CRYPTOGRAPHIC PROTOCOLS AND KEY DISTRIBUTION

Applied Cryptography

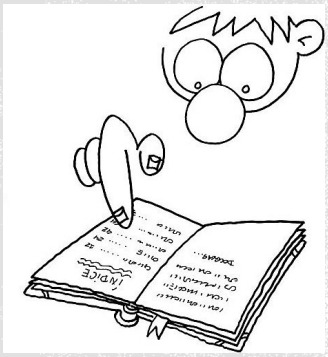
Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

CONTENTS

- Cryptographic protocols
  - A quick introduction
  - Examples of security issues from selected protocols
- Key distribution for symmetric ciphers
  - Needham-Schroeder protocol
  - Attacks and fixes
  - Kerberos
- Cryptographic protocols
  - BAN logic
  - Analysis of the Needham-Schroeder protocol



2

CRYPTOGRAPHIC PROTOCOLS: AN INTRODUCTION

WHAT IS A COMMUNICATION PROTOCOL?

- It is a system that allows two or more entities to exchange information
- It defines the rules, syntax, semantics and synchronization of communication and possible error recovery methods
- It can be implemented by hardware, software, or a combination of both
- It uses messages with well-defined formats (syntax) and each message has an exact meaning intended to elicit a response from a range of possible responses pre-determined for that particular situation (semantics)
  - **The specified behavior is typically independent of how it is to be implemented**
- Roughly, we can say that

protocols are to communication

what programming languages are to computation

WHAT IS A CRYPTOGRAPHIC PROTOCOL?

- It is a **communication protocol** (usually a small one, meaning that its specification is typically quite compact, e.g., few exchange of messages in given formats) **designed to secure communication** (various security goals) by **using cryptographic primitives** (e.g., ciphers, hash functions, …)
- Typical security goals/properties
  - **Secrecy**: May an intruder learn some secret message between the two honest participants Alice and Bob?
  - **Authentication**: Is the agent Alice really talking to Bob?
  - **Non-repudiation**: Alice sends a message to Bob. Alice cannot later deny having sent this message. Similarly, Bob cannot deny having received the message
- Cryptographic protocols are used everywhere: financial transactions, accessing online services, …

IMPORTANT REMARK ON ABSTRACTION (1)

- What is the exact meaning of the sentence "**The specified behavior [of a protocol] is typically independent of how it is to be implemented**" ?
- Preliminarily, observe that you really want to specify a protocol independently of its implementation because you want to abstract away a lot of details that are present in an implementation
  - Type of processors used by the entities involved in the protocol
  - Operating systems
  - Programming language
  - Libraries
  - … and many other details
- What we want is to characterize the "essence" of a protocol, i.e. what is the meaning of each message and how each entity should generate or react to a particular message without considering all the details listed above

- This is an instance of applying a very important engineering principle: separation of concerns
- First you focus on the essence of the cryptographic protocol, understand if it satisfies the security goal, and then consider the problem of implementing it

IMPORTANT REMARK ON ABSTRACTION (2)

- We are left with the problem of characterizing the "essence" of a protocol and come up with a mean to unambiguously define its syntax and semantics
- The problem admits at least two possible solutions that have in common one feature, namely that the syntax and semantics of (cryptographic) protocols together with their security goals can be precisely described by using appropriate mathematical objects
- In this way, it is possible to prove (in a mathematical way) that a certain protocol achieves a security goal under the assumption that a characterization of the attacker capabilities is also included in the specification of the protocol
- Typically, the attacker capabilities that are considered in the context of cryptographic protocols are the following: an (active) attacker can
  - intercept all messages sent on the network
  - compute messages
  - send messages on the network

Roughly speaking we can say that this kind of attacker coincides with the network

IMPORTANT REMARK ON ABSTRACTION (3)

- **Formal** or **Dolev-Yao** model (developed in 1983) assumes the following
  - The cryptographic primitives are black-boxes
  - The messages are expressions built out of the cryptographic primitives
  - Examples
    - $\{m\}k$ denotes the ciphertext obtained by encrypting the plaintext m with the key k
    - $(m1,m2)$ is the pairing of messages $m1$ and $m2$
  - The attacker is assumed to be able to compute only using the cryptographic primitives
- In this model, we say that cryptographic primitive cannot be broken or, equivalently, we assume that **cryptography is perfect**
- Needless to say this is an abstraction and maybe a coarse abstraction that ignores all the weaknesses that may be existing in available cryptographic primitives
- This implies that a proof of the security of a certain protocol in this model holds under the assumption that the attacker does not attempt to break the cryptographic primitives and, of course, that the protocol implementation correctly implements its specification

IMPORTANT REMARK ON ABSTRACTION (4)

- **Computational** model (developed beginning of the 80's) assumes the following
  - The messages are bit-strings
  - The cryptographic primitives are functions on bit-strings
  - The attacker is any probabilistic (polynomial-time) Turing machine
- The main difference with the Dolev-Yao model is that cryptographic primitives are no more perfect but are functions operating on bit-strings which is a less coarse abstraction; however, notice that it is still an abstraction since bit-strings are mathematical objects that are quite different from the bit-strings datatypes that we may find in programming languages
- A probabilistic (polynomial-time) Turing machine characterizes the capabilities of a computationally constrained attacker that can solve problems using randomized polynomial time algorithms including those required to break a certain cryptographic primitive

A **probabilistic Turing machine** is a non-deterministic Turing machine that chooses between the available transitions at each point according to some probability distribution (roughly by tossing a coin at each step in the computation)

IMPORTANT REMARK ON ABSTRACTION (5)

- Despite being more realistic than the Dolev-Yao model, the computational model still ignores several details that should be considered when implementing a cryptographic protocol including **side channel attacks** such as
  - timing
  - power consumption
  - noise
  - physical attacks against smart cards

  which can give additional information
- A side-channel attack aims to gather information from or influence the execution of a system by measuring or exploiting indirect effects of the system or its hardware rather than targeting the program or its code directly
  - In other words, side channel attacks do not exploit weaknesses in the implemented system but rather gain information from the execution of the system itself
  - Example: measuring power consumption may give hints on which bits is being computed by a circuit

IMPORTANT REMARK ON ABSTRACTION (6)

- If the computational model is more precise than the Dolev-Yao, why do we need to bother with the latter*?*
- There are several good reasons to keep considering the Dolev-Yao model
  - Security proofs are easier than those in the computational model
  - Several attacks are already found while considering the simpler model: this is so because the **logic underlying protocol exchanges is inherently difficult to get right**
    - This aspect is summarized in the abstract of the paper "Programming Satan's Computer"

Cryptographic protocols are used in distributed systems to identify users and authenticate transactions. They may involve the exchange of about 2–5 messages, and one might think that a program of this size would be fairly easy to get right. However, this is absolutely not the case: bugs are routinely found in well known protocols, and years after they were first published. The problem is the presence of a hostile opponent, who can alter messages at will. In effect, our task is to program a computer which gives answers which are subtly and maliciously wrong at the most inconvenient possible moment. This is a fascinating problem; and we hope that the lessons learned from programming Satan's computer may be helpful in tackling the more common problem of programming Murphy's.

- The paper is available at https://www.cl.cam.ac.uk/~rja14/Papers/satan.pdf

IMPORTANT REMARK ON ABSTRACTION (7)

- An attack in the Dolev-Yao model implies a (practical) attack in the computational model
- A proof in the Dolev-Yao model does not always imply a proof in the computational model

- The Dolev-Yao model allows for automated verification
- Most proofs in the computational model are manual although recently tools to partially automate the reasoning tasks have been made available

- Why is it considered important to at least partially automate the proof process*?*
  - *Hint*: how much manual attempts are expensive*?* how much expertise is needed*?*

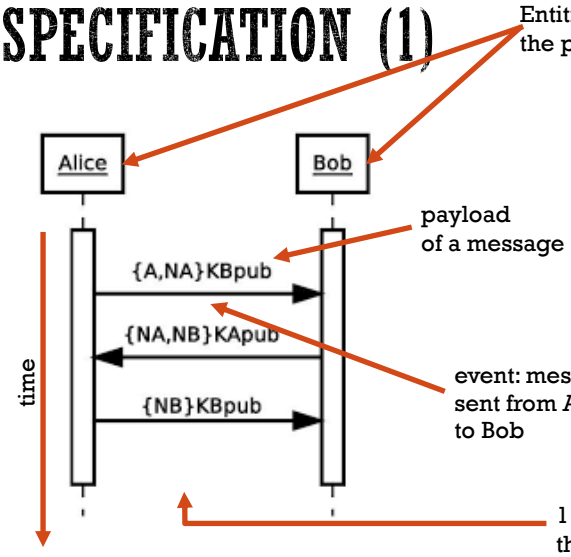
IN THIS SLIDE DECK

- A replay attack is when valid data transmission is maliciously repeated or delayed
- It can be seen as a very simple Men-In-The-Middle attack in which messages are intercepted but not altered as in the general case

- We are going to consider several different cases in which cryptographic protocols failed to achieve the security goals for which they were designed
- First, we consider failures derived from weaknesses of symmetric cryptographic primitives (RC4 and 3DES) or weakness in the design of the protocol (replay attack made practical by using AES in CBC model for ensuring integrity)
  - These can be seen as considering the protocols in the Computational model
- Then, we consider a very important class of cryptographic protocols (for key exchange) together with the possible attacks that can be mounted by exploiting weaknesses in the logic used to design the exchange of messages
  - These can be seen as considering the protocols in the Dolev-Yao model
  - We will also discuss some difficulties in implementing some of the protocols
- All the above is discussed informally
- In the last part, we consider how the reasoning in the Dolev-Yao model can be made precise by using formal logic

A MESSAGE SEQUENCE DIAGRAM OF A PROTOCOL SPECIFICATION (1)

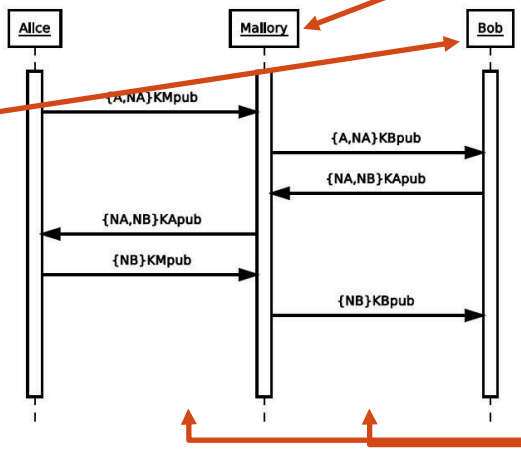
Reading (order matters)
1. Alice sends a message to Bob whose payload is the result of encrypting the pair consisting of the identifier A and the nonce NA with the key Kbpub
2. Bob sends a message to Alice whose payload is the result of encrypting the pair consisting of the noncences NA and NB with the key Kapub
3. Alice sends a message to Bob whose payload is the result of encrypting the nonce NB with the key KBpub



A very simple cryptographic protocol for key exchange

A MESSAGE SEQUENCE DIAGRAM OF A PROTOCOL SPECIFICATION (2)

Malicious entity interfering with the protocol



Honest entities involved in the protocol

An attack to the very simple cryptographic protocol for key exchange

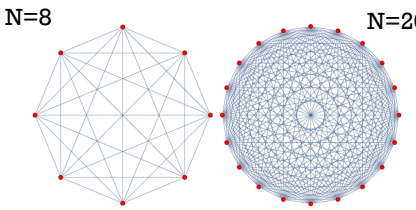
The attack by Mallory consists of pretending to be Bob with Alice and Alice with Bob during key exchange… afterwards all messages encrypted with the 2 exchanged keys can be easily decrypted by Mallory….

