KRY06 - MNG

Model 2019

Kryptografie

Část 6

Symetrická správa klíčů

Post 19/20

Souhrnné materiály

Ver 0.1

KRY

Kryptografie

Pre 2011/12
Pre 2015/16
2020 MNGprep

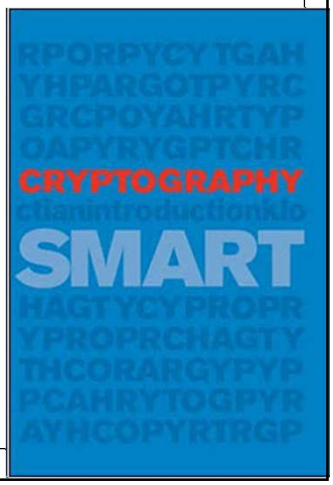
KRY 6

Symetrická správa klíčů

Učebnice

6

- Nigel Smart: Cryptography - An Introduction, 3rd Edition,
  - Mcgraw-Hill College, 3rd Edition, 2013
  - ISBN-10: 0077099877
- Kapitoly
  - Kapitola 9
    - » Zajímavá je pro nás kapitola 9 v rozsahu slajdů



The third edition is now online. You may make copies and distribute the copies of the book as you see fit, as long as it is clearly marked as having been authored by N.P. Smart.

Učebnice je v dokumentovém skladu

©Petr Hanáček

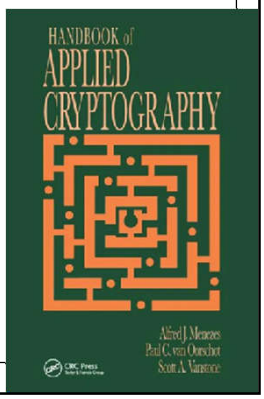
Učebnice

6

- Menezes, Van Oorschot, Vanstone: Handbook of Applied Cryptography, CRC Press, Hardcover, 816 pages, CRC Press, 1997.
- Kapitoly
  - Kapitola 12
    - » Zajímavé jsou pro nás podkapitoly 12.2, 12.3 a 12.4

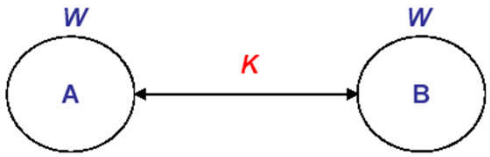
http://www.cacr.math.uwaterloo.ca/hac/

Učebnice je v dokumentovém skladu



©Petr Hanáček

Správa klíčů mezi dvěma účastníky



- Účastníci A a B mají dopředu domluven dlouhodobý sdílený klíč W_{AB} (často se označuje KEK_{AB} - Key Encrypting Key)
- Pomocí něj si dohodnou klíč relace K_{AB} .

©Petr Hanáček CLACRYPT Slide 4

Klíč relace

- **Def**
  - Dočasné tajemství
  - Používané po krátkou dobu a pak zničené
- **Motivace**
  - Omezit množství zašifrovaných zpráv pro analýzu útočníkem
  - Omezit časový rozsah dopadu prozrazení jednoho klíče
  - Zamezit potřebu pamatovat si množství klíčů – vytváříme klíče až když jsou potřeba
  - Nezávislost zabezpečení jednotlivých relací

©Petr Hanáček CLACRYPT Slide 5

Vlastnosti správy klíčů

- **Způsob autentizace**
  - Autentizace subjektu
  - Autentizace klíče
- **Vzájemnost autentizace**
  - Jednosměrná, obousměrná
- **„Čerstvost“ klíče**
- **Činnost: distribuce klíče, dohoda na klíči**
- **Efektivnost**
  - Počet přenesených zpráv
  - Počet přenesených bajtů
  - Složitost výpočtů účastníků
- **Požadavky na třetí nezávislou stranu (TTP)**
  - On-line, off-line
  - Míra důvěry v TTP

©Petr Hanáček CLACRYPT Slide 6

Útočník

- **Útoky**
  - Pasivní
  - Aktivní
- **Cíle útočníka**
  - Získat klíč relace pomocí odposlechu
  - Účastnit se běžícího protokolu a pomocí změn zpráv získat klíč relace
  - Iniciovat jeden nebo více běhů protokolu a kombinovat jejich zprávy pro realizaci jednoho z výše uvedených útoků
  - Bez schopnosti zjistit klíč relace podvést jednoho z účastníků ohledně identity účastníka, se kterým navazuje protokol
  - Přesvědčit jednoho z účastníků, že úspěšně navázal protokol s účastníkem jiným, než je útočník

 CLACRYPT Slide 7

Vytvoření klíče relace

one-pass

$M1 \quad A \rightarrow B: \quad E_W(t_A, B, K)$

Nezajišťuje čerstvost ani autentizaci

with challenge-response

$M1 \quad A \leftarrow B: \quad n_B$

$M2 \quad A \rightarrow B: \quad E_W(n_B, B, K)$

- n_B is a nonce (a "fresh" quantity)

Pouze jednosměrná autentizace

both parties contribute to the session key

$M1 \quad A \leftarrow B: \quad n_B$

$M2 \quad A \rightarrow B: \quad E_W(K_A, n_B, n_A, B)$

$M3 \quad A \leftarrow B \quad E_W(K_B, n_A, n_B, A)$

- n_A and n_B are nonces
- K_A and K_B are keying materiale
- $K = f(K_A, K_B)$

 CLACRYPT Slide 8

Útok replay, čerstvost

- Simple Replay
  Zkopírovat zprávu, později zaslat
- Suppressed Replay
  Odstranit zprávu, později zaslat
- Timed Replay
  Zkopírovat zprávu, zaslat ji v rámci času expirace
- Replay to Sender
  Zaslat zprávu zpět odesílateli (Reflection Attack, např. pro protokoly výzva-odpověď)

INTERLEAVED RUNS

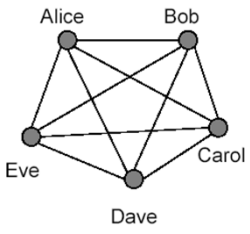
- **Kombinace zpráv z rozdílných (někdy i souběžných) běhů téhož protokolu**

Čerstvost

- Sequence Numbers
  Musí být udržovány pro každý kanál (např. útoky na uhodnutí ISN při útocích na TCP/IP)
- Timestamps
  Zpráva je přijmuta jenom pokud je časová značka dostatečně čerstvá. Jsou třeba synchronizované hodiny.
- Nonces
  Typicky náhodná čísla, zasílaná ve výzvě, vrácená v odpovědi.

©Petr Hanáček CLACRYPT Slide 9

n^2 problém distribuce

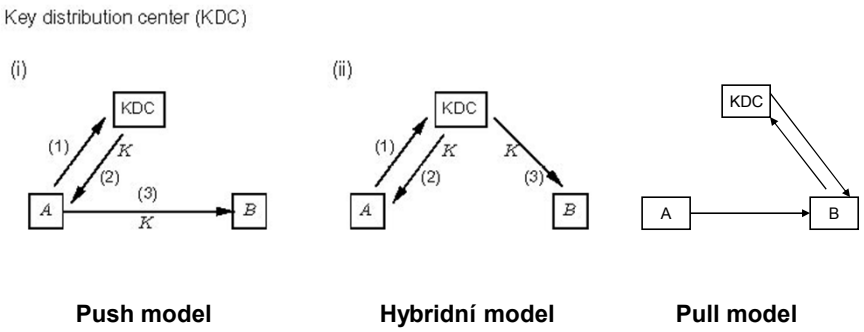


- **Každá dvojice uživatelů musí sdílet společný klíč**
- **Každý uživatel má (n-1) klíčů**
- **Počet klíčů je n*(n-1) nebo (n*(n-1)/2)**
- **Tj. cca n^2 klíčů**
- **Řešením může být důvěryhodná třetí strana – TTP, Trusted Third Party**
  - **KDC**
  - **KTC**
  - **CA**

©Petr Hanáček CLACRYPT Slide 10

Centralizovaná správa klíčů - KDC

- **KDC – Key Distribution Center**
- **A a B sdílejí klíče K_A a K_B s KDC**
- **KDC generuje klíč relace K_{AB}**

