ID: .....

Queen Mary
University of London

Friday 22 February 2019 9:00 to 10:50 AM

ECS655U Security Engineering Midterm (20%) **Duration:** 110 **minutes**

SOLUTIONS AND MARKING SCHEME

YOU ARE NOT PERMITTED TO READ THE CONTENTS OF THIS QUESTION PAPER UNTIL INSTRUCTED TO DO SO BY AN INVIGILATOR.

Instructions: Check whether the student ID on this page matches yours. This paper contains FOUR questions. **Answer all questions**. Cross out any answers that you do not wish to be marked.

Calculators are not permitted in this examination.

Possession of unauthorised material at any time when under examination conditions is an assessment offence and can lead to expulsion from QMUL. Check now to ensure you do not have any notes, mobile phones, smartwatches or unauthorised electronic devices on your person. If you do, raise your hand and give them to an invigilator immediately.

Exam papers must not be removed from the exam room.

Student ID Number: .....

Question	Points	Score
1	30	
2	30	
3	30	
4	30	
Total:	120	

Leave this table blank.

Examiner: Dr Arman Khouzani

Question 1

(a) This question is about basic security services and cryptographic primitives.

(i) Provide a short definition for each of the following security services:

[5 marks — basic]

▷ Data Origin Authentication

▷ Non Repudiation

Solution:

Data Origin AUthentication: the assurance that a given data has been created by an authorised entity and has not been modified since its creation. In other words, the assurance that a given entity is the source of a piece of data (in its entirety).

Non-Repudiation: the assurance that an entity cannot deny a previous commitment or action (in case of a dispute), e.g. creating a specific piece of data.

Marking scheme:

▷ 3 points for each, capped at 5 in total. The exact wording is not important as long as the concept is delivered.

(ii) Does "Non-Repudiation" need "Data-Origin Authentication"? In other words, do we need to have data origin authentication if we want to have non-repudiation? Support your answer with a brief explanation.

[5 marks — basic]

Solution: Yes: if we don't have a means to ascertain/prove the origin of a data, an entity can always deny to be its creator.

Marking scheme:

▷ 2 points for 'yes'. 3 points for any correct explanation.

(iii) Determine which one of the above two security services are provided by "digital signatures". Provide a brief explanation with your answer.

[5 marks — basic]

Solution: Both: "Non-Repudiation" is because if the signature verifies, it means that only the entity who has had the correct signing (private) key could have created the signature on this specific message, and the message has not changed ever since. So that entity cannot deny it because no other entity has had access to that signing (private) key. DOA follows up from Non-repudiation as well.

Marking scheme:

▷ 3 points for the correct identification of "both". 2 points if only one of them is mentioned without the other. 1 point if one of them is mentioned and explicitly said not the other! The remaining 2 points is for a(ny) correct explanation.

(b) This question is regarding a generic "symmetric-key" cipher. Using the following notations, answer each question.

Notation	Description
m	a message (plaintext)
c	a ciphertext
k	a secret key
$E(m,k)$	Encryption algorithm applied to message m (the first argument) using secret key k (the second argument)
$D(c,k)$	Decryption algorithm applied to ciphertext c (the first argument) using secret key k (the second argument)
$\forall$	a logic symbol which means "given any", or "for all". For example: $\forall k, c$ means given any key k and c (i.e., for any given k and any given c)

(i) Provide the "correctness" condition of this (symmetric-key) cipher. That is, what relation needs to hold for this cipher to be "correct"? Your answer can be very briefly stated in terms of the notations in the table. But if you cannot, express it in words using as much of the notations in the table as you can. **[5 marks — basic]**

Solution: $\forall k, m \quad D(E(m,k),k) = m$.