2 sessions of the protocol

37

THE PROBLEM OF KEY DISTRIBUTION FOR SYMMETRIC CIPHERS

KEY DISTRIBUTION



Possible pairs
N(N-1)/2

- Assume that a large number of people, processes, or systems that want to communicate with one another in a secure fashion
- Assume that this group of people/processes/systems is not static, i.e. individual entities may join or leave the group at any time
- **Naïve solution:** point-to-point key establishment
  - Each party physically exchange an encryption key with every one of the other parties
  - Afterwards, any two parties would be able to establish a secure communication link using the encryption key they possess for each other
- This solution is obviously **not feasible** for **large groups** of people/processes/systems, especially when **group membership evolves over time**

SESSION/EPHEMERAL KEY (1)

- Parties know each other (e.g., a client A has an account on server B)
- A and B a priori share a long term key W
- A and B want to establish a session key K

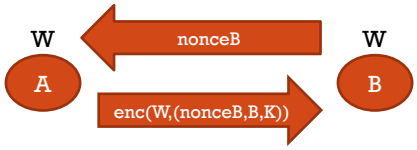
- Session key K is used for a communication session
- Session key K is used for bulk encryption
- Long term key W is used for key establishment

SESSION/EPHEMERAL KEY (2)

A **pseudorandom function** is an efficiently computable function emulating a random oracle (a function whose outputs are fixed completely at random)

- How to derive the session key K from the long term key W?

- **Unilateral challenge-response protocol**
  - A ← B : nonceB
  - A → B : enc(W, (nonceB, B, K))



- **Bilateral challenge-response protocol**
  - A ← B : nonceB
  - A → B : enc(W, (KA,nonceB,nonceA,B))
  - A ← B : enc(W,(KB,nonceA,nonceB,A))
  - A : K=kdf(KA,KB) and B : K=kdf(KA,KB)

kdf = key derivation function
A cryptographic hash function that derives one or more secret keys from a secret value such as a main key, a password, or a passphrase using a pseudorandom function

POINT-TO-POINT KEY ESTABLISHMENT

- PROS
  - **Security:** if a subject is compromised only its communications are compromised; communications between two other subjects are not compromised

- CONS
  - **Poor scalability**: the number of keys is quadratic in the number of subjects
  - **Poor handling of dynamic topologies**: a new member's joining and a member's leaving affect all current members

FOR SMALL NETWORKS…

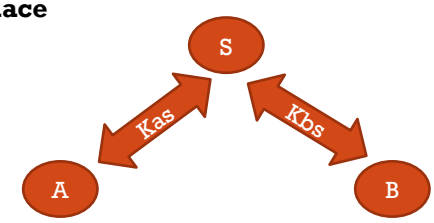
- … it is possible to store at every node of a network the "master" keys needed for communicating privately with each of the other N nodes in a network
  - Each node will store N − 1 such keys
- If the messages exchanged over the network are short, one may use these keys directly for encryption
- When the messages are of arbitrary length, a node A in the network can use the master key for another node B to first set up a session key and subsequently use the session key for the actual encryption of the messages
- For large networks this becomes impractical and we need to design an alternative solution based on a trusted 3rd party that mediates the establishment of a secure connection…

KEY DISTRIBUTION CENTER (KDC)

- A more efficient alternative
  - Provide every group member with a single key, called the **master key**, for securely communicating with a key distribution center (KDC)
  - When A(lice) wants to establish a secure communication link with B(ob)…
  - … A(lice) requests a session key from KDC for communicating with B(ob)
- The secure implementation of this idea requires to address the following issues
  - Assuming that A is the initiator of a session-key request to KDC, when A receives a response from KDC, **how can A be sure that the sending party for the response is indeed the KDC?**
  - Assuming that A is the initiator of a communication with B, **how does B know that some other party is not masquerading as A?**
  - **How does A know that the response received from B** is indeed from B and not from someone else masquerading as B?
  - **What should be the lifetime of the session key acquired by A for communicating with B?**

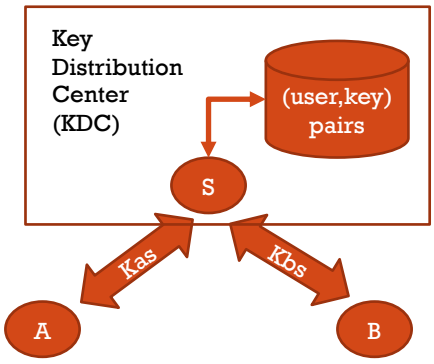
NEEDHAM-SCHROEDER KEY DISTRIBUTION

- A party A wants to establish a secure communication link with another party B
- Both A and B possess master keys Kas and Kbs, respectively, for communicating privately with a key distribution center (KDC)
- Needham-Schroeder is a **shared-key authentication protocol** designed to generate and propagate a session key, i.e., a shared key for subsequent symmetrically encrypted communication
- **There is no public key infrastructure in place**
- *Assumption*: both A and B have a shared communication key with S
- It is crucial that the key shared between A and S (Kas) is protected otherwise anyone can impersonate A (similarly for B)



KEY DISTRIBUTION AS TRUSTED 3RD PARTY

- Let S(erver) be a trusted 3rd party
- S allows pair of users to establish a session key
- Each user shares a long-term key, a priori with S
- The overall **number of long-term keys is linear in the number of users**
- S maintains a database containing pairs associating an identifier of a user and the key shared between the user and S
  - Examples: (A,Kas) and (B,Kbs)
- It guarantees the integrity and confidentiality of the database
- It is a **honest participant of the key distribution protocol**



Problem: how can keys stored in the KDC be protected?

NEEDHAM-SCHROEDER IN A NUTSHELL

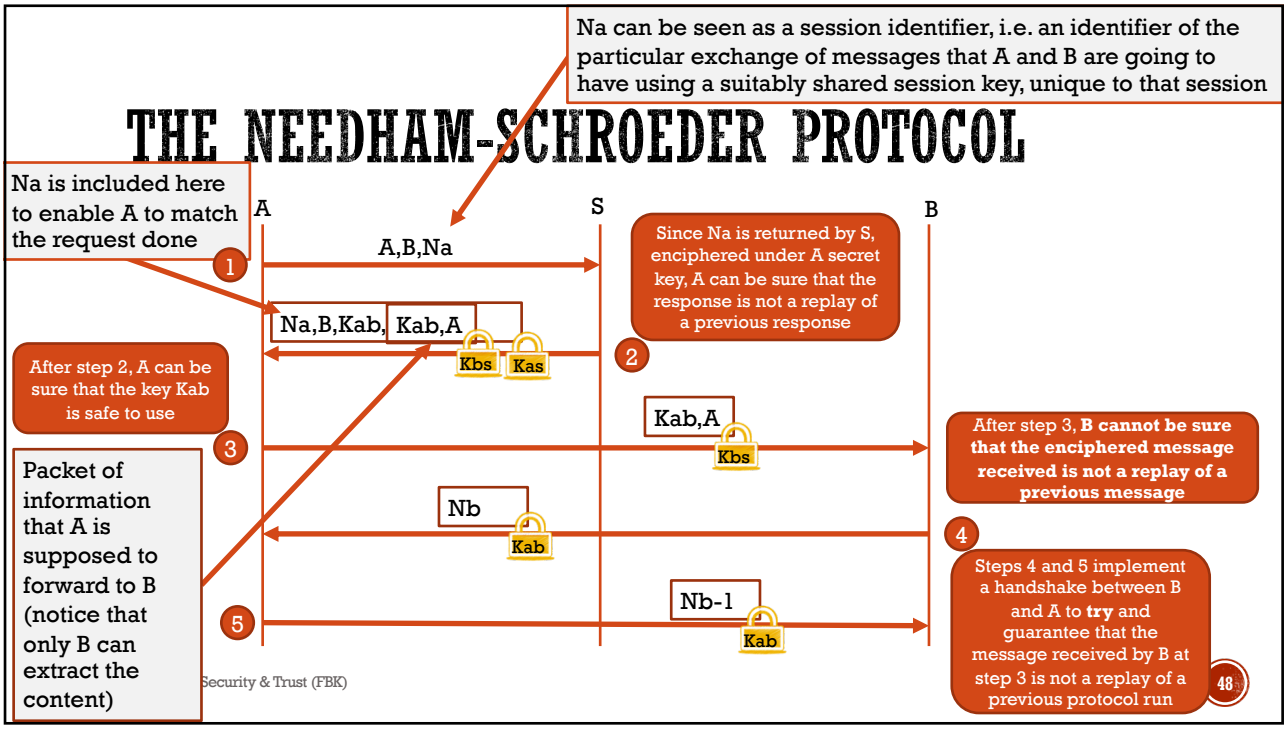
- **Assumption**: each user A has a (secret) key Kas which is only known to A and S
- If two users wish to establish a secure communication channel, one of them obtains a secret communication key from S and gives a copy to the other
- If a new key is obtained for each message exchange, users need not maintain a list of secret communication keys for all their correspondents
- Protocol specification (1st version)
  - A→S: (A,B,Na)
  - S →A: {(Na,B,Kab,{(Kab,A)}Kbs}Kas
  - A→B: {(Kab,A)}Kbs
  - B→A: {Nb}Kab
  - A→B: {Nb-1}Kab
- **What are Na and Nb?**

- X→Y:M X sending message M to Y
- (M1,M2) pairing of M1 and M2
- {M}k ciphertext obtained from plaintext M with key k

NEEDHAM-SCHROEDER AND NONCES

- Needham-Schroeder uses **nonces** (**numbers used once**),
  - Nonces are randomly generated values included in messages
- If a nonce is generated and sent by A in one step and returned by B in a later step, **A knows that B's message is fresh and not a replay from an earlier exchange**
- A nonce is not a timestamp
- The only assumption is that **it has not been used in any earlier interchange**

THE NEEDHAM-SCHROEDER PROTOCOL



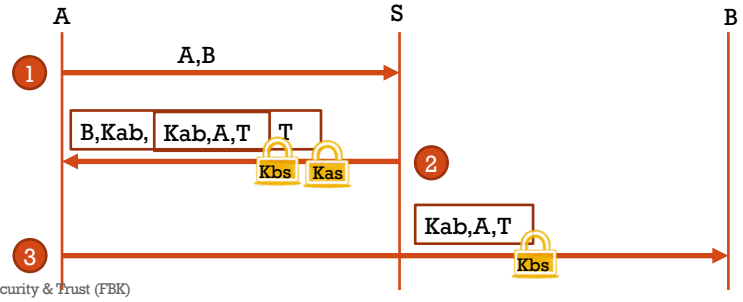
POSSIBLE REPLAY ATTACK

- **Denning** and **Sacco** pointed out that the compromise of a session key has bad consequences
- An intruder can reuse an old session key and pass it off as a new one as though it were fresh
- Suppose an attacker C
  - has cracked Past_Kab from last week's run of the protocol or was able to steal it by breaching either A or B and
  - is able to intercept and block message 3 from the current run of the problem session
- Then C can send { Past_Kab, A } Kbs to B as message 3, i.e. C→B: { Past_Kab, A } Kbs
  - ***At this point B believes it is receiving a message from A!***
- Thinking that A has initiated a new conversation, B requests a handshake from A, i.e. B→A: {Nb'} Past_Kab
- C intercepts the message, deciphers it, and impersonates A's response, i.e. C→A: {Nb'-1} Past_Kab
- From there on, C can send malicious messages to B pretending to be A

FIXING NEEDHAM-SCHRODER

- As long as dt1 + dt2 is less than the interval since the last use of the protocol, this method will protect against replays
- Since the timestamp T is enciphered under the private keys Kas and Kbs, impersonation of the S is impossible

- The replay attack has been fixed in the **Kerberos** protocol that we will see below
- The idea is to use **timestamps** (under the assumption that private keys are secure)
  - Replace the handshake (steps 4 and 5) by adding a timestamp to steps 2 and 3 as follows:



Problem and fix were discussed by Denning and Sacco in 1981

Both A and B can check $|Clock - T| < dt1 + dt2$ where

- Clock gives the local time
- dtl is an interval representing the normal discrepancy between the server's clock and the local clock
- dt2 is an interval representing the expected network delay time

SOME REMARKS (1)

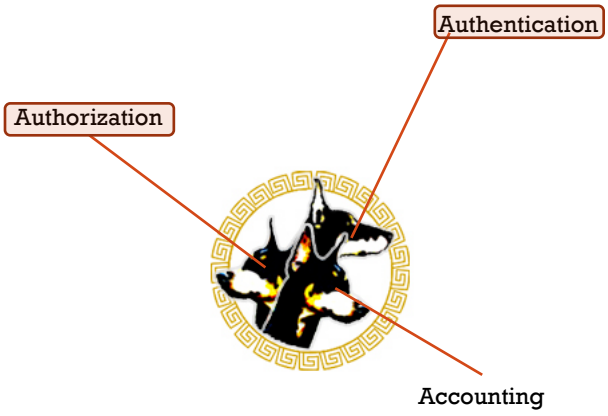
- For very large networks or network of networks, it is **not practical to have a single KDC service**
- **Better to have KDCs organized hierarchically**, with each local network serviced by its own KDC, and a group of networks serviced by a more global KDC, and so on
- A local KDC would distribute the session keys for secure communications between users/processes/systems in the local network
- **When a user/process/system desires a secure communication link with another user/process/system in another network, the local KDC would communicate with a higher level KDC and request a session key for the desired communication link**
- Such a hierarchy of KDCs simplifies the distribution of master keys
- A KDC hierarchy also limits the damage caused by a faulty or subverted KDC

SOME REMARKS (2)

- How can we discover if a cryptographic protocol achieves the security goals (e.g., authentication of principals) that it has been designed for?
- How can we assess that attackers cannot violate the security goals under the assumption that they have certain capabilities?
- We will see possible answers to these questions after discussing the Kerberos protocol based on Needham-Schroeder key exchange…

53

KERBEROS



- http://web.mit.edu/kerberos/
- http://web.archive.org/web/20071219054639/http://www.ietf.org/rfc/rfc4120.txt

54

KERBEROS USE CASE SCENARIO: PRINTING

CUPS

https://www.cups.org/

- IPP is a specialized Internet protocol for communication between client devices (computers, mobile phones, tablets, etc.) and printers (or print servers)
- It allows clients to submit one or more print jobs to the printer or print server, and perform tasks such as querying the status of a printer, obtaining the status of print jobs, or cancelling individual print jobs

- CUPS is the standards-based, open source printing system developed by Apple Inc. for macOS and other UNIX-like operating systems
- CUPS uses the Internet Printing Protocol (IPP) to support printing to local and network printers
- **Scenario**
  - A university computer network wants to provide printer services to its students
  - Printers are located at certain designated locations on the campus
  - Each student gets a "printer budget" on a semester basis
  - A student is allowed a certain number of free pages
  - When a student has used up his/her printer budget, he/she is expected to deposit money for additional pages
  - Printers are connected to machines (called "printer servers") that run the CUPS software

SCENARIO: SECURITY REQUIREMENTS

- When a print request is received, a printer server must first **authenticate the client**
  - Not all the client hosts on the campus may be authorized to send jobs to a printer
- A printer server must also **validate the print request received against the print budget** for the student who sent the request
- A printer server must enable a **confidential communication** link directly from the host where the print request originates to the printer in question
  - I.e. we do not want a student to send his/her job to the printer in plaintext
- A printer server would not want to route all the print jobs through itself since that would unnecessarily degrade the performance of the server

SCENARIO: MAIN SECURITY ISSUES

- Establish a **direct authenticated and confidential communication link between the host (**where a print request originates) **and the printer**
- Since **printers** generally are **rudimentary** when it comes to general purpose computing, you may **not expect** a printer to contain all of the **software** that generally is required (such as the SSL/TLS libraries) **for establishing such links**
- One certainly would **not want the students to establish password based connections with the printers** (for authentication) since such passwords are likely to be transmitted in **clear text over a network**

ALL MAIN SECURITY ISSUES ARE SOLVED BY DEPLOYING KERBEROS

- There is no reason to transmit passwords in clear text or otherwise since, as in the Needham-Schroeder protocol, Kerberos operates on the principle of shared secret keys
- If Kerberos is enabled in the CUPS software on the printer server, when adding a client host to the group of hosts allowed to send print jobs to the printers, a secret key (like the master keys in the Needham-Schroeder protocol) is created that will be shared by the client and the printer server
- The printer server will also possess a shared secret key for communicating with each of the printers it is in charge of
- Through the Kerberos protocol, the printer server will bring into existence a secret session key that would allow a student's laptop to send a print job directly to the printer over an encrypted link

KERBEROS IN A NUTSHELL (1)

- Network authentication protocol developed at MIT in (mid) 1980s
- Windows 2000 and later uses it as the **default authentication mechanism**
- **Purpose**: allow users and services to *authenticate* themselves to each other
  - I.e. allow them to demonstrate their identity to each other
- Many ways to establish one's identity to a service
  - Most familiar is the **user password**
  - When translated to a network, this approach has additional problem
    - The password must travel over that network in clear (i.e. unencrypted)
    - Anyone listening in on the network can intercept it, and use it to **impersonate** the legitimate user

KERBEROS IN A NUTSHELL (2)

- Key idea underlying Kerberos
  - Password can be viewed as a special case of a **shared secret**--something that the user and the service hold in common, and which (again ideally) only they know
  - Establishing identity should not require the user to actually reveal that secret by sending it over the network
- **Question**
  - how a user can prove it knows the password without sending it to the server?
- **Answer**
  - The shared secret is used as an encryption key
- **Simplest case**
  - user takes something freshly created, like a nonce (it need not be secret), encrypts it with the shared secret key and sends on to the service
  - service decrypts it with the shared key and recovers the nonce
- If the user used the wrong key, the nonce will not decrypt properly, and the service can reject the user's authentication attempt
- In no event does the user or the service reveal the shared key in any message passed over the network

WHICH CRYPTOGRAPHY IS USED?

- Kerberos, as defined in **RFC 4120**, uses **only symmetric cryptography**
  - So, there is only one key, which is shared by two endpoints
  - Kerberos supports **DES, AES, RC4**

https://www.ietf.org/rfc/rfc4120.txt

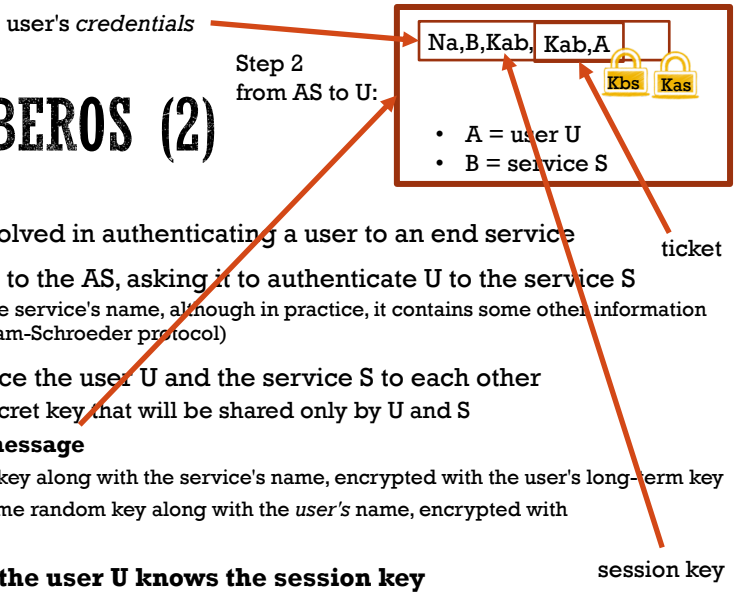
- Kerberos, by default, does not use public-key cryptography, but **RFC 4556** adds public-key cryptography to **the initial authentication phase**
  - More on this point, later…

http://www.ietf.org/rfc/rfc4556.txt

BASICS OF KERBEROS (1)

- Create a service whose sole purpose is to authenticate
- By doing this, it frees other services from having to maintain their own user account records
- Both users and services implicitly trust the Kerberos **Authentication Server (AS)** that mediates their interaction
- For this to work, both the **user and the service must have a shared secret key registered with the AS**
  - Such keys are called *long-term keys*, since they last for weeks or months

BASICS OF KERBEROS (2)

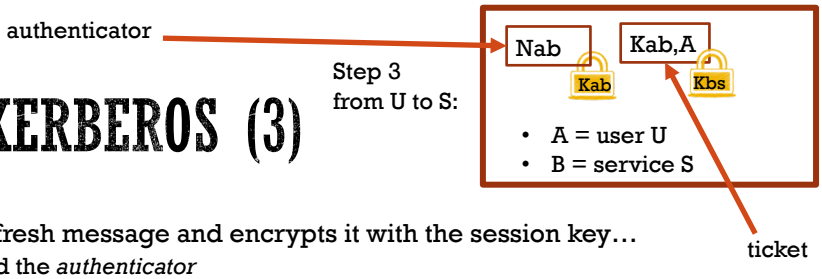


There are three basic steps involved in authenticating a user to an end service

1. The user U sends a request to the AS, asking it to authenticate U to the service S
   - This request consists only of the service's name, although in practice, it contains some other information (recall the nonce in the Needham-Schroeder protocol)
2. The AS prepares to introduce the user U and the service S to each other
   - It generates a new, random secret key that will be shared only by U and S
   - It sends the user a **two-part message**
     - One part contains the random key along with the service's name, encrypted with the user's long-term key
     - The other part contains that same random key along with the *user's* name, encrypted with the *service's* long-term key

- Notice that at this point, **only the user U knows the session key**
  - Provided it really is the user U and knows the appropriate long-term key

BASICS OF KERBEROS (3)



3. User U generates a fresh message and encrypts it with the session key…
   - This message is called the *authenticator*

   … then sends the authenticator, along with the ticket, to the service
   - The service S decrypts the ticket with its long-term key to recover the session key
   - In turn, the session key is used to decrypt the authenticator
- The service S trusts the AS, so it knows that only the legitimate user could have created such an authenticator
- This completes the authentication of the user to the service
- If the user U want the service S to be authenticated in return, then
  - The service S takes Nab from the authenticator, adds the service's own name to it, and encrypts the whole message with the session key
  - This is then returned to the user U

TICKET GRANTING SERVER (1)

- One of the inconveniences of using a password is that each time you access a service, you have to type it in
  - It can be a tremendous nuisance, if you have to access a variety of different services, and so the temptation is to use the same password for each service, and further to make that password easy to type
- Kerberos eliminates the problem of having passwords for each of many different services, but there is still the temptation of making the one password easy to type
- Kerberos resolves the last problem by introducing a new service, called the *ticket granting server* (TGS)
- Logically, TGS is distinct from the AS, although they may reside on the same physical machine
- They are often referred to collectively as the KDC--the Key Distribution Center, from Needham and Schroeder
- The purpose of the TGS is to add an extra layer of indirection so that the user only needs to enter a password once
  - The ticket and session key obtained from that password are used for all further tickets

TICKET GRANTING SERVER (2)

