EXAMINATION MARKING SCHEME

Module: Internetwork Security (EE548)
Date: **13-03-2007**

<table>
<tr><th>Q#</th><th></th><th>MARK</th></tr>
<tr><td>Q1a</td><td>Using the Vigenère square given, it is relatively straightforward to work out that the plaintext is “Well done this is the correct answer”.</td><td>[5 Marks]</td></tr>
<tr><td>Q1b</td><td>The 5x5 matrix should be:
<table><tr><td>S</td><td>E</td><td>C</td><td>U</td><td>R</td></tr><tr><td>I/J</td><td>T</td><td>Y</td><td>A</td><td>B</td></tr><tr><td>D</td><td>F</td><td>G</td><td>H</td><td>K</td></tr><tr><td>L</td><td>M</td><td>N</td><td>O</td><td>P</td></tr><tr><td>Q</td><td>V</td><td>W</td><td>X</td><td>Z</td></tr></table>Split into digrams and encrypt:

I will travel by sea in the morning:

PT: iw – il – lt – ra – ve – lb – ys – ea – in – th – em – or – ni - ng
CT: yq – dq – mi – ub – et – pi – ic – ut – yl – af – tv – pu – ly – wn</td><td>[8 Marks]</td></tr>
<tr><td>Q1c</td><td>Electronic Codebook Mode

This first mode is the simplest of all five modes. Figure 7.10 shows the scheme where it can be seen that a block of plaintext (which is the same size in each case) is encrypted with the same key K. The term codebook is used because, for a given key, there is a unique ciphertext for every block of plaintext. Therefore we can imagine a gigantic codebook in which there is an entry for every possible plaintext pattern showing its corresponding ciphertext. If the message is longer than the block length then the procedure is to break the message into blocks of the required length padding the last block if necessary. As with encryption, decryption is performed one block at a time, always using the same key.

The ECB method is ideal for small amounts of data such as an encryption key however for larger messages if the same plaintext block appears more than once then the same ciphertext is produced. This may assist an attacker.

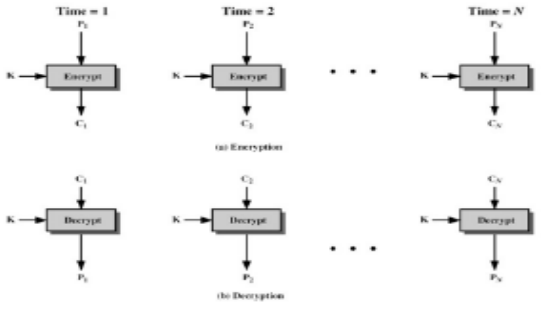
 

Figure 7.10: Electronic Codebook Mode (ECB)</td><td>[2 Marks]</td></tr>
</table>

Cipher Block Chaining (CBC) Mode

Cipher Block Chaining allows this by XORing each plaintext with the ciphertext from the previous round (the first round using an Initialisation Vector (IV)). The same key is used for each block. Decryption works as shown in the figure because of the properties of the XOR operation, i.e. IV XOR IV XOR P = P where IV is the Initialisation Vector and P is the plaintext. Obviously the IV needs to be known by both sender and received and it should be kept secret along with the key for maximum security.

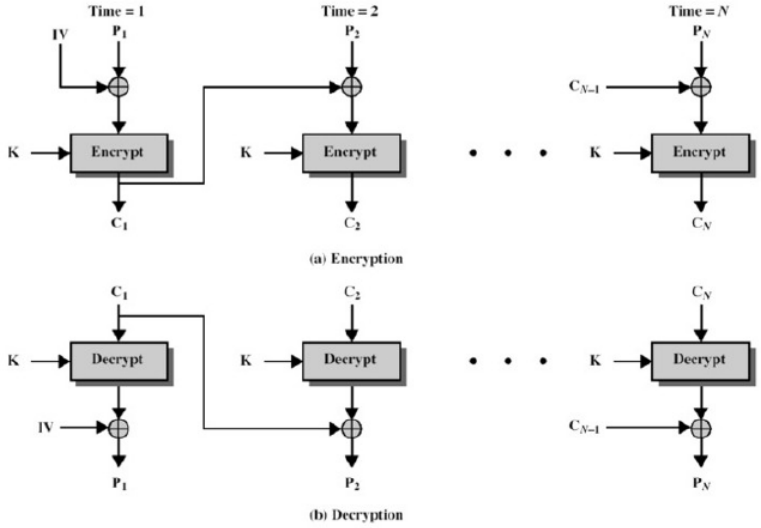


Figure 7.11: Cipher Block Chaining (CBC) Mode

[2 Marks]

Cipher Feedback (CFB) Mode

The Cipher Feedback and Output Feedback allows a block cipher to be converted into a stream cipher. This eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time.

Figure 7.12 shows the CFB scheme. In this figure it assumed that the unit of transmission is s bits; a common value is s = 8. As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext (which is split into s bit segments). The input to the encryption function is a shift register equal in length to the block cipher of the algorithm (although the diagram shows 64 bits, which is block size used by DES, this can be extended to other block sizes such as the 128 bits of AES). This is initially set to some Initialisation Vector (IV). The leftmost s bits of the output of the encryption function are XORed with the first segment of plaintext P_1 (also s bits) to produce the first unit of ciphertext C_1 which is then transmitted. In addition, the contents of the shift register are shifted left by s bits and C_1 is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted. Decryption is similar.

