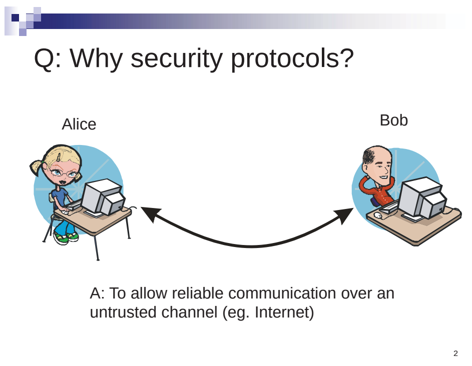
7 Security Protocols

7.1 Introduction

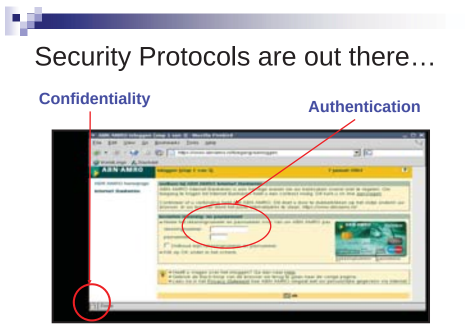
A security protocols is a 'recipe' for communication; it describes the sequence of messages that the participants in the communication should send. The goal of security protocols is to provide secure communication over an unsecure channel. Here secure can mean for example confidentiality of the data exchanged or proper authentication of the parties involved and their messages. In the example of internet banking you want both of these properties; the bank should be properly authenticated before you enter your login information and the requests for transfers by the client also need to be authenticated. Confidentiality is needed to protect the privacy of the client (as well as the login information).



Q: Why security protocols?

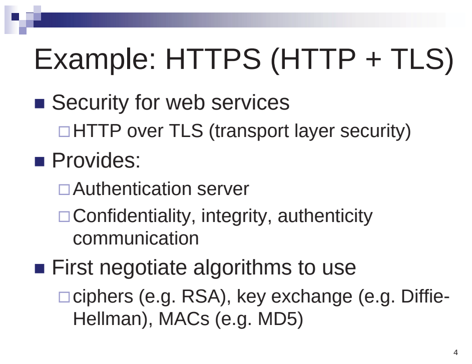
A: To allow reliable communication over an untrusted channel (eg. Internet)

2



Security Protocols are out there…

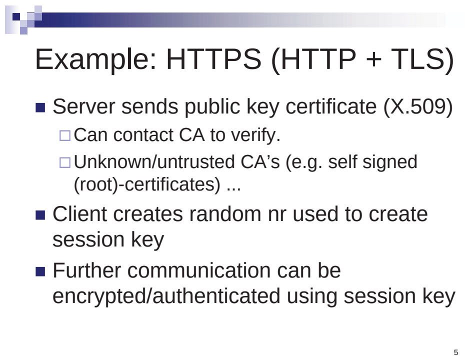
In earlier chapters we have seen mechanisms that can be used to achieve such properties; encryption, MACs, digital signatures, etc. A good security protocol combines these methods in a way that all desired security properties are achieved.



Example: HTTPS (HTTP + TLS)

- Security for web services
  - HTTP over TLS (transport layer security)
- Provides:
  - Authentication server
  - Confidentiality, integrity, authenticity communication
- First negotiate algorithms to use
  - ciphers (e.g. RSA), key exchange (e.g. Diffie-Hellman), MACs (e.g. MD5)

4



Example: HTTPS (HTTP + TLS)

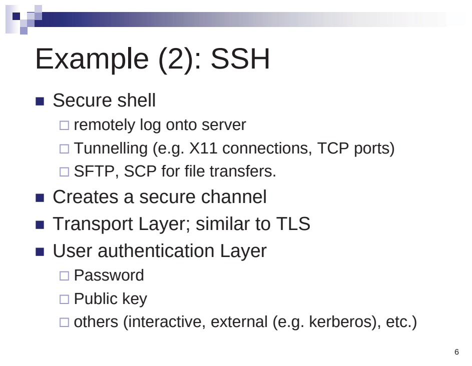
- Server sends public key certificate (X.509)
  - Can contact CA to verify.
  - Unknown/untrusted CA's (e.g. self signed (root)-certificates) ...
- Client creates random nr used to create session key
- Further communication can be encrypted/authenticated using session key

5

To 'security' browse on the web, HTTPS i.e. HTTP over a TLS connection, is used. With HTTPS the server is authenticated (based on a certificate authority tree using X5.09 certificates; see previous chapter) and confidentialilty, integrity and authenticity of the communication is provided. For the user the authenticity is a guarantee that they are indeed communicating with the correct server; the one that belongs to that webaddress (according to the CA). The client does not have certificates to prove their identity; the authenticity for the webserver only means that all messages come from the same (unidentified) party. To identify the party it needs to employ other mechanisms; e.g. having the user enter a username and password (see also Chapter 6 on Authentication) once the connection has been established.

Because the server has a (certified) public key but the client does not, in the setup of the connection we can send messages confidentially from the client to the server but not the other way around. As such the client should generate the secret (random number) on which the session key is based (any contribution from the server is to be seen as public information).

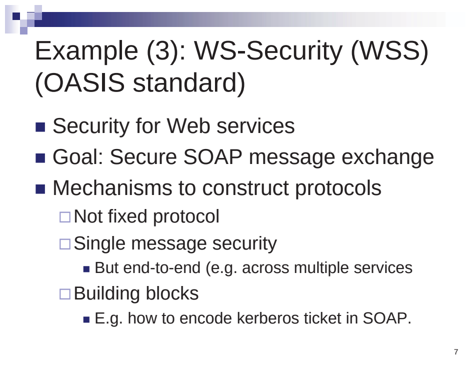
Another commonly used security protocol is Secure Shell (SSH) which is used to remotely log onto a server. It is a transport layer protocol that supports tunneling, in which traffic for e.g. X11 (graphical interface) connections and TCP ports can be forwarded over the secure SSH connection. SCP for securely copying files and SFTP, a secure alternative for FTP (File Transfer Protocol), also use SSH. SSH supports password authentication of the client but also a public key can be used; you place your public key on your account on the server. Your local SSH client can then connect to the server by using your private key instead of you entering a password. (Though you would likely want to protect your private key with a password/phrase but then against attackers with access to your machine rather than eavesdroppers on the network - i.e. a completely different attacker model.) External authentication (e.g. Kerberos) is also possible.