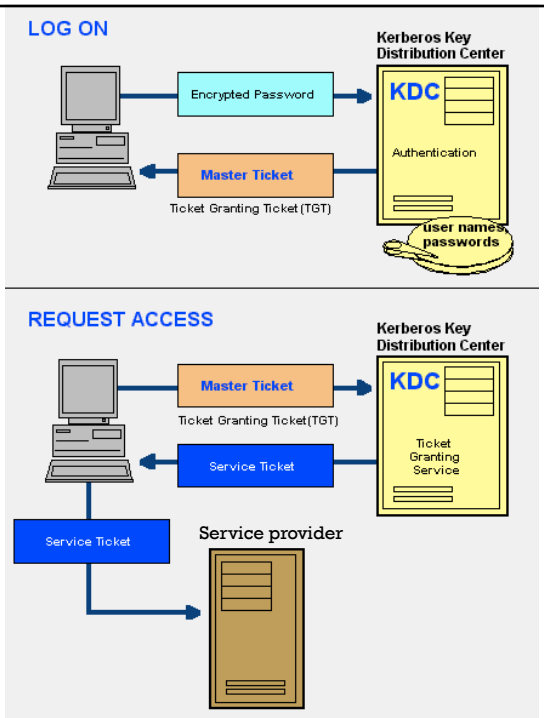
- Before accessing any regular service, the user requests a ticket from the AS to talk to the TGS
- This ticket is called the *ticket granting ticket* (TGT), also called the *initial/master ticket*
- The session key for the TGT is encrypted using the user's long-term key, so the password is needed to decrypt it from the AS's response to the user
- After receiving the TGT, any time that the user wishes to contact a service, he requests a *service ticket* not from the AS, but from the TGS
- The reply is encrypted not with the user's secret key, but with the session key that came with the TGT, so the user's password is *not* needed to obtain the new session key (the one that will be used with the end service)

Similar to visiting some workplaces

- One shows regular ID once to get a guest ID
- When willing to enter various rooms in the workplace, instead of showing your regular ID over and over again, which might make it vulnerable to being dropped or stolen, ones show the guest ID, which is only valid for a short time
- If a guest ID were stolen, one could get it invalidated and be issued a new one quickly and easily, which is much more difficult to do with your regular ID

- Passwords usually remain valid for months
- The TGT is valid only for a fairly short period, typically 8/10 hours
- Afterwards, the TGT is not usable by anyone, including the user or any attacker

66



The KDC is split in two parts

- one devoted to client authentication (LOG ON)
- the other provides security to the service providers (REQUEST ACCESS)

A client cannot gain direct access to the Ticket Granting Service and only the latter can provide a session key to communicate with a service provider

The bridge/indirection between the Authentication Service and the Ticket Granting Service is the Master Ticket.
This also avoids to repeatedly ask users to enter their passwords

67

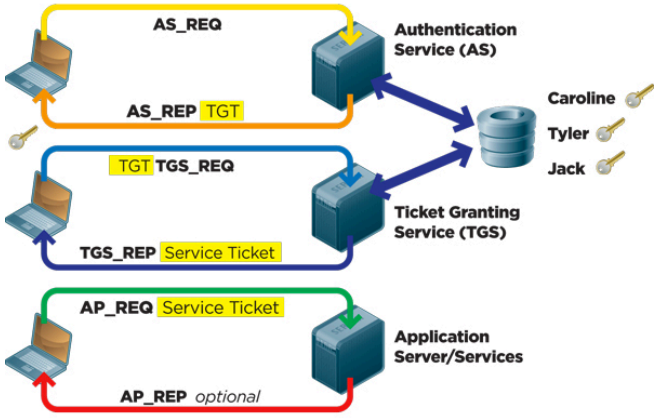
CROSS-REALM AUTHENTICATION

- So far, we have considered the case where there is a single AS and a single TGS
- As the network grows, the number of requests grows with it, and the AS/TGS becomes a bottleneck in the authentication process
  - The system does not *scale*
- It often makes sense to divide the world into distinct *realms* (often) corresponding to organizational boundaries
  - Each realm has its own AS and TGS
- To allow for cross-realm authentication (i.e. to allow users in one realm to access services in another), it is necessary first for the user's realm to register a *remote* TGS (RTGS) in the service's realm
- This now adds a new layer of indirection to the authentication procedure
  - First the user contacts the AS to access the TGS
  - Then the TGS is contacted to access the RTGS
  - Finally, the RTGS is contacted to access the end service

KERBEROS AND PUBLIC KEY CRYPTO

- When using symmetric key cryptography, for a user to use Kerberos, it must already be registered with a KDC
- Such a requirement can be circumvented with the use of public-key cryptography
- When the KDC (that is, the AS) generates its response, encapsulating the session key for the TGT, it does not encrypt it with the user's long-term key (which does not exist)
- Rather, it encrypts it with a randomly generated key, which is in turn encrypted with the user's public key
- The only key that can reverse this public-key encryption is the user's private key, which only it knows
- The user thus obtains the random key, which is in turn used to decrypt the session key, and the rest of the authentication can proceed as before

KERBEROS AUTHENTICATION IN BRIEF



71

KERBEROS: DETAILS

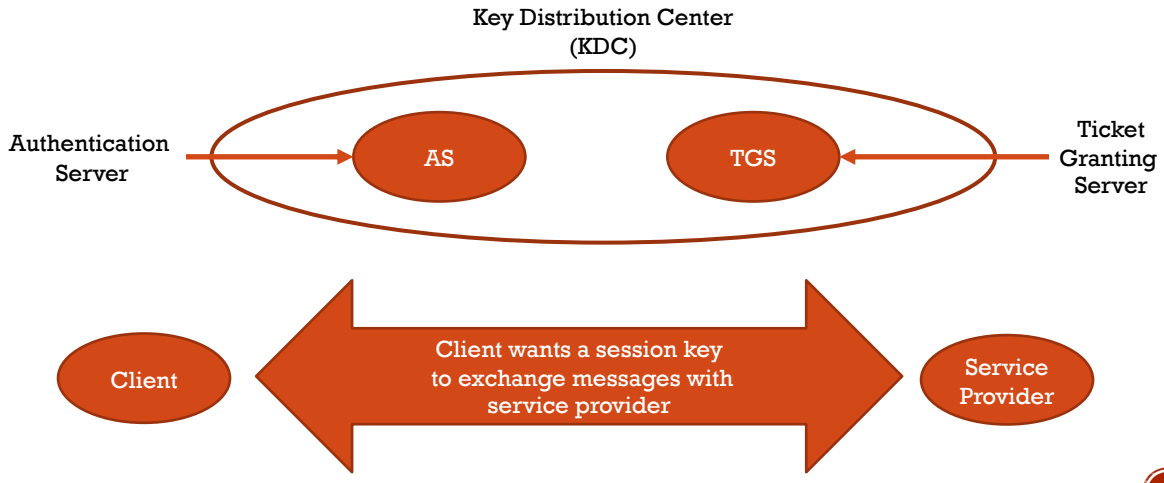
OVERVIEW

- The Kerberos protocol provides security for client-server interactions in a network
  - Examples of servers are printer servers, database servers, news servers, FTP servers, …
- The main difference between the Needham-Schroeder protocol and the Kerberos protocol is that the latter makes a distinction between the clients, on the one hand, and the service providers, on the other
  - No such distinction is made in the Needham-Schroeder protocol
- Another difference is that in the Kerberos protocol, the Key Distribution Center (KDC) is divided into two parts
  - one devoted to client authentication, called **Authentication Server (AS)**
  - the other in charge of providing security to the service providers, called **Ticket Granting Server (TGS)**

IMPORTANT REMARK

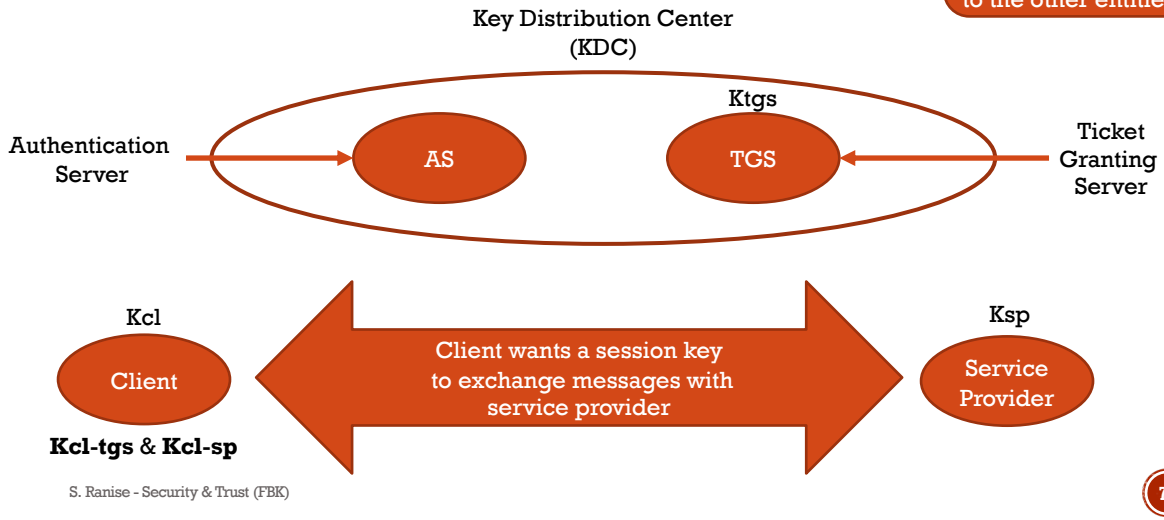
- A client cannot gain direct access to TGS and only the TGS can provide a session key to communicate with a service provider
- A client must first authenticate to AS and obtain from AS a session key for accessing TGS
- Consider the printing scenario
  - Separating the AS from the TGS and establishing an indirection between the two allows for **separation of concerns**
  - AS can concern itself exclusively with matters related to user authentication, which would include keeping up-to-date with who is allowed to use which printers
  - In a large organization (like a university)
    - The database of users and which printers and other resources the users are allowed to access is bound to be updated very frequently as new students join the university, graduating students leave, …
  - TGS can concern itself exclusively with issues related directly to the printers that include keeping track of the current load on each printer so that a user wanting access to a printer that already has too many jobs in its queue could be warned; etc

OVERVIEW OF BASIC NOTIONS (1)



OVERVIEW OF BASIC NOTIONS (2)

Notice that all keys are stored or generated by the AS and distributed to the other entities!

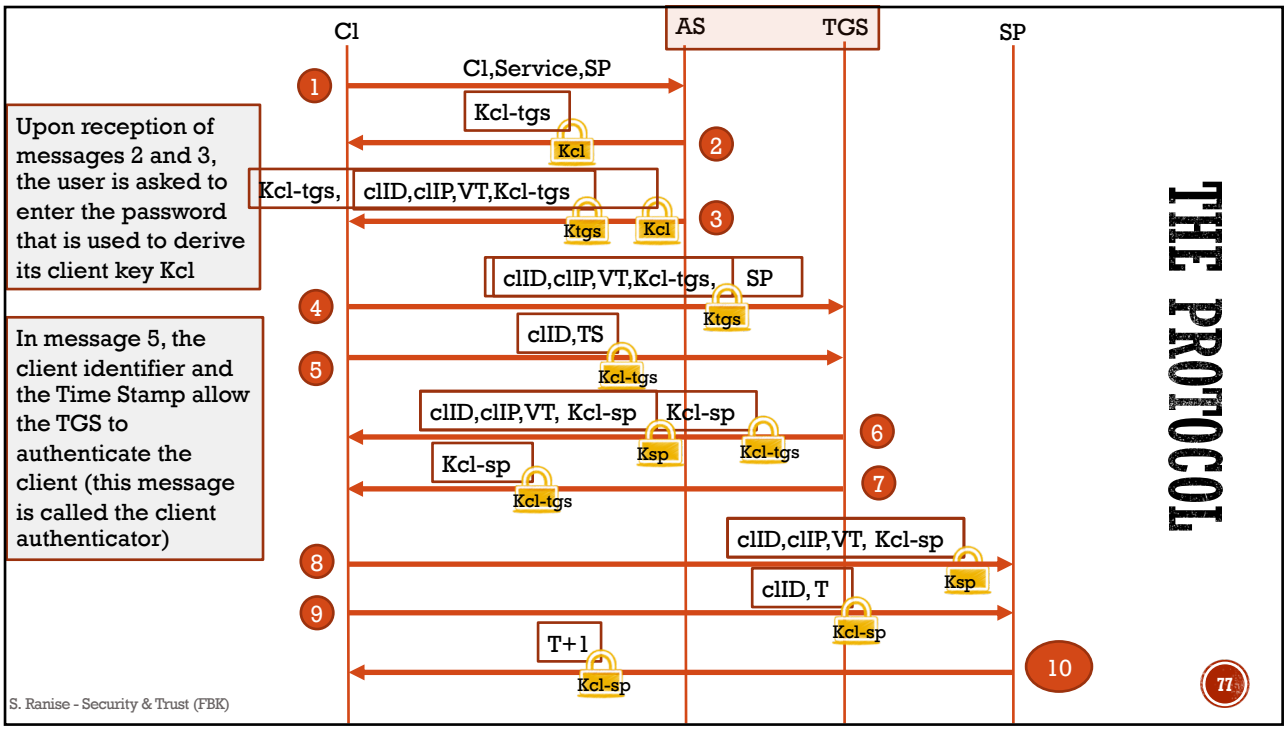


OVERVIEW OF BASIC NOTIONS (3)