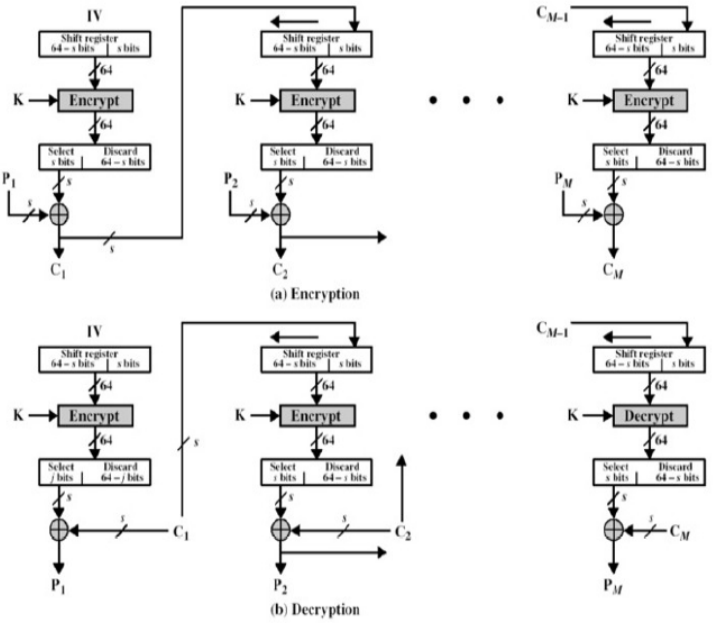


Figure 7.12: Cipher Feedback (CFB)

[2 Marks]

Output Feedback (OFB) Mode

The Output Feedback Mode is similar in structure to that of CFB, as seen in figure 7.13. As can be seen, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register. One advantage of the OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs in C_1 only the recovered value of P_1 is affected; subsequent plaintext units are not corrupted. With CFB, C_1 also serves as input to the shift register and therefore causes additional corruption downstream.

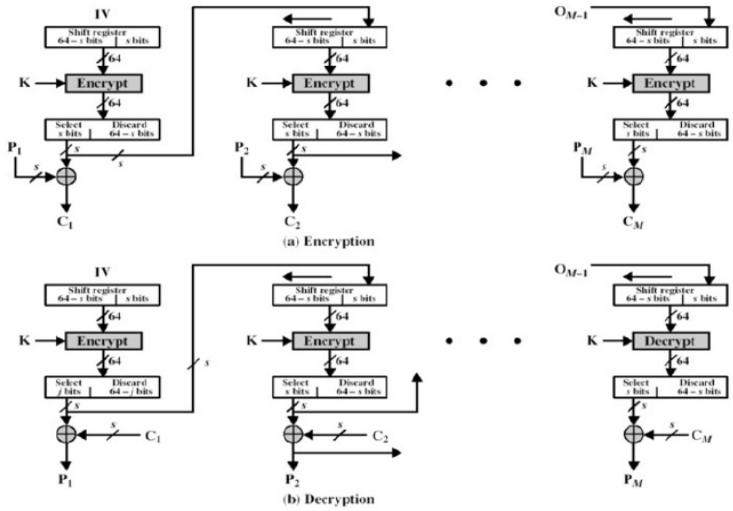


Figure 7.13: Output Feedback (OFB)

[2 Marks]

Counter Mode

This is a newer mode that was not listed initially with the above four. Interest in this mode has increased a good deal lately. A counter, equal to the plaintext block size is used. The only requirement stated in the standard is that the counter value must be different for each plaintext block that is encrypted. Typically, this counter is initialised to some value and then incremented by 1 for each subsequent block (modulo 2^b where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext to produce the ciphertext block; there is no chaining. For decryption, the

same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block. This mode contains a number of advantages including hardware efficiency, software efficiency, provable security (in the sense that it is at least as secure as the other modes discussed) and simplicity.

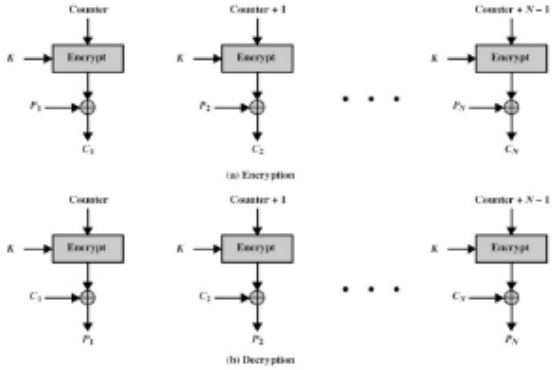


Figure 7.14: Counter (CTR) Mode

[2 Marks]

(+ 2 Marks for all five correct)

Q 2(a)