Example (2): SSH

- Secure shell
  - remotely log onto server
  - Tunnelling (e.g. X11 connections, TCP ports)
  - SFTP, SCP for file transfers.
- Creates a secure channel
- Transport Layer; similar to TLS
- User authentication Layer
  - Password
  - Public key
  - others (interactive, external (e.g. kerberos), etc.)

6

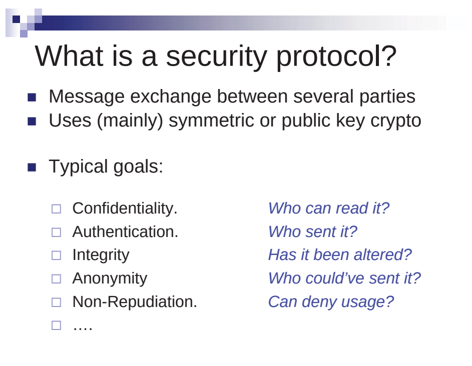


Example (3): WS-Security (WSS) (OASIS standard)

- Security for Web services
- Goal: Secure SOAP message exchange
- Mechanisms to construct protocols
  - Not fixed protocol
  - Single message security
    - But end-to-end (e.g. across multiple services
  - Building blocks
    - E.g. how to encode kerberos ticket in SOAP.

7

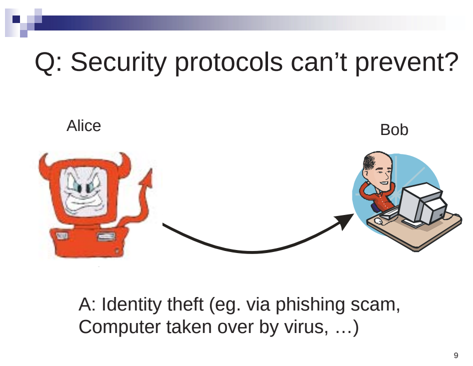
The Web Service Security (WS-Security or: WSS) standard by OASIS is technically not a protocol but rather provides building block and mechanisms to construct protocols. Its goal is to provide end-to-end security of individual (SOAP) messages. In a web services setting communication may not be simply between a client and a single web service; instead information may need to travel accross several web services. End-to-end security means that the data is protected along the entire chain, not just for a single step in the chain.



What is a security protocol?

- Message exchange between several parties
- Uses (mainly) symmetric or public key crypto
- Typical goals:

Confidentiality.	*Who can read it?*
Authentication.	*Who sent it?*
Integrity	*Has it been altered?*
Anonymity	*Who could've sent it?*
Non-Repudiation.	*Can deny usage?*
….	

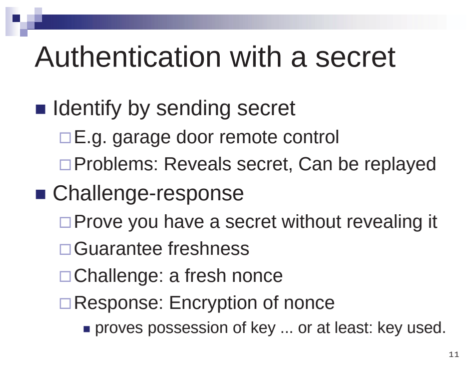


Q: Security protocols can't prevent?

A: Identity theft (eg. via phishing scam, Computer taken over by virus, …)

9

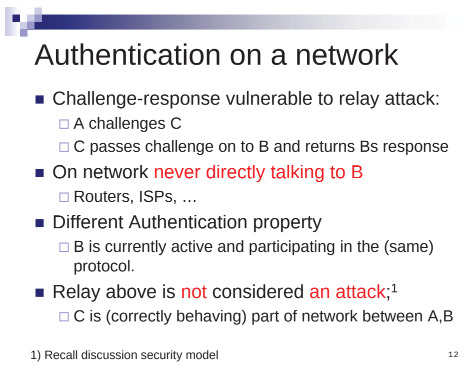
Above we have seen some examples of well-known/commonly used protocols. But what makes a protocol work; how can security goals be achieved?



Authentication with a secret

- Identify by sending secret
  - E.g. garage door remote control
  - Problems: Reveals secret, Can be replayed
- Challenge-response
  - Prove you have a secret without revealing it
  - Guarantee freshness
  - Challenge: a fresh nonce
  - Response: Encryption of nonce
    - proves possession of key ... or at least: key used.

11



Authentication on a network

- Challenge-response vulnerable to relay attack:
  - A challenges C
  - C passes challenge on to B and returns Bs response
- On network never directly talking to B
  - Routers, ISPs, …
- Different Authentication property
  - B is currently active and participating in the (same) protocol.
- Relay above is not considered an attack;[1]
  - C is (correctly behaving) part of network between A,B

1) Recall discussion security model

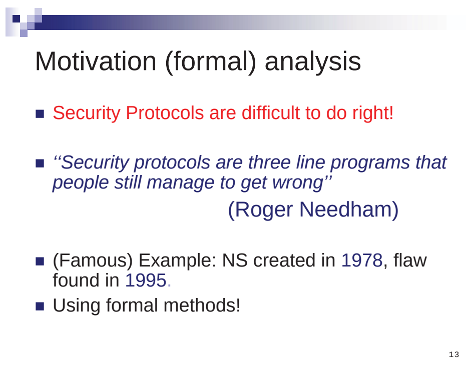
12

Lets try to construct a very simple protocol for authentication: I have a secret stored on a device/server and when I want to authenticate I send the secret. If the attacker is not able (remember security depends on the attacker model) to see messages between me and the device and cannot guess the secret (see e.g. the section on passwords, Section 6.2) then this solution works. On a computer network, however, we definitely need to consider attackers that are able to eavesdrop on the communication. With such an slightly stronger attacker model this approach is clearly not secure - after the first time I authentication the attacker also has the secret. Simply encrypting the secret will not help; the attacker does not learn the secret but can simply replay the same message to authenticate.