Key	Description
Kcl	secret key held by AS for the Client
Ktgs	• secret key held by AS for TGS • TGS also has this key
Ksp	• secret key held by AS for the Service Provider (SP) • SP also has access to this key
Kcl-tgs	**session key** that AS send to the Client for communicating with TGS
Kcl-sp	**session key** that TGS send to the Client for communicating with the SP

Each client and service register with the Authentication Server and are granted an identity and a secret password

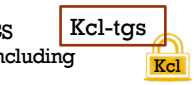
THE PROTOCOL



THE STEPS OF THE PROTOCOL (1)

The TGT is also called the **initial ticket** since it enable the client to subsequently obtain client-to-SP tickets from TGS

- Step 1
  - Client sends a request in plain text to the AS
  - This request is for a service that the Client expects from the SP
- Steps 2 and 3
  - AS sends back to the Client the following two messages encrypted with Kcl
  - In the database maintained by AS, Kcl is specific to the Client and will remain unchanged as long as the client does not alter its password
  - **This encryption key is not directly known to the Client**
  - The two messages are
    2. A session key Kcl-tgs that the client can use to communicate with TGS
    3. A **Ticket-Granting Ticket (TGT)** that is meant for delivery to TGS including
       - client's user ID (clID)
       - client's network address (clIP)
       - validation time (VT)
       - same Kcl-tgs session key
    - The ticket is encrypted with the KT GS secret key that the AS server maintains for TGS

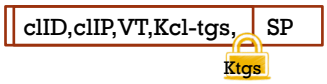


THE STEPS OF THE PROTOCOL (2)

- **After step 3 and before step 4**
  - When the client receives the messages at steps 2 and 3, it enters its password into a dialog box
  - An algorithm converts password into what would be the Kcl encryption key if the password is correct
  - The password is immediately destroyed and the generated key used to decrypt the messages received from AS
  - The decryption allows the Client to extract
    - the session key Kcl-tgs
    - the ticket meant for TGS from the information received from AS

THE STEPS OF THE PROTOCOL (3)

- Steps 4 and 5
  - The client sends the following two messages to TGS:
    4. The encrypted ticket meant for TGS followed by the ID of the requested service
    5. A Client Authenticator that is composed of the client ID and the timestamp, both encrypted with the Kcl-tgs session key
- **Before steps 6 and 7**
  - TGS recovers the ticket from message 4 above
  - From the ticket, it recovers the Kcl-tgs session key
  - The TGS then uses the session key to decrypt message 5 above that allows it to authenticate the Client



THE STEPS OF THE PROTOCOL (4)

- Steps 6 and 7
  - TGS sends back to the Client the following two messages
    6. A Client-to-Service Provider ticket that consists of
       - the Client ID (clID)
       - the Client network address (clIP)
       - the validation time (VT)
       - a session key Kcl-sp for the Client and the Service Provider encrypted with the Ksp key that is known to TGS
    7. The same Kcl-sp session key encrypted with the Kcl-tgs session key

THE STEPS OF THE PROTOCOL (5)

- **Before steps 8 and 9**
  - Client recovers the ticket meant for the service provider with the Kcl-tgs session key

- **Steps 8 and 9**
  - The client next sends the following two messages to the service provider:
    8. The Client-to-ServiceProvider ticket that was encrypted by TGS with the Ksp key

    9. An authenticator that consists of the Client ID and the timestamp, encrypted with the Kcl-sp session key

THE STEPS OF THE PROTOCOL (6)

- **Before step 10**
  - SP decrypts the ticket with its own Ksp key
  - It extracts the Kcl-sp session key from the ticket
  - Then uses the session key to decrypt message 9 received from the client

- **Step 10**
  - SP sends to the Client a message that consists of the timestamp in the authenticator received from the Client plus one, encrypted with Kcl-sp

THE STEPS OF THE PROTOCOL (7)

- **After step 10**
  - The client decrypts the message received from the Service Provider using the Kcl-sp session key and makes sure that the message contains the correct value for the timestamp
  - If that is the case, the client can start interacting with the Service Provider
- **Scenario of the printing service at a university campus**
  - It is the Kcl-sp key that allows a student's laptop to send his/her job directly to a printer over an encrypted connection
- **Important observation**
  - An additional advantage of separating AS from TGS (although they may reside in the same machine) is that the **Client needs to contact AS only once** for a Client-to-TGS ticket and the Client-to-TGS session key
  - These can subsequently be used for multiple requests to the different service providers in a network
  - This is an example of a **Single-Sign-On** experience

85

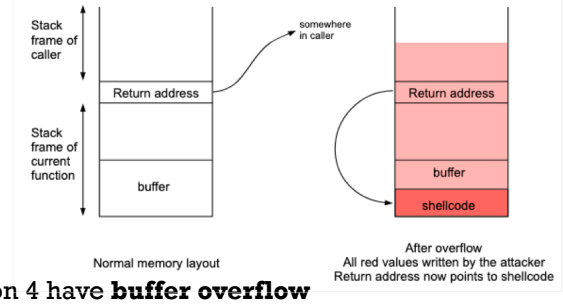
KERBEROS SECURITY ISSUES

COMPROMISE OF KERBEROS INFRASTRUCTURE (1)

- The primary way in which an attacker will attempt to compromise a Kerberos Infrastructure would be to **attack the Kerberos servers**
  - If an attacker can gain root access to a KDC, the attacker will have access to the database of shared keys of the principals
- Other methods include **replay attacks** and **password-guessing attacks**
  - A replay attack would involve intercepting or otherwise acquiring a Kerberos ticket and then fraudulently representing that ticket in an attempt to gain authentication
  - Password guessing in a Kerberos system could be done by intercepting Kerberos tickets from the network and then making a brute force attempt to decrypt the intercepted tickets
- An attacker may **exploit outdated software** in the infrastructure
  - There are many problems with Kerberos version 4
  - Most importantly, it has a basic protocol weakness in the encryption used because it included the use of DES in standard mode which allows attackers to intercept and modify the ciphertext of Kerberos tickets in an undetectable way
  - To prevent these attacks, Kerberos 5 has been modified to use triple DES in Cipher Block Chaining (CBC) mode

COMPROMISE OF KERBEROS INFRASTRUCTURE (2)

http://intronetworks.cs.luc.edu/current1/uhtml/security.html



- Many implementations of Kerberos version 4 have **buffer overflow vulnerabilities**
- While the reference implementations of version 5 fixed the buffer overflow weaknesses in version 4, **distributions of Kerberos 5 usually ship with software to provide backward compatibility to support legacy Kerberos 4 applications**
- Some of the backward compatibility code for version 4 support in Kerberos version 5 releases is still believed to be vulnerable to buffer overflow attacks
- Due to the protocol weaknesses in version 4 and the potential for buffer overflow attacks with version 4 implementations and version 4 backward compatibility code, **it is best not to support or use Kerberos version 4**

STORING PASSWORDS/SHARED KEYS (1)

- Kerberos uses a **Keytab file** to store passwords so that users can authenticate to Kerberos services without typing their passwords again
- **Passwords are not stored in raw format rather…**
- **… derived keys are generated based on passwords and such keys are stored instead**
- A keytab file is encrypted by AS (secret) key
- Kerberos support several different ways to derive and store keys from passwords including
  - RC4-HMAC
  - AES-128-CTS-HMAC-SHA1-96
  - AES-256-CTS-HMAC-SHA1-96

STORING PASSWORDS/SHARED KEYS (2)

- RC4-HMAC
  - The key is simply the MD4 hash of the password (after representing it in Unicode format)
  - MD4 (Message-Digest Algorithm) is a cryptographic hash function developed by Ronald Rivest in 1990
  - The security of MD4 has been severely compromised
    - The first full collision attack against MD4 was published in 1995 and several newer attacks have been published since then
    - As of 2007, an attack can generate collisions in less than 2 MD4 hash operations

Unicode is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems

https://tools.ietf.org/html/rfc3962

STORING PASSWORDS/SHARED KEYS (3)

- AES
  - When it comes to AES, things are more complicated than a simple hash
  - Although key generation is documented in RFC3962, the process is a bit involved
  - Roughly, the idea is to use the function **PBKF2**
  - PBKDF2 (Password-Based Key Derivation Function 2) are **key derivation functions with a sliding computational cost, used to reduce vulnerabilities to brute-force attacks**
- PBKDF2 applies a pseudorandom function, such as hash-based message authentication code (HMAC), to the input password along with a salt value and repeats the process many times to produce a derived key, which can then be used as a cryptographic key in subsequent operations
- The added computational work makes password cracking much more difficult, and is known as key stretching
  - When the standard was written in the year 2000 the recommended minimum number of iterations was 1000, but the parameter is intended to be increased over time as CPU speeds increase
  - The Kerberos standard in 2005 recommended 4096 iterations

ON REPLAY ATTACKS

- A replay attack occurs when an intruder steals the packet and presents it to the service as if the intruder were the user
  - The user's credentials are there -- everything needed to access a resource
- This is mitigated by the features of the Authenticator that contain additional data such as an encrypted
  - IP list,
  - the client's timestamp
  - the ticket lifetime
- **If a packet is replayed, the timestamp is checked**
- If the timestamp is earlier or the same as a previous authenticator, the packet is rejected because it is a replay
- In addition, the time stamp in the Authenticator is compared to the server time
  - It must be within few minutes (5 by default in Windows)
- While it is technically possible to steal the packet and present it to the server before the valid packet gets there, it is very difficult to do

ON TIME SYNCHRONIZATION (1)

- Since the security of Kerberos authentication is in part based upon the time stamps of tickets, it is critical to have accurately set clocks on Kerberos servers
- It is advisable to set a short lifetime for tickets to prevent attackers from performing successful brute force attacks or replay attacks
- If one allows for server clocks to drift, the network will become vulnerable to such attacks
- If clocks are not synchronized within a reasonable time window, Kerberos will report fatal errors and refuse to function
- Clients attempting to authenticate from a machine with an inaccurate clock will be failed by the KDC in authentication attempts due to the time difference with the KDC's clock

ON TIME SYNCHRONIZATION (2)

- The **Network Time Protocol (NTP)** is available for the time synchronization of servers
- NTP is capable of synchronizing client clocks within
  - milliseconds on LANs
  - tens of milliseconds over WANs
- NTP servers are divided by stratum
  - Primary NTP servers are classified as stratum 1: these should not be used for synchronization of client machines due to the relatively small number of them
  - Public stratum 2 servers are available for client machine synchronization and synchronize their own clocks with the public stratum 1 servers
  - It is advisable to set NTP and query against 3 stratum two servers

