forgery is likely. Assuming that A has sent a signed message M to B, $M \| s$ where $s = R_{KR_a}[H(M)]$:

1. An attacker intercepts A's signature and message.
2. The attacker finds another message, M' where $H(M) = H(M')$.
3. The attacker now has your signature s on the real message.

Repudiation is likely if the strong collision resistance property is not met. Assuming that A is to send a signed message M to B:

1. A chooses two messages M and M' where $H(M) = H(M')$.
2. A signs M by generating the signature.
3. A sends B $M \| s$.
4. Later A repudiates this signature, saying that it was really a signature on the message M'.

5.2 Construction Methods

Cryptographic checksum construction methods include message authentication code (MAC), hash functions and HMAC, among others. Below we take a deeper look into some of these construction methods.

5.2.1 Extension Method

In this method, each MD function processes a block of M, the output is the input for the next block. The output of the final block is the digest of the entire message.

5.2.2 Message Authentication Code (MAC)

This is a public function with a shared secret key that produces a fixed-length output i.e. $\text{MAC} = f_k(M)$.

It is block cipher based which means that it is slow and produces a short digest length. As before, the output of one block is the input to the next block. At the end we just take the last output value as our digest.

If only A and B know the secret key K, and if $\text{MAC} = \text{MAC}'$ (MAC' is the receivers digest), then the receiver can be assured that:

- The message has not been altered.
- The message is from the alleged sender.
- The message is of the proper sequence if the message includes a sequence number.
- The message is fresh, if the message includes a timestamp.

5.2.3 Hash Functions

Below is a table of commonly used hash functions:

	SHA-1	SHA-256	SHA-384	SHA-512
Hash value size	160	256	384	512
Block size	512	512	1024	1024
Word size	32	32	64	64
Security	80	128	192	256

Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a work-factor of approximately $2^{n/2}$.

5.2.4 HMAC

HMAC constructs MAC by applying a message and key to a cryptographic hash function in a nested manner i.e.

$$\text{HMAC}(K, M) = H[(K \oplus \text{opad}) \| H[(K \oplus \text{ipad}) \| M)]]$$

Where:

- **H**: hash function such as SHA-1
- **ipad**: a string by repeating the byte 0x36 (00110110) as often as necessary.
- **opad**: a string by repeating the byte 0x5c (01011100) as often as necessary.
- **K**: 512-bits secret key

6 Digital Signature Algorithms

A digital signature associates a mark unique to an individual with a body of text. It should be:

- Message-dependent: unreusable & ensures integrity
- Signer-dependent: unforgable & ensures authenticity
- Verifiable: others should be able to verofy that the signature is indeed from the originator.

A digital signature scheme uses public-key cryptography, typically RSA or DSA.

The main idea is that only A can sign a message because only A has access to their private key. Anyone can verify A's signature because everyone has A's public key. A digital signature scheme consists of a key generation algorithm, a signature generation algorithm and a signature verification algorithm.

6.1 RSA for Digital Signature - Problems

One problem with using RSA for digital signature is that RSA can only handle so much data and we may want to sign a message which is longer than RSA allows. One way to solve this would be to break the message up into smaller chunks but this would add a great deal of complexity and expense.

There is also a security issue:

- Let m be the message of which you want to forge a signature.
- Choose two messages x and y such that $xy = m \bmod n$.
- If you could obtain the signatures of x and y then you can easily forge a signature of m by $S(m) = S(y)S(x) \bmod n$

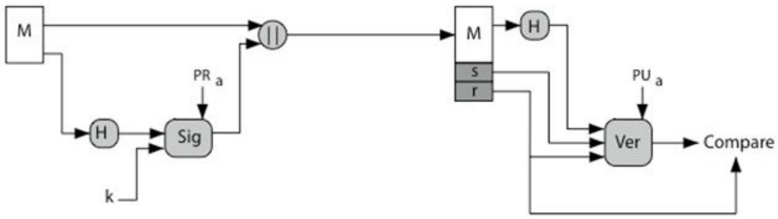
A solution to these problems is to use the 'hash-and-sign' paradigm where we sign the hash value of the message. Below is how we write the signature generation process and the signature verification process:

$$S = E_{KR_a}[H(M)]$$
$$h = D_{KU_a}[S] = D_{KU_a}[E_{KU_a}[H(M)]]$$

The remaining problem is how can we ensure that public keys are trustworthy? This is covered in the Public Key Infrastructure topic.