A challenge-response mechanism can be used instead to prove that I have the secret without revealing it. The server sends a challenge (e.g. a random number) and my response should be one that only I can make and only after I have received the challenge (e.g. secret+challenge; which should be sent encrypted to not reveal the secret). In this way upon receiving the response, the server can be sure that I got the challenge and that my response is fresh and not a response that I sent out in an earlier session.

What the server cannot know is how far away I am. When remotely accessing some service on the network it likely does not matter how far away the user is. When running a protocol between e.g. an OVchip card and a metro access gate one would like to be sure that the user (card) is actually at the gate. If a response comes within a millisecond the challenge travelled at most 0,001s * 300.000 km/s = 300km thus responder is at most 150km away. To actually get useful distance bounds one needs to use more advanced *distance bounding protocols* which are outside the scope of this course. (See the Master course Physical Aspects of Digital Security (2IC35) if you want to learn about distance bounding protocols.) In the protocol analysis treated here we will assume the location is irrelevant.

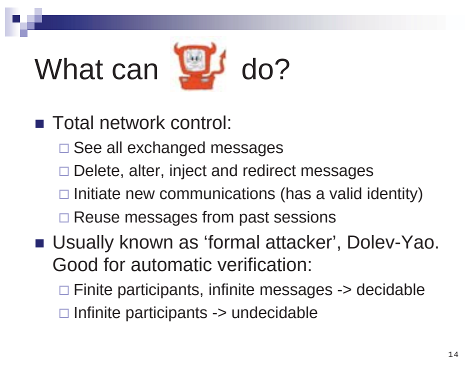
Authentication in our security model for communicating over a network becomes: Authentication of Bob to Alice is correctly achieved when: Bob is currently active and running the same protocol. Usually a protocol also tries to share some data (e.g. a message from Alice to Bob, random nounces to create session keys,etc.) between Alice and Bob. Secrecy of such data is thus also a common protocol goal.



Motivation (formal) analysis

- Security Protocols are difficult to do right!
- *''Security protocols are three line programs that people still manage to get wrong''*
  (Roger Needham)
- (Famous) Example: NS created in 1978, flaw found in 1995.
- Using formal methods!

13

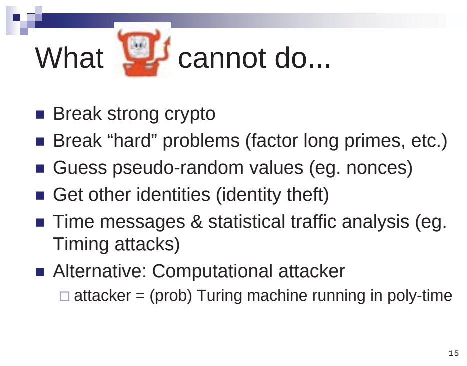


What can do?

- Total network control:
  - See all exchanged messages
  - Delete, alter, inject and redirect messages
  - Initiate new communications (has a valid identity)
  - Reuse messages from past sessions
- Usually known as 'formal attacker', Dolev-Yao. Good for automatic verification:
  - Finite participants, infinite messages -> decidable
  - Infinite participants -> undecidable

14

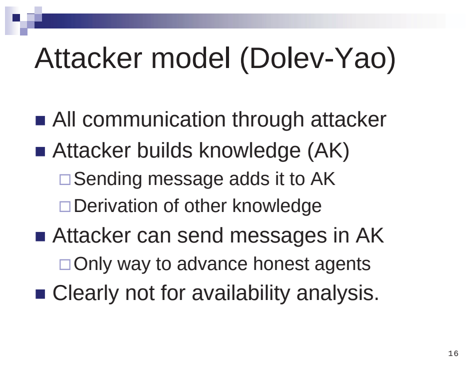
Eventhough the protocols may seem simple, it is easy to make mistakes. Formal analysis, sometimes supported with automated ways of checking models or proofs, helps to find mistakes.



What cannot do...

- Break strong crypto
- Break "hard" problems (factor long primes, etc.)
- Guess pseudo-random values (eg. nonces)
- Get other identities (identity theft)
- Time messages & statistical traffic analysis (eg. Timing attacks)
- Alternative: Computational attacker
  - attacker = (prob) Turing machine running in poly-time

15

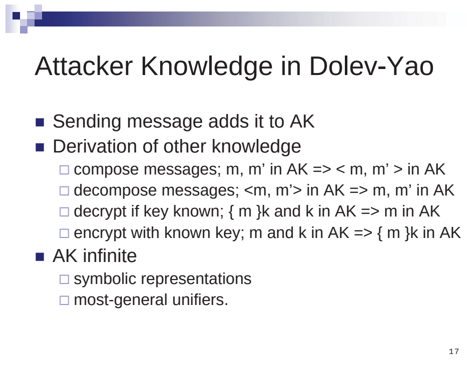


Attacker model (Dolev-Yao)

- All communication through attacker
- Attacker builds knowledge (AK)
  - Sending message adds it to AK
  - Derivation of other knowledge
- Attacker can send messages in AK
  - Only way to advance honest agents
- Clearly not for availability analysis.

16

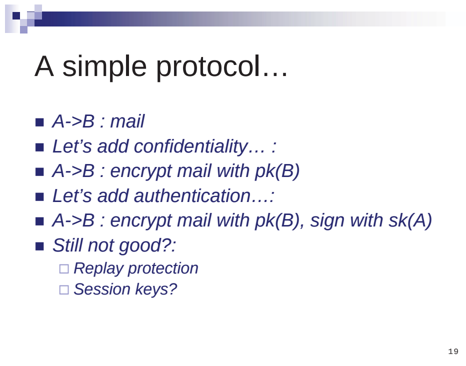
To do formal analysis we have to be very precise on what our security model is. We already looked at goals (e.g. authentication) is some details. The next thing we need is an attacker model. As the network is a dangerous place (see Chapter 3) we play it safe and assume the attacker has full control over the network; it can intercept (all) messages, block or changes them, and create new messages. The attacker may also have a (or several) valid identity on the network with which it can partake in protocol runs. On the otherhand we do need some tools to be able to build secure protocols. So we do not have to mix the analysis of the protocol logic with the analysis of the cryptography used we assume we have 'perfect' cryptography available; the attacker will not be able to decypher any encrypted messages without the key and will not be able to guess random numbers. This attacker model is called the *Dolev-Yao* model.



