COMP8411 - Cryptography and Network Security

Christopher Williamson

December 22, 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.

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

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:

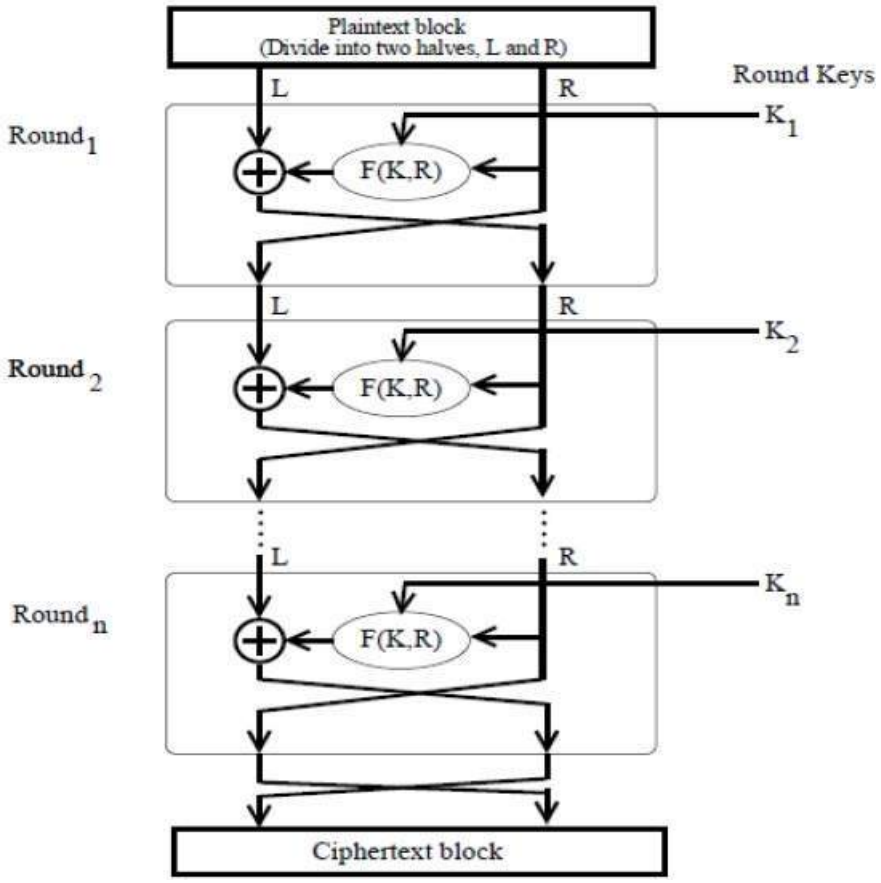
- Completeness - Each bit of the output should depend on every bit of the input and every bit of the key.
- Avalanche effect - Changing one bit in the input should change many bits in the output. Also, changing one key bit should result in the change of many bits in the output.

- 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. 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 MAC = 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