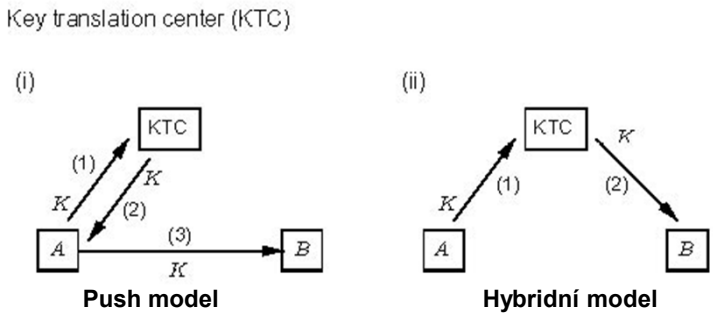


Push model **Hybridní model** **Pull model**

 CLACRYPT Slide 11

Centralizovaná správa klíčů - KTC

- **KTC – Key Translation Center**
- **A a B sdílejí klíče K_A a K_B s KDC**
- **Jedna z komunikujících stran generuje klíč relace K_{AB}**



Push model **Hybridní model**

 CLACRYPT Slide 12

Symetrická správa klíčů - vlastnosti

- **Nevýhody**
  - **Každá komunikace vyžaduje kontakt s TTP**
  - **TTP musí mít uloženo n dlouhodobých klíčů**
  - **TTP může číst všechny zprávy**
  - **Je-li TTP kompromitováno, je jakákoli komunikace nezabezpečená**
- **Oproti tomu asymetrická**
  - **TTP nemusí být kontaktováno před každou komunikací**
  - **TTP nemůže jednoduše odposlouchávat komunikaci**
  - **Důvěra v TTP postačuje menší (TTP může pouze dočasně změnit identitu účastníka)**

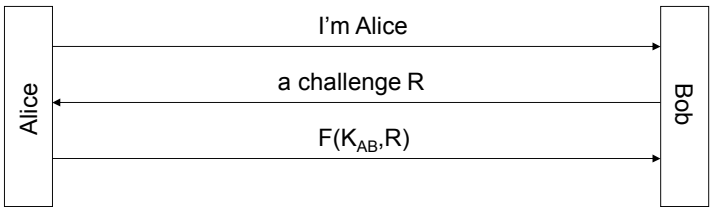
 CLACRYPT Slide 13

Základní protokoly (primárně autentizační)

 CLACRYPT Slide 14

Symetrický klíč, výzva-odpověď

Nejjednodušší protokol:

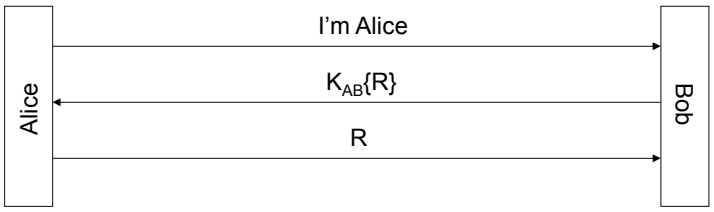


- **Autentizace není obousměrná**
- **Autentizovaný kanál se dá ukrást (connection hijacking)**
- **Pokud je K odvozeno z hesla, dá se provádět off-line hádání hesla**
- **Postačuje hašovací funkce**

 CLACRYPT Slide 15

Symetrický klíč, výzva-odpověď

Alternativní protokol:

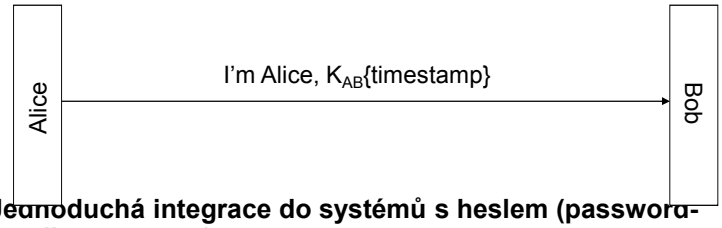


- **Vyžaduje reverzibilní kryptografii (hašovací funkce nestačí)**
- **Pokud R není dostatečně náhodné, lze realizovat slovníkový útok**

 CLACRYPT Slide 16

Symetrický klíč, časová značka

A one-message protocol:



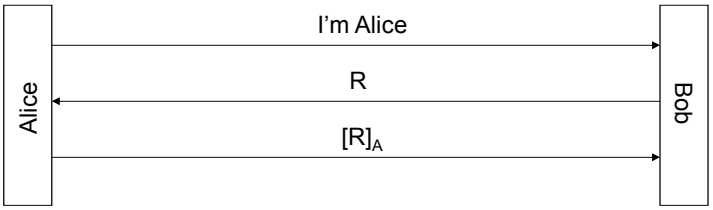
- **Jednoduchá integrace do systémů s heslem (password-sending systems)**
- **Velmi efektivní: jediná zpráva, bezstavový**
- **Je třeba dát pozor na replay**
- **Je třeba dát pozor, pokud je klíč sdílen mezi více servery**
- **Je třeba chránit hodiny**
- **Alternativně se dá použít s hašovací funkcí:**

  I'm Alice, timestamp, $H(K_{AB}$,timestamp)

 CLACRYPT Slide 17

Veřejný klíč, výzva-odpověď

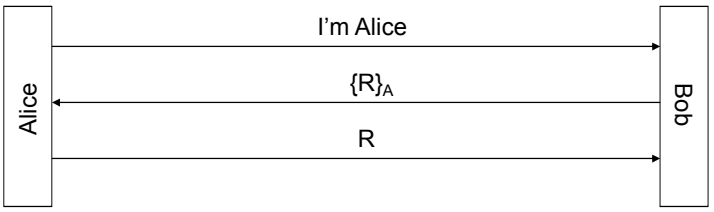
Pomocí podpisu



 CLACRYPT Slide 18

Veřejný klíč, výzva-odpověď

Pomocí dešifrování:



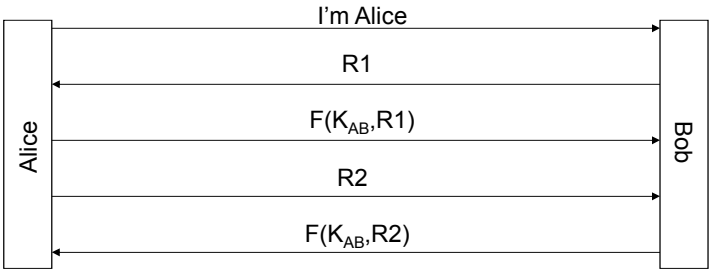
- **Problém: Bob (nebo Trudy) mohou vyžadovat po Alici, aby podepsala/dešifrovala jakýkoli text, který si oni zvolí**
- **Řešení:**
  - **Nikdy nepoužívat stejný klíč pro více účelů (pro přihlášení a pro podpis)**
  - **Používat formátované výzvy**

 CLACRYPT Slide 19

Oboustranná autentizace

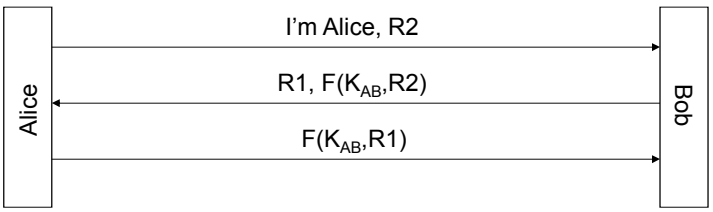
Alice a Bob se autentizují navzájem

Příklad protokolu:



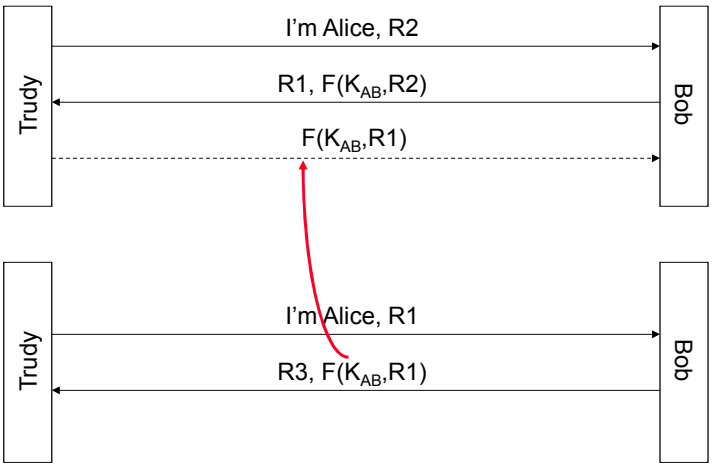
 CLACRYPT Slide 20

Ušetření některých zpráv



©Petr Hanáček CLACRYPT Slide 21

Reflexní útok (reflection attack):



