CS5285

Information Security for eCommerce

Dr. Gerhard Hancke

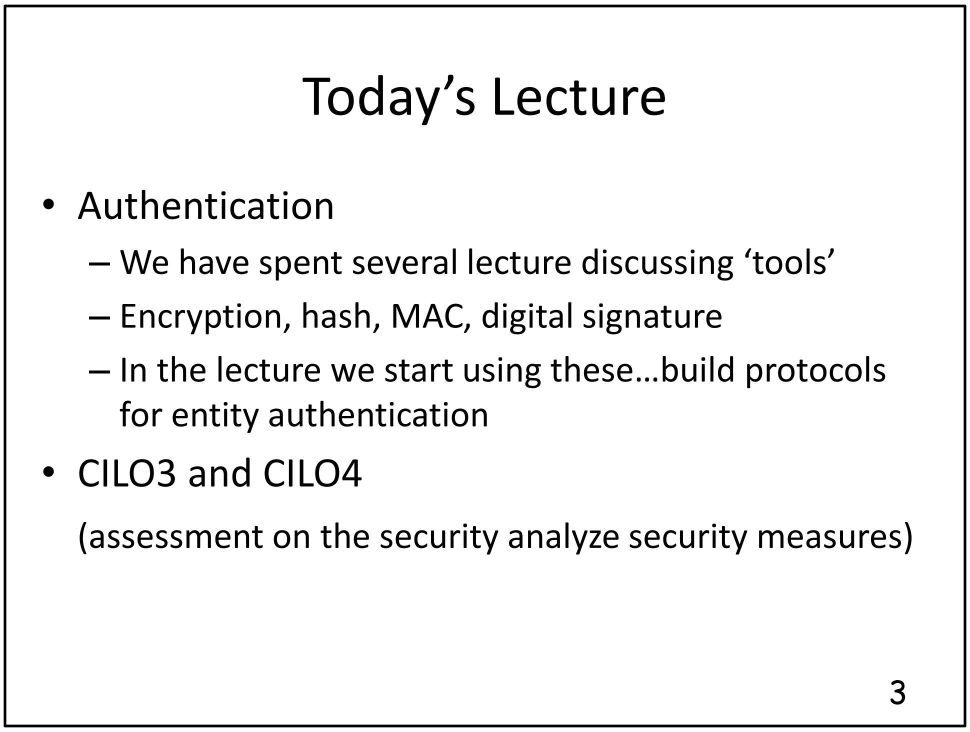
CS Department
City University of Hong Kong

1

Reminder of previous lecture

- Integrity (data origin authentication/non-repudiation)
  - Hash (known algorithm, no key)
    - Only detect accidental modification
    - Finding collisions should not be easy (birthday paradox >> n/2)
  - MAC (needs key)
    - Data origin authentication (only generated of key known)
    - HMAC or CBC-MAC
    - No non-repudiation (two parties can generate MAC)
  - Digital signature (asymmetric crypto system)
    - Sign with private key (only one person)
    - Verify with public key (anyone)
    - Look at RSA version (there are other signature schemes)
    - Only mechanism to provide non-repudiation (also origin auth.)
    - Only one person can generate signature

2



Today’s Lecture

- Authentication
  – We have spent several lecture discussing ‘tools’
  – Encryption, hash, MAC, digital signature
  – In the lecture we start using these…build protocols for entity authentication
- CILO3 and CILO4

  (assessment on the security analyze security measures)

3

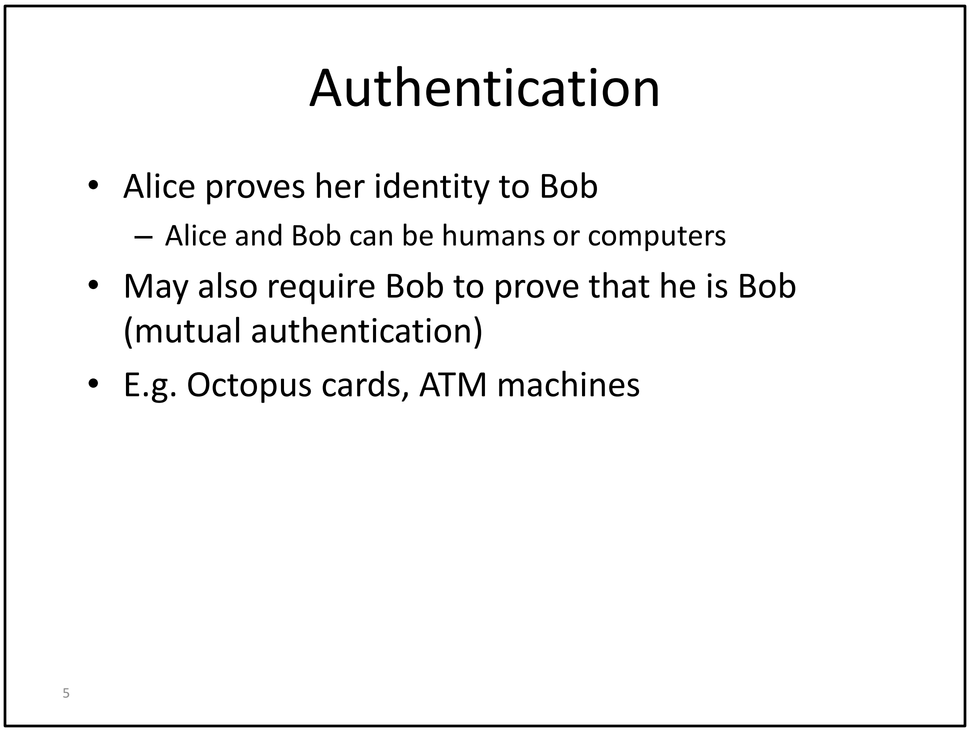
What crypto mechanisms do we have up to now?

Symmetric?
Encryption
MAC

Asymmetric
Encryption
Signature

Entity Authentication

4

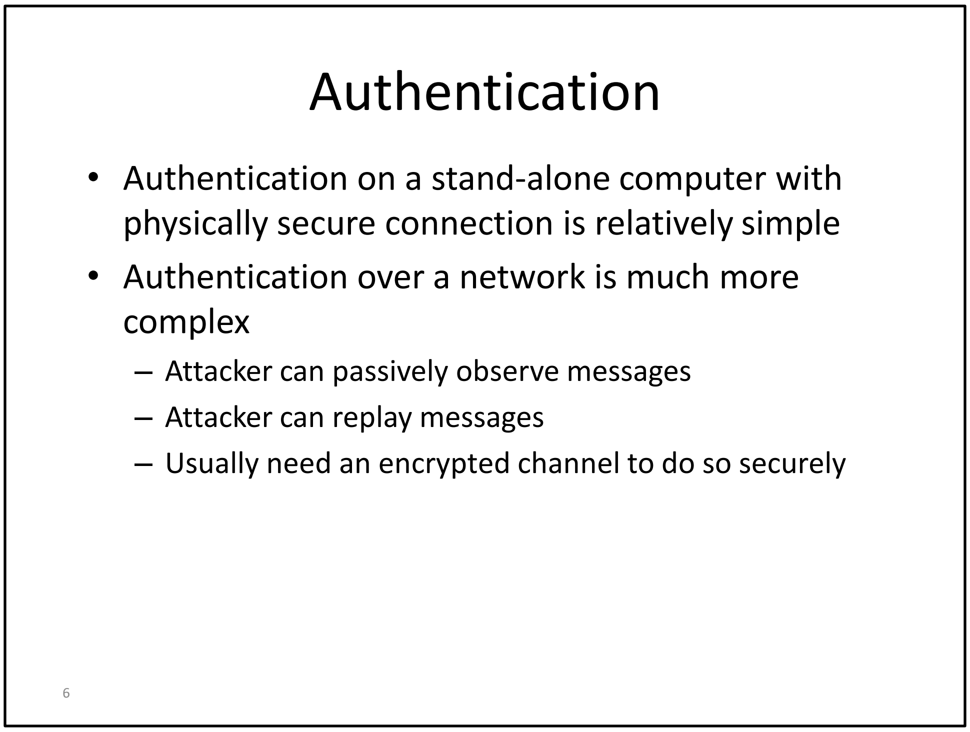


Authentication

- Alice proves her identity to Bob
  - Alice and Bob can be humans or computers
- May also require Bob to prove that he is Bob (mutual authentication)
- E.g. Octopus cards, ATM machines

5

Slides 5 to 8 is just some basic introduction. You only need to know that authentication if you are dealing directly within a physically secure environment is easy while authentication over an open network (like we usually do online needs little more thought).

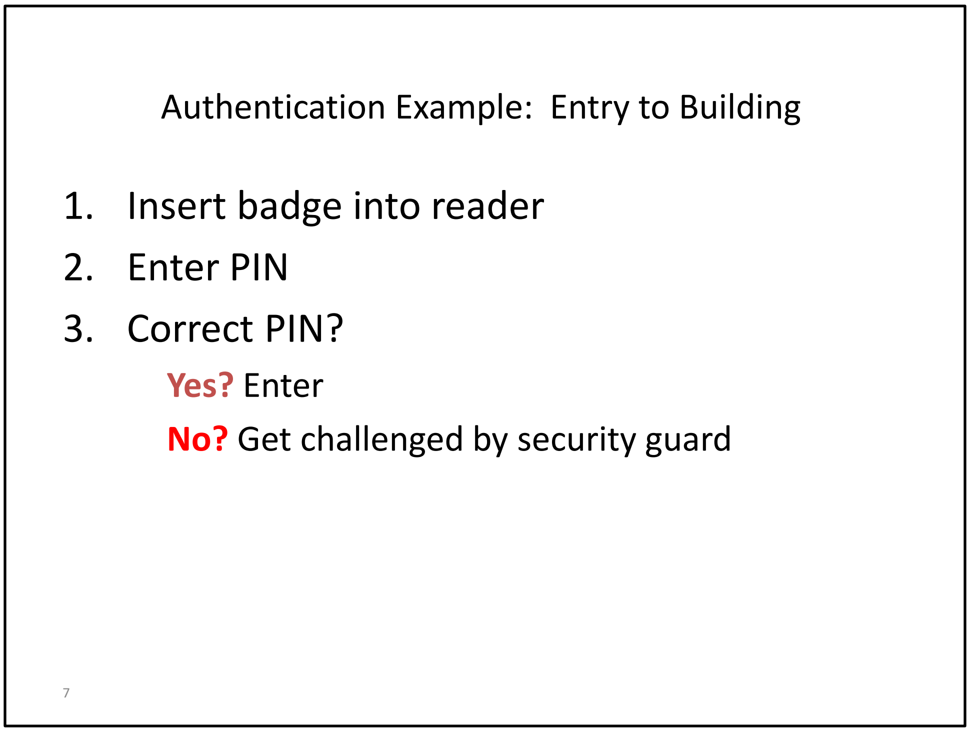


Authentication

- Authentication on a stand-alone computer with physically secure connection is relatively simple
- Authentication over a network is much more complex
  - Attacker can passively observe messages
  - Attacker can replay messages
  - Usually need an encrypted channel to do so securely

6

Why are closed systems easier? There is a trusted entity overseeing the process. For example, ATM/secure entry


Authentication Example: Entry to Building

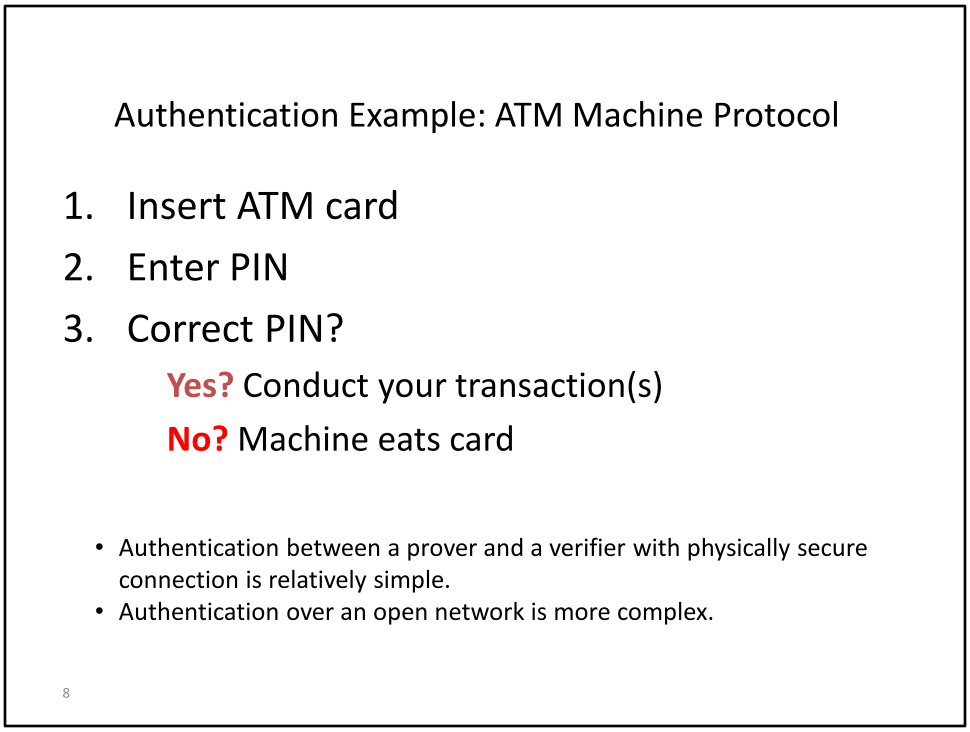
1. Insert badge into reader
2. Enter PIN
3. Correct PIN?

   Yes? Enter

   No? Get challenged by security guard

Why is trusted easier?

See you enter PIN off a piece of paper? Hanging out near gate waiting to see someone’s PIN?



Authentication Example: ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Correct PIN?

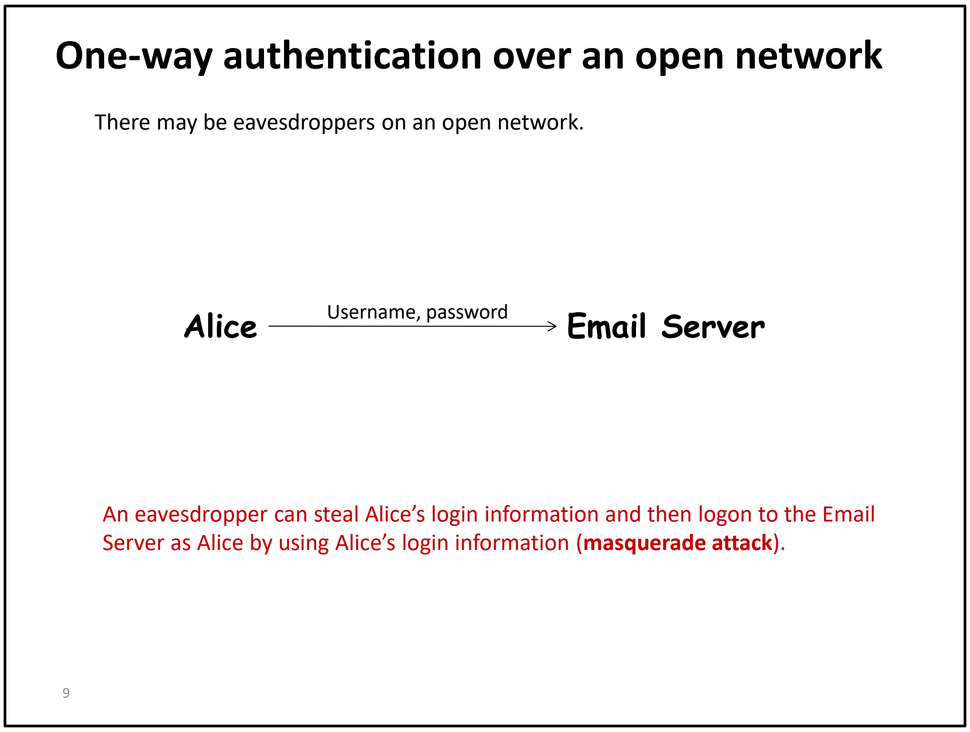
   Yes? Conduct your transaction(s)

   No? Machine eats card

- Authentication between a prover and a verifier with physically secure connection is relatively simple.
- Authentication over an open network is more complex.