Marking scheme: 5 points. 1 point only (so -4) if the order of encryption and decryption are reversed. (4) point if either for all messages, or for all keys is not mentioned (so 3 points if neither one is mentioned). If the key is missing (-1) for each encryption or decryption. (-1) for each mistake in the order of the arguments (-3 if inconsistent mistake in the orders). The following gets a full mark too
Alternative potential answers and their score:

- ▷ 5/5: $\forall k, m: \quad E(m,k) = c \Rightarrow D(c,k) = m$.
- ▷ 4/5: $\forall k, m: \quad E(m,k) = c \Leftrightarrow D(c,k) = m$.

(ii) Does the following statement hold for a correct symmetric cipher? That is, does a symmetric cipher have to satisfy the following property? Provide a brief reasoning.
[5 marks — medium]

$$\forall k, m_1, m_2: \quad m_1 \neq m_2 \Rightarrow E(m_1,k) \neq E(m_2,k)$$

Solution: Yes (correct): this is to ensure that all ciphertext can be decrypted without ambiguity.
(In detail: suppose not, and for some key k and distinct messages m_1, m_2 ($m_1 \neq m_2$), we have $E(m_1,k) = E(m_2,k) = c$. Then there is no way to decrypt c using k without

ambiguity, because both distinct m_1 and m_2 could have created it for this key. In other words, information is irreversibly destroyed!)

Marking scheme: 2 points for the correct identification and 3 points for a correct explanation. A brief explanation is enough, no need for a detailed one.

(iii) Answer the above question about the following statement. Again, is the following property necessary for a symmetric cipher? Support your answer with a brief reasoning.

[5 marks — medium]

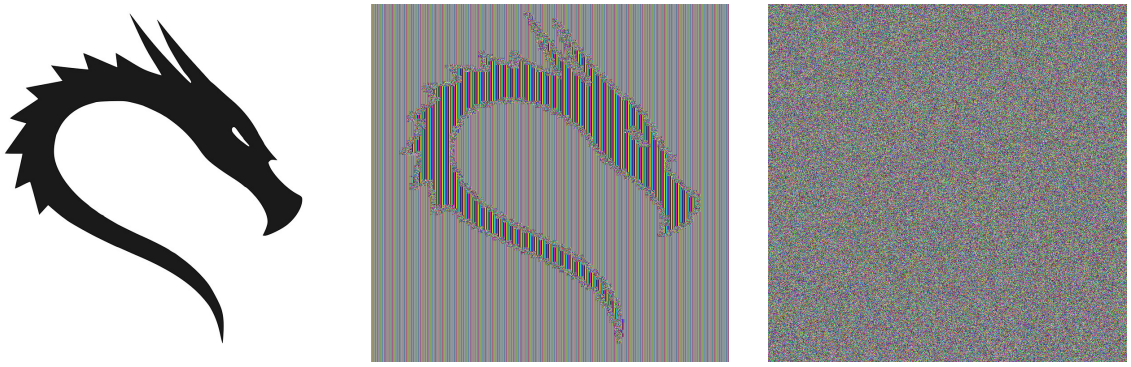
$$\forall k_1, k_2, m_1, m_2 : \quad m_1 \neq m_2, k_1 \neq k_2 \Rightarrow E(m_1, k_1) \neq E(m_2, k_2)$$

Solution: No (this is not a necessary condition): the same ciphertext can be created from different messages given different keys.
(detail: in fact, not only this is not necessary, this is a highly undesirable property! We do want the adversary that sees a ciphertext to have the ambiguity about which message could have created it, because many messages could have created that ciphertext given different keys, and the adversary cannot break the ambiguity because they don't have the key. Otherwise, if a ciphertext could only be created from one message only, the message is revealed!)

Marking scheme: 2 points for the correct identification and 3 points for a correct explanation. A brief explanation is enough, no need for a detailed one.

Question 2

(a) Figure 1a shows an image in a format in which each pixel is represented in one byte. Figures 1b and 1c show the result of AES encryption of that image using ECB and CBC modes of operations respectively (using the same key). Elaborate why we can see a pattern of the original image in Fig. 1b but not in Fig. 1c?