Attacker Knowledge in Dolev-Yao

- Sending message adds it to AK
- Derivation of other knowledge
  - compose messages; m, m' in AK => < m, m' > in AK
  - decompose messages; <m, m'> in AK => m, m' in AK
  - decrypt if key known; { m }k and k in AK => m in AK
  - encrypt with known key; m and k in AK => { m }k in AK
- AK infinite
  - symbolic representations
  - most-general unifiers.

17



A simple protocol…

- *A->B : mail*
- *Let's add confidentiality… :*
- *A->B : encrypt mail with pk(B)*
- *Let's add authentication…:*
- *A->B : encrypt mail with pk(B), sign with sk(A)*
- *Still not good?:*
  - *Replay protection*
  - *Session keys?*

19

The Dolev-Yao model is very suitable for automated formal analysis. As long as the number of (potential) participants is finite verification of properties like authentication is possible (decidable) even when considering an arbirary number of possible messages. An alternative attacker model is the computational attacker where an attacker is an arbitrary probabilistic turing machine. With such an attacker model we can also consider the cryptographic primatives being used (is encryption schema x combined with random number generator y suitable for protocol z?) but it is much harder to reason about. Here we will stick with the Dolev-Yao model.

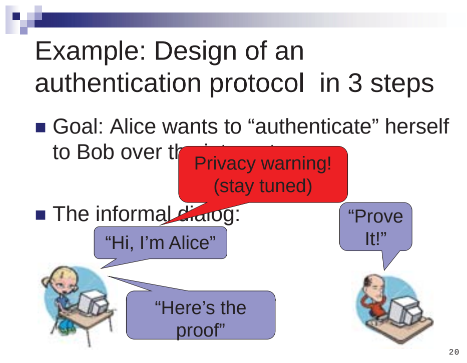
Crusial to analysing a protocol is the attacker knowledge. As the attacker has full control over the network, sending any message will add that message to the knowledge of the attacker. The attacker can combine this message with the knowledge she already has; e.g. decrypting a message if she knows the key. With possible goals and attacker model in place we can start designing and analysing security protocols.

To describe a protocol we list the sequence of messages that each participant in the protocol is intended to send. For instance in *A->B: mail* the initiator (for example Alice) takes the role A and the intended recipient (e.g. Bob) takes role B. Thus an email from Alice to Bob saying 'lets meet at five' would be one run of this protocol.

As the attacker can see messages, we may need to protect the confidentiality of the message which we could do by encrypting with it the public key of B. Here we assume that the participants in a protocol run already have the right public keys (recall the first lab session). Alice thus already knows the correct public key of Bob (recall the first lab session) and can send her message encrypted.

Though only Bob can now read the message, Bob cannot know whom it came from. (It is supposed to come from Alice as she is playing role A in this run of the protocol but it may as well have been created by the attacker.) To solve this Alice could sign her message. However, Bob is still not sure that this is a fresh message from Alice; the attacker could be replaying the same message that Alice sent last week.

A challenge-response mechanism could be achieve proper authentication. We may also try to achieve a shared secret which can be used to create a session key for further communication. The basic protocol design contains three steps: Alice tells Bob she wants to talk, Bob presents a challenge to Alice to know it is really her and Alice proofs this by answering the challenge.



Example: Design of an authentication protocol in 3 steps

- Goal: Alice wants to "authenticate" herself to Bob over th
- The informal dialog:

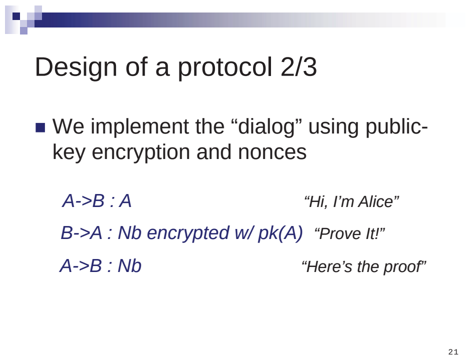
Privacy warning! (stay tuned)

"Hi, I'm Alice"

"Prove It!"

"Here's the proof"

20



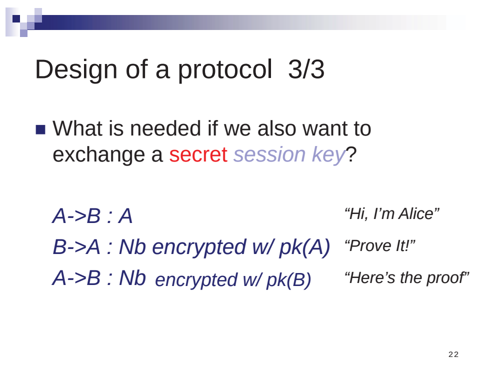
Design of a protocol 2/3

- We implement the "dialog" using public-key encryption and nonces

A->B : A	*"Hi, I'm Alice"*
B->A : Nb encrypted w/ pk(A)	*"Prove It!"*
A->B : Nb	*"Here's the proof"*

21

Our first attempt at implementing the basic design uses the public key of Alice and a randomly generated nonce to challenge her. Only Alice can decrypt the challenge and returning the nonce. The final message obviously reveals the nounce so it will not be a shared secret between Alice and Bob. We can try to address this by having Alice encrypt her answer with the public key of Bob. Is Alice now correctly authenticated to Bob? (i.e. if this is used as the 'meet at 5 today' protocol, can Bob now be certain that Alice wants to meet him today at 5?) Does the nounce they use remain secret (can they savely use it to encrypt following messages? e.g. where to meet, what to bring, etc.)

