Management of Secret Keys

Segurança Informática em Redes e Sistemas
2021/22

Ack: Ricardo Chaves, Miguel Pardal,
Miguel P. Correia, Carlos Ribeiro

Roadmap

- Introduction
- Generation and manual distribution
- Distribution with shared values
- Distribution without shared values
- Distribution with third parties
- Key renewal

Roadmap

- **Introduction**
- Generation and manual distribution
- Distribution with shared values
- Distribution without shared values
- Distribution with third parties
- Key renewal

Secret key management: problems

- Ciphered data is confidential,
  but only if the ciphering key is secret
  - **Distribution/storage** of the keys must assure its **confidentiality and integrity**
- The more unpredictable the generated keys are,
  the harder it is to "guess" their value
  - Key values should be as **random** as possible
- Computers are not good random generators
  - We need to discover and use random data and **random behaviors** in the system
- The excessive use of the keys eases their discovery
  - We need to quantify and impose **limits to the use of a key**

Secret key management: aspects

- Key generation
  - How and when should the secret key be created?
- Key distribution
  - How are the keys distributed to a limited number (typically, 2) of communicating parties?
- Key lifetime
  - For how long should a key be used?

Secret key management: Renewal of keys

- Goal
  - Minimize the cryptanalysis risk
  - Applicable to session keys and long-term keys
- Criteria
  - After a predetermined time interval
    - To avoid its discovery during its usage lifetime
      - That might allow to deterministically modify cryptograms and observe the plaintext
  - After a given amount of exchanged cipher data
    - To avoid the excessive use of the key

Roadmap

- Introduction
- **Generation and manual distribution**
- Distribution with shared values
- Distribution without shared values
- Distribution with third parties
- Key renewal

Generation of secret keys: principles

- Use good random values generators
  - Should be able to generate any of the key values acceptable for the ciphering algorithm
  - Equiprobability of all the key bits
    - Typically generated by pseudo-random generators
    - Validated by randomness test functions
  - Unpredictability of all the key bits
    - Should not be predictable even if the algorithm and all the generation history is known
  - Symmetric ciphers usually have a few weak keys
    - Must be discarded when returned by the random key generator