(a) plaintext (b) ciphertext using ECB mode (c) ciphertext using CBC mode

Figure 1: Encryption of pixel-encoded image (a) using the ECB mode of operation (b) and the CBC mode of operation of the AES block cipher.

[10 marks — basic]

Solution: AES, as a block cipher, works the following way: the plaintext – here, the picture in (1a) – is chopped into plaintext blocks (which for AES is 16 bytes long each). Then in ECB mode, each block is *independently* encrypted using the encryption key. In particular, if the same plaintext block is encounter, irrespective of its location in the plaintext, the same ciphertext block is produced. This is what is called lack of "positional dependency", that is, the ciphertext is not dependent on the position of the plaintext block, and only depends on the plaintext block and the key. So in the figure, wherever we have repeated plaintext blocks (16 bytes that are the same from different parts of the picture, e.g. a 16-byte block of white background), the same ciphertext is generated. This is why the pattern is visible in the resulting ciphertext in (1b). In contrast, the CBC mode using the encryption of the previous block and XOR it with the input block before encrypting it, so even if the plaintext block is the same, the input to the encryption block is dependent on all the previous encryptions, which introduces "positional dependency": so the same plaintext bock no longer produces the same ciphertext block, but rather the ciphertext depends on the position of the plaintext as well. This is why the ciphertext in (1c) looks random and no trace of the ciphertext is visible.

Marking scheme: 10 points for a complete answer. (8/10) if the explanation is generally correct but the connection to the picture is not made explicit. 5 points for just mentioning lack of positional dependency of ECB mode of operation as opposed to CBC. Note: the key is the same in both modes of operations: in CBC too, the encryption key is the same across different blocks! it is the input that is processed differently!

(b) Consider the following diagram describing the encryption algorithm in a particular mode of operation of a block cipher like AES, and answer the subsequent questions.

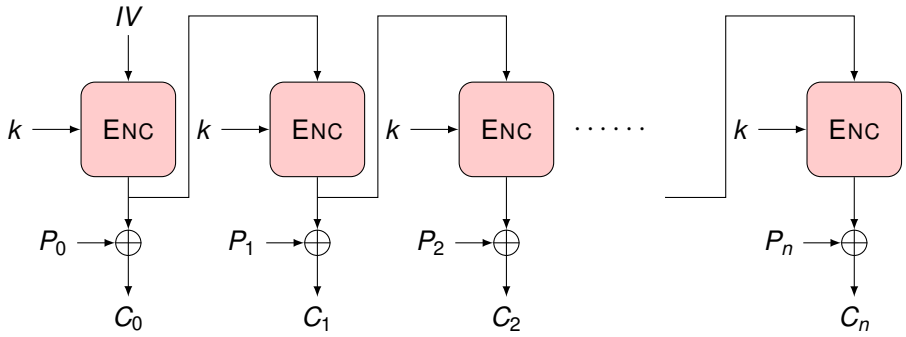
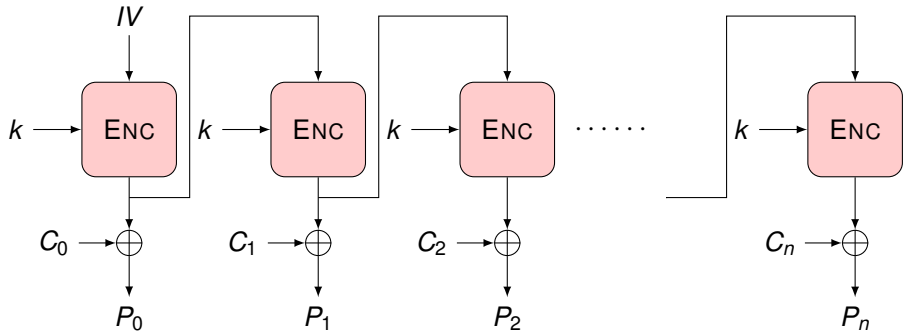


Figure 2: Q2(b) Notations: $P_0,\ldots,P_n$: blocks of plaintext. $C_0,\ldots,C_n$: blocks of ciphertext. k: cryptographic key. ENC: an encryption block. $\oplus$: XOR (eXclusive OR) operator. IV: initialisation vector.