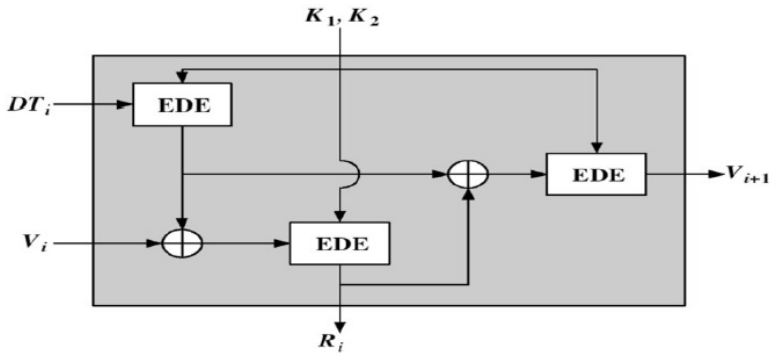
This is considered one of the strongest (cryptographically speaking) pseudorandom number generators. It is employed by a number of applications including financial security schemes and PGP (to be discussed later in the course). It is a U.S. Federal Information Processing Standard (FIPS) approved method. It makes use of Triple DES to produce random numbers. Although it uses triple DES three times, it only uses two keys as the same two keys are used three times. This is effectively the same as using a 112 bit key.

Figure 4.2 shows the scheme. The algorithm makes use of triple DES as mentioned and has the following inputs and output:

- *Input*: Two pseudorandom inputs drive the algorithm. One is a 64 bit representation of the current date and time updated on each number generation. The other is a 64 bit seed value initialized to some arbitrary value and updated during the generation process.
- *Keys*: The generator makes use of three triple DES encryption modules. All three make use of the same pair of 56 bit keys, which must be kept secret.
- *Output*: These are a 64 bit pseudorandom number (R_i) and a 64 bit seed value (V_{i+1}).

The strength of the PRNG of figure 4.2 can be attributed to the fact that there are a total of nine DES encryptions and a 112 bit key (effectively). As well as this, the scheme takes two pseudorandom inputs to begin with, both of which are not revealed (although it might be possible to work out DT_i). As a result the amount of information needed

by and attacker is formidable. Even if the attacker learns R_i, it would be impossible to deduce v_{i+1} due to extra EDE encryption.



[10 marks]

Q 2(b)

The linear congruential method was first introduced by D.H. Lehmer in 1948. The algorithm is parameterised as follows:

m	the modulus	$m > 0$
a	the multiplier	$0 < a < m$
c	the increment	$0 \leq c < m$
X_0	the starting value, or seed	$0 \leq X_0 < m$

The sequence of random numbers X_n is obtained via the following iterative equation:

$$X_{n+1} = (aX_n + c) \bmod m \tag{10}$$

If m,a,c and X_0 are integers then the technique will produce a sequence of integers with each integer in the range $0 \leq X_n < m$. The selection of a,c and m is critical in developing a good generator. For example, if $a = c = 1$ the sequence produced is no use as we simply obtain a sequence of integers incremented by 1 each time. However, consider $a = 7, c = 0, m = 32$ and $X_0 = 1$. The resultant sequence is also clearly unsatisfactory $\{7, 17, 23, 1, 7, \ldots\}$. Of the 32 possible values, only 4 are used (the sequence is said to have a period of 4). Changing $a = 5$ then the sequence is $\{5, 25, 29, 17, 21, 9, 13, 1, etc.\}$ which gives a period of 8. This is better but still far from what is desired.
The value m should be large so that there is potential for long sequences, usually m is nearly equal to the maximum nonnegative integer for a given computer e.g. 2^{31}.

The limitations of this system in cryptographic terms is that an opponent obtaining a small sequence of the output values can discover the parameters $(a, c, m)$to the compromise the system. For example, assume the attacker knows X_0, X_1, X_2, X_3 then it is possible to set up three simultaneous equations as follows:

$$X_1 = (a \cdot X_0 + c) \bmod m$$
$$X_2 = (a \cdot X_1 + c) \bmod m$$
$$X_3 = (a \cdot X_2 + c) \bmod m$$

As there are three equations in three unknowns, it is possible to solve for the three unknowns.

To make the sequence less reproducible, the sequence could be restarted after every N numbers. This can be achieved using the current clock value $(\bmod(m))$ as a new seed after N numbers (starting the new sequence). Another method could be to add the current value of the clock $(\bmod(m))$ to the random numbers produced.

[10 marks]

Q 2(c)

1. The function should be a full-period generating function (i.e. should generate all numbers between 0 and m before repeating).
2. The generated sequence should appear random. The sequence is not random at all but there is a wide variety of tests to assess the degree of randomness exhibited.
3. The function should implement efficiently with 32-bit arithmetic.

[5 marks]
(3*1.5 + .5)

Q3(a)

All cryptographic algorithms require time and space (memory) to execute. Clearly what is desired is an algorithm that executes as quickly as possible with the minimum number of resources possible, however, often this is not feasible. **Complexity Theory** deals with the resources required during computations to solve a given problem. The most common resources (for example) would be *time* and *space*. How much time is it going to take to solve a particular problem? Generally time is a valuable resource so we would like to keep it to a minimum. Also, how much memory (space) are we going to require to solve the problem? The PC in the lab only has 128MB so if it takes more than that we are going to have to buy more. Complexity theory allows us to work out how costly different algorithms (such as DES and RSA) are going to be. In fact, if you think about it, it also allows us to analyse attacks on cryptosystems as these will be algorithmic in nature as well. Using complexity theory we can therefore determine whether a particular attack is feasible against a cryptographic algorithm and can therefore give some indication of the security of an algorithm.