8

It is difficult to see that actual communication between card and machine (inside trusted box)



One-way authentication over an open network

There may be eavesdroppers on an open network.

$$\text{Alice} \xrightarrow{\text{Username, password}} \text{Email Server}$$

An eavesdropper can steal Alice's login information and then logon to the Email Server as Alice by using Alice's login information (**masquerade attack**).

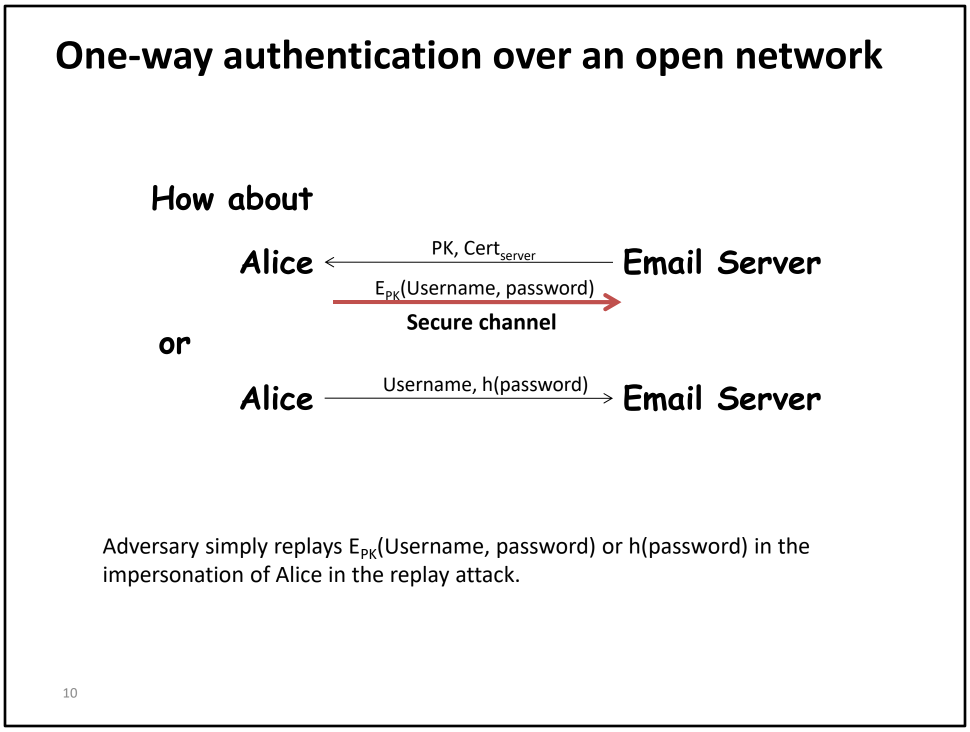
9

What can go wrong here?

Look at slides 9 and 10 and understand replay attack. If we were to use a value (even if encrypted and the attacker does not know the value) in the same way twice this means someone can record it and can pretend to be use later…

Study this is combination with slide 11 and understand why we need some freshness (make each authentication sequence different – easiest way to do so is to make the response depend on a challenge that is different each time).

This is the simplest of relay attacks, since username and password can just be reused.



One-way authentication over an open network

How about

$$\begin{tikzcd}[column sep=huge] \text{Alice} \arrow[r, shift left, red, "E_{PK}(\text{Username, password})", "\text{Secure channel}"'] & \text{Email Server} \arrow[l, shift left, "\text{PK, Cert}_{server}"'] \end{tikzcd}$$

or

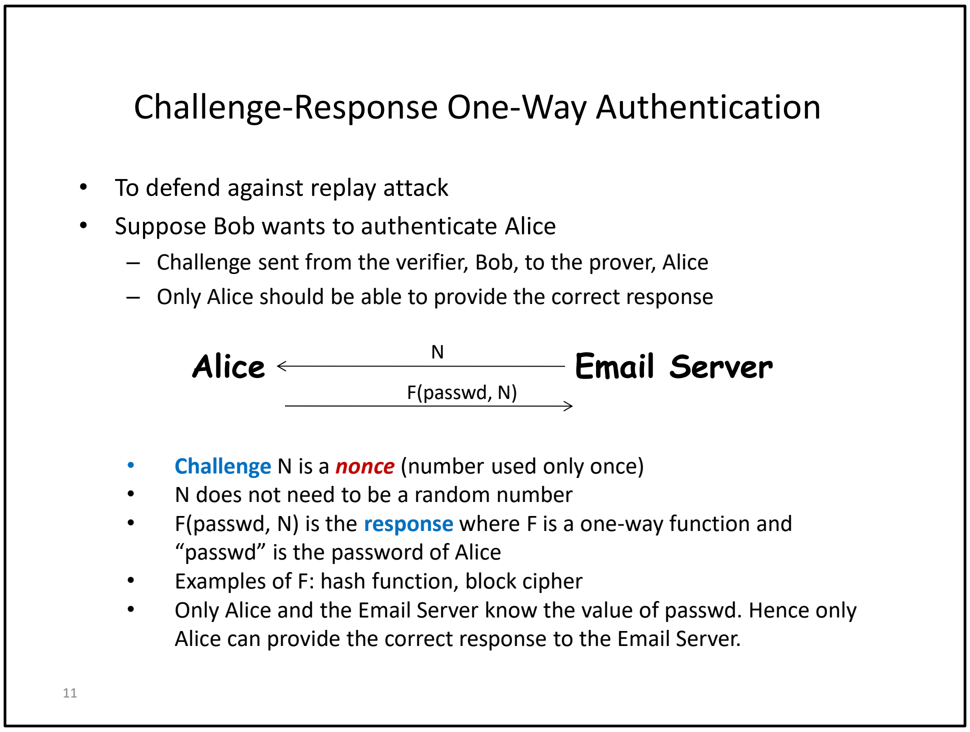
$$\begin{tikzcd}[column sep=huge] \text{Alice} \arrow[r, "\text{Username, h(password)}"] & \text{Email Server} \end{tikzcd}$$

Adversary simply replays E_{PK}(Username, password) or h(password) in the impersonation of Alice in the replay attack.

10

What goes right here? Ok password not leaked…

What can go wrong here? However, can still record the message and pretend to be Alice later. Here we do not know the password value, but the hashed or encrypted version can just as effectively be replayed.



Challenge-Response One-Way Authentication

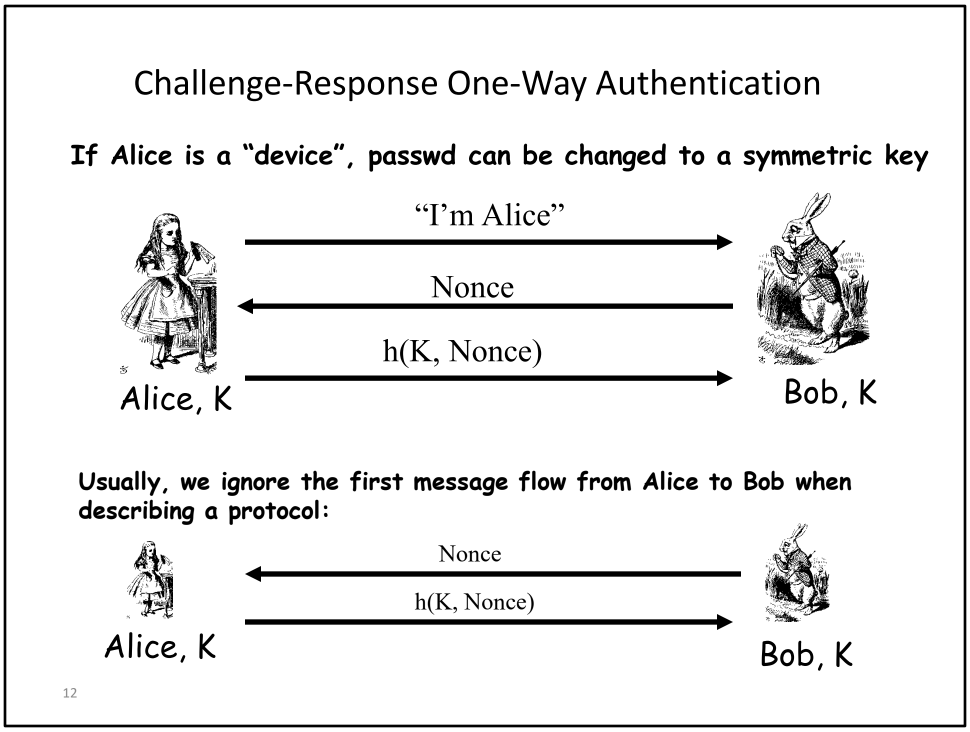
- To defend against replay attack
- Suppose Bob wants to authenticate Alice
  - Challenge sent from the verifier, Bob, to the prover, Alice
  - Only Alice should be able to provide the correct response

$$\begin{tikzcd}
\textbf{Alice} \arrow[r, bend right, "F(passwd, N)"'] & \textbf{Email Server} \arrow[l, "N"']
\end{tikzcd}$$

- **Challenge** N is a ***nonce*** (number used only once)
- N does not need to be a random number
- F(passwd, N) is the **response** where F is a one-way function and "passwd" is the password of Alice
- Examples of F: hash function, block cipher
- Only Alice and the Email Server know the value of passwd. Hence only Alice can provide the correct response to the Email Server.

11

You must realise that entity authentication relies on responding 'now', with a message that has just been generated. Later we will call this 'freshness'



Challenge-Response One-Way Authentication

If Alice is a "device", passwd can be changed to a symmetric key

$$\begin{tikzcd}
\text{Alice, K} \arrow[r, shift left=4, "\text{“I'm Alice”}"] \arrow[r, leftarrow, "\text{Nonce}"] \arrow[r, shift right=4, "\text{h(K, Nonce)}"'] & \text{Bob, K}
\end{tikzcd}$$

Usually, we ignore the first message flow from Alice to Bob when describing a protocol:

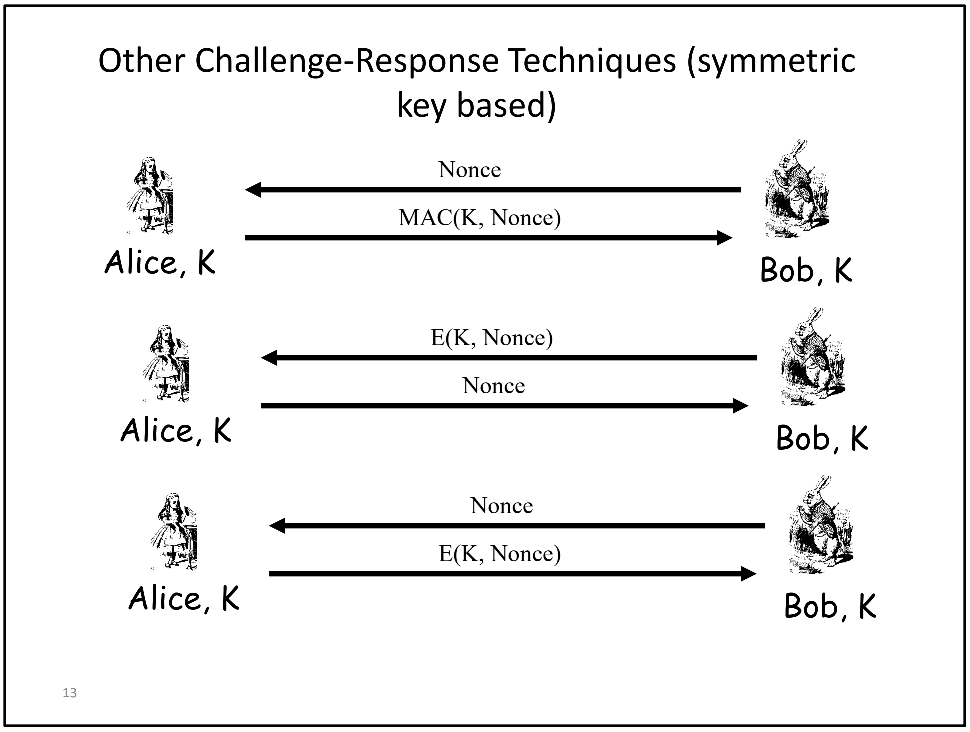
$$\begin{tikzcd}
\text{Alice, K} \arrow[r, shift left=2, leftarrow, "\text{Nonce}"] \arrow[r, shift right=2, "\text{h(K, Nonce)}"'] & \text{Bob, K}
\end{tikzcd}$$

12

Challenge response protocols can look a number of different ways (slide 12-15)

Look at each one and think what is important? How is this achieved?
Must be sure Alice generated the response (so only Alice must be able to generate the response) (formally this is data origin authentication)
Must be sure Alice generated the response in reaction to the challenge we sent *just now*. (formally this is freshness).

Freshness+data origin authentication = entity authentication



Other Challenge-Response Techniques (symmetric key based)

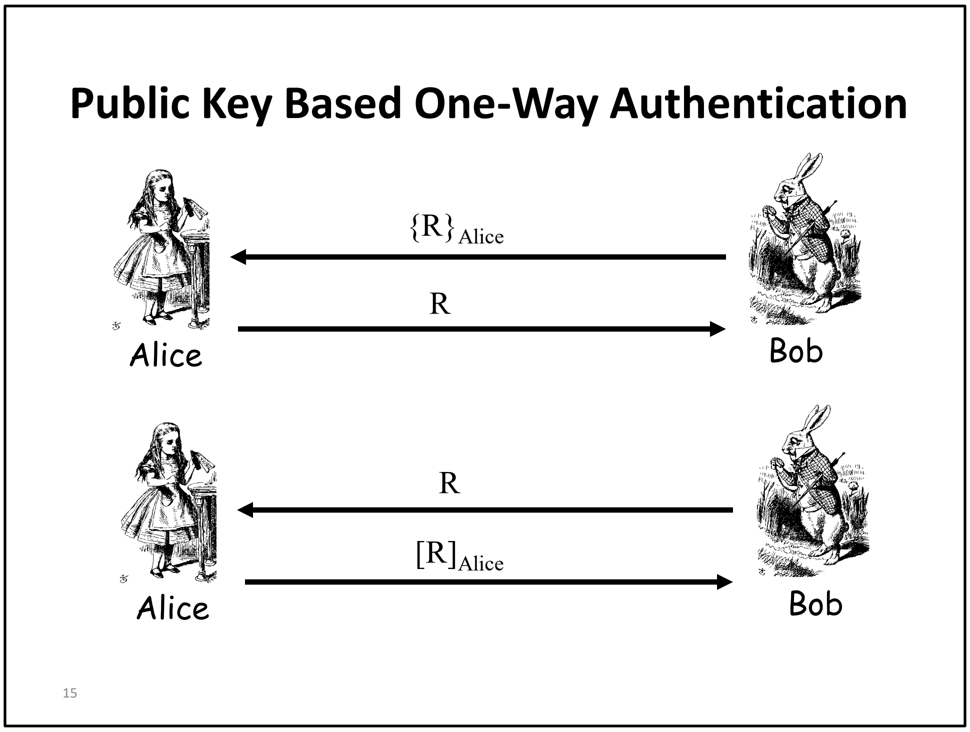
Is there some form of freshness in each protocol? Yes, the nonce.