(i) Draw the diagram of the corresponding decryption process (with clear labels). Hint: this mode of operation was not explicitly presented in class, but you should be able to answer this question from basic principles. Start from C_0 and try to recover P_0 from it, and then P_1, P_2, and from there a general P_i. **[10 marks — medium]**

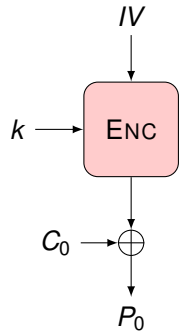
Solution:



This is enough for the answer. But if you want a guide on how to get to this answer: First, let's see how to get P_0: from the encryption figure, IV goes through the encryption block using key k, and the output is XORed with P_0 to yield C_0, so to get P_0 back, we need to XOR C_0 with the encryption of IV using key k (recall: XORing twice with the same thing cancels its effect!). Mathematically, if you wish:

$$C_0 = E_k(IV) \oplus P_0 \Rightarrow E_k(IV) \oplus C_0 = E_k(IV) \oplus E_k(IV) \oplus P_0 = P_0$$

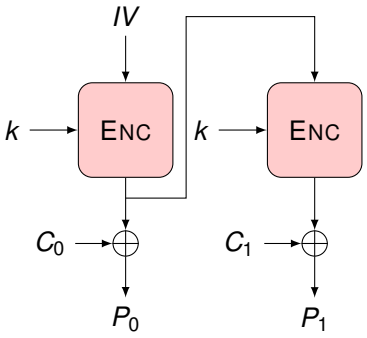
Therefore, so far we have:



Now, let's see how to get P_1: From encryption algorithm, we see that C_1 is produced by XORing P_1 with the output of Encryption of something coming from the previous stage with key k, which itself was the Encryption of the IV using key k, so again, if we want to get P_1 back, we need to XOR C_1 with the same thing (to cancel its effect, because XORing twice with the same block of bits cancels its effect!). Mathematically, if you wish:

$$C_1 = E_k\left(E_k(IV)\right) \oplus P_1 \Rightarrow E_k\left(E_k(IV)\right) \oplus C_1 = E_k\left(E_k(IV)\right) \oplus E_k\left(E_k(IV)\right) \oplus P_1 = P_1$$

which gives:



The next plaintexts follow similarly, and we get the entire algorithm as the answer.

Marking scheme: (10/10) if all correct including the labels. (4/10) points if only the first block is correct. (7/10) if all is correct except that ENC block is replaced by DEC block. (-2) if keys missing. (-2) if IV missing.

(ii) Analyse the "error-propagation" of this mode of operation (in Figure 2). That is, if a single bit of the ciphertext is flipped (e.g. due to noise in the channel), what would be the effect on the recovered plaintext at the receiver. **[5 marks — advanced]**

Solution: only one bit (and in the same position): this is because the value of the counter is not affected by the error in the cipher-block, so what comes out of the encryption block is correct. This correct block then gets XORed with a cipher-text that has only one bit error (in the same position).

Marking scheme:

▷ 5 points for correctly identifying 1 bit. 2 or 3 points if only mentioned one block is affected, depending on the reasoning.

(iii) Provide one advantage and one disadvantage of this mode of operation compared to the CTR (i.e., Counter) mode. **[5 marks — medium]**

Solution:

advantage does not require a reliable synchronised counter between sender and receiver (unlike CTR mode)

disadvantage cannot be parallelized (unlike CTR) – (not required for answer: this is because in this mode, encryption of a plaintext depends on the encryption of the previous plaintext-blocks, so it has to be done sequentially. Same holds for decryption process as well).