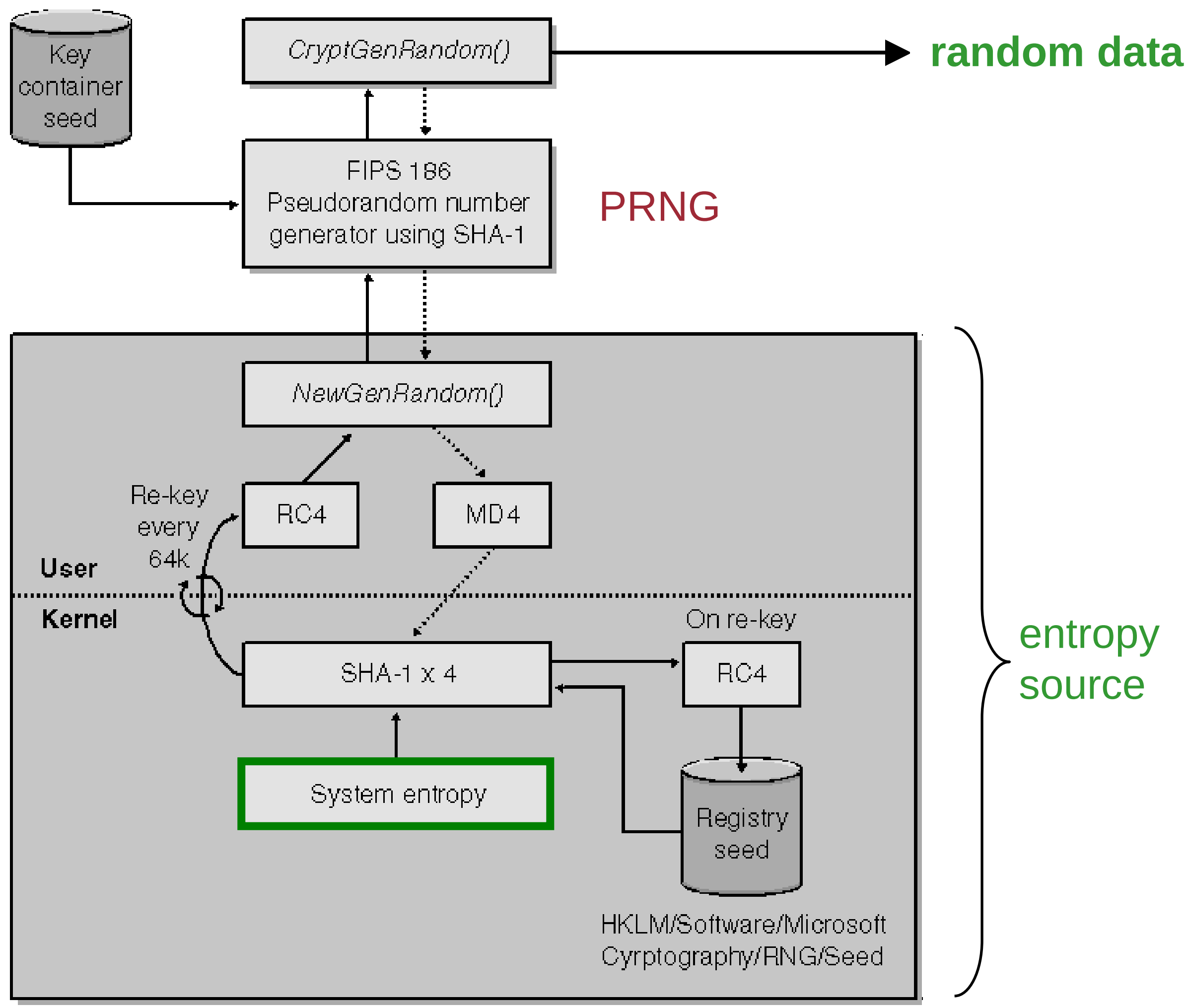
Random number generation: hardware

- Entropy: measure of randomness in a signal/random event
  - High entropy means high randomness (what we need)
  - Concept from Information Theory
- Hardware random number generator (HRNG)
  - A.k.a. true random number generator (TRNG)
  - Hardware device that gets entropy from a physical source

- Example physical sources:
  - Atmospheric noise – read by a radio receiver, for example
  - Thermal or quantum-mechanical noise
    - Amplified to provide a random electrical signal
    - e.g. thermal noise from a resistor
  - Nuclear decay radiation
    - e.g. some commercial smoke alarms, detected by a Geiger counter

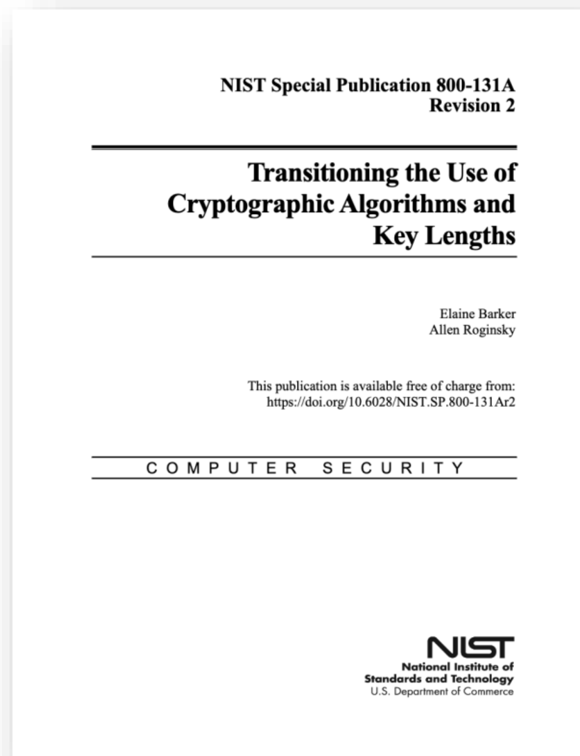
Random number generation:
Windows CryptGenRandom / BCryptGenRandom