Of course one could say that the time the execution of an algorithm takes depends on the speed of the particular computer on which it is running, as well as the amount of memory it has. This is true, so what we would like is a method of comparing different algorithms that is independent not only of the speed and amount of RAM a particular machine has, but also of the size of the input (which is normally denoted by n).

A measure of the efficiency of an algorithm is known as its **Time Complexity**. The time complexity of an algorithm is defined to be $f(n)$ if for all n and all inputs of length n, the algorithm takes at most $f(n)$ steps. Thus for a given processor speed and a given size of input (n) the time complexity is an *upper bound* on the execution time of the algorithm. However, the definition is not precise:

- The meaning of "step" is not precise: Is it a single processor machine instruction or a single high-level language machine instruction? Fortunately, each definition of "step" is in general related to each other by a multiplicative constant and for large n these constants are not important. What is important however is the speed at which the relative execution time is growing with increasing n. For example, In RSA, if we are concerned about whether to use a 50 digit ($n = 10^{50}$) or a 100 digit ($n = 10^{100}$) key, it is not necessary (or possible) to know exactly how long it would take to break each size of key. However in ballpark figures, we can determine the level of effort and how much extra relative effort is required for larger key sizes.
- An exact formula for $f(n)$ is generally not available but that is not important. We need only an approximation and are interested primarily in the rate of change of $f(n)$ as n increases to very large values.

There is a standard mathematical notation known as the "big O" notation which is used to characterise the time complexity of algorithms. By definition:-

$$f(n) = O(g(n)) \text{ iff } \exists\, a, M \in Z^+ \text{ such that } |f(n)| \leq a \times |g(n)| \text{ for } n \geq M \tag{6.3}$$

For example we might say that an algorithm runs in $O(n^2)$ time. This means that for an input of size n the running time of the algorithm will be proportional to n^2. So if we double the input size n the running time will increase by a factor of 4. It must be emphasised that the "big O" notation does not tell us the running time of an algorithm. It only tells us how its performance changes with the size of the input.

Anything that takes a fixed amount of resources takes $O(1)$. For example if the amount of memory required by an algorithm is always 64MB then we can say that the algorithm takes O(1) space (and similarly for time).

In general, the "big O" notation makes use of the term that grows fastest:

- $O(ax^7 + 3x^3 + sin(x)) = O(x^7)$
- $O(e^n + an^{10} = O(e^n))$
- $O(n! + n^{50}) = O(n!)$

	An algorithm with an input of size n is said to be a). Linear if the running time is $O(n)$. b). Polynomial if the running time is $O(n^t)$ for some constant t. c). Exponential if the running time is $O(t^{h(n)})$ where t is some constant and $h(n)$ is a polynomial in n. Generally a problem that can be solved in polynomial time is considered feasible whereas anything worse than polynomial time is considered computationally infeasible - especially exponential time. N.B. If n is small enough, even complex algorithms become feasible.	[12 marks]
Q3(b)	The result can be shown to be {37} and m(x) is not needed in this case. The value of m(x) given here is that used in AES.	[7 marks]
Q3(c)	Say gcd(a,b) = d. Therefore d|a and d|b. a can be written as: a = qb + r where q and r are the quotient and remainder respectively. Therefore, a – qb = r. Because d|a and d|b => d| (a - qb) => d|r but r = amodb and therefore d|a, d|b and d|amodb. Therefore, gcd(a,b) = gcd(b,amodb) = d.	[6 marks]

Q#		MARK
Q4(a)	The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and $n-1$ for some n. The scheme is as follows: 1. Two extremely large, equal length, random prime numbers, p and q, are chosen carefully. 2. The number n is found such that $n = p \times q$. 3. The Euler phi (totient) function is found from n using the property $\phi(n) = \phi(pq) = (p-1)(q-1)$ 4. The value e is chosen at random such that $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$. 5. The value d is computed using the congruence relation $de \equiv 1 \pmod{\phi(n)}$ (i.e. d is the multiplicative inverse of e modulo $\phi(n)$). The ciphertext is then produced by $$C = M^e \bmod n \quad (1)$$ where C is the ciphertext, M is the plaintext and the set {e, n} constitute the public key. The original plaintext can then be reconstructed by $$M = C^d \bmod n \quad (2)$$ where the set {d, n} constitute the private key and the other variables are as before. Equation 2 uses the fact that $$de \equiv 1 \pmod{\phi(n)} \quad (3)$$ and that $$C^{\phi(n)} \equiv 1 \pmod{n}, \quad \text{where} \quad gcd(C, n) = 1 \quad (4)$$ which is Euler's theorem. This can be seen as $$C^d \bmod n = (M^e \bmod n)^d \bmod n = M^{ed} \bmod n \quad (5)$$ From equation 3 it is clear that $$de = \phi(n)q + 1 \quad (6)$$ for some integer q. Equation 5 can then be written as $$C^d \bmod n = M^{ed} \bmod n = M^{\phi(n)q+1} \bmod n \quad (7)$$ which then becomes $$M^{\phi(n)q} M \bmod n \quad (8)$$	

which by Euler's theorem can be seen to be equal to

$$M \bmod n \tag{9}$$

With regard to key generation, before two parties can use a public key system, each must generate a pair of keys. This involves the following tasks:

- Determining two prime numbers p, q
- Selecting either e or d and calculating the other.