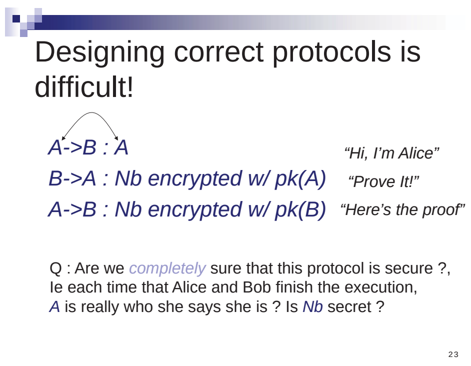


Design of a protocol 3/3

- What is needed if we also want to exchange a secret *session key*?

A->B : A	*"Hi, I'm Alice"*
B->A : Nb encrypted w/ pk(A)	*"Prove It!"*
A->B : Nb encrypted w/ pk(B)	*"Here's the proof"*

22



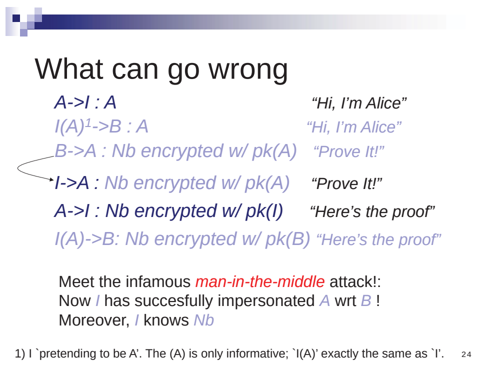
Designing correct protocols is difficult!

A->B : A	*"Hi, I'm Alice"*
B->A : Nb encrypted w/ pk(A)	*"Prove It!"*
A->B : Nb encrypted w/ pk(B)	*"Here's the proof"*

Q : Are we *completely* sure that this protocol is secure ?,
Ie each time that Alice and Bob finish the execution,
A is really who she says she is ? Is *Nb* secret ?

23

A problem can occur when Alice wants to mean Ivy who, as it turns out, is malicious. Ivy can misuse Alice's wish to meet to authenticate to Bob, making Bob think that Alice wants to meet him while Alice actually wants to meet Ivy.

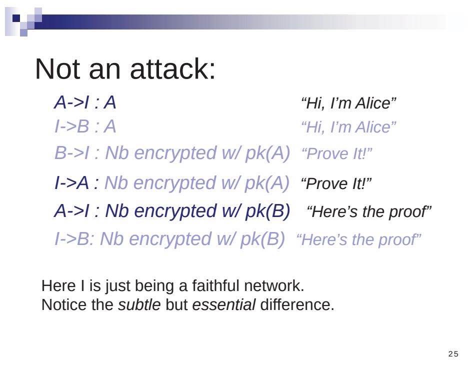


What can go wrong

A->I : A	*"Hi, I'm Alice"*
I(A)[1]->B : A	*"Hi, I'm Alice"*
B->A : Nb encrypted w/ pk(A)	*"Prove It!"*
I->A : Nb encrypted w/ pk(A)	*"Prove It!"*
A->I : Nb encrypted w/ pk(I)	*"Here's the proof"*
I(A)->B: Nb encrypted w/ pk(B)	*"Here's the proof"*

Meet the infamous *man-in-the-middle* attack!:
Now *I* has succesfully impersonated *A* wrt *B* !
Moreover, *I* knows *Nb*

1) I `pretending to be A'. The (A) is only informative; `I(A)' exactly the same as `I'. 24



Not an attack:

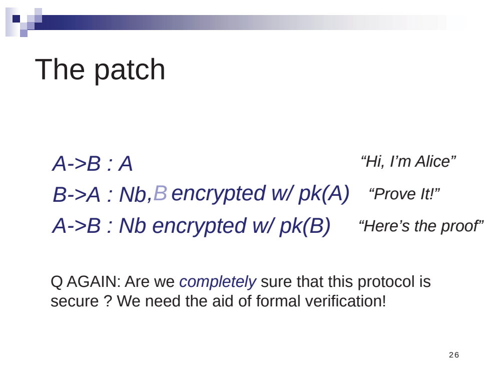
A->I : A	*"Hi, I'm Alice"*
I->B : A	*"Hi, I'm Alice"*
B->I : Nb encrypted w/ pk(A)	*"Prove It!"*
I->A : Nb encrypted w/ pk(A)	*"Prove It!"*
A->I : Nb encrypted w/ pk(B)	*"Here's the proof"*
I->B: Nb encrypted w/ pk(B)	*"Here's the proof"*

Here I is just being a faithful network.
Notice the *subtle* but *essential* difference.

25

When Alice sends her meeting request, Ivy forwards it to Bob (pretending to be Alice in the protocol run with Bob). Bob will send a challenge to check whether Ivy is actually Alice. Ivy cannot answer this challenge herself but can forward it to Alice (predenting it came from her). As Alice is waiting for a challenge to come from Ivy, she is trying to authenticate to Ivy afterall, she will answer this challenge, decrypting the nounce and returning it to Ivy encrypted with Ivy's public key. In this way Ivy obtains the nounce and can now answer the challenge from Bob by encrypting the nounce with his public key. Bob will now go to meet Alice and will think that any further communication (e.g. where to meet) using the nounce comes from her.

Note that the analysis of a security protocol can be quite subtle. For example, the second sequence above looks very similar to the man in the middle attack that we saw before but is actually quite different. Here Ivy is actually a good friend faithfully passing along messages between Alice and Bob. Alice and Bob agree to meet each other as the protocol intended. Also, Ivy does not obtain the nounce.

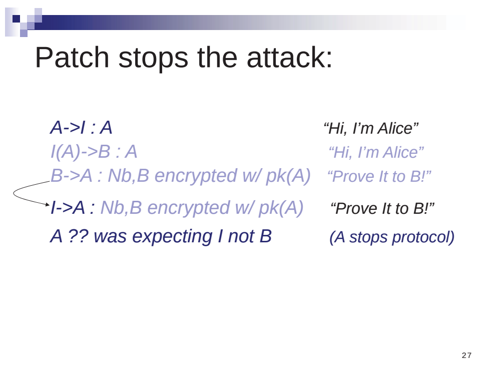


The patch

A->B : A	*"Hi, I'm Alice"*
B->A : Nb,B encrypted w/ pk(A)	*"Prove It!"*
A->B : Nb encrypted w/ pk(B)	*"Here's the proof"*

Q AGAIN: Are we *completely* sure that this protocol is secure ? We need the aid of formal verification!