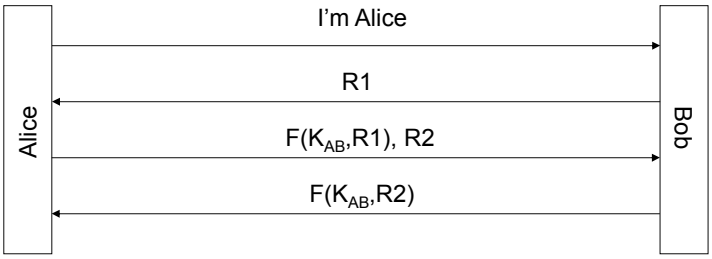
©Petr Hanáček CLACRYPT Slide 22

- **Řešení:**
  - **Různé klíče pro Alici a Boba**
  - **Formátované výzvy, různé pro Alici a Boba**
- **Jiný princip: iniciující strana by měla prokázat svou identitu první**

 CLACRYPT Slide 23

Další slabina: Trudy může provést slovníkový útok na K_{AB} vydávajíce se za Alici, bez odposlouchávání.

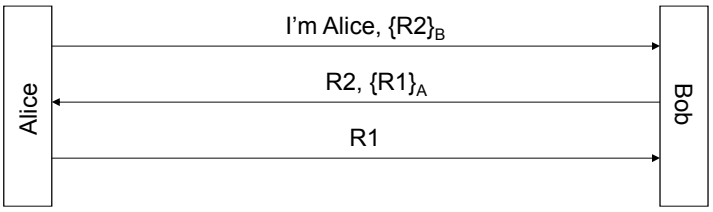
Řešení obou problémů – vyzývající strana se autentizuje první:



(Slovníkový útok je stále možný, pokud se Trudy může vydávat za Boba)

 CLACRYPT Slide 24

Vzájemná autentizace s PKC:



- **Problém: Jak si běžní uživatelé budou pamatovat veřejné a soukromé klíče?**
- **Je vhodné je získávat zašifrovaně ze serveru na základě autentizace heslem?**
  - **Útok na heslo**
  - **Obtížná změna**

 CLACRYPT Slide 25

Vytvoření klíče relace (sezení)

- **Je třeba pro autentizaci a šifrování zbytku relace Musí být:**
  - **Různý pro každou relaci**
  - **Neuhodnutelný útočníkem který odposlouchává**
  - **Nesmí být $K_{AB}\{x\}$ pro x, které je předpověditelné**
- **Se symetrickou kryptografií např.:**

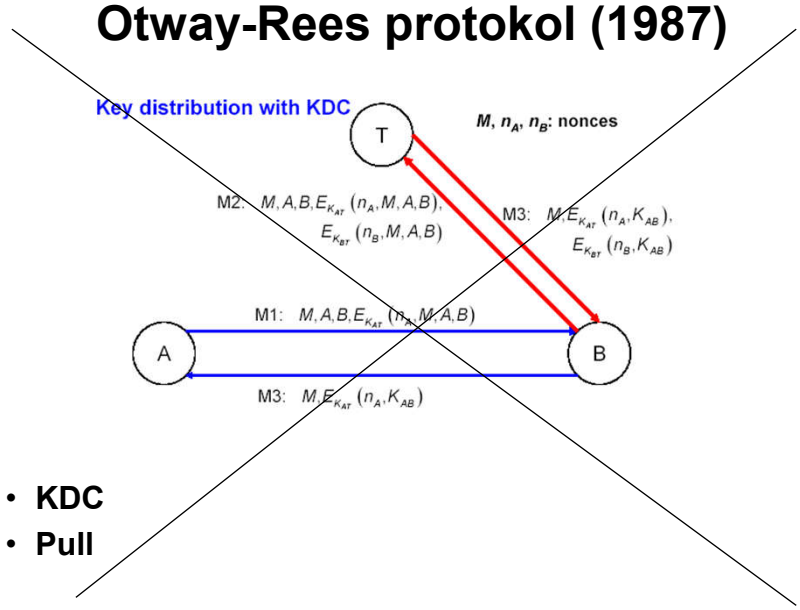
  $(K_{AB}+1)\{R\}$

  (Proč ne $K_{AB}\{R\}$ nebo $K_{AB}\{R+1\}$?)
- **S PKC se zašlou dodatečné náhodné výzvy $\{R\}_A$, $\{R\}_B$ a z nich se vtvoří klíč relace (jako u SSL).**

 CLACRYPT Slide 26

Příklady protokolů

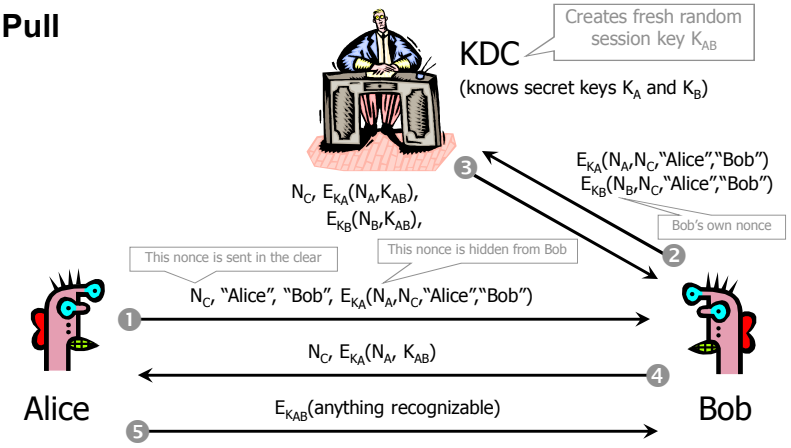
Otway-Rees protokol (1987)