Firstly, considering selection of p and q. Because the value $n = pq$ will be known to any opponent, to prevent the discovery of p, q through exhaustive methods, these primes must be chosen from a sufficiently large set (must be large numbers). On the other hand the method used for finding large primes must be reasonably efficient. At present there is no useful techniques that yields arbitrarily large primes. The procedure is to pick at random an odd number of the desired magnitude and test that it is prime. If not, repeat until a prime is found.
A variety of tests for primality have been developed, all of which are statistical in nature. The tests however may be run in such a way as to attain a probability of as near 1 as is desired that a particular number is prime. One of the more efficient algorithms is the Miller-Rabin scheme, which performs calculations on n, the candidate prime and a randomly chosen integer a. This procedure may be repeated as required.
In summary the procedure for picking a prime is as follows:

1. Pick an odd integer n at random (e.g., using a pseudorandom number generator).
2. Pick an integer $a < n$.
3. Perform the probabilistic primality test, (such as Miller-Rabin). If n fails the test then go to step 1.
4. If n passes a sufficient number of tests then accept it, otherwise go to step 3.

[12 Marks]

Q 4(b)

P=2,q=11 therefore n= 22 and phi(n) = 10. Also we have e=7 and we need to find its inverse modulo phi(n) which can be done using Euclids extended algorithm:

Q	A_1	A_2	A_3	B_1	B_2	B_3
-	1	0	20	0	1	7
2	0	1	7	1	-2	6
1	1	-2	6	-1	3	1

therefore, d = 3. We then have C=M^e mod n which means C = 18. The can be verified as M=C^d mod n where d is 3. M therefore equals 2.

[5 marks]

Q4(c)

1. $\mathbf{a}'$, a superincreasing vector (private, chosen).
2. m, an integer larger than the sum of all a_j's (private, chosen).
3. ω, an integer relatively prime to m (private, chosen).
4. ω^{-1}, the inverse of ω, modulo m (private, calculated).
5. $\mathbf{a}$, equal to $\omega\mathbf{a}' \pmod{m}$ (public, calculated).

The private key consists of the triple $(\omega^{-1}, m, \mathbf{a}')$ and the public key consists of the value of $\mathbf{a}$.

[8 marks]

Q5(a)

The Secure Sockets Layer (SSL) is a method for providing security for web based applications. It is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol but rather two layers of protocols as illustrated in figure 4. It can be seen one layer makes use of TCP directly. This layer is known as the **SSL Record Protocol**. This layer provides basic security services to various higher layer protocols. An independent protocol that makes use of this layer is the Hypertext Markup Language (HTTP) protocol. However three higher level protocols make use of this layer but are also part of the SSL stack. They are used in the management of SSL exchanges and are as follows:

1. SSL Handshake Protocol.
2. SSL Change Cipher Spec Protocol.
3. SSL Alert Protocol.

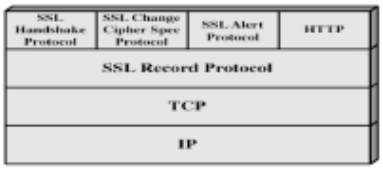


Figure 4: SSL protocol stack.

The **SSL record protocol** which is at a lower layer and offers services to these three protocols provides two services for SSL connections:

1. Confidentiality - using conventional encryption.
2. Message Integrity - using a Message Authentication Code (MAC).

In order to operate on data the protocol performs the following actions:

- It takes an application message to be transmitted and fragments it into manageable blocks. These block are $2^{14} = 16,384$ bytes or less.
- These blocks are then optionally compressed which must be lossless and may not increase the content length by more than 1024 bytes.
- A message authentication code is then computed over the compressed data using a shared secret key. This is then appended to the compressed (or plaintext) block.
- The compressed message plus MAC are then encrypted using symmetric encryption. Encryptions may not increase the content length by more than 1024 bytes, so that the total length may not exceed $2^{14} + 2048$. A number of different encryption algorithms are permitted.

- The final step is to prepend a header, consisting of the following fields:
  - Content type (8 bits) - The higher layer protocol used to process the enclosed fragment.
  - Major Version (8 bits) - Indicates major version of SSL in use.
  - Minor Version (8 bits) - Indicates minor version in use.
  - Compressed Length (16 bits) - The length in bytes of the compressed (or plaintext) fragment.

The overall format is shown in figure 5.