- Software random number generator
- Sources of entropy:
  - Ticks since boot
  - Current time
  - Several high-precision performance counters
  - Low-level system info (idle processing time, I/O read and write transfer counts, …)



Generation of secret keys: size

- What should the size of a secret key be? Depends on:
  - Algorithm strength
  - Lifetime of the key
  - Usage of the algorithm + key
  - Attacker's power
- Follow recommendations
  - ENISA, NIST,…


NIST Special Publication 800-131A
Revision 2

Transitioning the Use of Cryptographic Algorithms and Key Lengths

Elaine Barker
Allen Roginsky

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-131Ar2

COMPUTER SECURITY

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Manual distribution (1/2)

- Usefulness:
  - Personal keys – that authenticate a person (e.g. password)
  - Large sets of keys – to be used for long periods of time

- Common requirements:
  - Confidentiality: keys cannot be revealed during generation and distribution
  - Authenticity and integrity: the receiver must be able to check the key’s authenticity and integrity
  - All entities that may have access to the key should be considered
    - System administrators, key distributors, etc.

Manual distribution (2/2)

- Physical support
  - On volatile media
    - e.g. screen showing the new password to the users
  - On paper
    - Typically used to transmit personal keys
      - ATM (Multibanco) or VISA PINs
  - Writable media
    - USB drives, magnetic cards, smartcards

- Distribution
  - On-site
  - Hand-to-hand

Roadmap

- Introduction
- Generation and manual distribution
- **Distribution with shared values**
- Distribution without shared values
- Distribution with third parties
- Key renewal

Distribution with long-term shared secrets (1/3)

- Usefulness
  - Allow exchanging temporary secrets between entities that already share some secret information (long-term secrets)

- Nomenclature
  - Long-term shared secrets
    - Key Encrypting Keys, KEK
  - Temporary secrets to be shared
    - Sessions keys, Ks

Perfect Forward Secrecy (PFS)

- Is a desirable characteristic of a key agreement protocol
- Gives assurance that **session keys will not be compromised even if** the private key of the **server is compromised**
- Protects *past* sessions against *future* compromises of keys
  - By generating a unique session key for every session a user initiates, the compromise of a single session key will not affect any data other than that exchanged in the specific session protected by that particular key

Distribution with long-term shared secrets (2/3)

- Distribution

$$A \rightarrow B: \{Ks\}_{KEK}$$

- Encrypted using a symmetric cipher
- Guarantees authenticity under a set of assumptions:
  - Only A and B know KEK
  - B verifies the message freshness
    - Avoid replay attacks (see later)
  - B verifies the actual content of the message is $\{Ks\}_{KEK}$

Distribution with long-term shared secrets (3/3)

- Practical aspects to consider
  - KEKs should only be used to cipher session keys
    - In order to prevent cryptanalysis
      - The more session data is ciphered, the more the KEK is exposed
  - Perfect Forward Secrecy (PFS) is not assured
    - The disclosure of the KEK reveals all session keys that have been exchanged between the communicating parties
  - A session key should not be used as a KEK
    - Because, by definition, it is or will be extensively exposed by its repeated use

Distribution with shared public keys

- Similar to distribution of keys with shared secrets (keys)
  - No KEKs, but the public key of the receiver
  - Typically designated hybrid ciphers or hybrid encryption
    - Example: PGP (using RSA asymmetric keys)