SECURITY ISSUES IN SHORT

- If a server can be fooled about the correct time, old tickets can be reused
- Vulnerable to password guessing attacks (attacker collects tickets and does trial decryptions with guessed passwords)
- Active intruder on the network can cause denial of service by impersonation of Kerberos IP address
- Principals must keep their secret keys secret.
- If a user chooses a poor password, it is possible for an attacker to successfully mount an offline dictionary attack by repeatedly attempting to decrypt, with successive entries from a dictionary, messages obtained which are encrypted under a key derived from the user's password
- More limitations at https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.6087&rep=rep1&type=pdf

95

PROTOCOL SPECIFICATION & ANALYSIS

Or how to prove that a protocol achieves a certain security goal despite an attacker

1. Need to give a meaning to the exchange of a message, clarifying the temporal aspects (i.e. the fact that sending and receiving messages is highly asynchronous)
2. Give structured methods to prove or disprove that a given protocol satisfies a certain security property

NEEDHAM-SCRHOEDER AGAIN AS A PROTOCOL NARRATIVE

1. $A \longrightarrow S \ : \ A, B, N_A$
2. $S \longrightarrow A \ : \ \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \longrightarrow B \ : \ \{K_{AB}, A\}_{K_{BS}}$
4. $B \longrightarrow A \ : \ \{N_B\}_{K_{AB}}$
5. $A \longrightarrow B \ : \ \{N_B - 1\}_{K_{AB}}$

- A, S, and B are unique identifiers of principals Alice, (Key Distribution) Service, and Bob
- Na and Nb are nonces
- { M } k denotes the encryption of the message M with key k
- The comma is the operator for concatenation of parts of messages

DIGRESSION ON ALICE&BOB NOTATION (1)

- Alice&Bob–notation is commonly used to describe security protocols as sequences of message exchange steps of the form

  A → B : M

  where A and B are the parties involved in the message exchange and M is the message being exchanged
  - Intuitively, the above means that party A sends message M to party B
- A **protocol narrative** (or narration) is a simple sequence of message exchanges between the different participating principals and can be interpreted as the intended trace of the ideal execution of the protocol
- Are we sure that the meaning of protocol narratives is sufficiently clear?
- At a closer look, the intuitive semantics leaves a lot of details implicit…

DIGRESSION ON ALICE&BOB NOTATION (2)

- The following 4 are the main issues that need to be clarified

1. Need to make explicit what is known (public, private) before a protocol run, and what is to be generated freshly during a protocol run
   - Must make explicit the pieces of data known in advance by the agents (e.g., master keys) and those that are to be freshly generated during the course of a protocol run (e.g., nonces)
2. Need to make explicit which checks the individual principals are expected to carry out on the reception of messages
   - A message exchange is very abstractly characterized as A → B : M without remembering that message M is usually transmitted from A to B by passing through an asynchronous insecure network where a potential intruder can interfere
   - Once B receives a message, it may be just the expected one M according to the protocol, but it may also be an intended message received at the wrong moment and, worse, it may even be an unintended message forged by some malicious attacker
   - As a consequence, B needs to perform appropriate checks on the received message to be sure that it is the expected one

DIGRESSION ON ALICE&BOB NOTATION (3)

3. Principals act concurrently, in contrast to the apparently sequential idealized execution of a run according to a narration
   - Since the message exchange happens through an asynchronous and insecure network, potentially changing the order in which messages are received, may give rise to unexpected situations
4. Concurrency occurs also at the level of different protocol sessions, which may happen to be executed simultaneously while sharing principals across
   - Protocol narratives completely ignore this aspect

DIGRESSION ON ALICE&BOB NOTATION (4)

- A more precise characterization of the meaning of A → B : M could be the following
  - A **asynchronously** sends M towards B
  - B **receives some message** (intended to be M), and
  - finally B **checks that the message it just received indeed has the expected properties** (associated with M, from the point of view of B)
- The checks performed by B on the received message must verify in how far
  - the expectations of the recipient on the received message (as expressed statically in the narration) are matched according to the recipient's current knowledge
  - the gained knowledge is consistent with previously acquired knowledge
- Examples of checks are related to the type of messages
  - if a message is supposed to be built by using pairing, then it should be possible to obtain the components using projections
  - if a message is an encrypted piece of data, then it should be possible to decrypt it and recover the plaintext with the appropriate key (if it is known by the party)

DIGRESSION ON ALICE&BOB NOTATION (5)

To summarize…

- Due to the distributed nature of the system, protocols are highly asynchronous
  - A party to a protocol does not know anything about the current run of the protocol except the messages it has received and sent
  - Except for the initiator, other parties will not even know that they are participating until they receive their first message
- Each message sent must be of a form the recipient can identify and respond to

RELEVANT PROPERTIES FOR CRYPTOGRAPHIC PROTOCOL

- What are the goals of the protocol?
- What does the protocol actually achieve?
- Does it achieve its stated objective?
- Does it include unnecessary steps or messages?
- What assumptions are made?
- Does it encrypt items that could be sent in the clear?
- Is it susceptible to attack?
- What would an attack look like?

ATTACKS ON CRYPTOGRAPHIC PROTOCOLS

- What constitutes an attack?
- This question can be broken down in the following questions
  - Are both **authentication** and **secrecy** assured?
  - Is it possible to **impersonate** one or more of the parties?
  - Is it possible to interject messages from an earlier exchange (**replay attack**)?
  - What tools can an **attacker** deploy with which **capabilities**?
  - If any **key** is **compromised**, what are the **consequences**?
- Some protocols have been in use for years before someone noted a significant vulnerability

TYPES OF ATTACKS

- **Known-key attack**
  - Attacker obtains some keys used previously and uses this info in some malicious fashion
- **Replay**
  - Attacker records messages and replays them at a later time
- **Impersonation**
  - Attacker assumes the identity of one of the legitimate parties in a network
- **Man-in-the-Middle**
  - Attacker interposes itself between two parties and pretends to each to be the other
- **Interleaving attack**
  - Attacker injects spurious messages into a protocol run to disrupt or subvert it

ATTACKER CAPABILITIES

- The designer of a protocol should assume that an attacker can access all of the traffic and interject his own messages into the flow
- Can the attackers messages be arbitrary? Why not? What restrictions do we impose on the attacker?
- The protocol should be robust in the face of such a determined and resourceful attacker

Typically, it assumed the **Dolev-Yao model**

- The attacker can overhear, intercept, and synthesize any message and is only **limited by the constraints of the cryptographic methods used**
- In other words: "the attacker carries the message" but it cannot break the cryptographic primitives that are considered perfect

NEEDHAM-SCHROEDER ATTACK AGAIN

$$ii.1.\ A \longrightarrow S\ :\ A, B, N_A$$

$$ii.2.\ S \longrightarrow A\ :\ \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$

$$ii.3.\ I(A) \longrightarrow B\ :\ \{K'_{AB}, A\}_{K_{BS}}$$

$$ii.4.\ B \longrightarrow I(A)\ :\ \{N_B\}_{K'_{AB}}$$

$$ii.5.\ I(A) \longrightarrow B\ :\ \{N_B - 1\}_{K'_{AB}}$$

- I(A) is the intruder posing as A with B
- K'_{AB} is the session key stolen by the intruder I from a previous run of the protocol
- The problem is that B has no way to check the freshness of the session key just received

VERIFICATION OF CRYPTOGRAPHIC PROTOCOLS

INTRODUCTION

Security protocols are three-line programs that people still manage to get wrong.
Roger M. Needham

- Protocols can be notoriously difficult to get correct
  - Flaws have been discovered in protocols published many years before
- **Protocols often depend on assumptions that are not clearly stated**
- It would be nice to be able to reason with mathematical rigour about protocol correctness

- How do you characterize what an attacker can do?
  - We should at least assume that it can
    - see all messages sent
    - compose messages from anything visible to it
    - interject messages into the flow

Problems with the design of protocols:
- Lack of assumptions
- Lack of formal descriptions
- Lack of clarity

THREE MAIN APPROACHES TO PROTOCOL VERIFICATION

- **Belief logics** allow reasoning about what principals within the protocol should be able to infer from the messages they see
  - Allows abstract proofs, but may miss some important flaws
  - We will briefly present it in the following

- State exploration methods (**model checking**) treat a protocol as a state machine and conduct an exhaustive search checking that all reachable states are safe
  - For an introduction, see http://www-oldurls.inf.ethz.ch/personal/basin/pubs/security-modelchecking.pdf

- **Theorem proving** uses induction over potential traces of protocol execution
  - For an introduction, see https://www.cl.cam.ac.uk/~lp15/papers/Auth/index.html

BELIEF LOGICS

Example of rule of inference
Modus ponens
1. If A then B
2. A
3. Therefore B

Interestingly, one can write programs that allow to perform inferences automatically… athough they are not guaranteed to be complete, i.e. not all possible conclusions are derived!

- A belief logic is a formal system for reasoning about beliefs
  - Any logic consists of a set of logical operators and rules of inferences that allows us to symbolically derive true statements
- Idea for protocol verification
  - Build a sequence of message exchanges and generating a collection of belief statements
- Postulate some reasonable initial assumptions about the state of knowledge/belief of the principals…
- … combine all these to formally derive that certain security properties are ensured by the protocol

BAN LOGIC: INTRODUCTION (1)

- Introduced in 1989, after its inventors: **Burrows, Abadi, Needham**
- It is based on an agreed set of deduction rules for formally reasoning about the **authentication protocols** and is often referred to as a logic of authentication
- It is a mathematically rigorous method for verifying that two principals (people, computer, services) are entitled to believe they are communicating with each other and not the intruders
- **The logic cannot prove that a protocol is wrong**, i.e. to disprove that a security goal cannot be achieved
  - However, if one cannot prove a protocol correct, then it is wise to consider that protocol with great suspicion
- **It helps clarify the protocol's assumptions by formally stating them**
  - This is a general phenomenon that is intrinsic to the formalization process, i.e. the fact of using strictly specified inference rules immediately points out which hypotheses are missing…

BAN LOGIC: INTRODUCTION (2)

- It is a logical systems based on **principal beliefs and their evolution after message exchanges**
  - It concentrates on the beliefs of trustworthy parties involved in the protocol and
  - the evolution of these beliefs through communication processes

- **Digression on the concept of logical system**
  - A logical system is a **deductive system** together with additional axioms and a **semantics**
  - A **deductive system** consists of the **axioms** and **rules of inference** that can be used to derive theorems of the system
  - An **axiom** is a statement that is taken to be true, to serve as a premise or starting point for further reasoning and arguments
    - *Example*: It is possible to draw a straight line from any point to any other point
  - An **inference rule** is a function which takes premises, analyzes their syntax, and returns a conclusion
    - *Example*: Modus Ponens
  - The semantics gives the meaning to the expressions of the deductive system

BAN LOGIC: SOME OPERATORS

- P, A, and B identify principals
- X identifies messages
- K identifies keys
- { . } . denotes encryption

$P|\equiv X$: *(P believes X)* P is entitled to act as though X is true.

$A \triangleleft X$: *(A sees X)* someone has sent a message to A containing X so that he can read X and repeat it.

$A|\sim K$: *(A once said K)* at some time, A used key K.

$A|\sim X$: *(A once said X)* at some time, A uttered a message containing X.

$A \Longrightarrow X$: *(A has jurisdiction over X)* A is an authority on X and can be trusted on X.

$A \overset{K}{\longleftrightarrow} B$: *(A and B share key K)* A and B can use key K to communicate. The key is unknown to anyone else.

$\#(X)$: *(X is fresh)* meaning that X has not been sent before in any run of the protocol.