Are we sure Alice is sending the response?

In all three cases yes as it involves her using the shared key.

Public Key Notations and Assumption

- Encrypt M under Alice's public key: $\{\mathrm{M}\}_{\mathrm{Alice}}$
- Sign M with Alice's private key: $[\mathrm{M}]_{\mathrm{Alice}}$
- All public keys are assumed to be certified (e.g. digital certificates) and become publicly known.

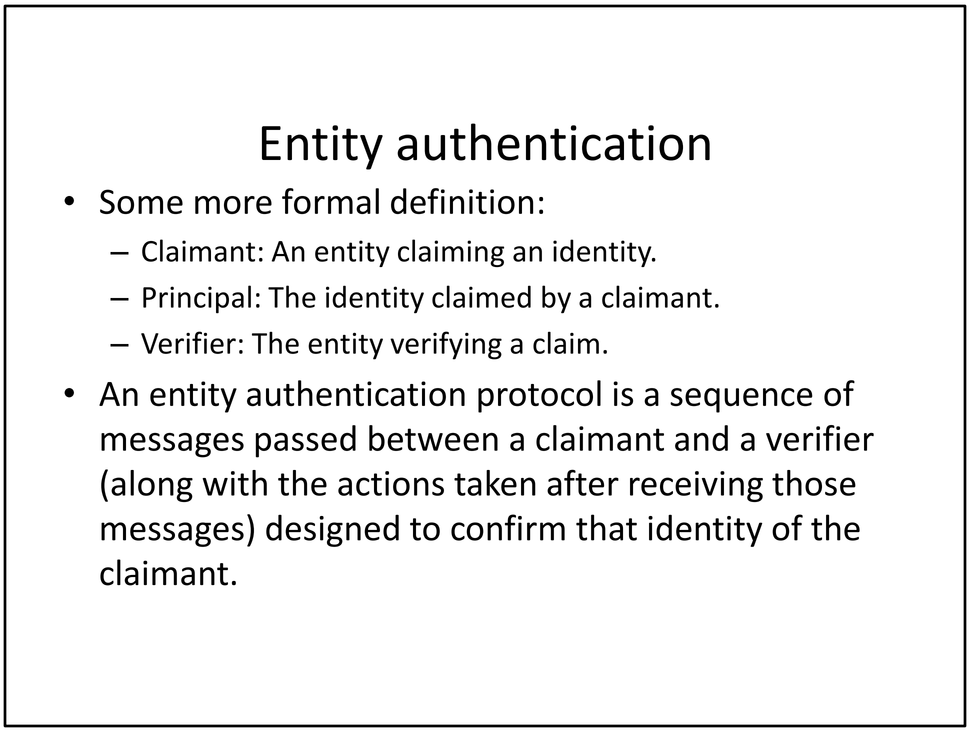


Public Key Based One-Way Authentication

Is there some form of freshness in each protocol? Yes, the nonce.

Are we sure Alice is sending the response?

In both cases yes as it involves her using her private key.



Entity authentication

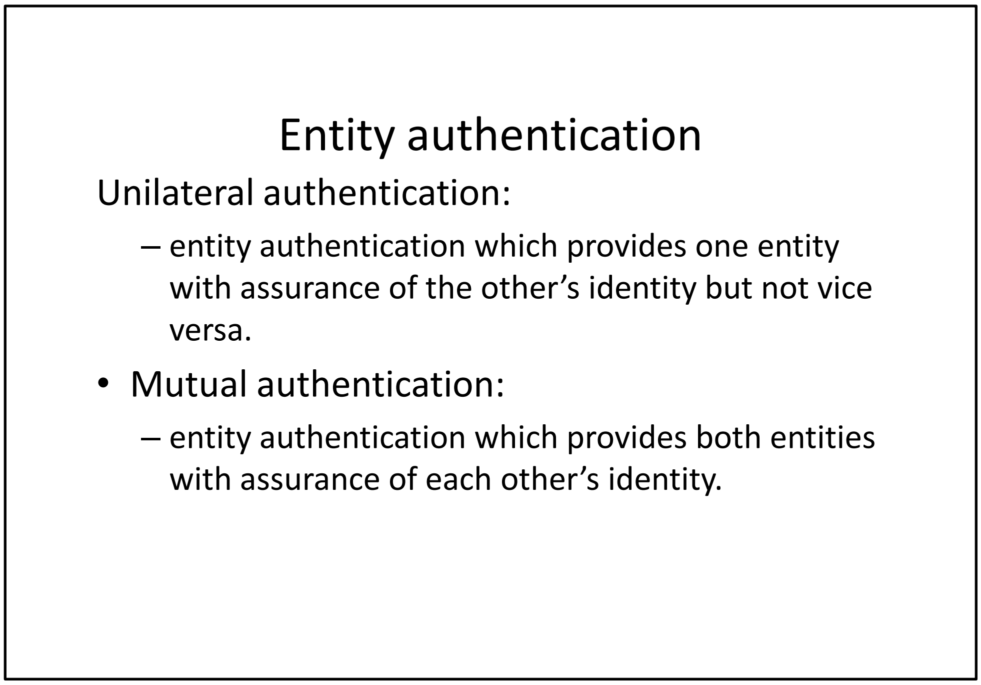
- Some more formal definition:
  - Claimant: An entity claiming an identity.
  - Principal: The identity claimed by a claimant.
  - Verifier: The entity verifying a claim.
- An entity authentication protocol is a sequence of messages passed between a claimant and a verifier (along with the actions taken after receiving those messages) designed to confirm that identity of the claimant.

You need to know the terminology claimant/principal and verifier. The claimant is the person the verifier is talking to, the principal is the logical identity the claimant says he is.

Additional notes for interest:
An entity authentication protocol is a sequence of “messages” passed between a claimant and a verifier (as well as the actions taken after the receiving these messages) that is designed to confirm that the claimant is the principal.

Our inspiration for the material in this section is the ISO/IEC 9798 standard. This is a six-part standard which gives general abstract entity authentication protocols based on a number of different technologies (e.g. symmetric cryptography, asymmetric cryptography, and manual data transfer).

These protocols are not “ready to use”; they have to be adapted for use in a particular context and so we study them as the basis for a large number of other, sector-specific entity authentication standards.



Entity authentication

Unilateral authentication:

- entity authentication which provides one entity with assurance of the other’s identity but not vice versa.

- Mutual authentication:
  - entity authentication which provides both entities with assurance of each other’s identity.

You should know this terminology

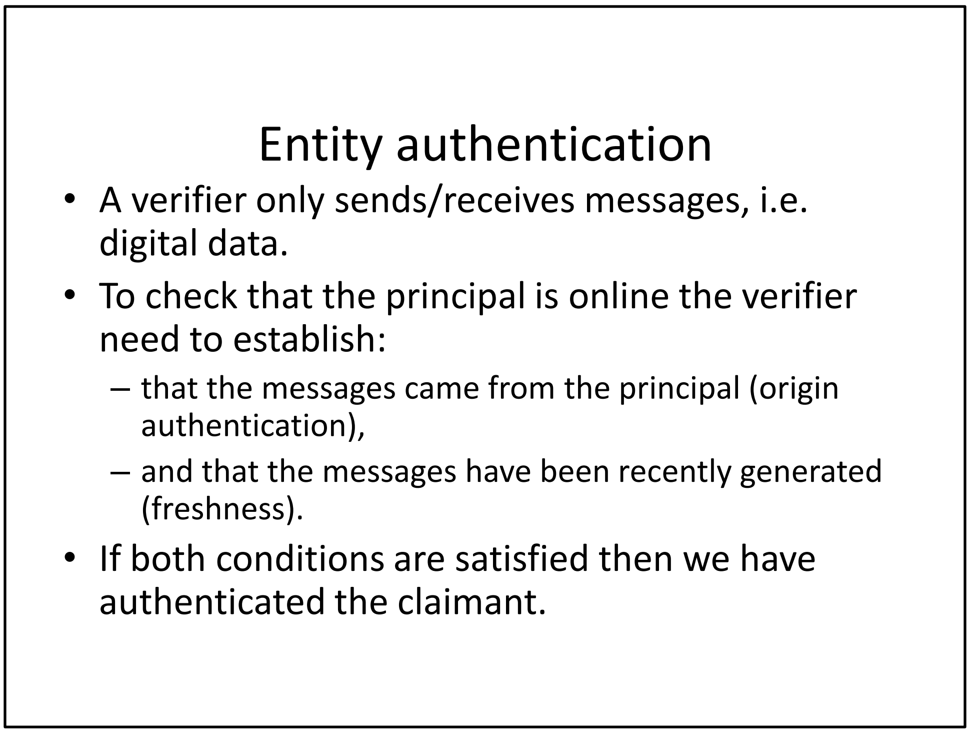
Additional notes for interest:
ISO 7498-2 defines entity authentication as ‘the corroboration that an entity is the one claimed’.

We can also distinguish between protocols providing *unilateral authentication* and those providing *mutual authentication*.

Unilateral authentication is defined as ‘entity authentication which provides one entity with assurance of the other’s identity but not vice versa’ and mutual authentication is defined as ‘entity authentication which provides both entities with assurance of each other’s identity’.

For mutual authentication, which entity is the verifier and which entity is the claimant?

They swop around – both entities are effectively both during the protocol execution.



Entity authentication

- A verifier only sends/receives messages, i.e. digital data.
- To check that the principal is online the verifier need to establish:
  - that the messages came from the principal (origin authentication),
  - and that the messages have been recently generated (freshness).
- If both conditions are satisfied then we have authenticated the claimant.

This slide again points out the requirements for entity authentication (we are sure message come from principal, we are sure message fresh)
This is very important – you need to always know these two requirements.

Quick revision: What mechanisms give us origin authentication? Symmetric MAC, Asymmetric Digital signature.

Additional notes for interest:

Typically, in a computer system, the verifier never meets the claimant. The only way that the verifier can gain any information about the claimant is through the messages (digital data) that they exchange.

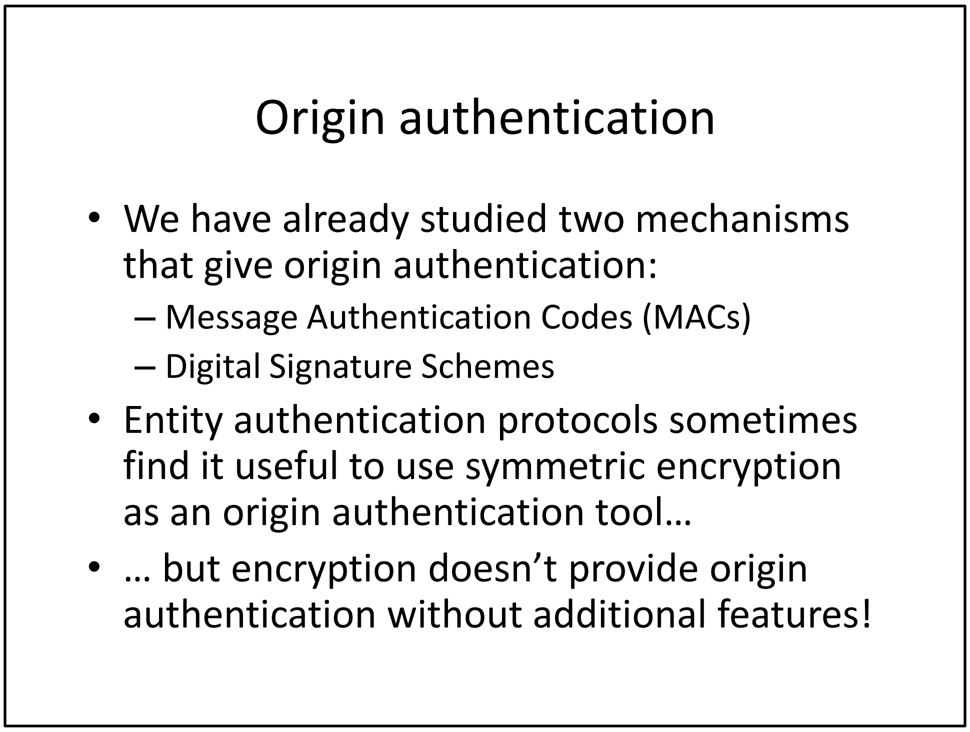
Therefore, the digital data has to convince the verifier that it was generated by the principal in response to the verifier's request.

There are two things we need to establish – the first is whether this message did come from the principal. This is standard origin authentication.

Is knowing that an entity created a message enough to prove the a principal is currently communicating with us? If Alice received a message with origin authentication proving that this message was put together by Bob. Is that enough to assume Bob is currently talking to Alice? No, Replay attack

So what do we need?
We also need to establish that this message was generated now…this is referred to as freshness.

So our characterisation of entity authentication is that there is both freshness and origin authentication.



Origin authentication

- We have already studied two mechanisms that give origin authentication:
  – Message Authentication Codes (MACs)
  – Digital Signature Schemes
- Entity authentication protocols sometimes find it useful to use symmetric encryption as an origin authentication tool…
- … but encryption doesn't provide origin authentication without additional features!

MAC and Digital signatures provide assurance the a specific person generated a message.

You must also know that when we are dealing with authentication – that encryption can give us data origin authentication (even they in general encryption does not provide any integrity). The reason is that in these protocols the verifier knows the expected content and format of the message (not the case for data receiver) – and if the plaintext does not make sense (as someone tried to modify ciphertext) in reject authentication.

Also look at slide 20 (it could be made better by introducing a dedicated MDC) – this is any measure that would indicate that something wrong with plaintext. Ideally a hash would work the best.

Additional notes for interest:

We have already studied two mechanisms that can be used to provide origin authentication:
• MACs. A MAC can provide origin authentication because only people who hold the correct secret key can compute the MAC. Hence, on receiving a correct MAC, you know that the message must have come from an entity that knows the secret key. If

you know that you didn't compute the MAC and only one other person has the key, then you must conclude that the message comes from that other key holder.

- Digital Signature Schemes. Only the holder of the private key can compute a digital signature, hence if you receive a correct digital signature, then you must conclude that it was sent by the holder of the private key.