6.2 DSA

Below is an illustration of the workings of the DSA algorithm:



k is a random number generated for this signature (it should be different for every signature generation). PR_a is the sender's private key and PR_a is the sender's public key.

6.2.1 Key Generation

First we generate our global public-key components that can be shared among a group of users:

- Choose p, a 512-bit to 2048-bit prime number.
- Choose q, a 160-bit prime number which divides $(p-1)$ (q is a prime divisor of $p-1$).
- $g = h^{(p-1)/q} \bmod p$ where h is any integer where $1 < h < (p-1)$ such that $h^{(p-1)/q} \bmod p > 1$.

Next, we choose a long-term private key, x, which should be a randomly chosen value in the range $0 < x < q$. Now we can compute our long-term public key:

$$y = g^x \bmod p$$
$$KU_a = \{p, q, g, y\}$$

6.2.2 The Signing Process

Once we have our public/private key pair, we can move onto the signing stage. First, we choose a random number for k. This key must be destroyed after use and must never be reused. We then compute two components:

- $r = (g^k \bmod p) \bmod q$

- $s = [k^{-1}(H(M) + xr)] \bmod q$
- Signature $= (r, s)$

We can then send $M \|$signature to the receiver.

6.2.3 Signature Verification

So now the receiver has got KU_a and $\{M', r', s'\}$. We compute:

- Message hash $H(M')$
- mod q inverse of s': $w = (s')^{-1} \bmod q$
- $u_1 = [H(M')w] \bmod q$
- $u_2 = (r')w \bmod q$
- $v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$

We then test that $v = r'$. If this is true, then the signature is verified.

6.3 RSA vs DSA

RSA	**DSA**
Security is based on difficulty of factoring large numbers	Security is based on difficulty of taking discrete logarithms
Can encrypt and sign	Can only sign messages
Faster than DSA in signature verification, and about the same in signature generation.	Some signature computation can be computed priori.
Can recover the message digest from the signature	Cannot recover the message digest from the signature
	Need to choose a unique secret number k for each message

7 Public Key Infrastructure

The major problem with wide-scale application of public-key cryptography is howcan we trust that a given public key belongs to the claimed entity? The solution is to have someone or some authority to sign one's public key (a digital certificate). A digital certificate is a statement certifying that this public key belongs to this identity and the owner with this identity possesses the corresponding private key.

7.1 Two Trust Models

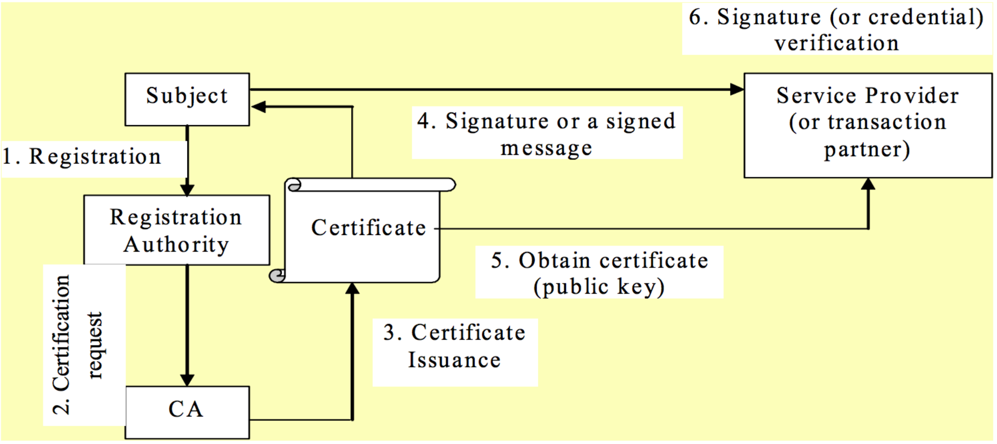
SPKI (Simple PKI) is a bottom-up approach:

- Uses a web-of-trust model.
- Public keys are signed/certified by friends or friends' friends.
- You are supposed to trust some of the friends.
- Used in the PGP solution.