BAN LOGIC: RULES (1)

A believes that A and B can use the shared secret key K to communicate

Someone has sent the message X encrypted with key K so that A can read it

Message meaning: If A believes (A share(K) B) and A sees $\{X\}_K$ then A believes(B said X).

$$\frac{A|\equiv (A \overset{K}{\longleftrightarrow} B), A \triangleleft \{X\}_K}{A|\equiv (B|\sim X)}$$

A believes that B has uttered a message containing X

BAN LOGIC: RULES (2)

A believes that X is fresh, i.e. X has not been seen in any previous protocol runto communicate

A believes that B has uttered a message containing X

Nonce verification: If A believes X is fresh and A believes B once said X, then A believes B believes X.

$$\frac{A|\equiv (\#(X)), A|\equiv (B|\sim X)}{A|\equiv (B|\equiv X)}$$

A believes that B believes X (in this protocol run!)

BAN LOGIC: RULES (3)

A believes that B is an authority on X and can be trusted on X

A believes that B believes X

Jurisdiction: If A believes B has jurisdiction over X and A believes B believes X, then A believes X.

$$\frac{A|\equiv (B \Longrightarrow X), A|\equiv (B|\equiv X)}{A|\equiv X}$$

A believes X

ANALYSIS OF PROTOCOLS

The steps of BAN logic to analyze the original protocol are as follows:

1. The protocol is transformed into some **idealized** form
2. Identify initial assumptions in the language of BAN logic
3. Use the axioms and rules of the logic to deduce new expressions
4. Interpret the statements proved by the process: are the goals achieved?

IDEALIZED PROTOCOLS (1)

- Each protocol step has the following form

$$A \rightarrow B : message$$

- What does this denote?
  - Principal A sends the message and principal B receives the (same) message
  - It is an *informal* notation, often ambiguous, and not appropriate for rigorous analysis
- How to fix it?
- Transform each protocol into an idealized form by omitting
  - the parts of the message that do not contribute to the beliefs of the recipient
  - clear text communication because it can be forged

IDEALIZED PROTOCOLS (2)

Example

$$A \rightarrow B : \{A, Kab\}Kbs$$

Is transformed into

$$A \xrightarrow{Kab} B : \{A \leftrightarrow B\}Kbs$$

So that when the message is sent to B, it is possible to derive that

$$B \lhd \{A \overset{Kab}{\leftrightarrow} B\}Kbs$$

The receiving principle becomes aware of the message (i.e. sees the message) and can act upon it

GOALS OF AUTHENTICATION PROTOCOLS

- Authentication rests on communication protected by shared session key, so the goals of authentication may be reached between A and B if there is a shared K such that

Key authentication

$$A \mid\equiv A \overset{K}{\leftrightarrow} B \qquad B \mid\equiv A \overset{K}{\leftrightarrow} B$$

- Some authentication protocols achieve this final goal:

Key confirmation

$$A \mid\equiv B \mid\equiv A \overset{K}{\leftrightarrow} B \qquad B \mid\equiv A \mid\equiv A \overset{K}{\leftrightarrow} B$$

Key freshness

$$A \mid\equiv \#(A \overset{K}{\leftrightarrow} B) \qquad B \mid\equiv \#(A \overset{K}{\leftrightarrow} B)$$

121

APPLICATION TO NEEDHAM-SCHROEDER

FORMALIZATION (1)

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. $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

implicit

A and B share key Kab

Kab is fresh

1. omitted since all components are plaintext
2. $S \rightarrow A : \{N_a, (A \overset{K_{ab}}{\longleftrightarrow} B), \#(A \overset{K_{ab}}{\longleftrightarrow} B), \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b, (A \overset{K_{ab}}{\longleftrightarrow} B)\}_{K_{ab}}$
5. $A \rightarrow B : \{N_b, (A \overset{K_{ab}}{\longleftrightarrow} B)\}_{K_{ab}}$

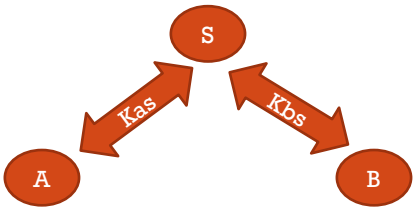
FORMALIZATION (2.1)

Assumptions about **initial state** of the Needham-Schroeder protocol

- Kas is a shared key by A and S; both A and S believes this
- Kbs is a shared key by B and S; both B and S believes this
- Kab is a shared key by A and B; only S believes this

$A|\equiv A \overset{K_{as}}{\longleftrightarrow} S \qquad B|\equiv B \overset{K_{bs}}{\longleftrightarrow} S \qquad S|\equiv A \overset{K_{as}}{\longleftrightarrow} S$

$S|\equiv B \overset{K_{bs}}{\longleftrightarrow} S$



FORMALIZATION (2.2)

Assumptions about **initial state** of the Needham-Schroeder protocol

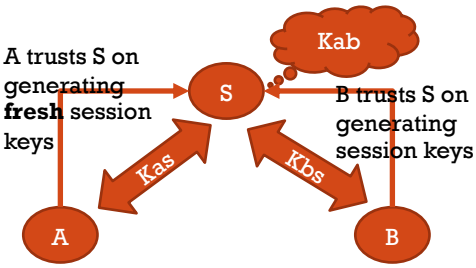
$S|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$

- Kab is a shared key by A and B; only S believes this
  - But see below…

$A|\equiv (S \Longrightarrow A \overset{K}{\longleftrightarrow} B)$ $B|\equiv (S \Longrightarrow A \overset{K}{\longleftrightarrow} B)$

$A|\equiv (S \Longrightarrow \#(A \overset{K}{\longleftrightarrow} B))$

For any key K:
- S is an authority to say that K is a shared key by A and B, and it is trusted to say this; both A and B believes this
- S is an authority to say that K is a fresh shared key by A and B, and it is trusted to say this; only A believes this



FORMALIZATION (2.3)

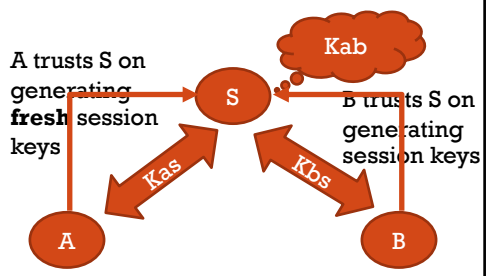
Assumptions about **initial state** of the Needham-Schroeder protocol

- Na is fresh and A believes this
- Nb is fresh and B believes this
- Kab is a fresh shared key by A and B; S believes this

$A|\equiv \#(N_a)$ $B|\equiv \#(N_b)$ $S|\equiv \#(A \overset{K_{ab}}{\longleftrightarrow} B)$

$B|\equiv \#(A \overset{K}{\longleftrightarrow} B)$

For any key K:
- B believes that K is a fresh shared key by A and B



- Needham and Schroeder did not realize they were making this assumption
- **They were criticized for it as it is pretty a strong assumption!**
- ***It also paves the way to an attack…***

Notice that A and B trusts S in different ways:
- A trusts also the fact that S is able to generate fresh keys
- B does not trust this and assumes that any shared key with A is fresh

2. $S \rightarrow A : \{N_a, (A \overset{K_{ab}}{\longleftrightarrow} B), \#(A \overset{K_{ab}}{\longleftrightarrow} B), \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$

DEDUCTION (1) [SHORT VERSION]

From step 2 of the (idealized) protocol:

$$A \triangleleft \{N_a, (A \overset{K_{ab}}{\longleftrightarrow} B), \#(A \overset{K_{ab}}{\longleftrightarrow} B), \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$$

The *Nonce Verification Rule* says:

$$\frac{A|\equiv (\#(X)), A|\equiv (S|\sim X)}{A|\equiv (S|\equiv X)}$$

Since A believes N_a to be fresh, we get:

$$A|\equiv (S|\equiv A \overset{K_{ab}}{\longleftrightarrow} B)$$

DEDUCTION (2) [SHORT VERSION]

The *Jurisdiction Rule* says that:

$$\frac{A|\equiv (S \Longrightarrow X), A|\equiv (S|\equiv X)}{A|\equiv X}$$

From this we obtain:

$$A|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

Key freshness (for A) $A|\equiv \#(A \overset{K_{ab}}{\longleftrightarrow} B)$

DEDUCTION (3) [SHORT VERSION]

Since A has also seen the part of the message encrypted under B's key, he can send it to B. B decrypts the message and obtains:

$$B|\equiv (S|\sim A \overset{K_{ab}}{\longleftrightarrow} B) \qquad 3\ A \rightarrow B : \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}$$

meaning that B believes that S once sent the key.

At this point, we need the final dubious assumption:

$$B|\equiv \#(A \overset{K}{\longleftrightarrow} B)$$

With it, we can get:

$$B|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

Key freshness (for B) derived from built in belief of B

DEDUCTION (4)

$$4\ B \rightarrow A : \{N_b, (A \overset{K_{ab}}{\longleftrightarrow} B)\}_{K_{ab}}$$
$$5\ A \rightarrow B : \{N_b, (A \overset{K_{ab}}{\longleftrightarrow} B)\}_{K_{ab}}$$

From the last two messages, we can infer the following.

Key authentication

$$A|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$$
$$B|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

Key confirmation

$$A|\equiv (B|\equiv A \overset{K_{ab}}{\longleftrightarrow} B)$$
$$B|\equiv (A|\equiv A \overset{K_{ab}}{\longleftrightarrow} B)$$

FINAL REMARKS (1)

- The assumption $B|\equiv \#(A \overset{K}{\longleftrightarrow} B)$ is dubious because it is different from all other freshness assumptions since those were all based on values that the believing principal had herself generated
- This one expresses B's belief that a random value someone else has generated is fresh!
- The consequence of this strange assumption is the attack described above where an attacker is assumed to be able to compromise an old session key
  - Since B is ready to accept the freshness of any key generated by S, it is willing to accept also the compromised key that can be replayed by the attacker that was able to compromise it

FINAL REMARKS (2)

- The attack relies on the fact that B has no way to actually be assured that message 3 is fresh
- An attacker could spend whatever time is needed to break the session key Kab…
- … as long as it can do so within the lifetime of Kbs, it can run the replay attack described above
- B will think it has confirmed sharing Kab with A, when in reality A is not present and the attacker knows the key

- Notice that the **replay attack is not directly uncovered by a BAN analysis of the protocol**; **rather the analysis shows that the protocol cannot achieve any sort of authentication for B without making the dubious assumption that underlies the attack**

FINAL REMARKS (3)

- BAN logic is a belief system and it is much different from a knowledge system
- Knowledge systems have an inference rule of the form

  "If a principal knows p, then p is true"
- Belief systems do not have this rule, since a belief in p says nothing about the truth or falsity of p

- In BAN logic, it is assumed that all principals taking part in a protocol are honest, in the sense that each principal believes in the truth of each message it sends.
- Unfortunately, this assumption is far from being reasonable in presence of attackers

133

ANALYSIS OF NEEDHAM-SCHROEDER PROTOCOL

Details

NOTATION

P |~ X: P *once said* X: P at some time sent a message including the statement X. It is not known when the message was sent(in the past or in the current run of the protocol) but P believed that X was true when it send the message.

P |⇒ X: P controls X. P has jurisdiction over X. P is a trusted authority on the truth of X.

#(X): X is *fresh*. Using the logic, time is divided into two *epoch*, the past and the present. The present begins with the start of the current execution of the current protocol. X is fresh if it is not contained in any message sent in the past.

P |≡ X: P *believes* X. P would be entitled to believe X. The principal P may act assuming X to be true.

P ◁ X: P *sees* X. P can read the contents of X (possibly after decryption, assuming P has the needed keys) and P can include X in messages to other principals.