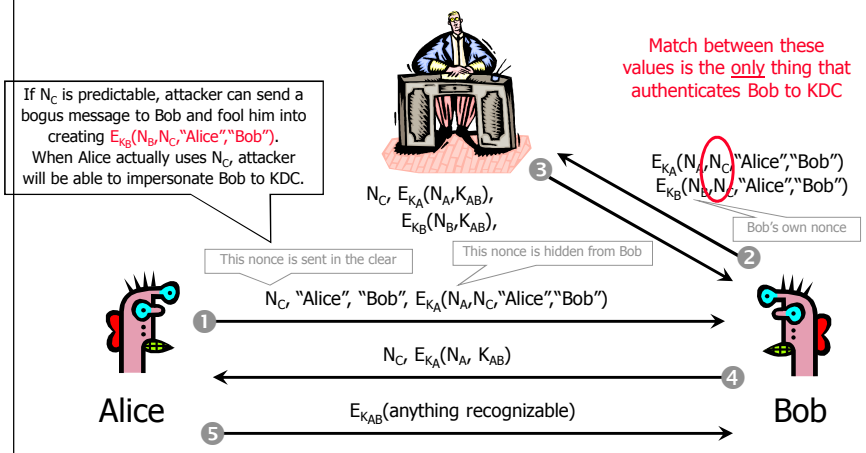
- **KDC**
- **Pull**

Protokol Otway-Rees

- KDC
- Pull

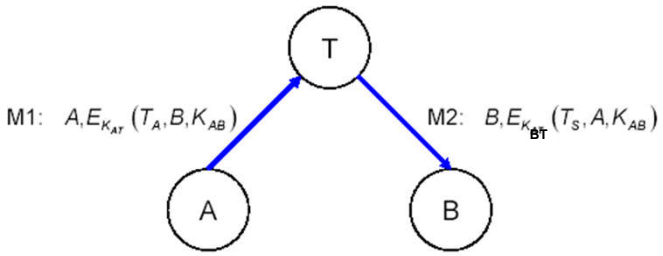


Analýza Otway-Rees



- **Poznatek: náhodnost náhodných výzev je zásadní**

Wide-Mouthed Frog

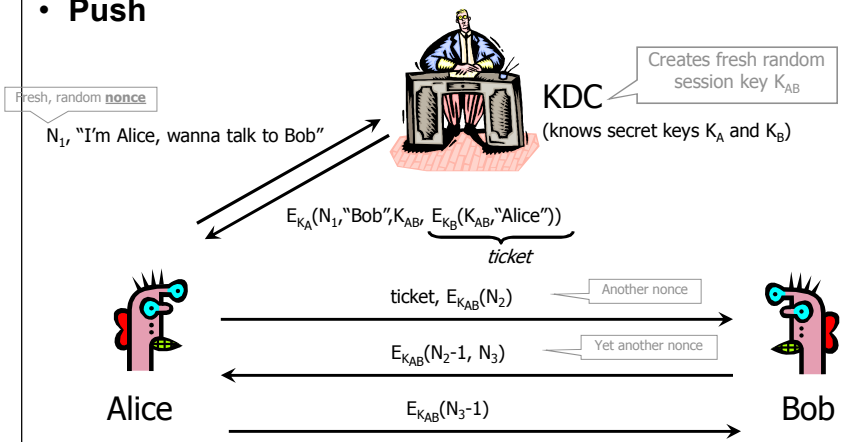


- **KTC**
- **Hybridní model**
- **Časové značky**
- **B věří, že A generuje vhodné klíče**
- **Později objeven problém s periodickou žádostí o přešifrování klíče a „prodlužováním“ ¨časové známky**

©Petr Hanáček CLACRYPT Slide 31

Needham-Schroeder

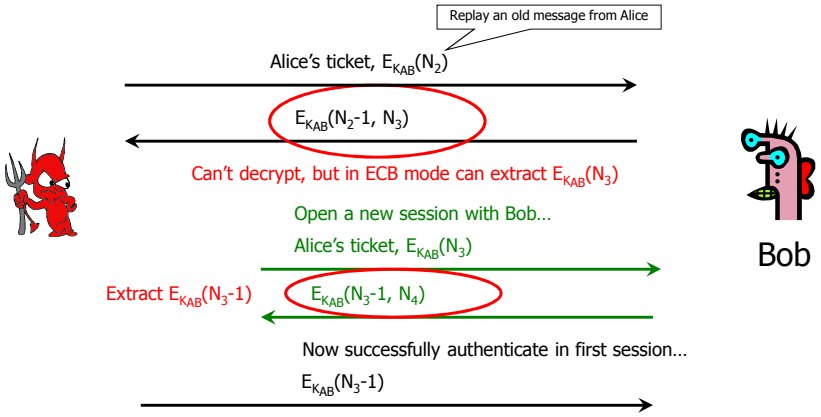
- **KDC**
- **Push**



©Petr Hanáček CLACRYPT Slide 32

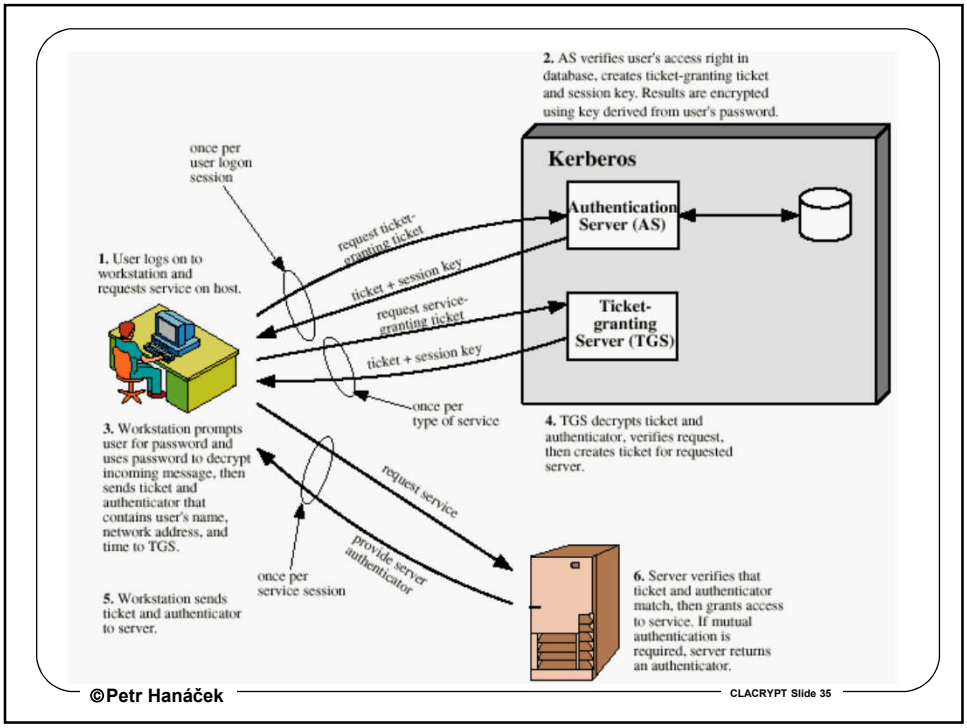
Weird Reflection Attack

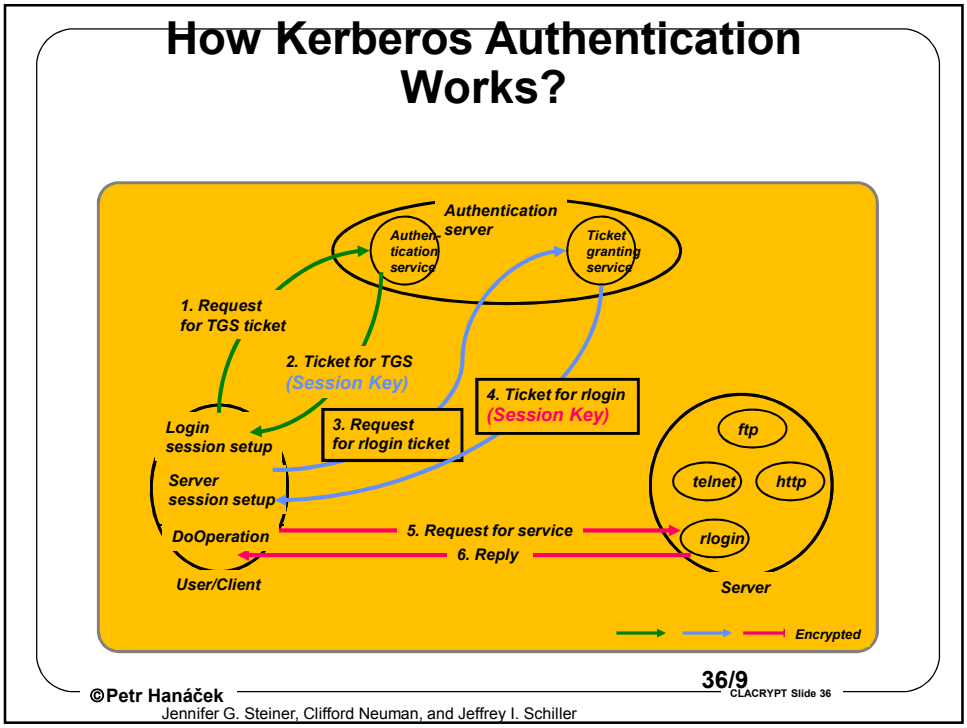
- **Předpokládejme symetrickou šifru v ECB režimu…**
  - **Obecně nevhodné**