Note: you didn't need to get the previous parts correctly to be able to answer this part correctly, if you think strategically!

Marking scheme:

▷ 3 points for a correct advantage and 3 points for a correct disadvantage, capped at 5 in total. If a correct advantage or disadvantage is mixed with incorrect ones, the point for the correct one will be capped at 2 instead.

Question 3

(a) A cryptographic hash function on an input has created the following output in hex (base 16) format (40 hexadecimal digits):

`644cf7d904b7f65b536627b54cb4f9ba0840fc28`

(i) On average, how many random inputs should you compute with this cryptographic hash function in order to find an input that gives you this specific output with at least 50% chance? **[5 marks — advanced]**

> **Solution:** The length of the output is $40 \times 4 = 160$ bits. So in order to get this specific output with probability higher than 1/2, we need to try 2^{159} random inputs on average. (Note that this is not about finding "any" collision, for which the birthday paradox applies, but rather it is to find a specific given output (so a first pre-image attack), for which, brute force needs about 2^{n-1} random tries for a chance of at least 1/2 where n is the length of the output in bits.)
>
> ---
>
> *Marking scheme:*
>
> ▷ Acceptable answers are 2^{159}. 4/5 for 2^{160} (or equivalently 16^{40}). For a correct answer, no explanation is required. Example wrong answers (which will receive 0/5) are: 2^{80}, or 2^{40}, 2^{20}, etc.

(ii) If a single bit of the input changes, how many of the output hex characters do you expect to change on average? (Recall that each hex character/digit represents 4 bits.)

[5 marks — advanced]

> **Solution:** If there is any change in the input (one bit or any number of bits), the output should be as if it is completely randomly picked again, so on average half of the bits are going to change. But this question is asking about the hex characters. For a hex character to stay the same, all of the 4 bits that it represent should stay the same. The probability that each bit stays the same is $\frac{1}{2}$, so the probability that the entire 4 bit of a hex stay the same is $\left(\frac{1}{2}\right)^4 = \frac{1}{16}$. So the probability that a hex character changes is $1 - \frac{1}{16} = \frac{15}{16}$. So on average, the number of hex characters that changes are $40 \times \frac{15}{16} = 37.5$
>
> ---
>
> *Marking scheme:*
>
> ▷ 5/5 for the correct answer. 3/5 if the answer is the majority or almost all, etc. But no points for "half". 2/5 for "all".

(b) A digital signature has a signing and a verification algorithm. Focusing on the verification algorithm, identify its input and output arguments (what input arguments does it take and what output(s) does it produce?). **[5 marks — basic]**

Solution:

3 inputs message (of arbitrary length e.g. byte-array type), digital signature (byte-array of fixed known length), verification key (byte-array of known length).

1 output True/False (Boolean) on whether or not the signature verifies (if the digital signature is created using the signing key corresponding to the verification key on this specific message).

Marking scheme:

▷ -2 points for each mistake/or missing argument.
▷ -1 if one of the inputs by mistake is said to be the hash algorithm.

(c) This part is regarding Public key certificates.

(i) What are the essential components of a public key certificate (without which, the certificate cannot serve its basic functionality)?

[5 marks — medium]

Solution: The "essential" ones are the following:

1. Name/Identity of the owner;
2. Public key of the owner;
3. Name of the issuer, i.e., the immediate Certificate Authority; (along with its own chain of trust, if it is an intermediary)
4. Digital signature of the issuer on the rest.

(Note: these are essential because it would be impossible to verify a certificate without any of them. Other ingredients are not functionally essential.)

Marking scheme:

▷ (-1) point for each of the above missing (but of course, 0 if all of them are missing!). No points for non-essential ones (e.g. the expiration date, the public key of the issuer, etc) with the exception of "purpose of use", which gets 0.5 point.

(ii) Briefly describe the procedure of verifying a certificate (What do we need to do to verify a certificate?)