However, for reasons we will discuss later, it is sometimes convenient to use symmetric encryption as an origin authentication tool. This is against everything we talked about in the Security Algorithms section, where we demonstrated that most encryption schemes do not provide <u>any</u> data integrity or origin authentication tools.

Why does it not give us data integrity? You can fiddle with the ciphertext – I decrypt it and you essentially fiddled with the plaintext. I do not know what was sent. No origin authentication? Because origin authentication goes hand and hand with integrity – no integrity no origin authentication.

Origin authentication

- Encryption checks the integrity of a message by checking that it "makes sense".
- It is hard for a computer to check whether a message makes sense or not.
- Append a *manipulation detection code* (MDC) to the message before encryption.
- A message "makes sense" if the MDC is correct for the decrypted message.
- What is MDC? Have we dealt with one before?
  - Could also be known or expected data in special case
  - A protocol could be design the receiver knows value

Additional notes for interest to understand this concept:

Therefore, if we are to use encryption as an origin authentication mechanism, then we must make sure that it is impossible for an attacker to create a new valid encrypted message or to alter an existing encrypted message.

We do this by checking whether the decrypted message "makes sense" or not. However, if we are to do this in the obvious way, then we have to make some assumptions about the messages that are being sent, for example that they are written in English using ASCII characters. This is rarely practical!

Instead, we use the notion of a *manipulation detection code* or MDC.

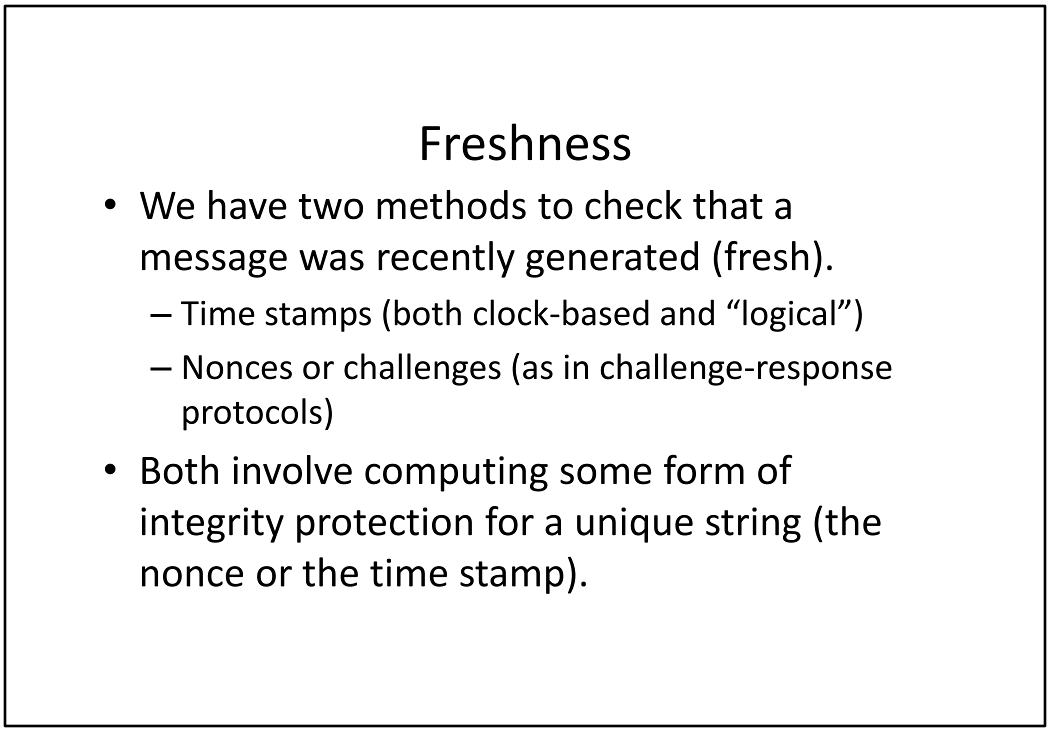
An MDC is block of data appended to the end of a message before it is encrypted. Typically, it is the hash of the message, although in some circumstances it may be a MAC of the message computed using a separate key.

Upon receiving a message, an entity first decrypts it, and then checks that it "makes sense" by checking that the MDC code appended to the end of the message is correct for that message.

It is hoped that any entity who wishes to change the encrypted message will not know how to change the MDC, and so any changes will be detectable.

Note that if we're using an unkeyed MDC, like a hash function or MDC that anyone does know how to change, then it is best practice opinion that this assumption can only hold if we use CBC mode encryption.

The alternative to using an MDC is to use a specific mode of operation that is designed to give origin authentication – i.e. an authenticated encryption scheme. These are given in ISO/IEC 19772 and were briefly discussed in the Security Algorithms section of the standard.



Freshness

- We have two methods to check that a message was recently generated (fresh).
  - Time stamps (both clock-based and "logical")
  - Nonces or challenges (as in challenge-response protocols)
- Both involve computing some form of integrity protection for a unique string (the nonce or the time stamp).

There are two major methods for checking that a message is fresh (i.e. that it was recently generated). These are by the use of nonces and the use of time-stamps.

Additional notes for interest:

Why must freshness data be integrity protected?

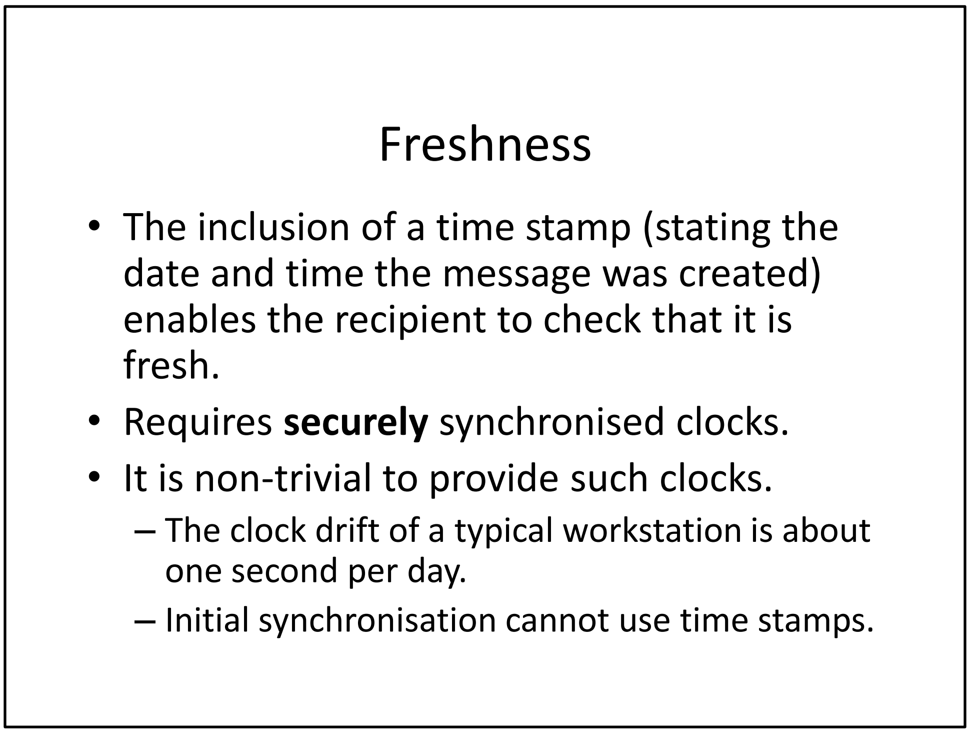
Otherwise this whole exercise is useless – we need to have assurance as to who created this freshness (tie the entity to this freshness).

Otherwise we just send a nonce, and the nonce comes back…who sent the nonce back? Anyone…

We get a message with a time stamp…who can create a time stamp? Anyone…

This does not give us any indication as to freshness of a response from a specific entity.

What if we get (timestamp) + MAC, or signature of a time stamp – do we have some assurance as to the entity and freshness of the response?



Freshness

- The inclusion of a time stamp (stating the date and time the message was created) enables the recipient to check that it is fresh.
- Requires **securely** synchronised clocks.
- It is non-trivial to provide such clocks.
  - The clock drift of a typical workstation is about one second per day.
  - Initial synchronisation cannot use time stamps.

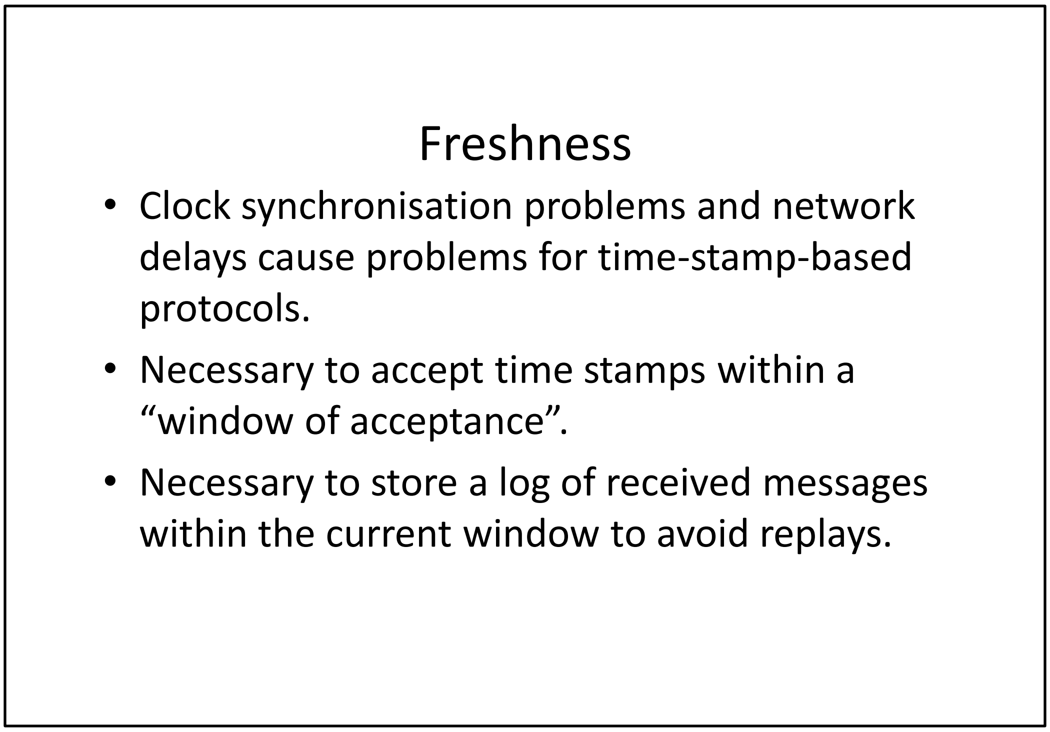
Know the advantages and disadvantages of time stamps and also study slide 23 on the need for a ‘window of acceptance’.

Clearly, the inclusion of a time stamp (a bit string representing the time and date of the creation of the message) enables the recipient to check when the message was created, i.e. that it is fresh.

However, it does have a significant disadvantage in that both the sender and the receiver must have securely synchronised clocks.

Providing such clocks is not necessarily easy, especially when you consider that a typical workstation might lose or gain up to one second per day. Hence, we must find a way to update user’s clocks securely.

However, this is not easy. To securely update a user’s clock (or initially synchronise user’s clock) would requires some form of entity authentication to make sure that the update comes from the appropriate source, and this update cannot make use of time stamps. There are two solutions to this problem:
• The update protocol should make use of nonces for freshness.
• The update protocol should make use of some reliable third party source of accurate time (such as national radio broadcast time).



Freshness

- Clock synchronisation problems and network delays cause problems for time-stamp-based protocols.
- Necessary to accept time stamps within a "window of acceptance".
- Necessary to store a log of received messages within the current window to avoid replays.

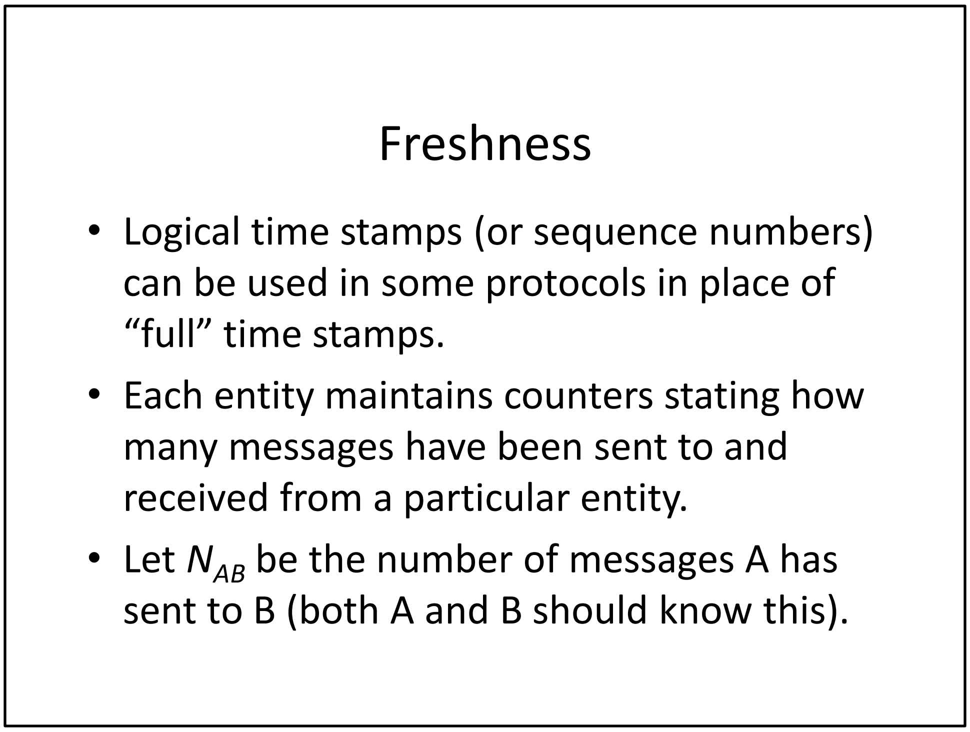
Given that it is difficult to create two completely synchronised clocks, and that it takes a certain amount of time for a message to cross a network, it is very unlike that a message will arrive at precisely the time stated by the time stamp.

Therefore, it is necessary to define a "window of acceptance" on either side of the receiver's current clock value. If they receive a message whose time stamp declares that it was created within the window of acceptance, then the receiver will accept that this message is fresh.

How can this be a problem?

Lets say my windows is 10 seconds, and I receive a message 1s after the stated timestamp? Is this valid? Lets say the attacker records this message and replays is 4 second later – is this still a valid message?

So this could allow an attacker to replay a message to a receiver any number of times providing that its time stamp lies within the window of acceptance. This problem can be solved by keeping a record of all the messages that are received by the receiver and which are still fresh (i.e. that there time stamps are within the window of acceptance), and refusing to accept any message more than once.