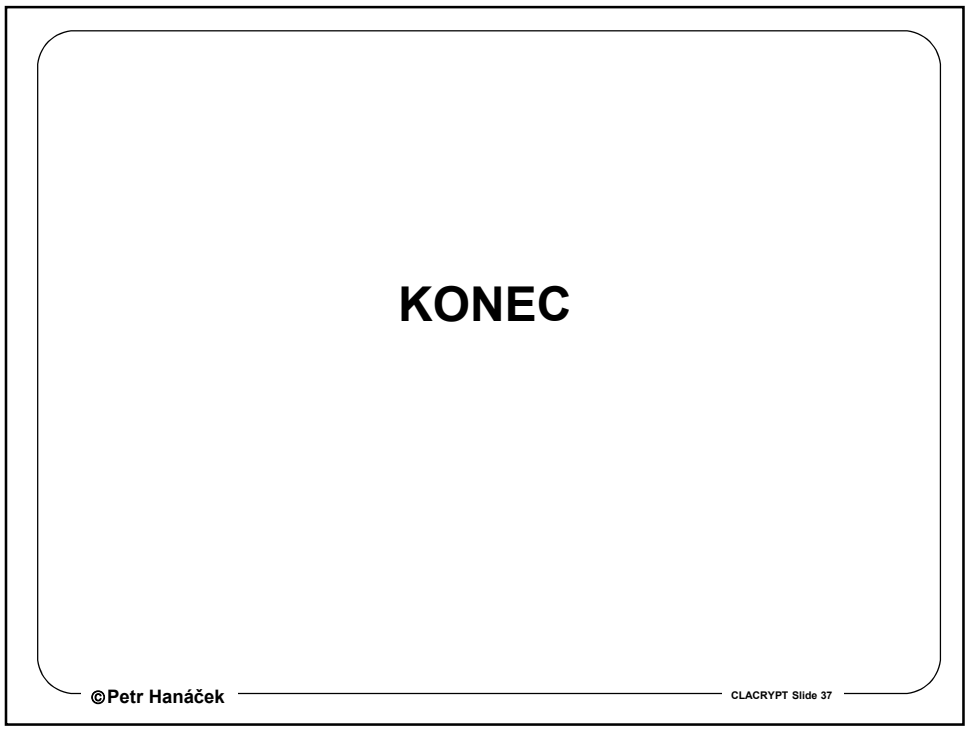


Kerberos v4 a v5

- **Protokol pro autentizaci a bezpečnou komunikaci počítačů propojených komunikační sítí**
- **Žadatelům se přidělují bezpečné „tickety“ které slouží pro navázání bezpečné komunikace**
- **Standardní formáty zabezpečených zpráv**
- **Zprávy zachycují i jména, doby expirace,…**
- **Mechanismus centrálního KDC (nazývá se AS – Authentication Server)**
- **Podporuje „slave KDC“, které mohou provádět pouze omezené operace (TGS – Ticket Granting Server)**


2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
1. User logs on to workstation and requests service on host.
request ticket-granting ticket
ticket + session key
request service-granting ticket
ticket + session key
once per type of service
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
request service
provide server authenticator
once per service session
5. Workstation sends ticket and authenticator to server.
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.
©Petr Hanáček
CLACRYPT Slide 35


How Kerberos Authentication Works?
Authentication server
Authen-tication service
Ticket granting service
1. Request for TGS ticket
2. Ticket for TGS (Session Key)
4. Ticket for rlogin (Session Key)
3. Request for rlogin ticket
Login session setup
Server session setup
DoOperation
5. Request for service
6. Reply
User/Client
ftp
telnet
http
rlogin
Server
Encrypted
©Petr Hanáček
36/9
CLACRYPT Slide 36
Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller


KONEC
©Petr Hanáček
CLACRYPT Slide 37

CHAPTER 9

Symmetric Key Distribution

Chapter Goals

- To understand the problems associated with managing and distributing secret keys.
- To learn about key distribution techniques based on symmetric key based protocols.
- To introduce the formal analysis of protocols.

1. Key Management

To be able to use symmetric encryption algorithms such as DES or Rijndael we need a way for the two communicating parties to share the secret key. In this first section we discuss some issues related to how keys are managed, in particular

- key distribution,
- key selection,
- key lifetime,

But before we continue we need to distinguish between different types of keys. The following terminology will be used throughout this chapter and beyond:

- **Static (or long-term) Keys:** These are keys which are to be in use for a long time period. The exact definition of long will depend on the application, but this could mean from a few hours to a few years. The compromise of a static key is usually considered to be a major problem, with potentially catastrophic consequences.
- **Ephemeral, or Session (or short-term) Keys:** These are keys which have a short life-time, maybe a few seconds or a day. They are usually used to provide confidentiality for the given time period. The compromise of a session key should only result in the compromise of that session's secrecy and it should not affect the long-term security of the system.

1.1. Key Distribution. Key distribution is one of the fundamental problems with cryptography. There are a number of solutions to this problem; which one of these one chooses depends on the overall situation.

- **Physical Distribution:** Using trusted couriers or armed guards, keys can be distributed using traditional physical means. Until the 1970s this was in effect the only secure way of distributing keys at system setup. It has a large number of physical problems associated with it, especially scalability, but the main drawback is that security no longer rests with the key but with the courier. If we can bribe, kidnap or kill the courier then we have broken the system.
- **Distribution Using Symmetric Key Protocols:** Once some secret keys have been distributed between a number of users and a trusted central authority, we can use the trusted authority to help generate keys for any pair of users as the need arises. Protocols to perform this task will be discussed in this chapter. They are usually very efficient but

have some drawbacks. In particular they usually assume that both the trusted authority and the two users who wish to agree on a key are both on-line. They also still require a physical means to set the initial keys up.

- **Distribution Using Public Key Protocols:** Using public key cryptography, two parties, who have never met or who do not trust any one single authority, can produce a shared secret key. This can be done in an on-line manner, using a key exchange protocol. Indeed this is the most common application of public key techniques for encryption. Rather than encrypting large amounts of data by public key techniques we agree a key by public key techniques and then use a symmetric cipher to actually do the encryption.

To understand the scale of the problem, if our system is to cope with n separate users, and each user may want to communicate securely with any other user, then we require

$$\frac{n(n-1)}{2}$$

separate secret keys. This soon produces huge key management problems; a small university with around 10 000 students would need to have around fifty million separate secret keys.

With a large number of keys in existence one finds a large number of problems. For example what happens when your key is compromised? In other words someone else has found your key. What can you do about it? What can they do? Hence, a large number of keys produces a large key management problem.

One solution is for each user to hold only one key with which it communicates with a central authority, hence a system with n users will only require n keys. When two users wish to communicate they generate a secret key which is only to be used for that message, a so-called session key. This session key can be generated with the help of the central authority using one of the protocols that appear later in this chapter.

1.2. Key Selection. The keys which one uses should be truly random, since otherwise an attacker may be able to determine information simply by knowing the more likely keys and the more likely messages, as we saw in a toy example in Chapter 5. All keys should be equally likely and really need to be generated using a true random number generator, however such a good source of entropy is hard to find.

Whilst a truly random key will be very strong, it is hard for a human to remember. Hence, many systems use a password or pass phrase to generate a secret key. But now one needs to worry even more about brute force attacks. As one can see from the following table, a typical PIN-like password of a number between 0 and 9999 is easy to mount a brute force attack against, but even using eight printable characters does not push us to the 2^{80} possibilities that we would like to ensure security.

Key size	Decimal digits	Printable characters
4	$10^4 \approx 2^{13}$	$10^7 \approx 2^{23}$
8	$10^8 \approx 2^{26}$	$10^{15} \approx 2^{50}$

One solution may be to use long pass phrases of 20–30 characters, but these are likely to lack sufficient entropy since we have already seen that natural language is not very random.

Short passwords based on names or words are a common problem in many large organizations. This is why a number of organizations now have automatic checking that passwords meet certain criteria such as

- at least one lower case letter,
- at least one upper case letter,
- at least one numeric character,
- at least one non-alpha-numeric character,
- at least eight characters in length.

But such rules, even though they eliminate the chance of a dictionary attack, still reduce the number of possible passwords from what they would be if they were chosen uniformly at random from all choices of eight printable characters.

1.3. Key Lifetime. One issue one needs to consider when generating and storing keys is the key lifetime. A general rule is that the longer the key is in use the more vulnerable it will be and the more valuable it will be to an attacker. We have already touched on this when mentioning the use of session keys. However, it is important to destroy keys properly after use. Relying on an operating system to delete a file by typing `del`/`rm` does not mean that an attacker cannot recover the file contents by examining the hard disk. Usually deleting a file does not destroy the file contents, it only signals that the file's location is now available for overwriting with new data. A similar problem occurs when deleting memory in an application.

1.4. Secret Sharing. As we have mentioned already the main problem is one of managing the secure distribution of keys. Even a system which uses a trusted central authority needs some way of getting the keys shared between the centre and each user out to the user.