26



Patch stops the attack:

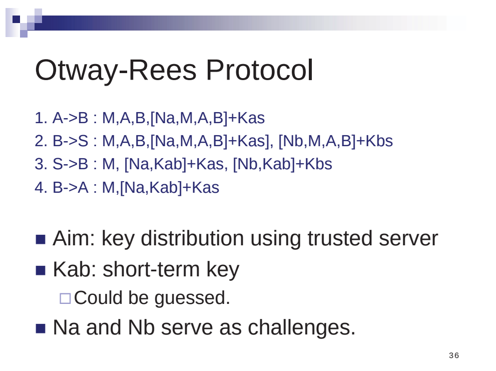
A->I : A	*"Hi, I'm Alice"*
I(A)->B : A	*"Hi, I'm Alice"*
B->A : Nb,B encrypted w/ pk(A)	*"Prove It to B!"*
I->A : Nb,B encrypted w/ pk(A)	*"Prove It to B!"*
A ?? was expecting I not B	*(A stops protocol)*

27

So how can we protect against the man in the middle attack we found? The problem comes from the fact that Alice does not know that the challenge came from Bob. We can solve this by simply telling her; including the identity of the challenger in the challenge. Now Ivy's attack will no longer work; when she forwards the challenge to Alice, Alice will see that it is a challenge from Bob and not the challenge from Ivy that she was expecting - she will thus not answer it. Note that Alice cannot be sure that the challenge came from Bob as anyone can construct such a challenge; however, Bob when getting a response to his challenge, can know for sure that Alice intended this to be for him.

The adjustment makes the attack above impossible. However, are there no other attack possible? In informal argument for correct authentication (of Alice to Bob) should at least argue the following points: (1) A secret of Alice is used; a challenge that only Alice can answer ensures that the attacker cannot complete the authentication without involving Alice (2) Freshness of Alice's response; the secret has to be used in this session and not be e.g. replayed from an earlier session. (3) Alice's response is meant to authenticate her to Bob; when Alice is not trying to authenticate to Bob it should not be possible to trick her into answering the challenge for the attacker.

We addressed point 3 above by adding Bob's name in the challenge that is sent to Alice. But is this always sufficient to prevent Alice from being confused? Let us consider another example protocol: the Otway-Rees Protocol for session key distribution using a trusted server.

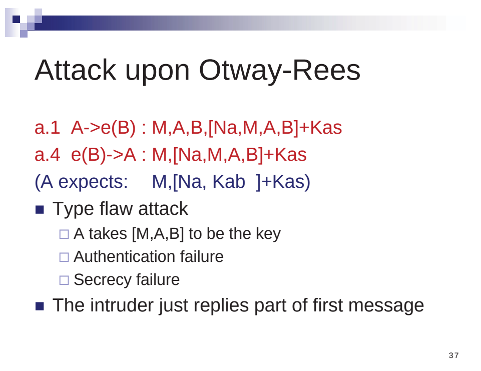


Otway-Rees Protocol

1. A->B : M,A,B,[Na,M,A,B]+Kas
2. B->S : M,A,B,[Na,M,A,B]+Kas], [Nb,M,A,B]+Kbs
3. S->B : M, [Na,Kab]+Kas, [Nb,Kab]+Kbs
4. B->A : M,[Na,Kab]+Kas

- Aim: key distribution using trusted server
- Kab: short-term key
  - Could be guessed.
- Na and Nb serve as challenges.

36



Attack upon Otway-Rees

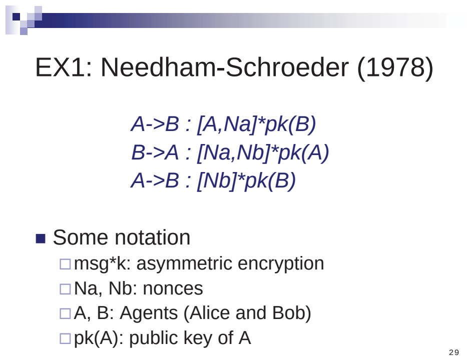
a.1 A->e(B) : M,A,B,[Na,M,A,B]+Kas
a.4 e(B)->A : M,[Na,M,A,B]+Kas
(A expects: M,[Na, Kab]+Kas)

- Type flaw attack
  - A takes [M,A,B] to be the key
  - Authentication failure
  - Secrecy failure
- The intruder just replies part of first message

37

The protocol uses a trusted server to create a short term session key for communication between Alice and Bob. Alice and Bob already share (long term) keys with the trusted server. The server is trusted in that we assume it will behave correctly; generate good random keys, not leak information to the attacker (other then what is leaked by correctly following the protocol) nor misuse the information it has. The server, for example, will know the key used by Alice and Bob to communicate but this is not considered to be a problem.

A so called *type flaw attack* is possible against Otway-Rees. The attacker sends a message to Alice which is of a different type than she is expecting. Depending on the implementation, however, Alice may not be able to tell that this is the case. As part of step 4, Alice is expecting to receive her nounce and a key encrypted with the key that she shares with the server. An attacker will not be able to create such a message. However, there are other messages that are encrypted with this key that the attacker could use: the first message of the same run of that protocol for example. If the attacker sends back the message part from message 1 then it will contain the rights nounce (Na). The remainder of the message $[M, A, B]$ could then by mistake be accepted as being Kab. Care needs to be taken that messages cannot be misused elsewhere in the protocol.