X509 PKI is a top-down approach:

- Public keys are signed/certified by trusted authorities, Certification Authorities (CAs).
- A CA or CA hierarchy digitally sign keys in a top-down manner.

Below is a diagram showing the connections between PKI entities:


6. Signature (or credential) verification
Subject
Service Provider (or transaction partner)
4. Signature or a signed message
1. Registration
Registration Authority
Certificate
5. Obtain certificate (public key)
2. Certification request
3. Certificate Issuance
CA

7.2 Main Functions

7.2.1 SystemSetup

A credential service provider should get the policy, prodedures and services ready, including key generation/update, certificates issuance, distribution and revocation, possibly key recovery, and potential interaction with other providers.

7.2.2 SubjectRegistration

During this process, a subject makes itself known to a CA.

7.2.3 KeyGeneration

A pair of crypto keys are generated either by the subject or by the CA, and the CA will certify the public key of the pair.

7.2.4 CredentialIssuance

The CA issues a certificate for a subject's public key.

7.2.5 CredentialVerification

This is performed when a credential is used to access a service or to perform a transaction

7.2.6 CredentialRevocation

If the private key associated to the public key certified in the certificate is compromised or suspected compromised, then the certificate should be revoked.

7.2.7 Cross-certification

Is an operation to allow a pair of CAs to establish a trust relationship through the signing of each other's public keys in a certificate called cross-certification.

7.2.8 SubjectRegistration

Enrollment: An applicant, e.g. Alice, may need to provide the following information:

- Proof of Alice's identity
- Alice's public key

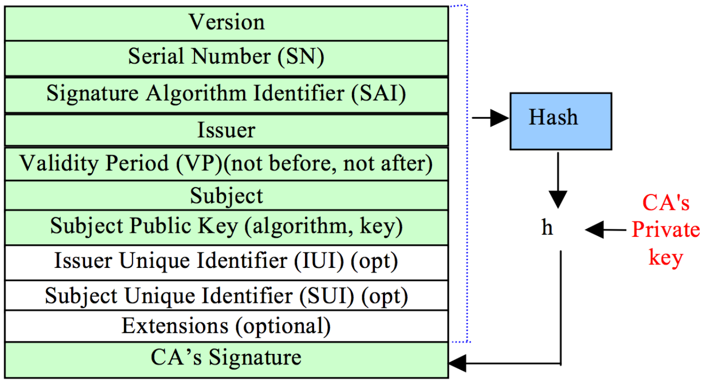
Authenticates applications:

- Share information with a third-party database
- Personal appearance

A Data Repository, typically a LDAP directory is where certificates and revocation status are officially stored

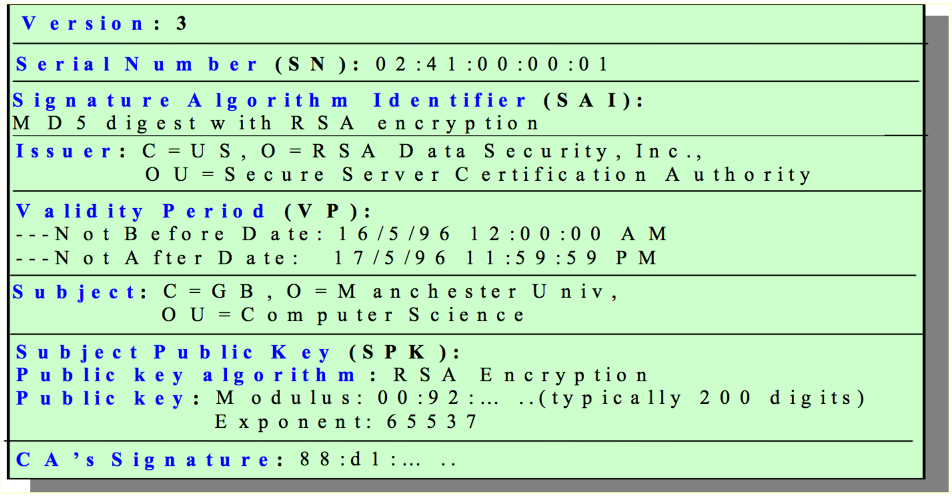
7.3 The X.509 v3 Certificate Format

The image below is the structure of this type of certificate format:


Version
Serial Number (SN)
Signature Algorithm Identifier (SAI)
Issuer
Validity Period (VP)(not before, not after)
Subject
Subject Public Key (algorithm, key)
Issuer Unique Identifier (IUI) (opt)
Subject Unique Identifier (SUI) (opt)
Extensions (optional)
CA's Signature
Hash
h
CA's Private key

- **Version:** current values are v1, v2 & v3
- **SN:** a number unique to the issuer (CA)
- **SAI:** identifies the algorithm, such as RSA or DSA, used by the CA to sign the certificate
- **Issuer:** the issuer's name
- **VP:** a range of time when the certificate is valid
- **Subject:** the subject's name
- **SPK:** the subject's public key and parameters, and the identifier of the algorithm with which the key is used.
- **IUI:** to allow the reuse of issuer names over time
- **SUI:** to allow the reuse of subject names over time
- **Ext:** provide a way to associate additional information for subjects, public keys, managing the certification hierarchy and certification revocation lists.

The next image is an example:



Version: 3

Serial Number (SN): 02:41:00:00:01

Signature Algorithm Identifier (SAI):
MD5 digest with RSA encryption

Issuer: C=US, O=RSA Data Security, Inc.,
OU=Secure Server Certification Authority

Validity Period (VP):
---Not Before Date: 16/5/96 12:00:00 AM
---Not After Date: 17/5/96 11:59:59 PM

Subject: C=GB, O=Manchester Univ,
OU=Computer Science

Subject Public Key (SPK):
Public key algorithm: RSA Encryption
Public key: Modulus: 00:92:... ..(typically 200 digits)
Exponent: 65537

CA's Signature: 88:d1:... ..

7.4 Certificate Revocation Lists - Why Do We Need It?

A certificate has a validity period but what if a private key corresponding to a public key certified in a certificate is compromised before the expiration date? We are now vulnerable to repudiation attacks.

Certificate revocation lists is a mechanism to let the world know that a certificate is no longer valid. It is a black list of revoked certificates. A CRL is a data structure, digitally signed by the issuing CA, containing:

- Dateand time of the CRL publication
- Name of the issuing CA
- Serial numbers of all the revoked certificates.

Using CRL is not that straightforward. The issuing CA needs to keep the CRL up-to-date. A certificate-using application should obtain the most recent CRL and ensure that the certificate serial number is not on the CRL list; in other words, a certificate is said to be valid iff the following verifications are positive:

- It has a valid CA signature
- It has not expired
- It is not listed in the CA's most recent CRL

There are also some scalability issues.

7.5 Top-down Certificate Hierarchy

In most cases we use more than one CA, as using one root key to sign certificates is too risky if that one key is compromised and also it is not scalable when the user base is large. In some cases, certificate managements may resemble the management structure of an organisation.

The certificate hierarchy starts with a root CA with a root cert/key. They create more keys, sign them with the root key and delegate them to subordinate CAs. Validating a certificate possibly involves validating a chain of certificates (called a chain of trust).

The CA hierarchy would look something like this:

For certificate chain validation we verify that all the digital signatures are signed by all subordinate CAs in a bottom up manner until you rech the root CA's signature, or until you reach a subordinate CA that you can trust. So, for Alice's certificate chain:

$$\{\text{CERT}_{\text{Alice}}\}S_{\text{DeptQ}} + \{\text{CERT}_{\text{DeptQ}}\}S_{\text{1stCorp}} + \{\text{CERT}_{\text{1stCorp}}\}S_{\text{RootCA}}$$

If Bob wishes to authenticate a message signed by Alice, he can proceed 'up' the certificate chain until he finds a certificate he can trust.