[5 marks — medium]

Solution: First, if there is any expiration date, or the name on the certificate does not match the presenter, or the certificate is relocated (is listed in a CRL), then it is rejected as invalid. Otherwise, we get to the essential part:
The essential part is the following: If the issuer is a trusted CA, it means we already

have its verification key, so we can use that to verify the signature on the certificate (input the digital signature on the certificate, the rest of the certificate as the message, and the verification key). If the signature verifies, the certificate is valid. If the issuer is not immediately trusted, then this process should be repeated for its own certificate until a trusted CA is reached in its chain of trust.

Marking scheme:

▷ If the essential part is described, complete point. Otherwise, if the non-essential part is described only, 2/5.

(iii) Suppose a certificate verifies. What conclusion can we make?

[5 marks — medium]

Solution: Only proves that the owner named in the certificate indeed owns that public key, i.e., that public key indeed belongs to this owner named. Nothing more!

Marking scheme:

▷ Anything else gets no points.

Question 4

(a) Consider the following cryptographic protocol:

- Alice (the sender) and Bob (receiver) have established two symmetric keys k_1 and k_2. That is, k_1 and k_2 are only known by (both) Alice and Bob.
- Both parties have also already agreed on the choice of a strong symmetric-key encryption algorithm E, and a strong message-authentication-code algorithm MAC.
- Alice (the sender) performs the following on her message m, and sends the output y to Bob over a public channel:

$$y = E_{k_1}(m) \,||\, MAC_{k_2}\left(E_{k_1}(m)\right)$$

(i) Describe the corresponding process in the receiver; i.e., what does Bob do to y upon receiving it?

[5 marks — basic]

Solution:
Bob first de-concatenates the two parts of y, as the second part is supposed to be a MAC which has a known fixed length. Bob then uses the symmetric key k_2 a compute a fresh MAC of the first part on his own. He then compares this with the second part of the y and if they don't match, he rejects the message. If they do match, he proceeds by passing the first part of y into the decryption algorithm with key k_1 and accepts the result as the legitimate message sent by Alice.

Marking scheme:

- ▷ 5 points for a correct identification of the steps.
- ▷ 3 points if it is said that Bob first decrypts the first part using k_1 and then computes MAC of the first part (and accepts if it matches the second part, rejects otherwise.)
- ▷ Only 1 point if it is said that Bob decrypts the first part using k_1 and then computes the MAC using k_2 on the result of the decryption (or if it left ambiguous on what exactly is the MAC computed).
- ▷ (-1) point for each wrong key, or for each time the key is not explicitly mentioned. So if the answer is: "first Bob de-concatenates y into first and second parts, and then computes the MAC of the first part and compares against the second part, if they match, he decrypts the first part". This will only get (3/5).
- ▷ (-3) if it is claimed that decryption or MAC is generated on the entire y.

(ii) For each of the following security services, determine whether our protocol provides it. Each of your answers should be supported by a brief but clear justification:

- Confidentiality
- Data Integrity

[5 marks — basic]

Solution:

- confidentiality: yes, through encryption, since only an entity who has k_1 can decrypt the message and see the plaintext.
- data integrity: yes, through the use of MAC: only a person with knowledge of k_2 could have created a correct MAC on the first part.

Marking scheme:

- ▷ 3 points for each correct entry as a pair of correct identification and correct explanation; Capped at 5 in total.
- ▷ 2 point for an entry with correct identification but incomplete explanation.
- ▷ 1 point for an entry with just a correct identification but no explanation.

(b) Suppose Alice wants to access her photos on her account on `wonderland.wl`, which are located at the following url: `wonderland.wl/users/alice/photos`. As a means of authentication, she needs to create the following HTTP request:

```
GET /users/alice/photos HTTP/1.1
Host: wonderland.wl
Authentication: hmac username:[digest]
```

in which:

```
digest = base64encode(hmac("sha512","key","GET||/users/alice/photos"))
```

where "key" is a shared secret key between Alice and the server, and "sha512" is the hashing algorithm to be used in computing the HMAC. Note that `base64encode` is just a text-based encoding of binaries.