One possible solution is key splitting (more formally called *secret sharing*) where we divide the key into a number of shares

$$K = k_1 \oplus k_2 \oplus \cdots \oplus k_r.$$

Each share is then distributed via separate routes. The beauty of this is that an attacker needs to attack all the routes so as to obtain the key. On the other hand attacking one route will stop the legitimate user from recovering the key.

We will discuss secret sharing in more detail in Chapter 23.

2. Secret Key Distribution

Recall, if we have n users each of whom wish to communicate securely with each other then we would require

$$\frac{n(n-1)}{2}$$

separate long-term key pairs. As remarked earlier this leads to huge key management problems and issues related to the distribution of the keys. We have already mentioned that it is better to use session keys and few long-term keys, but we have not explained how one deploys the session keys.

To solve this problem the community developed a number of protocols which make use of symmetric key cryptography to distribute secret session keys, some of which we shall describe in this section. Later on we shall look at public key techniques for this problem, which are often more elegant.

2.1. Notation. We first need to set up some notation to describe the protocols. Firstly we set up the names of the parties and quantities involved.

- **Parties/Principals:** A, B, S.
  Assume the two parties who wish to agree a secret are A and B, for Alice and Bob. We assume that they will use a trusted third party, or TTP, which we shall denote by S.
- **Shared Secret Keys:** K_{ab}, K_{bs}, K_{as}.
  K_{ab} will denote a secret key known only to A and B.
- **Nonces:** N_a, N_b.
  Nonces are numbers used only once, they should be random. The quantity N_a will denote a nonce originally produced by the principal A. Note, other notations for nonces are possible and we will introduce them as the need arises.

- **Timestamps:** T_a, T_b, T_s.
  The quantity T_a is a timestamp produced by A. When timestamps are used we assume that the parties try to keep their clocks in synchronization using some other protocol.

The statement

$$A \longrightarrow B : M, A, B, \{N_a, M, A, B\}_{K_{as}},$$

means A sends to B the message to the right of the colon. The message consists of

- a nonce M,
- A the name of party A,
- B the name of party B,
- a message $\{N_a, M, A, B\}$ encrypted under the key K_{as} which A shares with S. Hence, the recipient B is unable to read the encrypted part of this message.

Before presenting our first protocol we need to decide the goals of key agreement and key transport, and what position the parties start from. We assume all parties, A and B say, only share secret keys, K_{as} and K_{bs} with the trusted third party S. They want to agree/transport a session key K_{ab} for a communication between themselves.

We also need to decide what capabilities an attacker has. As always we assume the worst possible situation in which an attacker can intercept any message flow over the network. She can then stop a message, alter it or change its destination. An attacker is also able to distribute her own messages over the network. With such a high-powered attacker it is often assumed that the attacker *is* the network.

This new session key should be fresh, i.e. it has not been used by any other party before and has been recently created. The freshness property will stop attacks whereby the adversary replays messages so as to use an old key again. Freshness also can be useful in deducing that the party with which you are communicating is still alive.

2.2. Wide-Mouth Frog Protocol. Our first protocol is the Wide-Mouth Frog protocol, which is a simple protocol invented by Burrows. The protocol transfers a key K_{ab} from A to B via S, it uses only two messages but has a number of drawbacks. In particular it requires the use of synchronized clocks, which can cause a problem in implementations. In addition the protocol assumes that A chooses the session key K_{ab} and then transports this key over to user B. This implies that user A is trusted by user B to be competent in making and keeping keys secret. This is a very strong assumption and the main reason that this protocol is not used much in real life. However, it is very simple and gives a good example of how to analyse a protocol formally, which we shall come to later in this chapter.

The protocol proceeds in the following steps, as illustrated in Fig. 1,

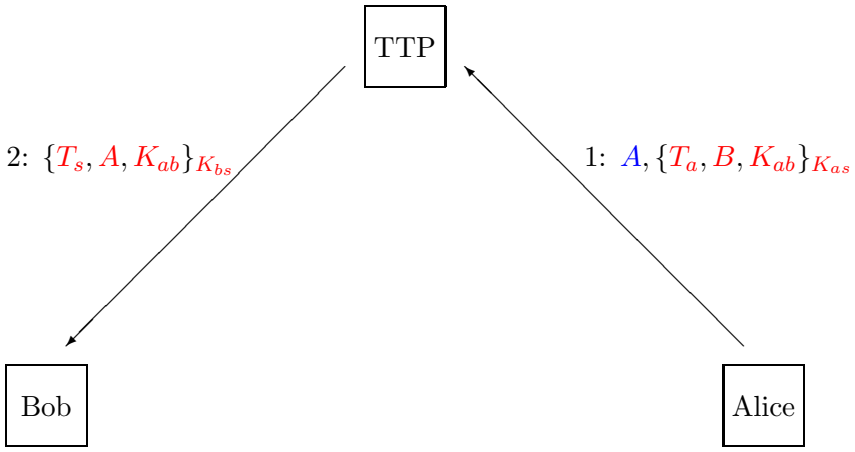
$$A \longrightarrow S : A, \{T_a, B, K_{ab}\}_{K_{as}},$$
$$S \longrightarrow B : \{T_s, A, K_{ab}\}_{K_{bs}}.$$

On obtaining the first message the trusted third party S decrypts the last part of the message and checks that the timestamp is recent. This decrypted message tells S he should forward the key to the party called B. If the timestamp is verified to be recent, S encrypts the key along with his timestamp and passes this encryption onto B. On obtaining this message B decrypts the message received and checks the time stamp is recent, then he can recover both the key K_{ab} and the name A of the person who wants to send data to him using this key.

The checks on the timestamps mean the session key should be recent, in that it left user A a short time ago. However, user A could have generated this key years ago and stored it on his hard disk, in which time Eve broke in and took a copy of this key.

We already said that this protocol requires that all parties need to keep synchronized clocks. However, this is not such a big problem since S checks or generates all the timestamps used in the

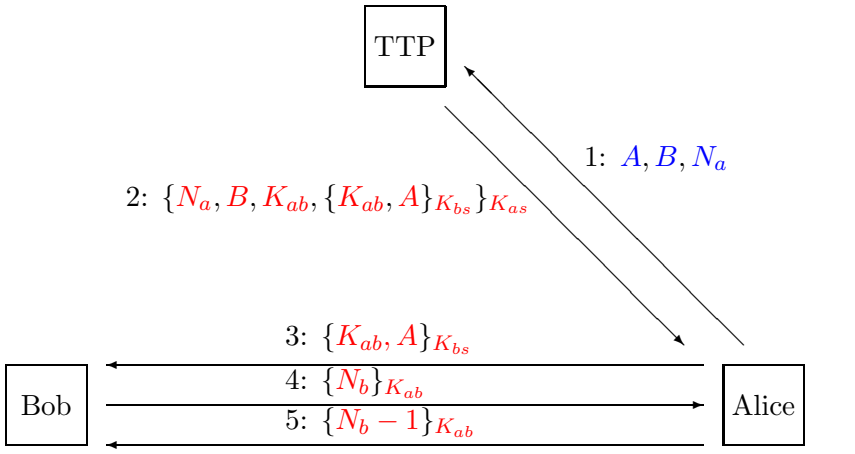
FIGURE 1. Wide-Mouth Frog protocol



protocol. Hence, each party only needs to record the difference between its clock and the clock owned by S. Clocks are then updated if a clock drift occurs which causes the protocol to fail.

This protocol is really too simple; much of the simplicity comes by assuming synchronized clocks and by assuming party A can be trusted with creating session keys.

FIGURE 2. Needham–Schroeder protocol



2.3. Needham–Schroeder Protocol. We shall now look at more complicated protocols, starting with one of the most famous namely, the Needham–Schroeder protocol. This protocol was developed in 1978, and is one of most highly studied protocols ever; its fame is due to the fact that even a simple protocol can hide security flaws for a long time. The basic message flows are described as follows, as illustrated in Fig. 2,

$$
\begin{aligned}
A &\longrightarrow S : A, B, N_a, \\
S &\longrightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}, \\
A &\longrightarrow B : \{K_{ab}, A\}_{K_{bs}},
\end{aligned}
$$

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

We now look at each message in detail, and explain what it does.