8 Key Management

Key management is the hardest part of cryptography. How can we generate them so that the can not be easily guessed? How can we securely store keys so that they can not be easily stolen? How could keys be delivered to their intended recipients securely? How could two entities agree on, or establish, a key securely and how are keys revoked and replace?

For symmetrical ciphers, how can we keep the keys secret and for public-key ciphers, how can we ensure that the public keys are trustworthy?

Below is a table which shows the number of possible keys given various constraints as well as the time it would take to crack them given 10^6 attempts per second:

	6-bytes	8-bytes
Lowercase letters (26)	$3.1 * 10^8$	$2.1 * 10^{11}$
	5 min	2.4 days
Lowercase letters & digits (36)	$2.2 * 10^9$	$2.8 * 10^{12}$
	36 mins	33 days
Alphanumeric characters (62)	$5.7 * 10^{10}$	$2.2 * 10^{14}$
	16 hours	6.9 years
Printable characters (95)	$7.4 * 10^{11}$	$6.6 * 10^{15}$
	8.5 days	210 years
ASCII characters (128)	$4.4 * 10^{12}$	$7.2 * 10^{16}$
	51 days	2300 years

8.1 Key Generation

Good keys are random numbers, a sequence of numbers where an observer can not predict one of the numbers given all of the others in the sequence. Oridinary random number generation functions e.g. `java.util.Random`, may not be good enough for this. A reliable random source, or a cryptographically secure pseudo-random number generator, e.g. `SecureRandom` class in `java.security` package should be used.

Some pseudo-random numbers are generated from a strong mixing function that takes two or more inputs that have some randomness (e.g. CPU load, arrival times of network packets), but produces an output bit of which depends on some nonlinear function of all the bits of the inputs. Cryptographic hashing functions and encryption algorithms (e.g. MD5, SHA and DES) are all examples of the strong mixing function.

8.2 Key Storage

You must protect the key to the same degree as all the data it encrypts. Attackers may defeat access control mechanisms, so you should encryt the file containing the key. Ideally, a key should never appear unencrypted outside the encryption device.

The key may be resident in memory, so attackers may be able to read it if they could get access to the machine. Therefore, you can use a physical token to store the key such as a smart card, and protect the token with a PIN number. The card could be stolen so therefore you can split they key into two halves and store one half on the card and the other in the machine.

8.3 Session Key Establishment

The more often a symmetric key is used, the more likely it is to be cracked. Therefore we should generate and use a symmetric key for one session only. It is desirable to use diferent session keys in different sessions as this can:

- Limit available ciphertexts for cyptanalysis
- Limit exposure (both in time period and amount of data) in an event of key compromise
- Avoid long-term storage of a large number of secret keys by only creating them when needed.

To establish a session key, one way it to use a key agreement protocol. A shared secret is derived by the parties as a function of information contributed by each, such that no party can predetermine the resulting value (Diffie-Hellman protocol).

Another way is to use key transportation protocols. If we're not using a public-key cipher session keys are generated and distributed with the help of a third party (Needham-Schroeder protocol). If we're using a public-key cipher one party creates a session key and securely transfers it to the other party using the recipients public key.

With this being said, there are other issues that sould be considered.
Entity and key authentication:

- Assurance: no other party could gain access to the established session key.
- Key confirmation: asking the other entity to demonstrate that he has the knowledge of the key by either producing a one way hash value of the key or encrypting some known data (e.g. nonce) with the key.

Key freshness: assurance that the key is fresh i.e. not used before.