$$\mathbf{A \rightarrow B: \{Ks\}_{KB\text{-}Pub}}$$

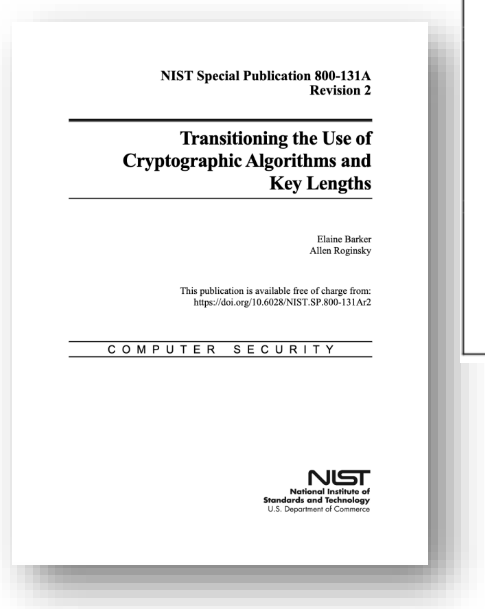
- Does not assume authentication
  - The receiver's public key is used to send the secret
  - Anyone can know the receiver's public key

- Practical aspects to be considered
  - Perfect Forward Secrecy (PFS) not assured: disclosure of the receiver's private key (!) reveals all session keys exchanged

Distribution of secret keys: key size

Table 5: Approval Status for the RSA-based Key Agreement and Key Transport Schemes

Scheme	Implementation Details	Status
SP 800-56B Key Agreement and Key Transport schemes	**len**(n) < 2048	Disallowed
	len(n) ≥ 2048	Acceptable
Non-SP 800-56B-compliant Key Agreement and Key Transport schemes	**len**(n) < 2048	Disallowed
	PKCS1-v1_5 padding	Deprecated through 2023 Disallowed after 2023
	Other non-compliance with SP 800-56B	Deprecated through 2020 Disallowed after 2020


NIST Special Publication 800-131A
Revision 2

Transitioning the Use of Cryptographic Algorithms and Key Lengths

Elaine Barker
Allen Roginsky

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-131Ar2

COMPUTER SECURITY

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Roadmap

- Introduction
- Generation and manual distribution
- Distribution with shared values
- **Distribution without shared values**
- Distribution with third parties
- Key renewal

Distribution without sharing values

- Diffie-Hellman (DH) algorithm
  - DH is seminal asymmetric cryptography algorithm, published in 1976
    - Allows generating a shared key
    - But it is **not** an encryption algorithm
  - In practice, DH values have to be shared:
    - The shared values are not secret, they are public
    - The sharing is ephemeral (temporary), not long-term
  - Participants start with two public parameters: q and α
    - q is a large prime; operations are done modulo q
    - α is the exponentiation base
      - α is a primitive root modulo q, i.e., for every integer a coprime to q, there is an integer *k* such that $\alpha^k \equiv a \pmod{q}$

Diffie-Hellman algorithm (1/3)

- Algorithm (α and q are public)
  - A and B generate random and secret values a and b
  - A computes $\mathbf{y_A = \alpha^a \bmod q}$
    B computes $\mathbf{y_B = \alpha^b \bmod q}$
  - A and B exchange y_A and y_B (public values of DH)
  - A computes $\mathbf{K_s = y_B^a \bmod q}$ $= (\alpha^b \bmod q)^a \bmod q = \alpha^{ba} \bmod q = \alpha^{ab} \bmod q$
  - B computes $\mathbf{K_s = y_A^b \bmod q}$ $= (\alpha^a \bmod q)^b \bmod q = \alpha^{ab} \bmod q$
- The security of the scheme is based on the complexity of the discrete logarithm problem
  - Knowing α, q, y_A and y_B it is unfeasible to obtain a, b and K_s
  - Specifically, it is unfeasible to compute $a = \log_\alpha(y_a)$ (same for b)
- Elliptic curve version exists: ECDH

Diffie-Hellman algorithm (2/3)

- Distribution does not provide authentication
  - Vulnerable to man-in-the-middle attacks
- To authenticate y_A and y_B with digital signatures:
  - A and B must know the other's public key
  - A needs to send y_A with a **signature** obtained with its private key
    - And B needs to sign its value with its private key
  - Example: PGP (with DH/DSS asymmetrical keys)

Diffie-Hellman algorithm (3/3)

- Practical aspects to be considered
  - If both secret values a and b are ephemeral (e.g., used only once), then there is Perfect Forward Secrecy
  - If a or b are long-term secret values and one of them is disclosed, then all keys they helped generate are revealed