- The first message tells S that A wants a key to communicate with B.
- In the second message S generates the session key K_{ab} and sends it back to A. The nonce N_a is included so that A knows this was sent after her request of the first message. The session key is also encrypted under the key K_{bs} for sending to B.
- The third message conveys the session key to B.
- B needs to check that the third message was not a replay. So he needs to know if A is still alive, hence, in the fourth message he encrypts a nonce back to A.
- In the final message, to prove to B that she is still alive, A encrypts a simple function of B's nonce back to B.

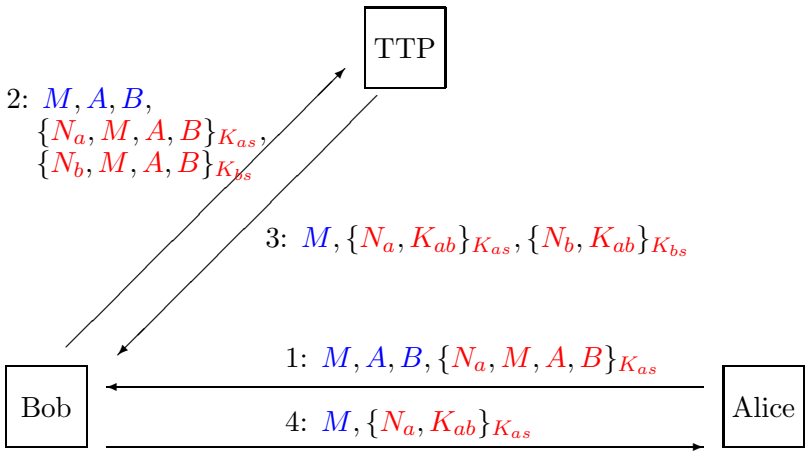
The main problem with the Needham–Schroeder protocol is that B does not know that the key he shares with A is fresh, a fact which was not spotted until some time after the original protocol was published. An adversary who finds an old session transcript can, after finding the old session key by some other means, use the old session transcript in the last three messages involving B. Hence, the adversary can get B to agree to a key with the adversary, which B thinks he is sharing with A.

Note, A and B have their secret session key generated by S and so neither party needs to trust the other to produce 'good' keys. They of course trust S to generate good keys since S is an authority trusted by everyone. In some applications this last assumption is not valid and more involved algorithms, or public key algorithms, are required. In this chapter we shall assume everyone trusts S to perform correctly any action we require of him.

2.4. Otway–Rees Protocol. The Otway–Rees protocol from 1987 is not used that much, but again it is historically important. Like the Needham–Schroeder protocol it does not use synchronized clocks, but again it suffers from a number of problems.

As before two people wish to agree a key using a trusted server S. There are two nonces N_a and N_b used to flag certain encrypted components as recent. In addition a nonce M is used to flag that the current set of communications are linked. The Otway–Rees protocol is shorter than the Needham–Schroeder protocol since it only requires four messages, but the message types are very different. As before the server generates the key K_{ab} for the two parties.

FIGURE 3. Otway–Rees protocol



The message flows in the Otway–Rees protocol are as follows, as illustrated in Fig. 3,

$$
\begin{aligned}
A &\longrightarrow B : M, A, B, \{N_a, M, A, B\}_{K_{as}}, \\
B &\longrightarrow S : M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{N_b, M, A, B\}_{K_{bs}}, \\
S &\longrightarrow B : M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}, \\
B &\longrightarrow A : M, \{N_a, K_{ab}\}_{K_{as}}.
\end{aligned}
$$

Since the protocol does not make use of K_{ab} as an encryption key, neither party knows whether the key is known to each other. We say that Otway–Rees is a protocol which does not offer *key confirmation*. Let us see what the parties do know: A knows that B sent a message containing a nonce N_a which A knows to be fresh, since A originally generated the nonce. So B must have sent a message recently. On the other hand B has been told by the server that A used a nonce, but B has no idea whether this was a replay of an old message.

2.5. Kerberos. We end this section by looking at Kerberos. Kerberos is an authentication system based on symmetric encryption with keys shared with an authentication server; it is based on ideas underlying the Needham–Schroeder protocol. Kerberos was developed at MIT around 1987 as part of Project Athena. A modified version of this original version of Kerberos is now used in Windows 2000.

The network is assumed to consist of clients and a server, where the clients may be users, programs or services. Kerberos keeps a central database of clients including a secret key for each client, hence Kerberos requires a key space of size $O(n)$ if we have n clients. Kerberos is used to provide authentication of one entity to another and to issue session keys to these entities.

In addition Kerberos can run a ticket granting system to enable access control to services and resources. The division between authentication and access is a good idea which we shall see later echoed in SPKI. This division mirrors what happens in real companies. For example, in a company the personnel department administers who you are, whilst the computer department administers what resources you can use. This division is also echoed in Kerberos with an authentication server and a ticket generation server TGS. The TGS gives tickets to enable users to access resources, such as files, printers, etc.

Suppose A wishes to access a resource B. First A logs onto the authentication server using a password. The user A is given a ticket from this server encrypted under her password. This ticket contains a session key K_{as}. She now uses K_{as} to obtain a ticket from the TGS S to access the resource B. The output of the TGS is a key K_{ab} , a timestamp T_S and a lifetime L. The output of the TGS is used to authenticate A in subsequent traffic with B.

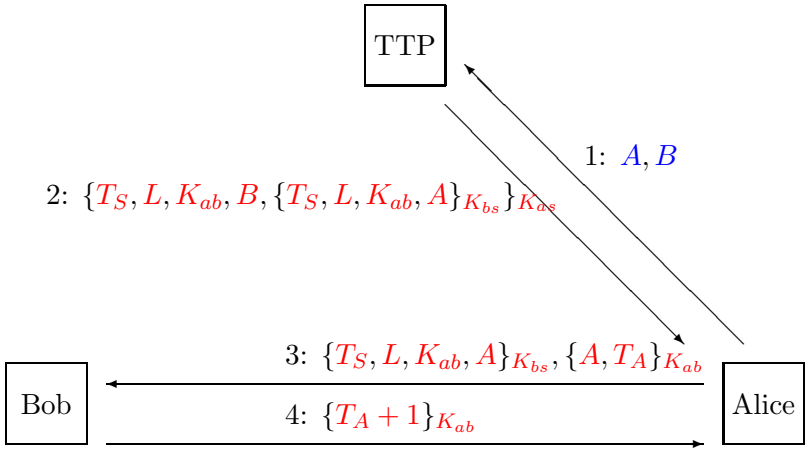
The flows look something like those given in Fig. 4,

$$
\begin{aligned}
A &\longrightarrow S : A, B, \\
S &\longrightarrow A : \{T_S, L, K_{ab}, B, \{T_S, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}, \\
A &\longrightarrow B : \{T_S, L, K_{ab}, A\}_{K_{bs}}, \{A, T_A\}_{K_{ab}}, \\
B &\longrightarrow A : \{T_A + 1\}_{K_{ab}}.
\end{aligned}
$$

- The first message is A telling S that she wants to access B.
- If S allows this access then a ticket $\{T_S, L, K_{ab}, A\}$ is created. This is encrypted under K_{bs} and sent to A for forwarding to B. The user A also gets a copy of the key in a form readable by her.
- The user A wants to verify that the ticket is valid and that the resource B is alive. Hence, she sends an encrypted nonce/timestamp T_A to B.
- The resource B sends back the encryption of $T_A + 1$, after checking that the timestamp T_A is recent, thus proving he knows the key and is alive.

We have removed the problems associated with the Needham–Schroeder protocol by using timestamps, but this has created the requirement for synchronized clocks.

FIGURE 4. Kerberos



3. Formal Approaches to Protocol Checking

One can see that the above protocols are very intricate; spotting flaws in them can be a very subtle business. To try and make the design of these protocols more scientific a number of formal approaches have been proposed. The most influential of these is the BAN logic invented by Burrows, Abadi and Needham.

The BAN logic has a large number of drawbacks but was very influential in the design and analysis of symmetric key based key agreement protocols such as Kerberos and the Needham–Schroeder protocol. It has now been supplanted by more complicated logics and formal methods, but it is of historical importance and the study of the BAN logic can still be very instructive for protocol designers.