8.4 Diffie-Hellman Protocol

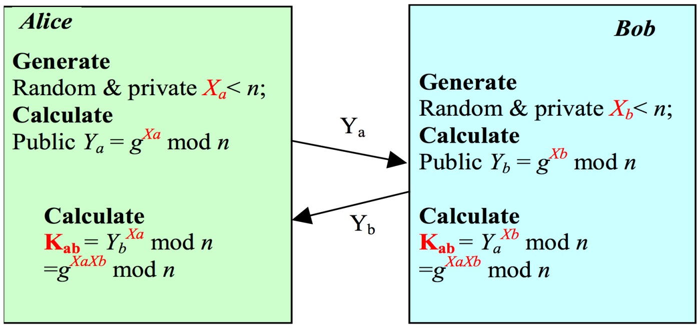
BH was the first public-key algorithm ever invented back in 1976. The DH key exchange protocol allows two parties who have never met before to echange protocol messages in public and collectively generate a key that is private to them, and none of the parties could predetermine the key.

Its security is based on the difficulty of calculating discrete logarithms in a finite field:

- Given integers y and g and prime number n, compute x such that $y = g^x \bmod n$.
- Solutions known for small n.

- Solutions computationally infeasible as n grows large.

Assuming that two parties, Alice and Bob, take part in the exchange. They agree on two large integers, g and n. n is a prime number that serves as the modulus and g is a ranodm number that serves as the basis with $1 < g < n$. g and n do not have to be secret. Alice has a private key X_a and a public key Y_a and so does Bob, X_b and Y_b. The image below shows how the algorithm works:


Alice
Generate
Random & private $X_a < n$;
Calculate
Public $Y_a = g^{Xa} \bmod n$
Calculate
$K_{ab} = Y_b^{Xa} \bmod n$
$= g^{XaXb} \bmod n$
Y_a
Y_b
Bob
Generate
Random & private $X_b < n$;
Calculate
Public $Y_b = g^{Xb} \bmod n$
Calculate
$K_{ab} = Y_a^{Xb} \bmod n$
$= g^{XaXb} \bmod n$

This resists passive attacks such as eavesdropper as caluclating a discrete logarithm is a computationally hard problem. There is one problem though. Neither party knows who it shares the secret with so it is vulnerable to active man-in-the-middle attacks.

8.5 Distribution Without Use of PKC

Approach one: Given n users, we provide $n(n-1)/2$ keys. The problem with this is that as n increases, the number of keys needed becomes untenable for everyone, the n^2 problem.

Approach two: Use a key distribution centre (KDC) or server which also includes a key hierarchy where there are master keys (for long-term use) and session keys.

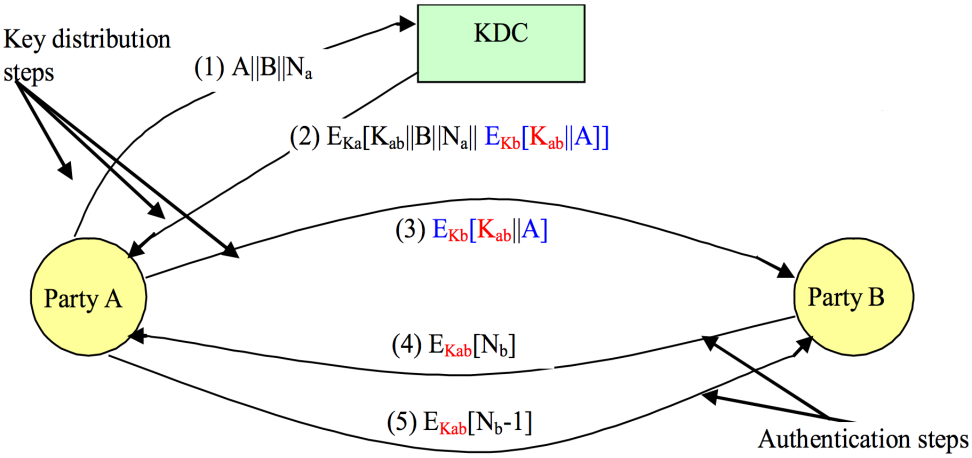
A unique master key, shared between either a pair of users or a KDC, is for session key distribution. The session key is to secure the communication. The advantage of using approach two is that it reduces the scale of the problem. The n^2 problem is reduced to an n problem therefore making the system more scalable.

The disadvantage of approach two is that there is a need to trust the KDC. The KDC has enough information to impersonate anyone to anyone. If the system is compromised, all the resources in the system are vulnerable.