Perfect Forward Secrecy (PFS) with DH

- PFS means that the session keys from past sessions are not compromised
  - Even if all secrets of the system are compromised in the present

- PFS means that the messages exchanged between the parties in the past will remain protected

- Way to achieve this:
  - Use DH with ephemeral a and b random values

Roadmap

- Introduction
- Generation and manual distribution
- Distribution with shared values
- Distribution without shared values
- **Distribution with third parties**
- Key renewal

Distribution with a trusted third party (1/3)

- Trusted third parties (Key Distribution Centers)
  - Act as mediators between the communicating parties
    - Distribute credentials for a secure interaction
  - Simplifies the management of long-term shared secrets
    - Avoids need of sharing a secret between any two communicating parties
  - Allows the authentication to be centralized
    - Central point of knowledge of shared secrets

- Assumptions
  - Third party always acts correctly (“trusted”)
    - Do not disclose nor incorrectly use the secrets they know
    - Generate unpredictable/random session keys
  - Are secure, i.e., manage to protect the secrets they store

Distribution with a trusted third party (2/3)

K_A shared between A and KDC; same for K_B

- Distribution
  - Pull model

    1: A → KCD: A, B

    2: KDC → A: $\{K_s\}_{K_A}$, $\{A, K_s\}_{K_B}$

    3: A → B: A, $\{A, K_s\}_{K_B}$

    A ⇔ B: $\{M\}_{K_s}$
  - Push model

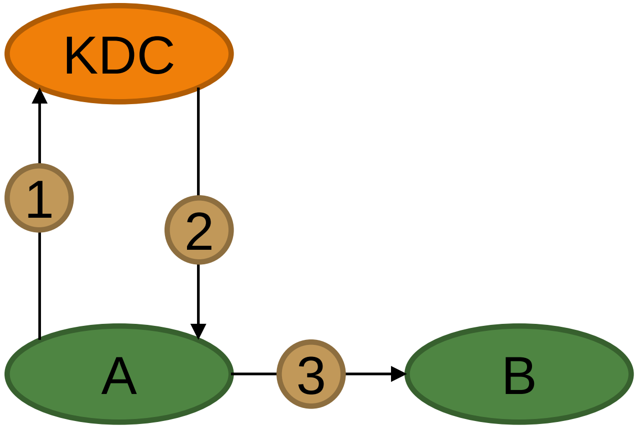
    1: A → B: A

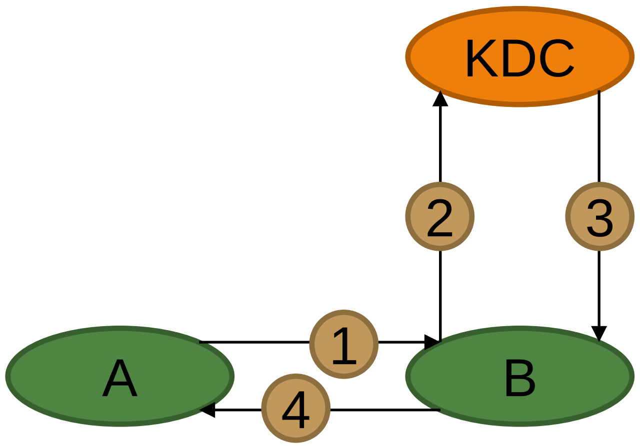
    2: B → KDC: A, B

    3: KDC → B: $\{K_s\}_{K_B}$, $\{B, K_s\}_{K_A}$

    4: B → A: $\{B, K_s\}_{K_A}$

    A ⇔ B: $\{M\}_{K_s}$





Distribution with a trusted third party (3/3)

- Distribution assumes authentication
  - Only those who share a key with the KDC can obtain session key
  - When B receives $\{A, Ks\}_{K_B}$ it is assured that it is receiving a key K_s to communicate with A
- Problems to be solved
  - Message authentication
    - Origin, content, freshness
  - Cooperation between different KDCs
    - Facilitate the key exchange between entities known by different KDCs
- Practical aspects to be considered
  - Perfect Forward Secrecy (PFS) is not assured