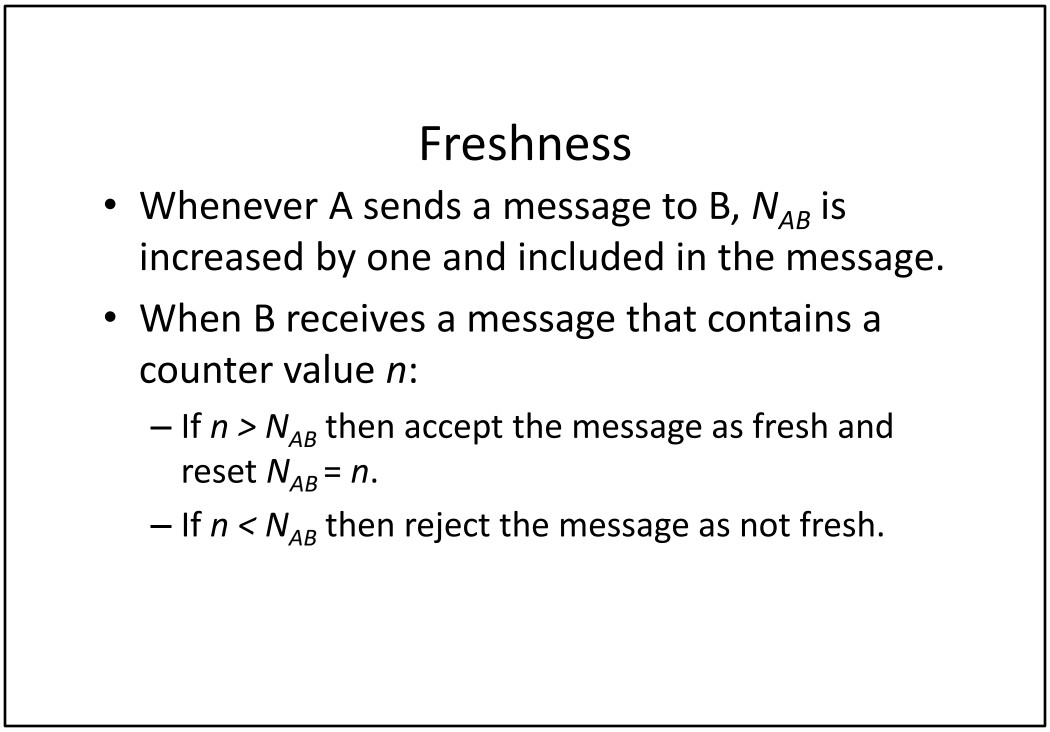


Freshness

- Logical time stamps (or sequence numbers) can be used in some protocols in place of “full” time stamps.
- Each entity maintains counters stating how many messages have been sent to and received from a particular entity.
- Let N_{AB} be the number of messages A has sent to B (both A and B should know this).

You should study the concept of logical time stamps (slide 24-25). These are simply sequence numbers, rather than time. The same concept applies – except an old message is not old in time, but contains a number that is smaller than the current counter value.

So in this case both parties maintain two counters. A maintains a counter for messages sent to B, and A maintains a counter for messages received from B.



Freshness

- Whenever A sends a message to B, N_{AB} is increased by one and included in the message.
- When B receives a message that contains a counter value n:
  - If $n > N_{AB}$ then accept the message as fresh and reset $N_{AB} = n$.
  - If $n < N_{AB}$ then reject the message as not fresh.

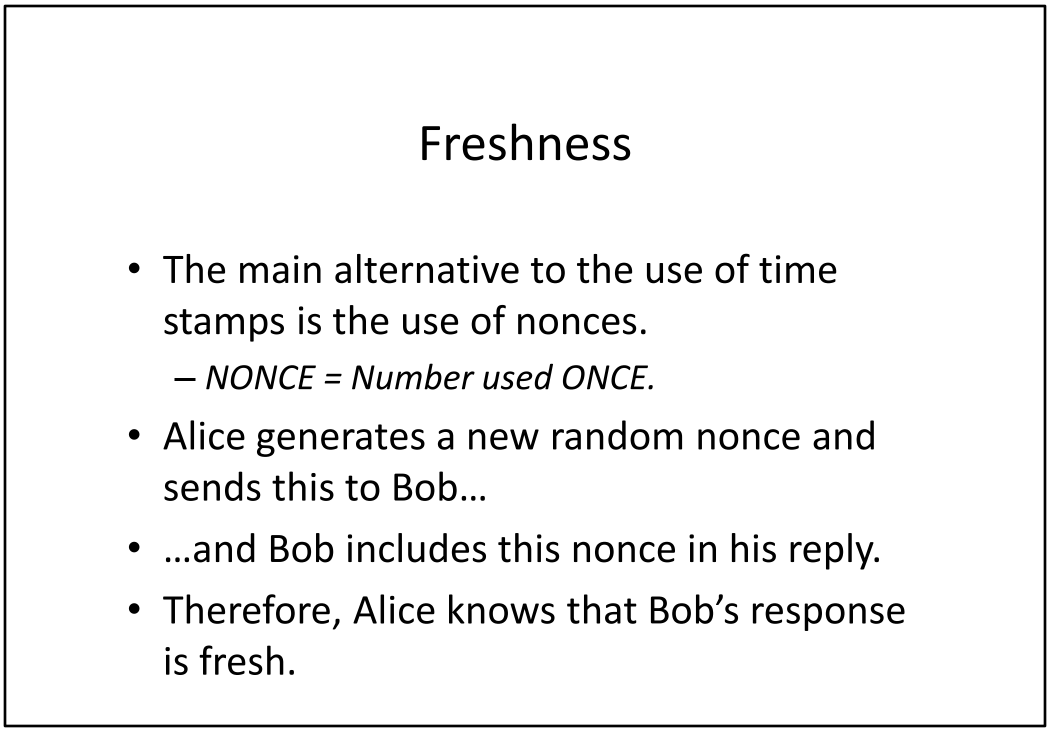
Every time A sends a message it sends the current value of the counter and then increments the counter. B check the value with his locally stored counter value for messages received from A. If the message value is greater than the counter then fresh, and we set counter to this value.

Why? We might lose a message, the message value is greater than counter – but it is not right just to increment counter by one.

On the other hand if the message value is smaller or equal to counter it is old.

Logical time stamps have to be used with care, because they are very vulnerable to “preplay” attacks. However, if used carefully, then they can be very effective. Logical time stamps are used in practice, for example in 3GPP mobile phone authentication protocols.

What are preplay attacks? I can predict the next value – it is a counter, so I send a message with that value. Remember - data integrity/origin authentication on sources of freshness is important.



Freshness

- The main alternative to the use of time stamps is the use of nonces.
  - *NONCE = Number used ONCE.*
- Alice generates a new random nonce and sends this to Bob…
- …and Bob includes this nonce in his reply.
- Therefore, Alice knows that Bob's response is fresh.

This is essentially more information on challenge-response approach we already talked about at start of lecture (you must know how nonces ensure freshness)

Nonce-based protocols work in quite a different way. They show that a message is fresh by including something in that message that could not have been known a long time ago.

In this case, the nonce is a random data value (that has never been used as a nonce before) that has been provided by Alice.

Nonce-based protocols are also known as challenge-response protocols, as the inclusion of a nonce (provided during the challenge) in the response indicates that the response was generated after the challenge.

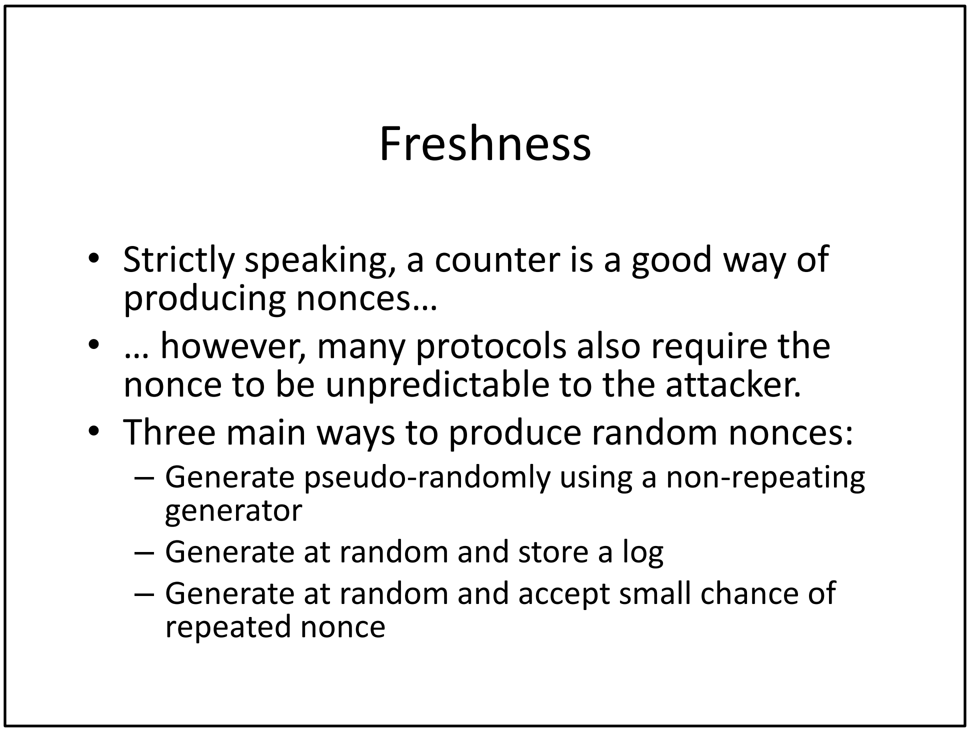
Whose responsibility is it to ensure that a nonce isn't used twice?
The nonce provider must ensure that a nonce is not used twice.

In general, to whom does a nonce provide freshness protection?
The nonce provider.

Suppose Alice generates a nonce and sends it to Bob; Bob returns the nonce to Alice; and Alice send a last message back to Bob. Can Bob conclude that Alice's last message is fresh?

What if Alice’s message contains the nonce? No. Alice (or someone claiming to be her controls the nonce).
What if along with Alice’s nonce Bob sends a nonce, and that nonce is included in the message? Yes.

If both Alice and Bob require freshness guarantees (say, for mutual authentication), then both Alice and Bob will need to use nonces.



Freshness

- Strictly speaking, a counter is a good way of producing nonces…
- … however, many protocols also require the nonce to be unpredictable to the attacker.
- Three main ways to produce random nonces:
  - Generate pseudo-randomly using a non-repeating generator
  - Generate at random and store a log
  - Generate at random and accept small chance of repeated nonce

If a nonce is ever reused, then it may be possible to replay old messages and have them accepted as fresh. Since nonces only provide freshness guarantees to the person who generated them, it is up to that person to make sure that the nonce they generate has not been used before. This can be done either by using a random bit generator with a long output (and accepting that there is a very small chance that a nonce will be repeated) or by using some more technical means.

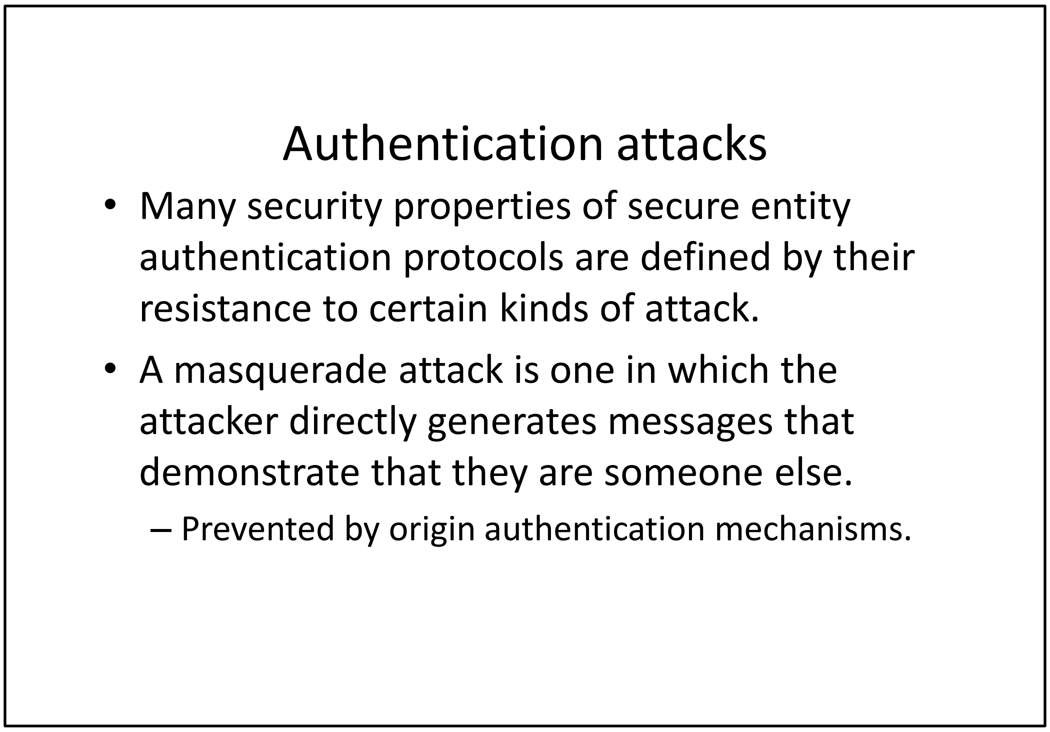
Strictly speaking a counter is a good way to produce a nonce, if the counter is long enough it will never repeat. This is the main requirement. However, some protocols require the a nonce is unpredictable. So this the more general requirements is that a nonce does not repeat, and it is not predictable.

What are the main advantages and disadvantages of pseudo-random generation?
Psuedo-random generators easier to implement that true random number generators,
Could be subject to cryptanalysis, still need a good quality seed.

What are the main advantages and disadvantages of generate-and-log?
Good random source, lots of overhead – how may nonces are we going to store?

What are the main advantages and disadvantages of generate-and-accept?
Good random source (no storage), there is the chance that a nonce repeats – this can

be reduced by making nonce sufficiently long.