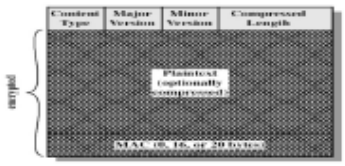
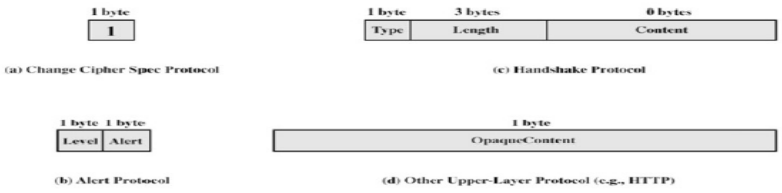


Figure 5: SSL record format.

The "content type" above is one of four types. The three higher level protocols given above that make use of the SSL record and a fourth known as "application_data". The first three are described next.

1. The **Change Cipher Spec Protocol** consists of a single message which consists of a single byte with the value 1. This is to cause the pending stage to be copied in to the current state which updates the cipher suite to be used on this connection.
2. The **Alert Protocol** is used to convey SSL-related alerts to the peer entity. It consists of two bytes the first of which takes the values 1 (warning) or 2 (fatal). If the level is fatal SSL immediately terminates the connection. The second byte contains a code that indicates the specific alert.
3. The **Handshake Protocol** allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in an SSL record. This protocol is used before any application data is sent. It consists of a series of messages exchanged by the client and server all of which have the format shown in figure 6. Each message has three fields:
   (a) Type (1 byte): Indicates one of 10 messages such as "hello_request".
   (b) Length (3 bytes): The length of the message in bytes.
   (c) content($\geq$ 0 byte): The parameters associated with this message such version of SSL being used.



The Handshake Protocol is shown in figure 7. This consists of four phases:

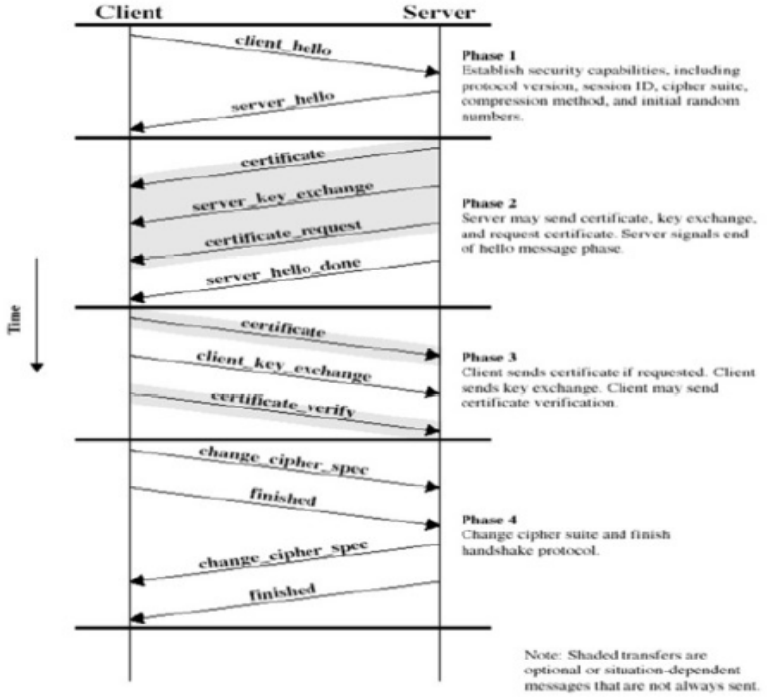
(a) Establish security capabilities including protocol version, session ID, cipher suite, compression method and initial random numbers.

(b) Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

(c) Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

(d) Change cipher suite and finish handshake protocol.

Q5(b)

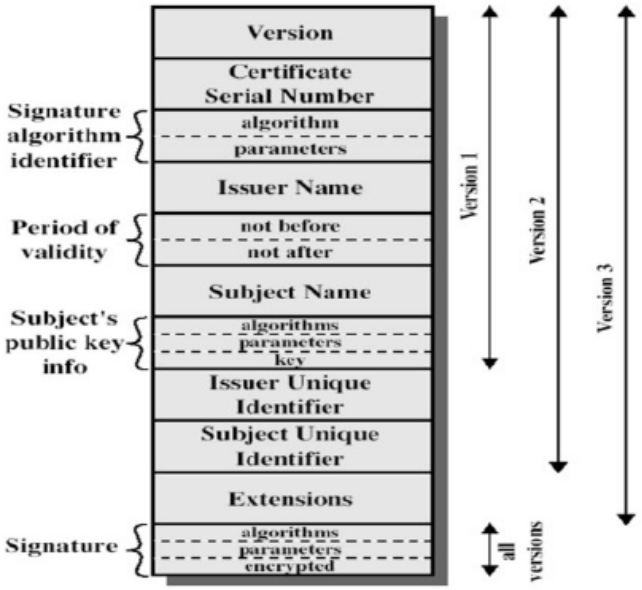


[12 marks]

- Version: Differentiates among successive versions of the certificate format;
- Serial Number: An integer value that is unambiguously associated with this certificate;
- Signature: The algorithm used to sign the certificate together with any parameters;
- Issuer name: X.500 name of the CA that created and signed this certificate;
- Period of validity: Two dates, the first and last on which the certificate is valid;
- Subject name: The name of the user to whom the certificate refers;
- Subject's public key information: The public key of the subject plus an identifier of the algorithm for which the key is to be used together with any associated parameters;
- Issuer unique identifier: An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities;
- Subject's unique identifier: An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities;
- Extensions: A set of one or more extension fields;
- Signature: Covers all of the other fields of the certificate. It contains the hash code of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifier.