8.5.1 The Needham-Schroeder Protocol

The Needham-Schroeder protocol uses approach two that's mentioned above. Both parties, A and B share a secret key with the KDC, K_a and K_b. A and B wish to establish a secure communication channel, i.e. establish a shared one-time session key K_{ab} for use between A and B.

N_a and N_b are nonces (random challenges), generated by A and B respectively, to keep the request fresh. Below is a diagram showing how the Needham-Schroeder protocol works.



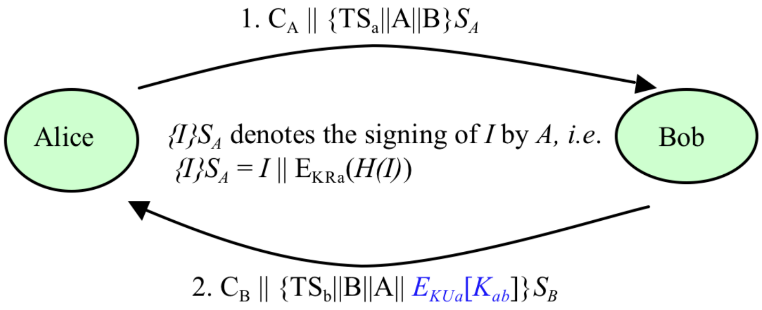
1. A sends a request to KDC for a session key to establish a secure channel with B.
2. KDC generates a random number K_{ab}, and replies with the response containing the session key K_{ab}, the original request from A matching the response with the request and an item (the session key and A's identity) which only B can view.
3. A forwards the item to B. At this point, the session key is securely delivered to A and B, and they may begin secure communication.
4. B sends a nonce N_b to A encrypted using the new session key.
5. A responds with $N_b - 1$.

Steps 4 and 5 assure B that the message received in 3 was not a reply, i.e. to authenticate A.

8.6 Distribution using PKC

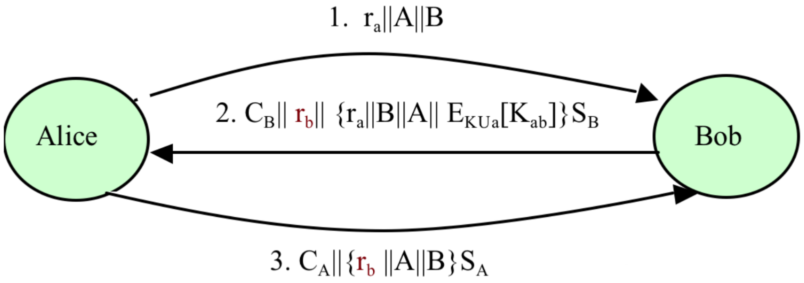
8.6.1 Two-passes Method

In this method there is secret key distribution with mutual authentication using a public key cryptosystem and timestamps:


1. $C_A \,||\, \{TS_a||A||B\}S_A$
Alice
$\{I\}S_A$ denotes the signing of I by A, i.e.
$\{I\}S_A = I \,||\, E_{KRa}(H(I))$
Bob
2. $C_B \,||\, \{TS_b||B||A||\, E_{KUa}[K_{ab}]\}S_B$

8.6.2 Three-passes Method

In this method there is symmetrical key distribution with mutual authentication using digital signatures and nonces:


1. $r_a||A||B$
2. $C_B||\, r_b||\, \{r_a||B||A||\, E_{KUa}[K_{ab}]\}S_B$
Alice
Bob
3. $C_A||\{r_b\, ||A||B\}S_A$

8.7 Summary of Secret Key Establishment Protocols

Protocols	ThirdParty	Timestamps	EntityAuth	Messages
Diffie-Hellman	No	No	None	2
Needham-Schroeder	KDC (online)	No	Symmetric encryption	5
Kerberos	KDC (online)	Yes	Symmetric encryption	4
X.509 (2 pass)	CA (offline)	Yes	mutual	2
X.509 (3 pass)	CA (offline)	No, but with nonce	mutual	3

9 Authentication

Some methods for user identification/authentication are:

- Where you are: location authentication based on IP addresses etc.
- Something you know: passwords, PIN.
- Something you have: keys, smart cards etc.
- Something you are: biometrics, voice recognition.