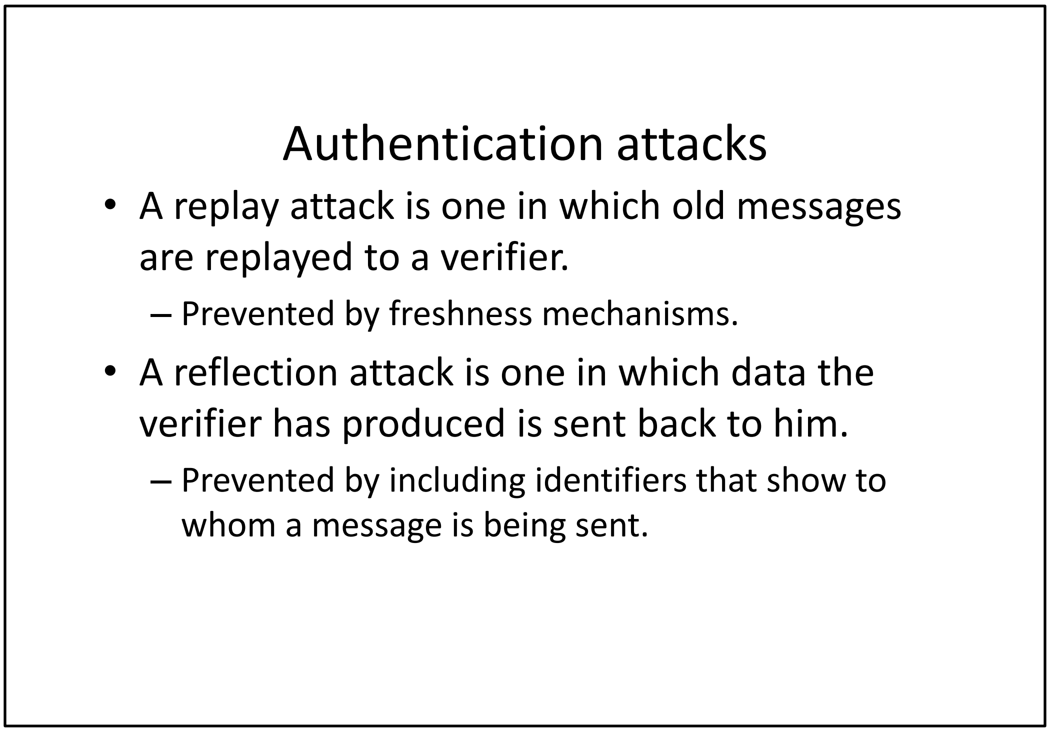


Authentication attacks

- Many security properties of secure entity authentication protocols are defined by their resistance to certain kinds of attack.
- A masquerade attack is one in which the attacker directly generates messages that demonstrate that they are someone else.
  - Prevented by origin authentication mechanisms.

So if you read academic papers on protocols it is common to have a basic security analysis section – this is most of a time a list of attacks applicable to this protocol application and then whether this protocol is resistant to this attack or not.

You must be able to recognise the basic three attacks in a given protocol
Masquerade
Replay
Reflection



Authentication attacks

- A replay attack is one in which old messages are replayed to a verifier.
  - Prevented by freshness mechanisms.
- A reflection attack is one in which data the verifier has produced is sent back to him.
  - Prevented by including identifiers that show to whom a message is being sent.

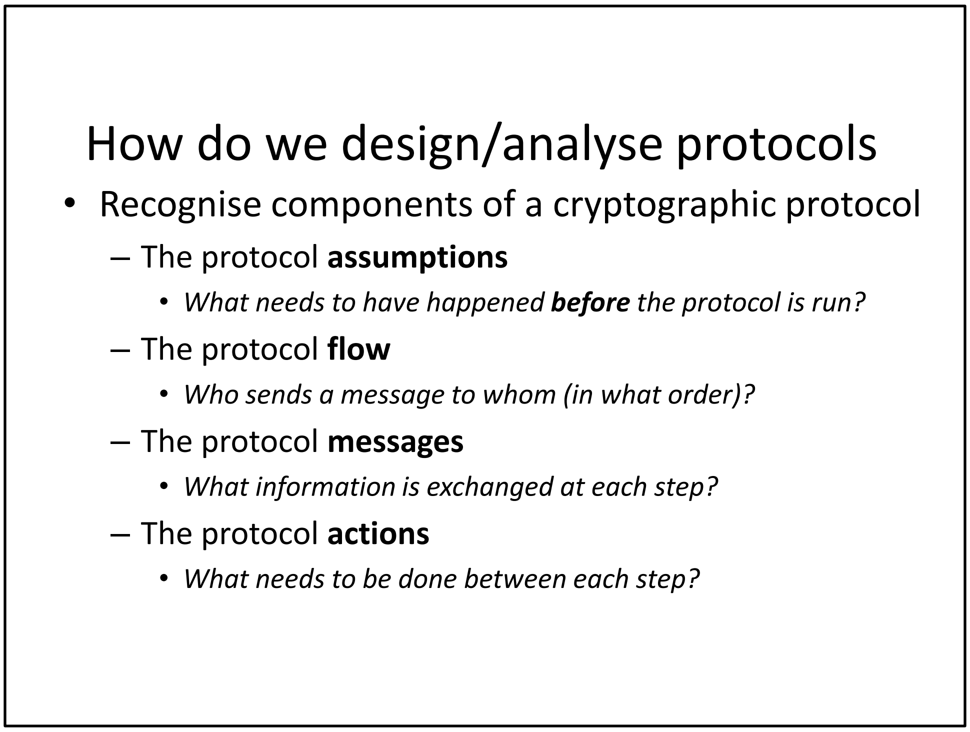
A replay attack – read the slide. So an attacker records a message and sends it at a later stage.

It is sometimes difficult to grasp the concept of a reflection attack(see example on slide 39). It's is best served with an example. Suppose we had an authentication scheme that used a MAC and a shared symmetric key. The verifier sends a random nonce and the claimant responds with the MAC of the nonce under the shared secret key.

We can attack this systems as follows. Suppose there is a session in which the attacker is claiming to be the principal.
- The attacker receives a random nonce from the verifier.
- The attacker starts a new session with the verifier and immediately challenges the verifier to prove his identity.
- The attacker sends the verifier the nonce he receive in the first step.
- The verifier, eager to prove his identity, responds with the MAC of the nonce computed using the shared secret key.
- The attacker closes down this session.
- The attacker sends the MAC he has just received back to the verifier as his authentication information for the first session.

How do we solve this?

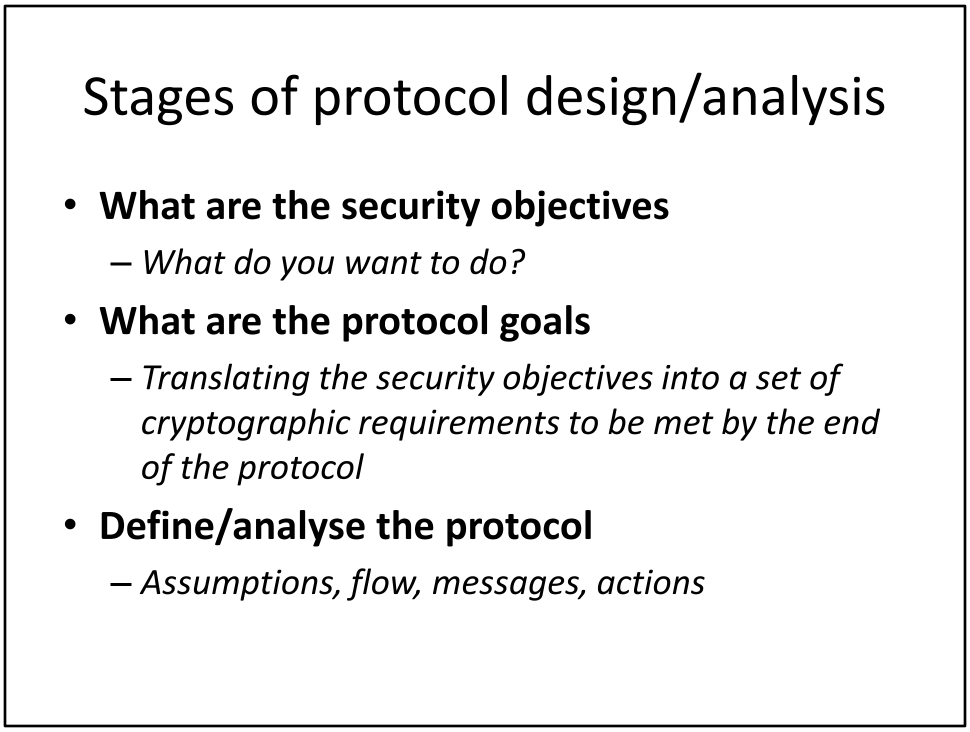
There are lots of ways to prevent this type of attack, but one of the simplest is to include in the message an identifier which states who is being identified by whom. The above attack won't work if the MAC is computed on the nonce and the name of the person who is being verified. The MAC value that the attacker receives in step 4 will not be the same MAC value that the attacker has to send in step 7.



How do we design/analyse protocols

- Recognise components of a cryptographic protocol
  - The protocol **assumptions**
    - *What needs to have happened* ***before*** *the protocol is run?*
  - The protocol **flow**
    - *Who sends a message to whom (in what order)?*
  - The protocol **messages**
    - *What information is exchanged at each step?*
  - The protocol **actions**
    - *What needs to be done between each step?*

You do not have to memorise this slide like ‘Name four components of a protocol’, but if I give you a protocol and ask what are the assumptions? You must be able to tell me the things that are assumed to happen, or the actions that take place that are not explicitly shown.



Stages of protocol design/analysis

- **What are the security objectives**
  - *What do you want to do?*
- **What are the protocol goals**
  - *Translating the security objectives into a set of cryptographic requirements to be met by the end of the protocol*
- **Define/analyse the protocol**
  - *Assumptions, flow, messages, actions*

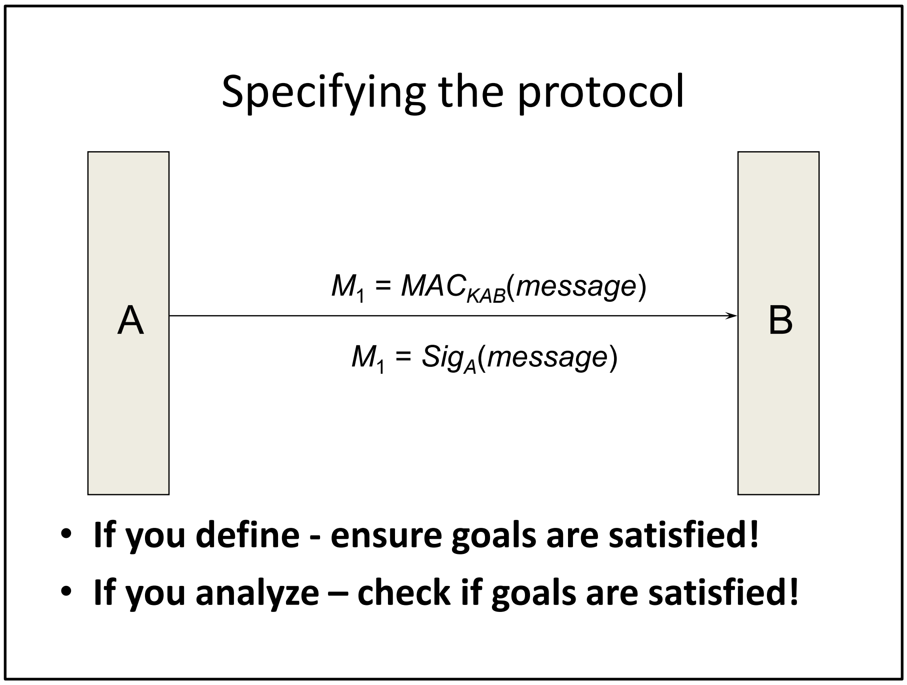
Note the difference between objectives and goals

The one is your real world problem.

The second is your technical requirement.

Very simple example

- **Defining the security objectives**
  - *Bob wants to make sure that Alice was the source of a electronic purchase contract*
  - *Bob wants to the contract to be enforceable at later date (Alice should not deny it)*
- **Determining the protocol goals**
  - *Bob requires data origin authentication of the message received from Alice*
  - *Bob requires non-repudiation of the message received from Alice*



Specifying the protocol

A $\xrightarrow[M_1 = Sig_A(message)]{M_1 = MAC_{KAB}(message)}$ B

- **If you define - ensure goals are satisfied!**
- **If you analyze – check if goals are satisfied!**

First the MAC

Assumptions? A and B share KAB

Actions? B verify MAC

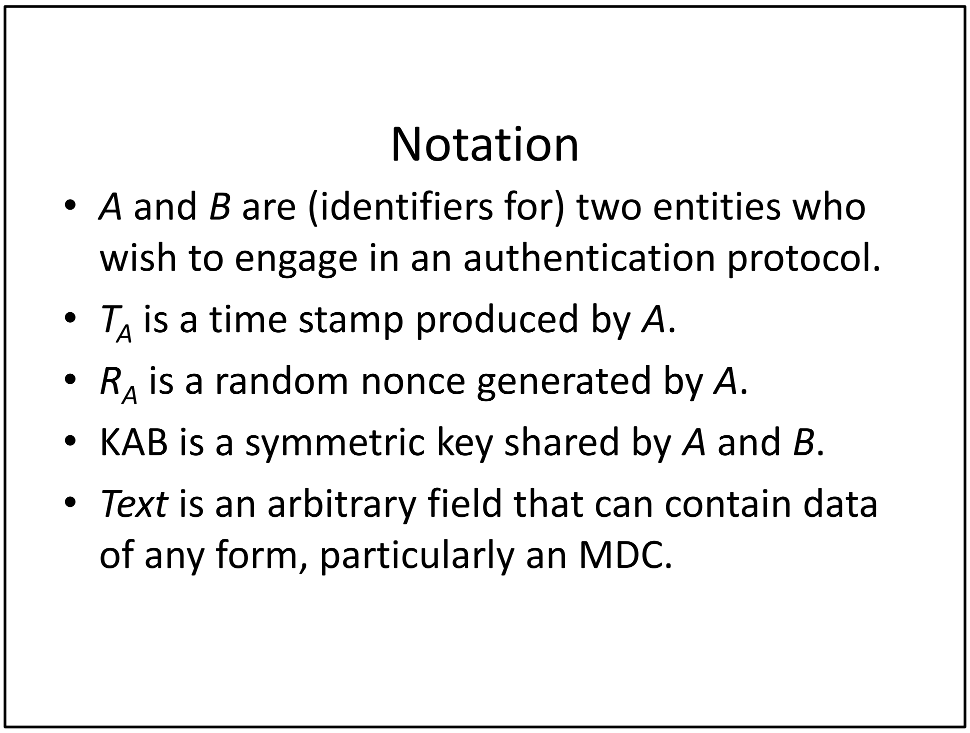
Goals satisfied (see slide 32)? No, we have data origin authentication but not non-repudiation

Signature

Assumptions? B has A public key

Actions? B verify signature

Goals satisfied? Yes, both data origin authentication and non-repudiation



Notation

- *A* and *B* are (identifiers for) two entities who wish to engage in an authentication protocol.
- T_A is a time stamp produced by *A*.
- R_A is a random nonce generated by *A*.
- KAB is a symmetric key shared by *A* and *B*.
- *Text* is an arbitrary field that can contain data of any form, particularly an MDC.

All examples are real world protocols from the ISO/IEC 9798 authentication standard.

Please work through the example protocols and in each case decide:
Mutual or unilateral
Freshness? Why?
Data origin authentication? Why?