$P \overset{K}{\leftrightarrow} Q$: K is a shared key for P and Q. K is a secure key for communication between P and Q, and it will never be discovered by any principal except for P or Q, or a principal trusted by either P or Q.

$\{X\}_K$: X *encrypted under* K. It represents the message X encrypted using the key K.

RULES (1)

- **Message meaning rule:** Rule concerns the interpretation of messages. This rule helps to explain the origin of the messages.

$$\frac{P \mid\equiv Q \overset{K}{\leftrightarrow} P,\ P \triangleleft \{X\}_K}{P \mid\equiv Q \mid\sim X}$$

RULES (2)

- **Nonce-verification rule:** This rule checks that a message is recent, and also checks if the sender still believes in it.

$$\frac{P \mid\equiv \#(X),\ P \mid\equiv Q \mid\sim X}{P \mid\equiv Q \mid\equiv X}$$

RULES (3)

- **Jurisdiction rule:** This rule states what it means for a principal to be the trusted authority on the truth of X.

$$\frac{P \mid\equiv Q \Rightarrow X,\ P \mid\equiv Q \mid\equiv X}{P \mid\equiv X}$$

RULES (4)

- **Belief Rule:** The rule states that a principal believes a collection of statements if and only if it believes each of the statements individually.

Example:

A) $\dfrac{P \mid\equiv X,\ P \mid\equiv Y}{P \mid\equiv (X,Y)}$ B) $\dfrac{P \mid\equiv (X,Y)}{P \mid\equiv X}$

C) $\dfrac{P \mid\equiv Q \mid\equiv (X,Y)}{P \mid\equiv Q \mid\equiv X}$

RULES (5)

- **Saying rule:** This rule says that a principal sees all the components of every message it sees, provided that the principal knows the necessary key

A) $\dfrac{P \triangleleft (X,Y)}{P \triangleleft X}$ B) $\dfrac{P \mid\equiv Q \overset{K}{\leftrightarrow} P,\ P \triangleleft \{X\}_K}{P \triangleleft X}$

RULES (6)

- **Freshness Rule:** This rule states that any message with a fresh component is also fresh.

$$\frac{P \mid\equiv \#(X)}{P \mid\equiv \#(X, Y)}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

Original version without idealization

Message 1 $A \rightarrow S$:	A, B, N_A
Message 2 $S \rightarrow A$:	$\{N_A, B, K_{AB}, \{K_{AB}, A\}K_{BS}\} K_{AS}$
Message 3 $A \rightarrow B$:	$\{K_{AB}, A\}K_{BS}$
Message 4 $B \rightarrow A$:	$\{N_B\}K_{AB}$
Message 5 $A \rightarrow B$:	$\{N_B - 1\}K_{AB}$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

Corresponding idealized protocol is as follows:

Message 2 $S \rightarrow A$: $\{N_A, (A \overset{Kab}{\leftrightarrow} B), \# (A \overset{Kab}{\leftrightarrow} B), \{A \overset{Kab}{\leftrightarrow} B\}K_{bs}\} K_{as}$

Message 3 $A \rightarrow B$: $\{A \overset{Kab}{\leftrightarrow} B\}K_{bs}$

Message 4 $B \rightarrow A$: $\{N_B, (A \overset{Kab}{\leftrightarrow} B)\}K_{ab}$

Message 5 $A \rightarrow B$: $\{N_B, (A \overset{Kab}{\leftrightarrow} B)\}K_{ab}$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

Initial assumptions:

$A \mid\equiv A \overset{Kas}{\leftrightarrow} S$ $\quad$ $B \mid\equiv B \overset{Kbs}{\leftrightarrow} S$

$S \mid\equiv A \overset{Kas}{\leftrightarrow} S$ $\quad$ $S \mid\equiv B \overset{Kbs}{\leftrightarrow} S$

$S \mid\equiv A \overset{Kab}{\leftrightarrow} B$

$A \mid\equiv (S \mid\Rightarrow A \overset{Kab}{\leftrightarrow} B)$ $\quad$ $B \mid\equiv (S \mid\Rightarrow A \overset{Kab}{\leftrightarrow} B)$

$A \mid\equiv (S \mid\Rightarrow \#(A \overset{Kab}{\leftrightarrow} B))$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

More assumptions(continued)

$A \mid\equiv \#(N_a)$ $\quad$ $B \mid\equiv \#(N_b)$

$S \mid\equiv \#(A \overset{K_{ab}}{\leftrightarrow} B)$ $\quad$ $B \mid\equiv \#(A \overset{K_{ab}}{\leftrightarrow} B)$

NOTE: The assumption $B \mid\equiv \#(A \overset{K_{ab}}{\leftrightarrow} B)$ meaning B believes in the freshness on the key is an assumption that the authors of the Needham-Schroeder protocol did not realize they were making.

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

Now we can apply the logical postulate rules to each message with assumptions

Recall message 2:

Message 2 $\quad S \rightarrow A: \{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}\}K_{as}$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

1) Recall the Assumption:

$$A \mid\equiv A \overset{K_{as}}{\leftrightarrow} S$$

With this Assumption and message 2, now we can say:

$$A \lhd \{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}\}K_{as}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

Now apply the logical postulate, the Message-meaning rule

Recall message-meaning rule is:

$$\frac{P \mid\equiv Q \overset{K}{\leftrightarrow} P, P \lhd \{X\}_k}{P \mid\equiv Q \mid\sim X}$$

Applying this postulate to the previous assumption and derivation, we derive that:

$$A \mid\equiv S \mid\sim \{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}\}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

2) Recall the Assumption:

$$A \mid\equiv \#(N_a)$$

Now we can apply the Freshness rule, recall that it is:

$$\frac{P \mid\equiv \#(X)}{P \mid\equiv \#(X,Y)}$$

Now we can derive that:

$$A \mid\equiv \#\{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}\}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

3) We can use a combination of the above derived rules together with Nonce-verification rule which is:

$$\frac{P \mid\equiv \#(X),\quad P \mid\equiv Q \mid\sim X}{P \mid\equiv Q \mid\equiv X}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

3) We can use the above derived rules stating that:

$$A \mid\equiv \#\{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_a}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}\}$$

together with:

$$A \mid\equiv S \mid\sim \{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}\}$$

and the Nonce-verification to obtain:

$$A \mid\equiv S \mid\equiv \{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}\}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

4) We can use the belief rule which is:

$$\frac{P \mid\equiv Q \mid\equiv (X,Y)}{P \mid\equiv Q \mid\equiv X}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

We can use this belief rule combined with the above derived statement stating that:

$$A \mid\equiv S \mid\equiv \{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}\}$$

to further derive that:

$$A \mid\equiv S \mid\equiv (A \overset{K_{ab}}{\leftrightarrow} B)$$

and that:

$$A \mid\equiv S \mid\equiv \#(A \overset{K_{ab}}{\leftrightarrow} B)$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

5) Recall the Assumptions:

$$A \mid\equiv (S \mid\Rightarrow A \overset{K_{ab}}{\leftrightarrow} B) \qquad A \mid\equiv (S \mid\Rightarrow \#(A \overset{K_{ab}}{\leftrightarrow} B)$$

and the previous derivations stating that:

$$A \mid\equiv S \mid\equiv (A \overset{K_{ab}}{\leftrightarrow} B) \qquad A \mid\equiv S \mid\equiv \#(A \overset{K_{ab}}{\leftrightarrow} B)$$

We can apply the jurisdiction postulate to these assumptions.

Recall jurisdiction rule:

$$P \mid\equiv Q \mid\Rightarrow X,\ P \mid\equiv Q \mid\equiv X$$

$$P \mid\equiv X$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

Applying the assumptions above to the postulates we finally get:

$$A \mid\equiv (A \overset{K_{ab}}{\leftrightarrow} B) \quad \text{and} \quad A \mid\equiv \#(A \overset{K_{ab}}{\leftrightarrow} B)$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

Now we can apply the logical postulate rules to the next message with assumptions

Recall message 3:

Message 3 $A \rightarrow B: \{A \overset{Kab}{\leftrightarrow} B\}K_{bs}$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

1) Recall the Assumption:

$$B \mid\equiv S \overset{K_{bs}}{\leftrightarrow} B$$

From this we can deduce that:

$$B \triangleleft \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}$$

We can now apply the message meaning rule which is

$$\frac{P \mid\equiv Q \overset{K}{\leftrightarrow} P,\ P \triangleleft \{X\}_k}{P \mid\equiv Q \mid\sim X}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

And we can derive:

$$B \mid\equiv S \mid\sim \{A \overset{K_{ab}}{\leftrightarrow} B\}K_{bs}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

2) Recall the Assumption:

$$B \mid\equiv \#(A \overset{Kab}{\leftrightarrow} B)$$

Also recall the derived formula from above stating:

$$B \mid\equiv S \mid\sim \{A \overset{Kab}{\leftrightarrow} B\}K_{bs}$$

We can apply the Nonce-verification rule which is:

$$\frac{P \mid\equiv \#(X),\ P \mid\equiv Q \mid\sim X}{P \mid\equiv Q \mid\equiv X}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

And we can derive:

$$B \mid\equiv S \mid\equiv \{A \overset{Kab}{\leftrightarrow} B\}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

3) Recall the Assumption:

$$B \mid\equiv (S \mid\overset{K_{ab}}{\Rightarrow} A \leftrightarrow B)$$

Also recall the derived formula above stating:

$$B \mid\equiv S \mid\equiv \{A \overset{K_{ab}}{\leftrightarrow} B\}$$

We can apply the jurisdiction rule which is:

$$\frac{P \mid\equiv Q \mid\Rightarrow X, \quad P \mid\equiv Q \mid\equiv X}{P \mid\equiv X}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

And we can derive:

$$B \mid\equiv \{A \overset{K_{ab}}{\leftrightarrow} B\}$$

Now we can apply the logical postulate rules to the next message with assumptions

Recall message 4:

Message 4 $\quad B \rightarrow A: \{N_b, (A \overset{\mathbf{K_{ab}}}{\leftrightarrow} B)\}K_{ab}$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

1) We can then say that:

$$A \lhd \{N_b, \overset{K_{ab}}{(A \leftrightarrow B)}\}\, K_{ab}$$

We can use the saying rule, which is:

$$\frac{P \lhd (X,Y)}{P \lhd X}$$

We can then derive that:

$$A \lhd \{\overset{K_{ab}}{(A \leftrightarrow B)}\}\, K_{ab}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

2) Recall a previous result we obtained:

$$A \mid\equiv \overset{K_{ab}}{(B \leftrightarrow A)}$$

Also recall the result that we just obtained the previous step:

$$A \lhd \{\overset{K_{ab}}{(A \leftrightarrow B)}\}K_{ab}$$

We can apply the message meaning rule:

$$\frac{P \mid\equiv \overset{K}{Q \leftrightarrow P}, P \lhd \{X\}_k}{P \mid\equiv Q \mid\sim X}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

Finally, we can deduce that:

$$A \mid\equiv B \overset{K_{ab}}{\mid\sim} (A \leftrightarrow B)$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

3) Recall a previous result we obtained:

$$A \overset{K_{ab}}{\mid\equiv} \#(A \leftrightarrow B)$$

Also recall the result that we just obtained the previous step:

$$A \mid\equiv B \overset{K_{ab}}{\mid\sim} (A \leftrightarrow B)$$

We can apply the nonce-verification rule:

$$\frac{P \mid\equiv \#(X), \qquad P \mid\equiv Q \mid\sim X}{P \mid\equiv Q \mid\equiv X}$$

PROTOCOL ANALYSIS

Needham-Schroeder Protocol

We then obtain:

$$A \mid\equiv B \mid\equiv (A \overset{K_{ab}}{\leftrightarrow} B)$$

In similar manner, we can also derive that:

$$B \mid\equiv A \mid\equiv (A \overset{K_{ab}}{\leftrightarrow} B)$$