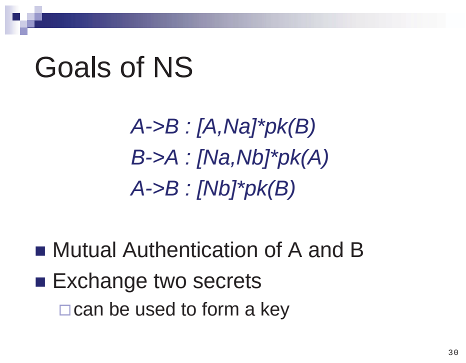


EX1: Needham-Schroeder (1978)

*A->B : [A,Na]*pk(B)*
*B->A : [Na,Nb]*pk(A)*
*A->B : [Nb]*pk(B)*

- Some notation
  - msg*k: asymmetric encryption
  - Na, Nb: nonces
  - A, B: Agents (Alice and Bob)
  - pk(A): public key of A

29



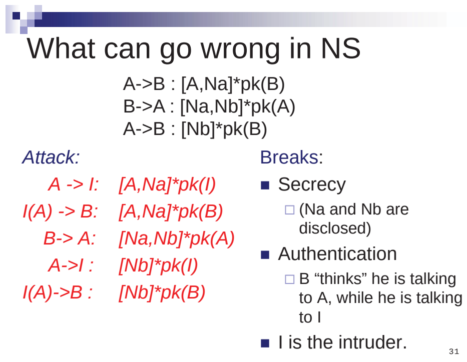
Goals of NS

*A->B : [A,Na]*pk(B)*
*B->A : [Na,Nb]*pk(A)*
*A->B : [Nb]*pk(B)*

- Mutual Authentication of A and B
- Exchange two secrets
  - can be used to form a key

30

The Needham-Shroeder protocol aims to achieve mutual authentication between Alice and Bob. It basically runs the authentication protocol we designed before in both directions. It also aims to maintain secrecy of the nounces used so they can be used to create a session key.



What can go wrong in NS

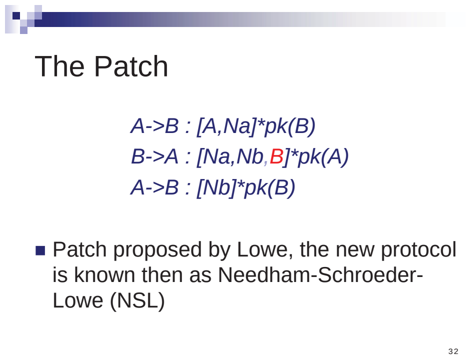
A->B : [A,Na]*pk(B)
B->A : [Na,Nb]*pk(A)
A->B : [Nb]*pk(B)

Attack:

A -> I:	*[A,Na]*pk(I)*
I(A) -> B:	*[A,Na]*pk(B)*
B-> A:	*[Na,Nb]*pk(A)*
A->I :	*[Nb]*pk(I)*
I(A)->B :	*[Nb]*pk(B)*

Breaks:
- Secrecy
  - (Na and Nb are disclosed)
- Authentication
  - B "thinks" he is talking to A, while he is talking to I
- I is the intruder.

31



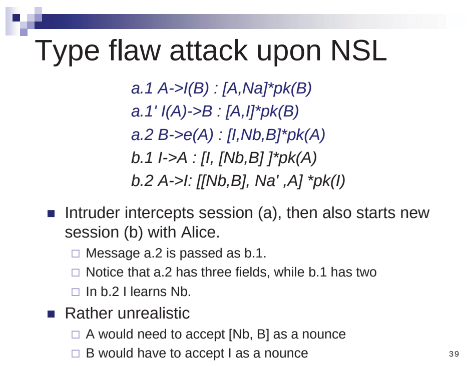
The Patch

*A->B : [A,Na]*pk(B)*
*B->A : [Na,Nb,B]*pk(A)*
*A->B : [Nb]*pk(B)*

- Patch proposed by Lowe, the new protocol is known then as Needham-Schroeder-Lowe (NSL)

32

The flaw is also very similar as the one we saw before; Alice can be tricked into answering a challenge from Bob eventhough she is not trying to authenticate to Bob but rather to Ivy. The patch is also the same; add identity B to the challenge from B. On NSL there is also a potential type flaw attack (though the type confusions that needs to happen seems unlikely to occur in practise).

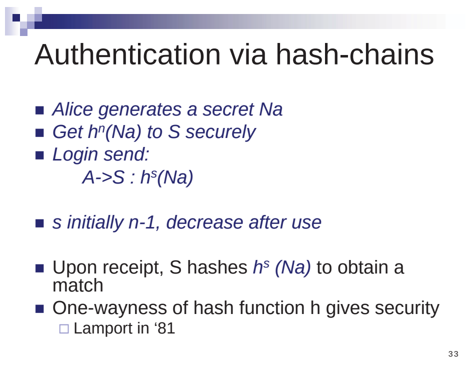


Type flaw attack upon NSL

*a.1 A->I(B) : [A,Na]*pk(B)*
*a.1' I(A)->B : [A,I]*pk(B)*
*a.2 B->e(A) : [I,Nb,B]*pk(A)*
*b.1 I->A : [I, [Nb,B]]*pk(A)*
*b.2 A->I: [[Nb,B], Na' ,A] *pk(I)*

- Intruder intercepts session (a), then also starts new session (b) with Alice.
  - Message a.2 is passed as b.1.
  - Notice that a.2 has three fields, while b.1 has two
  - In b.2 I learns Nb.
- Rather unrealistic
  - A would need to accept [Nb, B] as a nounce
  - B would have to accept I as a nounce

39

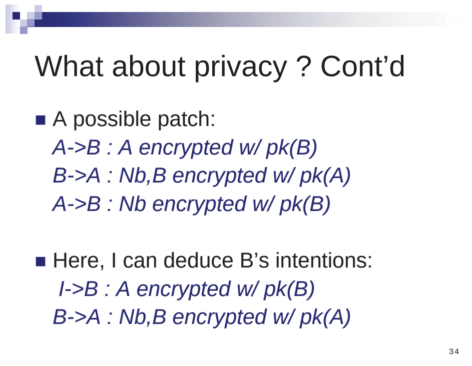


Authentication via hash-chains

- *Alice generates a secret Na*
- *Get h^n(Na) to S securely*
- *Login send:*
  A->S : h^s(Na)
- *s initially n-1, decrease after use*
- Upon receipt, S hashes *h^s (Na)* to obtain a match
- One-wayness of hash function h gives security
  - Lamport in '81