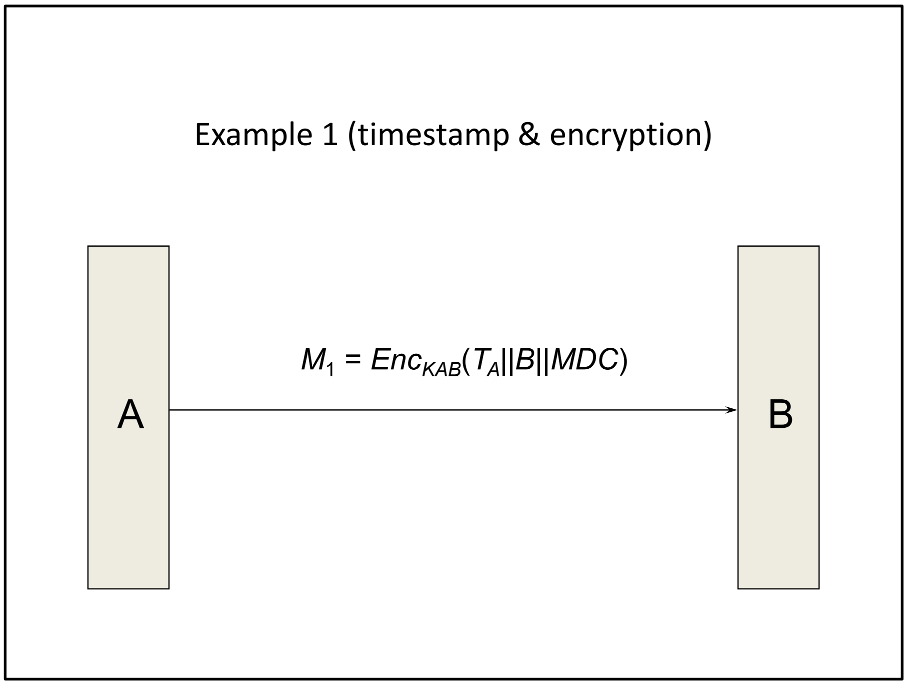
You should not be memorising protocols – you should understand how they work. Practice checking for freshness and data origin authentication. You might see a different protocol in the exam or mid-term than here. If given a protocol to analyse you must be able to answer the questions above: who is authenticated to who? Why – show that freshness and data origin authentication properties has been met.

Do not complicate things – if I give you a protocol and ask you what is wrong with it, it will be masquerade/replay or reflection.

These are all real protocols! I give you references to the standard they are specified in (no need to remember this details)

Notation

- $Enc_{KAB}(X)$ denotes the encryption of data *X* using a key *KAB* that is shared between *A* and *B*. We assume this is “integrity protected” encryption.
- $MAC_{KAB}(X)$ denotes a cryptographic check value (MAC) of data *X* using a key *KAB* that is shared between *A* and *B*.
- $Sig_{A}(X)$ denotes the signature (with appendix) computed by *A* on the data *X*.


Example 1 (timestamp & encryption)
A
B
$M_1 = Enc_{KAB}(T_A||B||MDC)$

Assumptions?
Sync clock
KAB shared

Goals?
Mutual or unilateral?

Note at this point – we are not doing all the protocols in the standard – there are a lot and some of them are essentially the same (we only select a few and hope you can identify what to look out for).

This example can be found in clause 5.1.1 of ISO/IEC 9798-2.

In each case go through protocol first.

So do we have data origin authentication? yes, especially since we assumed integrity protected encryption, possible MDCs, message redundancy.

Do we have freshness? Yes, timestamp.

WHAT HAPPENS IF WE REMOVE TA? Is it still secure? No, Replay attack

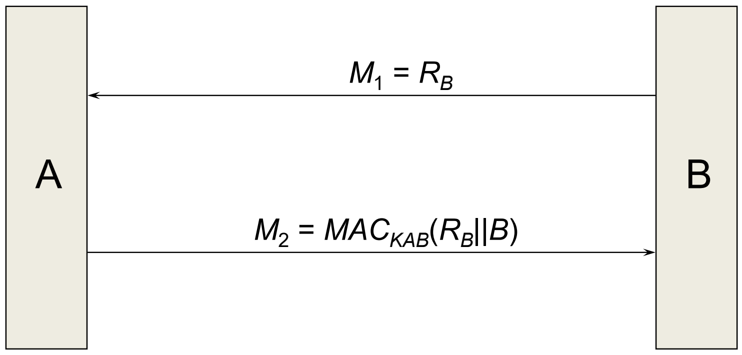
It is based on the use of time-stamps (for freshness) and encryption (for origin and integrity checking). It provides *unilateral authentication* (*B* can check *A*'s identity, but not vice versa).

When *B* receives the message from *A*, *B* deciphers the enciphered string. *B* checks three things:

- that the deciphered message 'makes sense' (has the appropriate redundancy) – comes from possible MDC, is sent to B and good guess for TA,
- that the time-stamp is within its current window (and, using its 'log', that a similar message has not recently been received),
- that *B*'s name is correctly included.

If all three checks are correct, then *B* accepts *A* as valid.

Example 2 (nonce & MAC)



This example can be found in clause 5.1.2 of ISO/IEC 9798-4.

Assumptions?
KAB shared

Goals?
Mutual or unilateral?

So, where does the freshness come from? Nonce B

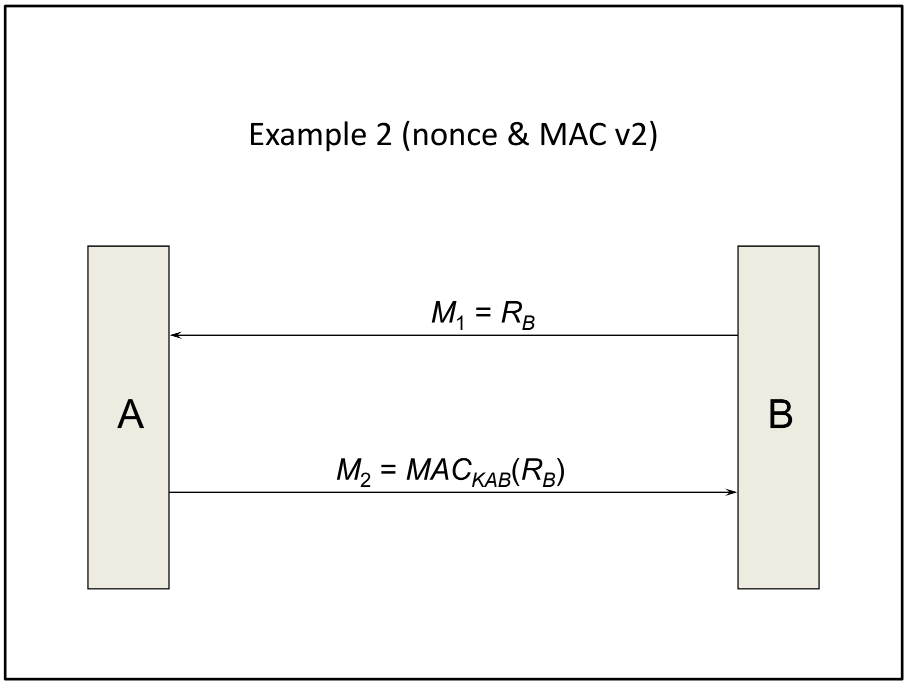
Where does data origin authentication come from? MAC

It is based on the use of nonces (for freshness) and a data integrity mechanism (for origin and integrity checking). It provides *unilateral authentication* (*B* can check *A*'s identity, but not vice versa).
When *B* sends the message M_1, *B* stores the nonce R_B. When *B* receives message M_2, *B* first assembles the string $R_B||B$ and then computes $MAC_{\mathrm{KAB}}(R_B||B)$, using the shared secret K_{AB}. *B* checks one thing:
- that the newly computed check value agrees with the one in message M_2,

If the check is correct, then *B* accepts *A* as valid.

Note also that the nonce check here is implicit. B checks that A has responded with the correct nonce by checking the MAC value. A never actually sends the nonce back to B.

The only time we have to perform an explicit nonce check is when we use symmetric encryption as our origin authentication protocol method. Why? Because then our integrity is tied to our formatting, redundancy – if we decrypt and the resultant plaintext is the nonce we were looking for then we trust that nobody messed with the message.



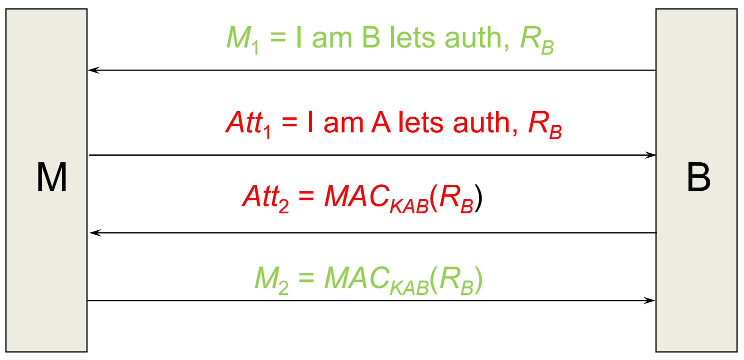
Why do we need the B?

Why is the identifier **B** there? Reflection attack.

Could we replace the MAC with a signature? Yes, in fact then is would be the protocol described in clause 5.1.2 in 9798-3 (instead of the protocol here which is the same clause in part 2).

If we change it to a signature do we really still need the B identifier (See slide 40)? No, reflection attack would not be possible. Reflection works because over RB as both A and B will create the same MAC. If I reflect back on A a challenge with RB the response signed by A then A cannot use that as a response to As challenge.

Example 2 (Reflection attack)



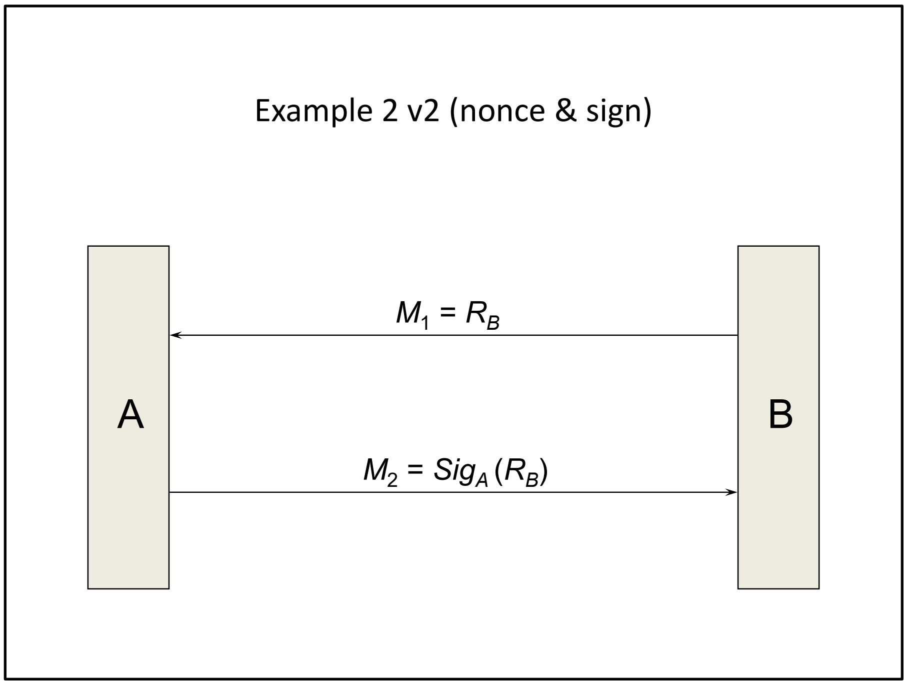
M
B
M_1 = I am B lets auth, R_B
Att_1 = I am A lets auth, R_B
$Att_2 = MAC_{KAB}(R_B)$
$M_2 = MAC_{KAB}(R_B)$

You should be able to recognise this issue in a protocol – if a protocol does not indicate the sender or recipient of a message then this attack is a possibility?

How do you spot other attacks?

Masquerade – there is not data origin authentication….

Replay – there is not strong freshness in the protocol!



This is only a made up example to show how reflection is mostly an issue with symmetric origin authentication method. However, it would be better to have message 2 as $Sig_A(R_B,B)$ even though the signature mitigates reflection. The reason is that even though B might never use same R_B value again someone else might. So if C uses this number in future to try and authenticate A, an attacker could just use this message 2 from this previous exchange with B.

Assumptions: B knows A public key

Mutual or unilateral: Unilateral

Freshness: Nonce

Data origin authentication: Signature by A

Can we still do a reflection attack?

No, change the messages on slide 39 and you will see that M ends up with a message signed by B, when he needs to respond with message signed by A.

Attack fails…

Example 3 (timestamp & signatures)

$$\begin{tikzcd} A \arrow[r, bend left, "M_1 = T_A||Sig_A(T_A||B)"] & B \arrow[l, bend left, "M_2 = T_B||Sig_B(T_B||A)"] \end{tikzcd}$$

This example can be found in clause 5.2.1 of ISO/IEC 9798-3. 5.2.1 in 9798-2 is the same except with MACs.

Does both A and B know messages are fresh an why? Timestamps from both parties.

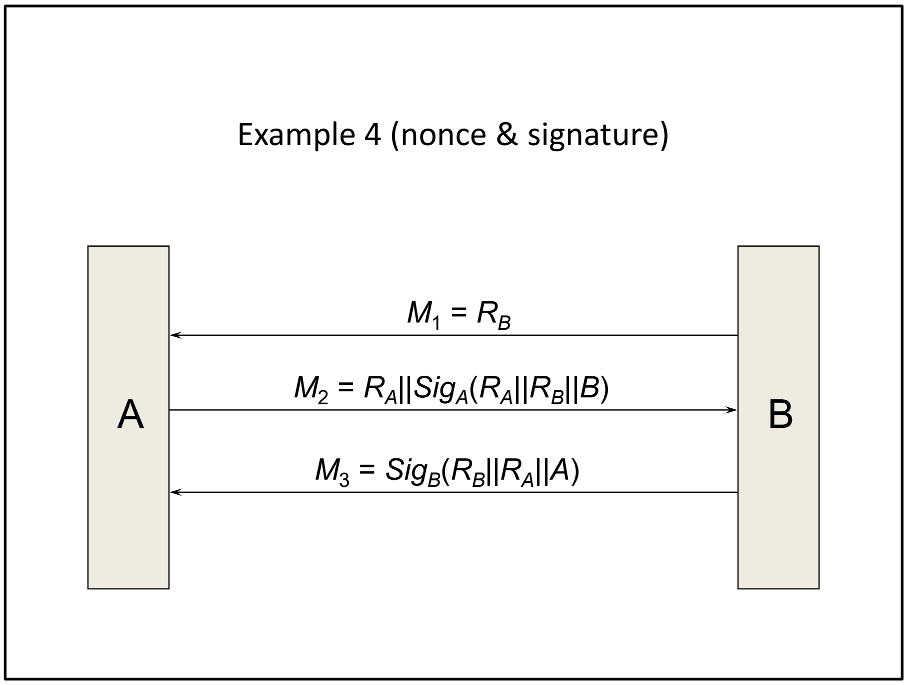
Integrity/data origin? Signature.

It is based on the use of time-stamps (for freshness) and digital signature (for origin and integrity checking). It provides *mutual authentication* (*B* can check *A*'s identity and vice versa).

When *B* receives M_1, *B* first assembles the string $T_A||B$. *B* then checks two things (using a copy of *A*'s public verification key):
• that the time-stamp T_A is within its current window (and, using its 'log', that a similar message has not recently been received),
• that the signature in M_1 is a valid signature on the string $T_A||B$.
If the checks are correct, then *B* accepts *A* as valid and sends message M_2.

When *A* receives M_2, *A* assembles the string $T_B||A$ and checks 2 things:
• that the time-stamp T_B is within its current window (and, using its 'log', that a similar message has not recently been received),
• that the signature in M_2 is a valid signature on the string $T_B||A$.
If the checks are correct, then *A* accepts *B* as valid.