EXAMINATION MARKING SCHEME

Module: Internetwork Security (EE548)
Date: **13-03-2007**

Q#		MARK
	 Figure 8: The X.509 certificate.	[8 marks]
	Five requirements not satisied by version two are: 1. The Subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user. 2. The subject field is also inadequate for many applications, which typically recognise entities by an Internet e-mail address, a URL, or some other Internet related identification. 3. There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy. 4. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability to a particular certificate. 5. It is important to be able to identify separately different keys used by the same owner at different times. This feature supports key life cycle management, in particular the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.	[5 marks]

EXAMINATION MARKING SCHEME

Module: Internetwork Security (EE548)
Date: **13-03-2007**

Q#		MARK
Q6(a)	In order to carry out a DPA attack, a method must be devised to produce a random set of J plaintext inputs that can be sent to the cryptosystem for encryption[1]. On receiving these plaintext inputs, pi_j, $1 \leq j \leq J$, the board will begin to run its algorithm and draw varying amounts of power. These power fluctuations can be sampled using a digital sampling oscilloscope which should be capable of sampling at about 20-30 times the clock frequency being used. The waveforms observed for each pi_j can be represented as a matrix wf_{jk}[2], where $1 \leq k \leq K$. A second column matrix, co_j, can also be used to represent the ciphertext output. In practice, each row of wf_{jk} would probably be stored as a separate file for ease of processing. Having captured each power waveform and ciphertext output, a function known as a *partitioning function*, D(.), must now be defined. This function will allow division of the matrix wf_{jk} into two sub-matrices $wf0_{pk}$ and $wf1_{qk}$ containing P and Q rows respectively, with $1 \leq p \leq P$ and $1 \leq q \leq Q$ where $P + Q = J$. Provided that the inputs pi_j were randomly produced, then $P = Q = J/2$ as $J \rightarrow \infty$ (i.e. the waveforms will be divided equally between the two sets). The partitioning function allows the division of wf_{jk} because it calculates the value of a particular bit, at particular times, during the operation of the algorithm. If the value of this bit is known, then it will also be known whether or not a power bias should have occurred in the captured waveform. For a 1, a bias should occur, and for a 0 it shouldn't. Separating the waveforms into two separate matrices (one in which the bias occurred and another in which it didn't) will allow averaging to reduce the noise and enhance the bias (if it occurred). For randomly chosen plaintexts, the output of the D(.) function will equal either a 1 or 0 with probability $\frac{1}{2}$ (this is just another statement of the fact that $P = Q = J/2$ as $J \rightarrow \infty$). An example of a partitioning function is: $$D(C_1, C_6, K_{16}) = C_1 \oplus SBOX1(C_6 \oplus K_{16}) \quad (11)$$ where SBOX1(.) is a function that outputs the target bit of S-box 1 in the last round of DES (in this case it's the first bit), C_1 is the one bit of co_j that is exclusive OR'ed with this bit, C_6 is the 6 bits of co_j that is exclusive OR'ed with the last rounds subkey and K_{16} is the 6 bits of the last round's subkey that is input into S-box 1. The value of this partitioning function must be calculated at some point throughout the algorithm. So, if the values C_1, C_6 and K_{16} can be determined, it will be known whether or not a power bias occurred in each waveform. It is assumed that the values C_1 and C_6 can be determined and the value of the subkey K_{16} is the information sought. To find this, an exhaustive search needs to be carried out. As it is 6 bits long, a total of $2^6 = 64$ subkeys will need to be tested. The right one will produce the correct value of the partitioning bit for every plaintext input. However, the incorrect one will only produce the correct result with probability $\frac{1}{2}$. In this case, the two sets $wf0_{pk}$ and $wf1_{qk}$ will contain a randomly[3] distributed collection of waveforms which will average [1] This method must be automated as the number of random plaintext inputs will be quite large. [2] The subscripts j and k are used to identify the plaintext number causing the waveform and the time sample point within that particular waveform, respectively. [3] Provided the plaintext inputs are randomly chosen.	