33

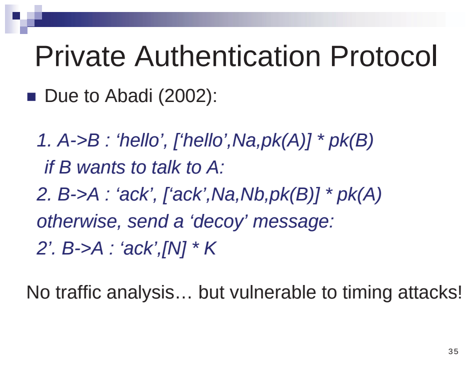
So far we have seen symmetric and asymmetric cryptography being used. Hashes are also commonly used tools in protocols. For example one way to 'reuse' a secret is by using hash chains. First $h^n(N_a)$ (i.e. the hash function h applied n times to random nounce N_a) is somehow securely sent from Alice to S. Once this is achieved it can be used n times to authenticate Alice. When Alice wants to authenticate she sends $h^i(N_a)$ where i is initially $n-1$ and decreased after each authentication. The server can easly check the value by hashing it and comparing it to the stored value for Alice. As the hash is collission resistant other values than $h^i(N_a)$ will not hash to Alices values. The hash is a one-way function so if the attacker learns value $h^i(N_a)$, the values N_a through $h^{i-1}(N_a)$ will still be secret.



What about privacy ? Cont'd

- A possible patch:
  A->B : A encrypted w/ pk(B)
  B->A : Nb,B encrypted w/ pk(A)
  A->B : Nb encrypted w/ pk(B)
- Here, I can deduce B's intentions:
  I->B : A encrypted w/ pk(B)
  B->A : Nb,B encrypted w/ pk(A)

34



Private Authentication Protocol

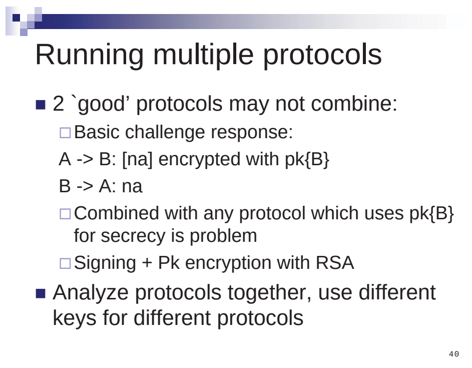
- Due to Abadi (2002):

*1. A->B : 'hello', ['hello',Na,pk(A)] * pk(B)*
if B wants to talk to A:
*2. B->A : 'ack', ['ack',Na,Nb,pk(B)] * pk(A)*
otherwise, send a 'decoy' message:
*2'. B->A : 'ack',[N] * K*

No traffic analysis… but vulnerable to timing attacks!

35

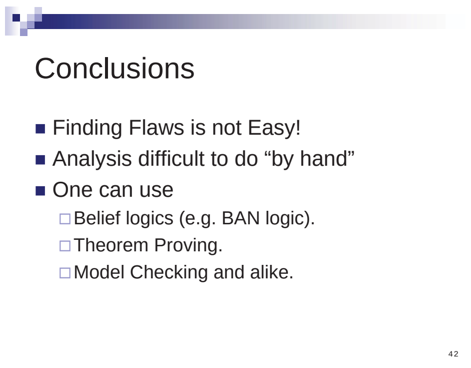
We also mentioned privacy issues with our little the authentication protocol design. Alice's identity is sent in the clear. We can solve this by encrypting it with Bob's public key. Yet even then there is a potential privacy issue: an attacker could determine whether Bob is willing to talk to Alice. Anyone, including the attacker, can send the message $[A] * pk(B)$. Eventhough the attacker cannot answer the challenge from Bob, the very fact that Bob sends a challenge is sufficient. The solution to this is to have Bob always send a reply as is done in Abadi's private authentication protocol.



Running multiple protocols

- 2 `good' protocols may not combine:
  - Basic challenge response:
    A -> B: [na] encrypted with pk{B}
    B -> A: na
  - Combined with any protocol which uses pk{B} for secrecy is problem
  - Signing + Pk encryption with RSA
- Analyze protocols together, use different keys for different protocols

40



Conclusions

- Finding Flaws is not Easy!
- Analysis difficult to do "by hand"
- One can use
  - Belief logics (e.g. BAN logic).
  - Theorem Proving.
  - Model Checking and alike.

42

Another aspect is the use of multiple protocols. Two protocols that are by themselves correct may became vulnerable/flawed when combined. A basic challenge response protocol using the public key of Bob, combined with a protocol that relies on secrecy of messages encrypted with this public key are obviously flawed when combined. To address this we should analyse all protocols that use the same keys together. Using different keys for the challenge response and the secrecy protocols would solve the problem while analysing them together would reveal the weakness so it can be addressed. (How would you address this weakness?)

7.2 Exercises

1. Consider again the online music store of the previous chapters. Review your requirements analysis, adding security protocol considerations; threats and countermeasures where appropriate.

2. Mutual authentication protocol:

   (a) A->B: A, K

   (b) B->A: {B, K, Nb}pk(A)

   (c) A->B: {Nb}K

   where A and B are agent identities, K is a fresh symmetric key and Nb is a nonce, pk(X) is the public key of agent X, and {M}pk(X) is message M encrypted with the public key of X using a public-key algorithm (as in message 2).

   The protocol above aims at mutual authentication of agents A and B. Authentication of B to A is not accomplished, since A is not really presenting a challenge to B.

   (a) Provide an attack that allows the intruder to impersonate agent B and fool A into thinking she's communicating with B

   (b) Provide a fix for the previous flaw, and explain why you think it avoids the mentioned vulnerability.

   (c) Which secrets, if any, are shared between A and B after the original protocol and after the fixed protocol?

   (d) Is authentication of A to B accomplished? (Explain your answer.)

   (e) Is the "B" really necessary in message 2. B->A: "B",K,Nbpk(A)? (Explain your answer.)