The main idea of BAN logic is that one should concentrate on what the parties believe is happening. It does not matter what is actually happening, we need to understand exactly what each party can logically deduce, from its own view of the protocol, as to what is actually happening. Even modern approaches to modelling PKI have taken this approach and so we shall now examine the BAN logic in more detail.

We first introduce the notation

- $P| \equiv X$ means P *believes* (or is entitled to believe) X.
  The principal P may act as though X is true.
- $P \triangleleft X$ means P *sees* X.
  Someone has sent a message to P containing X, so P can now read and repeat X.
- $P| \sim X$ means P *once said* X and P believed X when it was said.
  Note this tells us nothing about whether X was said recently or in the distant past.
- $P| \Rightarrow X$ means P has *jurisdiction* over X.
  This means P is an authority on X and should be trusted on this matter.
- $\#X$ means the formula X is *fresh*.
  This is usually used for nonces.
- $P \overset{K}{\leftrightarrow} Q$ means P and Q may use the *shared key* K to communicate.
  The key is assumed good and it will never be discovered by anyone other than P and Q, unless the protocol itself makes this happen.

- $\{X\}_K$, as usual this means X is *encrypted* under the key K.
  The encryption is assumed to be perfect in that X will remain secret unless deliberately disclosed by a party at some other point in the protocol.

In addition, conjunction of statements is denoted by a comma.

There are many postulates, or rules of inference, specified in the BAN logic. We shall only concentrate on the main ones. The format we use to specify rules of inference is as follows:

$$\frac{A, B}{C}$$

which means that if A and B are true then we can conclude C is also true. This is a standard notation used in many areas of logic within computer science.

Message Meaning Rule

$$\frac{A| \equiv A \overset{K}{\leftrightarrow} B, A \triangleleft \{X\}_K}{A| \equiv B| \sim X}.$$

In words, if both

- A believes she shares the key K with B,
- A sees X encrypted under the key K,

we can deduce that A believes that B once said X. Note that this implicitly assumes that A never said X.

Nonce Verification Rule

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

In words, if both

- A believes X is fresh (i.e. recent),
- A believes B once said X,

then we can deduce that A believes that B still believes X.

Jurisdiction Rule

$$\frac{A| \equiv B| \Rightarrow X, A| \equiv B| \equiv X}{A| \equiv X}.$$

In words, if both

- A believes B has jurisdiction over X, i.e. A trusts B on X,
- A believes B believes X,

then we conclude that A also believes X.

Other Rules

The belief operator and conjunction can be manipulated as follows:

$$\frac{P| \equiv X, P| \equiv Y}{P| \equiv (X, Y)}, \quad \frac{P| \equiv (X, Y)}{P| \equiv X}, \quad \frac{P| \equiv Q| \equiv (X, Y)}{P| \equiv Q| \equiv X}.$$

A similar rule also applies to the 'once said' operator

$$\frac{P| \equiv Q| \sim (X, Y)}{P| \equiv Q| \sim X}.$$

Note that $P| \equiv Q| \sim X$ and $P| \equiv Q| \sim Y$ does not imply $P| \equiv Q| \sim (X, Y)$, since that would imply X and Y were said at the same time. Finally, if part of a formula is fresh then so is the whole formula

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

We wish to analyse a key agreement protocol between A and B using the BAN logic. But what is the goal of such a protocol? The minimum we want to achieve is

$$A| \equiv A \overset{K}{\leftrightarrow} B \text{ and } B| \equiv A \overset{K}{\leftrightarrow} B,$$

i.e. both parties believe they share a secret key with each other.

However, we could expect to achieve more, for example

$$A| \equiv B| \equiv A \overset{K}{\leftrightarrow} B \text{ and } B| \equiv A| \equiv A \overset{K}{\leftrightarrow} B,$$

which is called key confirmation. In words, we may want to achieve that, after the protocol has run, A is assured that B knows he is sharing a key with A, and it is the same key A believes she is sharing with B.

Before analysing a protocol using the BAN logic we convert the protocol into logical statements. This process is called *idealization*, and is the most error prone part of the procedure since it cannot be automated. We also need to specify the assumptions, or axioms, which hold at the beginning of the protocol.

To see this in 'real life' we analyse the Wide-Mouth Frog protocol for key agreement using synchronized clocks.

3.1. Wide-Mouth Frog Protocol. Recall the Wide-Mouth Frog protocol

$$A \longrightarrow S : A, \{T_a, B, K_{ab}\}_{K_{as}},$$
$$S \longrightarrow B : \{T_s, A, K_{ab}\}_{K_{bs}}.$$

This becomes the idealized protocol

$$A \longrightarrow S : \{T_a, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{as}},$$
$$S \longrightarrow B : \{T_s, A| \equiv A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}.$$

One should read the idealization of the first message as telling S that

- T_a is a timestamp/nonce,
- K_{ab} is a key which is meant as a key to communicate with B.

So what assumptions exist at the start of the protocol? Clearly A, B and S share secret keys which in BAN logic becomes

$$A| \equiv A \overset{K_{as}}{\leftrightarrow} S, \qquad S| \equiv A \overset{K_{as}}{\leftrightarrow} S,$$
$$B| \equiv B \overset{K_{bs}}{\leftrightarrow} S, \qquad S| \equiv B \overset{K_{bs}}{\leftrightarrow} S.$$

There are a couple of nonce assumptions,

$$S| \equiv \#T_a \text{ and } B| \equiv \#T_s.$$

Finally, we have the following three assumptions

- B trusts A to invent good keys,

$$B| \equiv (A| \Rightarrow A \overset{K_{ab}}{\leftrightarrow} B),$$

- B trusts S to relay the key from A,

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

- A knows the session key in advance,

$$A| \equiv A \overset{K_{ab}}{\leftrightarrow} B.$$

Notice how these last three assumptions specify the problems we associated with this protocol in the earlier section.

Using these assumptions we can now analyse the protocol. Let us see what we can deduce from the first message

$$A \longrightarrow S : \{T_a, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{as}}.$$

- Since S sees the message encrypted under K_{as} he can deduce that A said the message.
- Since T_a is believed by S to be fresh he concludes the whole message is fresh.
- Since the whole message is fresh, S concludes that A currently believes the whole of it.
- S then concludes

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

  which is what we need to conclude so that S can send the second message of the protocol.

We now look at what happens when we analyse the second message

$$S \longrightarrow B : \{T_s, A| \equiv A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}.$$

- Since B sees the message encrypted under K_{bs} he can deduce that S said the message.
- Since T_s is believed by B to be fresh he concludes the whole message is fresh.
- Since the whole message is fresh, B concludes that S currently believes the whole of it.
- So B believes that S believes the second part of the message.
- But B believes S has authority on whether A knows the key and B believes A has authority to generate the key.

So we conclude

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

and

$$B| \equiv A| \equiv A \overset{K_{ab}}{\leftrightarrow} B.$$

Combining with our axiom $A| \equiv A \overset{K_{ab}}{\leftrightarrow} B$ we conclude that the key agreement protocol is sound. The only requirement we have not met is that

$$A| \equiv B| \equiv A \overset{K_{ab}}{\leftrightarrow} B,$$

i.e. A does not achieve confirmation that B has received the key.

Notice what the application of the BAN logic has done is to make the axioms clearer, so it is easier to compare which assumptions each protocol needs to make it work. In addition it clarifies what the result of running the protocol is from all parties' points of view.

Chapter Summary

- Distributing secret keys used for symmetric ciphers can be a major problem.
- A number of key agreement protocols exist based on a trusted third party and symmetric encryption algorithms. These protocols require long-term keys to have been already established with the TTP, they may also require some form of clock synchronization.
- Various logics exist to analyse such protocols. The most influential of these has been the BAN logic. These logics help to identify explicit assumptions and problems associated with each protocol.

Further Reading

The paper by Burrows, Abadi and Needham is a very readable introduction to the BAN logic and a number of key agreement protocols, much of our treatment is based on this paper. For another, more modern, approach to protocol checking see the book by Ryan et. al., this covers an approach based on the CSP process algebra.

M. Burrows, M. Abadi and R. Needham. *A Logic of Authentication.* Digital Equipment Corporation, SRC Research Report 39, 1990.

P. Ryan, S. Schneider, M. Goldsmith, G. Lowe and B. Ruscoe. *Modelling and analysis of security protocols.* Addison–Wesley, 2001.