What could possibly happen if we leave B and A out of the message? Do we worry about reflection? Probably not as the Signature already gives indication of direction of message. What could happen if we only have $M_1 = T_A||Sig_A(T_A)$? Well, we could take that message and send it to anyone not B (within window of acceptance) and pretend to be A.


Example 4 (nonce & signature)
A
B
$M_1 = R_B$
$M_2 = R_A||Sig_A(R_A||R_B||B)$
$M_3 = Sig_B(R_B||R_A||A)$

Note that RB and RA switches around.

This example can be found in clause 5.2.2 of ISO/IEC 9798-3.

How does B know the message from A is fresh? Nonce R_B
How does A know the message from B is fresh? Nonce R_A

Data origin authentication? Signature.

Mutual or unilateral? Mutual.

It is based on the use of nonces (for freshness) and digital signature (for origin and integrity checking). It provides *mutual authentication* (*B* can check *A*'s identity and vice versa).
When *B* sends M_1, *B* stores the nonce R_B. When *A* sends M_2, *A* stores the nonces R_A and R_B. When *B* receives M_2, *B* first assembles the string $R_A||R_B||B$. *B* then checks one thing (using a copy of *A*'s public verification key):

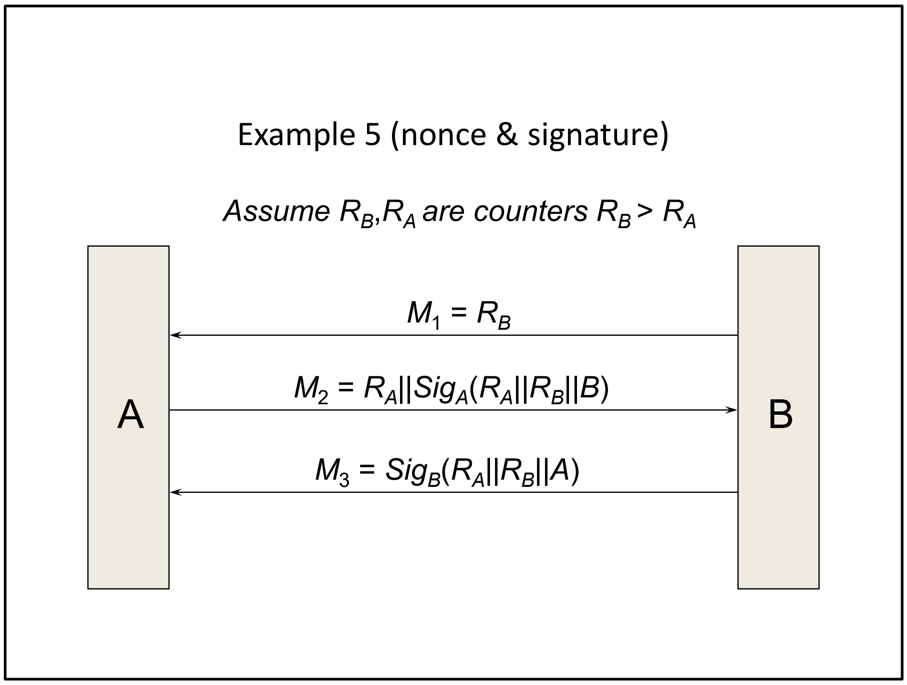
- that the signature in M_2 is a valid signature on the string $R_A||R_B||B$.

If the check is correct, then *B* accepts *A* as valid and sends message M_3.

When A receives M_3, A assembles the string $R_B||R_A||A$ and checks one thing:

- that the signature in M_3 is a valid signature on the string $R_B||R_A||A$.

If the check is correct, then A accepts B as valid.



Example 5 (nonce & signature)

Assume R_B,R_A are counters $R_B > R_A$

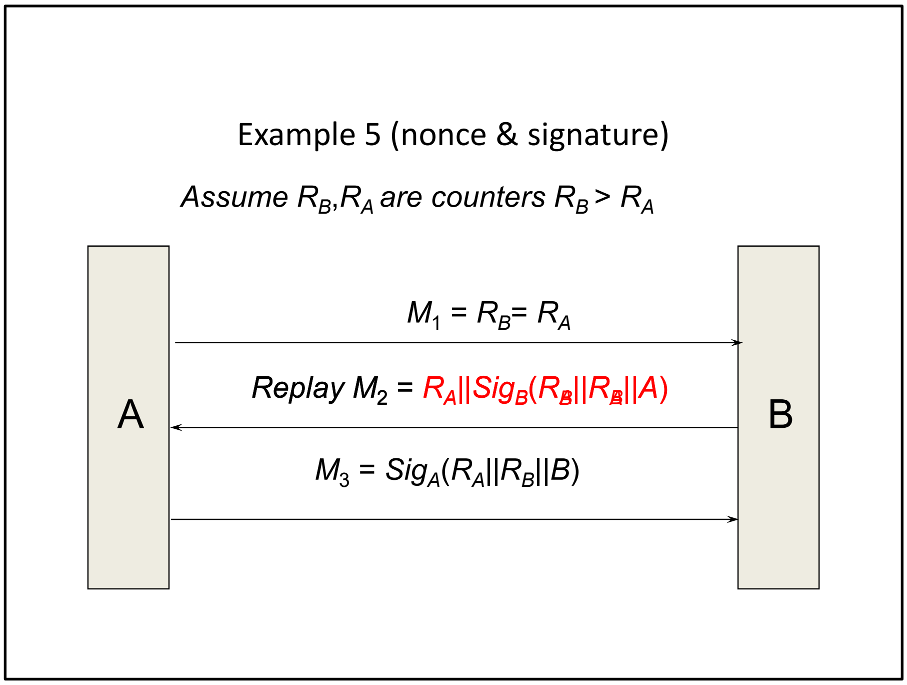
$$\begin{tikzcd}
A \arrow[r, shift left=4, leftarrow, "M_1 = R_B"] \arrow[r, "M_2 = R_A||Sig_A(R_A||R_B||B)"] \arrow[r, shift right=4, leftarrow, "M_3 = Sig_B(R_A||R_B||A)"'] & B
\end{tikzcd}$$

Anyone care to guess why nonces are swopped? Senders nonce first, indicate direction

Remember a nonce is not a random, it is a never repeat – we assume A never repeat or B never repeats, not A never repeats a number B had earlier.

RB and RA swop around protects the nonce and makes it harder to replay (remember the it could be a counter or a logical timestamp…)

Lets say I record this transaction knowing that A's counter or logical timestamp is less than that of B?



Example 5 (nonce & signature)

Assume R_B,R_A are counters $R_B > R_A$

A

B

$M_1 = R_B = R_A$

Replay $M_2 = R_A||Sig_B(R_B||R_B||A)$

$M_3 = Sig_A(R_A||R_B||B)$

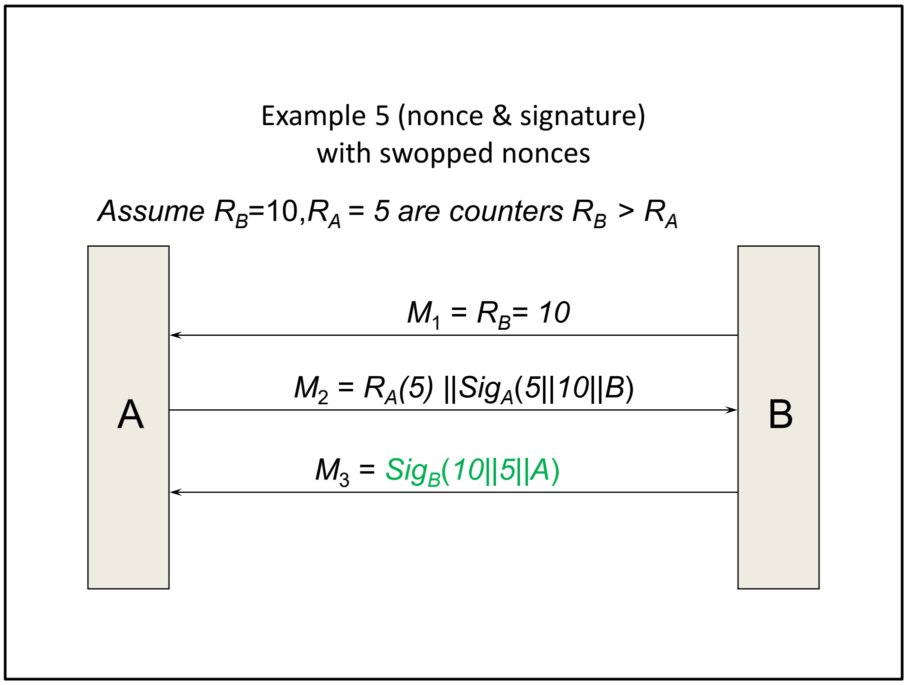
You have to consider slide 43 and 44 together. If is makes more sense if you think about ‘B’ here as M the attacker (you rename B->M (B is not actually involved, someone is pretending to be B).

An attacker records the protocol on slide 43 – here B initiated the protocol and sent RB. RB is a nonce so B makes sure he never uses it again. So trying to use this material to pretend that we are A authenticating to B is not useful.

The problem is that in future A might want to authenticate B and end up using RA equal to previous RB of the recorded execution. This is what is happening on this slide.

A starts to authenticate by sending M1= RB (this is a nonce generate by RA but equal to value RB on slide 43). The attacker M (pretending to be B) sees it is the same as the one he recorded. Now he can construct a valid M2 by replaying what he recorded on slide 43. He responds with RA (this is strictly RB as it is should be nonce generated by B but has same value as previous RA). He also has a message RA||RB||A signed by B from previous message M3, so he can reuse this to construct message M2. The attacker therefore convinces A that he is B.

However, if we kept the protocol as specified on slide 42 this would not happen as the format of the messages in M2 and M3 is different.

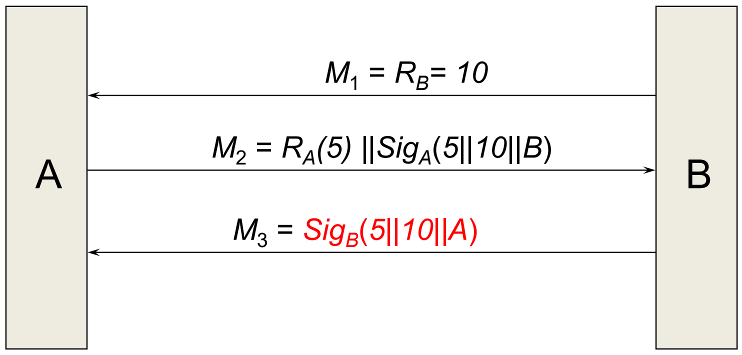

Example 5 (nonce & signature)
with swopped nonces
Assume R_B=10,R_A = 5 are counters $R_B > R_A$
A
B
$M_1 = R_B = 10$
$M_2 = R_A(5) \| Sig_A(5\|10\|B)$
$M_3 = Sig_B(10\|5\|A)$

To start with – RA and RB are required to be nonces, but the design allows either for:
Random numbers, which we consider infeasible to repeat across parties – or at least not at a predictable time – it is possible that if I watch A for years that she so happens to use same number but unlikely.
Counters, which should not repeat for any one party but it does mean that we will cases where different parties do use the same value as the counter increments.

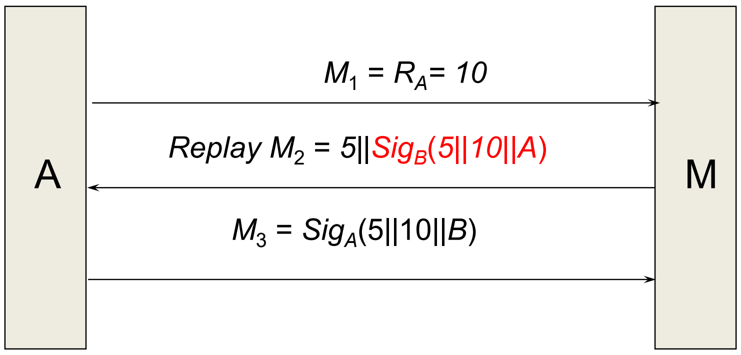
For counters, we assume that RB will never again be 10, RA will never again be 5, but RA could in future be 10.

Example 5 (nonce & signature)
without swopped nonces

Assume R_B=10,R_A = 5 are counters $R_B > R_A$



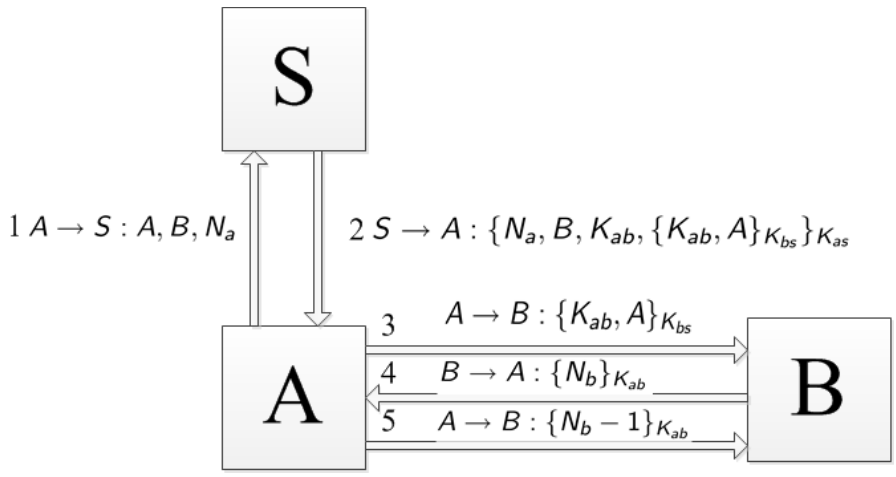
Later A so happens to use M1 that is equal to message M1 that B used before. This allows M to pretend to be B.



Now A reuses 10 for some reason as the original nonce in M_1. This means M_3 from the protocol execution with unswopped nonces could be reused in M2 here (with attacker M free to choose whatever here nonce is so can set to 5).

You would not be able to use the swopped nonce M_3. That said if we used random numbers and A decides to use 5 again for some reason then the nonces would not help.

Just for fun…


S
$1\ A \rightarrow S : A, B, N_a$
$2\ S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
$3\ A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
$4\ B \rightarrow A : \{N_b\}_{K_{ab}}$
$5\ A \rightarrow B : \{N_b - 1\}_{K_{ab}}$
A
B

If you saw this how would you analyse it?

Assumptions A and B share key with S

Is S authenticated to A? Yes, Freshness (NA), Data origin authentication? Known message encrypted KAS

Is A authenticated to S? No

Is S authenticated to B? No

Is B authenticated to A? Not really, Message 4 (Nb encrypted with KAB), only B could retrieve and use KAB from message 3 but we do not know what Nb is supposed to be. It would have been better for message 3 to be Na,{Kab,A}Kbs and 4 to be {Na,Nb}Kab. Then A and B would be mutually authenticated.

Is A authenticated to B? Yes, message 5. Freshness in nonce, only A can create message with KAB.

The end!

?

Any questions…

49