out to the same result. The differential trace (discussed below) will thus show a power bias for the correct key only. Of course it means that 64 differential traces are needed but this is a vast improvement over a brute force search of the entire 56 bit key.
Mathematically, the partitioning of wf_{jk} can be represented as

$$wf0_{pk} = \{wf_{jk} | D(.) = 0\} \tag{12}$$

and

$$wf1_{qk} = \{wf_{jk} | D(.) = 1\} \tag{13}$$

Once the matrices $wf0_{pk}$ and $wf1_{qk}$ have been set up, the average of each is then taken producing two waveforms $awf0_k$ and $awf1_k$ both consisting of K samples. By taking the averages of each, the noise gets reduced to very small levels but the power spikes in $wf1_{pk}$ will be reinforced. However, averaging will not reduce any periodic noise contained within the power waveforms and inherent to the operations on the cryptographic board. This can largely be eliminated by subtracting $awf0_{pk}$ from $awf1_{qk}$ (this can be thought of as demodulating a modulated signal to reveal the "baseband", where the periodic noise is the "carrier"). The only waveform remaining will be the one with a number of bias points identifying the positions where the target bit was manipulated. This trace is known as a *differential trace*, ΔD_k.
Again, in mathematical terms, the above can be stated as

$$awf0_k = \frac{1}{P} \sum_{wf_{jk} \in wf1} wf_{jk} = \frac{1}{P} \sum_{p=1}^{P} wf0_{pk} \tag{14}$$

and

$$awf1_k = \frac{1}{Q} \sum_{wf_{jk} \in wf0} wf_{jk} = \frac{1}{Q} \sum_{q=1}^{Q} wf1_{qk} \tag{15}$$

The differential trace ΔD_k is then obtained as

$$\Delta D_k = awf1_k - awf0_k \tag{16}$$

The last five equations can now be condensed into one:

$$\Delta D_k = \frac{\sum_{k=1}^{K} D(.) wf_{jk}}{\sum_{k=1}^{K} D(.)} - \frac{\sum_{k=1}^{K} (1 - D(.)) wf_{jk}}{\sum_{k=1}^{K} (1 - D(.))} \tag{17}$$

As $J \to \infty$, the power biases will average out to a value ϵ which will occur at times k_D - each time the target bit D was manipulated. In this limit, the averages $awf0_k$ and $awf1_k$ will tend toward the expectation $E\{wf0_k\}$ and $E\{wf1_k\}$, and equations 16 and 17 will converge to

$$E\{wf1_k\} - E\{wf0_k\} = \epsilon, \qquad \text{at times} \quad k = k_D \tag{18}$$

and

$$E\{wf1_k\} - E\{wf0_k\} = 0, \qquad \text{at times} \quad k \neq k_D \tag{19}$$

Therefore, at times $k = k_D$, there will be a power bias ϵ visible in the differential trace. At all other times, the power will be independent of the target bit and the differential trace will tend towards 0.
The above will only work if the subkey guess was correct. For all other guesses the partitioning function will separate the waveforms randomly, and equations 18 and 19 will condense to

$$E\{wf1_k\} - E\{wf0_k\} = 0, \qquad \forall k \tag{20}$$

As mentioned above, 64 differential traces are needed to determine which key is the correct one. Theoretically, the one containing bias spikes will allow determination of the correct key however, in reality the other waveforms will contain small spikes due to factors such as non-random choices of plaintext inputs, statistical biases in the S-boxes and a non-infinite number of waveforms collected. Generally however, the correct key will show the largest bias spikes and can still be determined quite easily.
The other 42 bits from the last round's subkey can be determined by applying the same method to the other 7 S-boxes[4]. A brute force search can then be used to obtain the remaining 8 bits of the 56 bit key.

[13 marks]

EXAMINATION MARKING SCHEME

Module: Internetwork Security (EE548)
Date: **13-03-2007**

Q#		MARK
Q6(b)	The following could be used as mitigation techniques for power attacks in general: 1. Timing Randomisation: This involves placing random time delays into the software so that a power analysis will not be possible. With random delays introduced, a steady trigger will not be sufficient to allow the averaging to work and will therefore act as a countermeasure. 2. Internal power supplies/power supply filtering: This would be another method that could be used to reduce the possibility of a power attack. For example, Adi Shamir proposes building a simple capacitance network into each smartcard to allow the fluctuations to be contained within the smartcard itself thereby preventing power attacks. 3. Data masking: One of the methods proposed consists of *masking* the intermediate data (i.e. mask the input data and key before executing the algorithm). This would make the power fluctuations independent of the actual data. 4. Tamper Resistance: This involves placing some detection/prevention system around the device to stop intruders gaining access to the power fluctuations. 5. Fail Counters: A differential power analysis attack requires the attacker to obtain a significant number of power waveforms. In order to do this the attacker must have the ability to run quite a few encryptions on the system under attack. If the number of encryptions were limited to a certain number then the attacks would become increasingly difficult. 6. Removal of conditional elements: One of the main features used to attack the square and multiply algorithm is the fact that it has a conditional multiplication that depends on the value of the exponent bit being operated upon. One suggested countermeasure is to implement this multiplication in every round (regardless of the value of the bit) and to only do a register update when the bit is a 1.	[8 marks]

Q6(c)	1. Unconditionally Secure defines a system where the ciphertext generated does not contain enough information to determine uniquely the corresponding plaintext no matter how much ciphertext is available or how much computation power the attacker has. 2. Computationally Secure defines a system where the cost of breaking the cipher exceeds the value of the encrypted information and the time required to break the cipher exceeds the useful lifetime of the information 3. A cryptanalytic attack is one that tries to attack the mathematical weaknesses of the algorithm 4. An implementation attack is one that tries to attack the actual implementation of the cryptographic system such as a smart card.	[4 marks]