These methods combined may be used for a higher level of assurance.

9.1 Pasword-based Authentication

The unix system chooses not to store plaintext passwords, rather it stored encrypted/hashed passwords in the password file. Storing passwords for all the system users plainly visible in a password file is vulnerable to theft and accidental disclosure. THe hashing algorithm is a one-way function, called `crypt()` that is modified based upon the DES algorithm. It uses salt to make the DES based one-way function different from DES and to make dictionary attacks harder to succeed. The crypt algorithm:

- Using DES with the first 7 bits of the first 8 characters of password as the key.
- Iterated 25 times on constant string 0s; making the process slower.
- Using salt to perturb the DES algorithm, so that DES chip can not be used to (dictionary) attack the algorithm. It makes precompiled dictionary attacks harder (by a factor of 4096) and it prevents an identical password from producing the same encrypted password.
- The final 64 bits are unpacked into a string of 11 printable characters, called *the encrypted password*

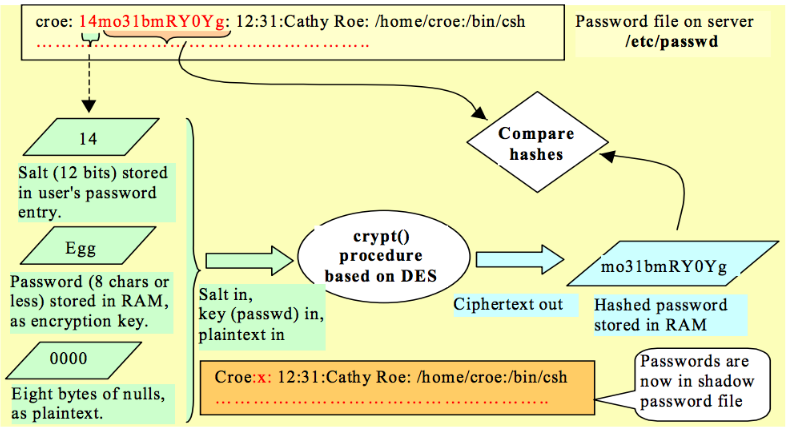
Most of today's implementations support MD5 in addition to the legacy DES-based `crypt()` function.

When a user have an account created or they change their password, the `/bin/password` program:

- Selects a salt based on the time of day; the DES salt is a 12-bit number, between 0 and 4095, that is converted into a two-character string and is stored in the `/etc/passwd` file along with the encrypted password.
- The password is used as the encryption key to encrypt a block of zero bits using `crypt()` to generate the encrypted password.

When a user tries to log in, the program /bin/login takes the password the user typed, and the salt from the password file, to generate a fresh encrypted password, and compares the newly generated one with the one stored in the /etc/passwd file. If the two encrypted results match, the system lets you in.

In the unix password file, each entry in the file is for one account and has several fields separated by colons, an image of this is below:


croe: 14mo31bmRY0Yg: 12:31:Cathy Roe: /home/croe:/bin/csh
Password file on server
/etc/passwd
Compare hashes
14
Salt (12 bits) stored in user's password entry.
Egg
Password (8 chars or less) stored in RAM, as encryption key.
0000
Eight bytes of nulls, as plaintext.
crypt() procedure based on DES
Salt in, key (passwd) in, plaintext in
Ciphertext out
mo31bmRY0Yg
Hashed password stored in RAM
Croe:x: 12:31:Cathy Roe: /home/croe:/bin/csh
Passwords are now in shadow password file

- User name (croe)
- Password (14mo31bmRY0Yg): the hashed password preceded by a salt value (14) to be used with the password.
- User ID (UID=12): a number assigned to this user name for system use in identifying the account.
- Group ID (GID=31): a number for the user's group.
- Home directory (/home/croe)
- Shell (/bin/csh): the user's default shell program.