There is unfortunately no SSL/TLS and we have to send this request over TCP/IP directly. Can an attacker (Eve) that can sniff the traffic use this request to `POST` a new photo to Alice's account? Explain your answer.

[5 marks — basic]

Hint: Eve needs to be able to construct the following message (a http post request with the correct header):

```
POST /users/alice/photos HTTP/1.1
Host: wonderland.wl
Authentication: hmac username:[digest]
```

where

```
digest = base64encode(hmac("sha512","key","POST||/users/alice/photos"))
```

Solution: No: even though Eve has the hmac of the message `GET||/users/alice/photos`, she cannot create a correct hmac of a different message `POST||/users/alice/photos` without having the secret "key".

Marking scheme: 2 points for the correct identification and 3 points for a correct explanation. A brief but correct explanation is enough, no need for a detailed one.

(c) Now suppose that the system is upgraded and in the new system, Alice needs to create the following digest instead:

```
digest = base64encode(hmac("sha512", "key",
                    "GET||/users/alice/photos||<timestamp>||<nonce>"))
```

where `<nonce>` is a pseudorandom number generated by Alice, and the timestamp is a global time-stamp in the internet, e.g. `28 FEB 2019 10:00:00`.

Alice then sends the following `GET` request to view her photos:

```
GET /users/alice/photos HTTP/1.1
Host: wonderland.wl
Authentication: hmac username:<nonce>:[digest]
Timestamp: 28 FEB 2019 10:00:00
```

(i) What problem is the new upgrade trying to resolve? Explain with a specific example scenario.

[5 marks — medium]

Solution: The previous version lacked entity authentication and was susceptible to replay attacks. Specifically, Eve could resend (replay) the same message of Alice and view her photos. Also, in case Alice ever sends a post request, Eve then will be able to post pictures using that header as well. In the new version, the previously captured digest are no loner valid because of the introduced freshness mechanisms.

Marking scheme: 5 points if a detailed replay attack scenario in the previous version is mentioned. 3 points if only lack of entity authentication is mentioned without giving an explicit scenario. 2 points if only lack of freshness mechanism is mentioned.

(ii) Why does Alice need to include the `nonce` "in the clear" as well as using it inside the digest? (pay attention to the line "`Authentication: hmac username:<nonce>:[digest]`").

[5 marks — medium]

Solution: Because the nonce is created by Alice and the server needs it in order to be able to verify the MAC (the server needs to create the same input to create its own MAC on it and compare against the provided digest.)

Marking scheme: 5 points. No points if the answer is for "freshness" or "further strength" or such. The questions is explicitly about why the nonce is passed in the clear *as well as* in the digest. That is, what will go wrong if the nonce is used in the digest but not passed along to the server.

(iii) What happens if we had only used the <nonce> and not timestamp in this new mechanism? What about if we had only used the time-stamp?

[5 marks — advanced]

Solution: This is actually two (not-so-easy) questions wrapped into one:

- *if we had only used <nonce> and not timestamp*: since the nonce is created by Alice (and not the server, as in a challenge-response protocol), this way of freshness does not work! There is no practical way for the server to check whether this "nonce" is really a fresh one, or a replayed one by Eve from a past interaction.
- *if we had only used timestamp and not <nonce>*: Note that the server has to accept the time-stamp if it is within an permissible window (to allow for the network and processor latencies, etc). This is a window of exposure within which Eve can replay the request. To counter this, the server can make the window of acceptance very narrow, but this would make the authentication mechanism too sensitive to the delays, and many legitimate request may also be rejected. By including the <nonce>, the server can maintain a list of previously received nonces "within a sliding time-window" and reject a request if a nonce is not new.

Marking scheme: 3 points for a correct explanation of each, capped at 5 in total.

End of questions