Replay attacks

- Messages copied and later resent
- Avoided by guaranteeing message freshness
  - Sequence numbers
  - Timestamps
  - Challenge/Response

Replay attacks (1/3)

- Sequence numbers
  - Sender adds counter value to message content and increments it
  - Receiver checks if counter value received is ok
- Problems
  - Participants need to keep synchronized counters
  - Difficult when message loss or duplication occurs

Replay attacks (2/3)

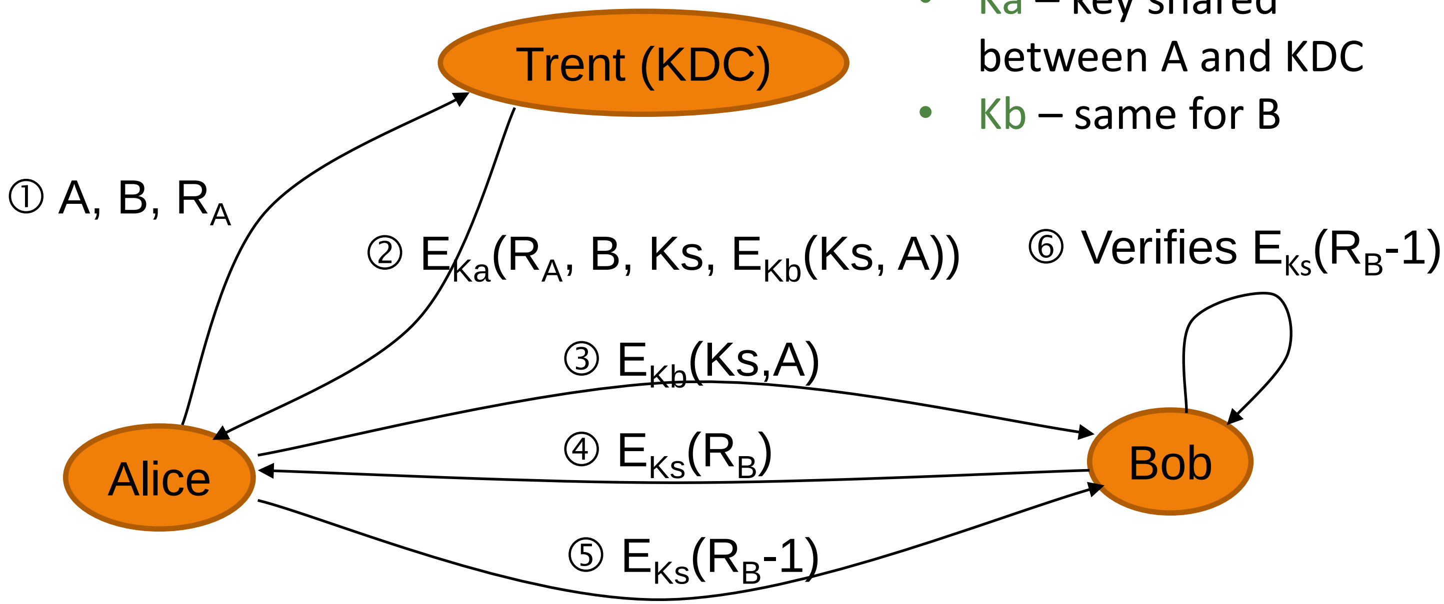
- Timestamps
  - Messages contain a timestamp
  - Messages are only accepted if their timestamps are within a given timeframe
- Frequently used (e.g., in Kerberos), however, problems exist:
  - Clock must be synchronized
  - Tolerance to network delay

Replay attacks (3/3)

- Challenge/Response
  - The communication initiator sends a nonce (number used only once)
  - and waits for that nonce (or its transformation) to come in the reply
- Easy to implement but:
  - More messages are required
  - Needs for both parties to be active
    - Not applicable to communications without a connection

Needham-Schroeder (NS)

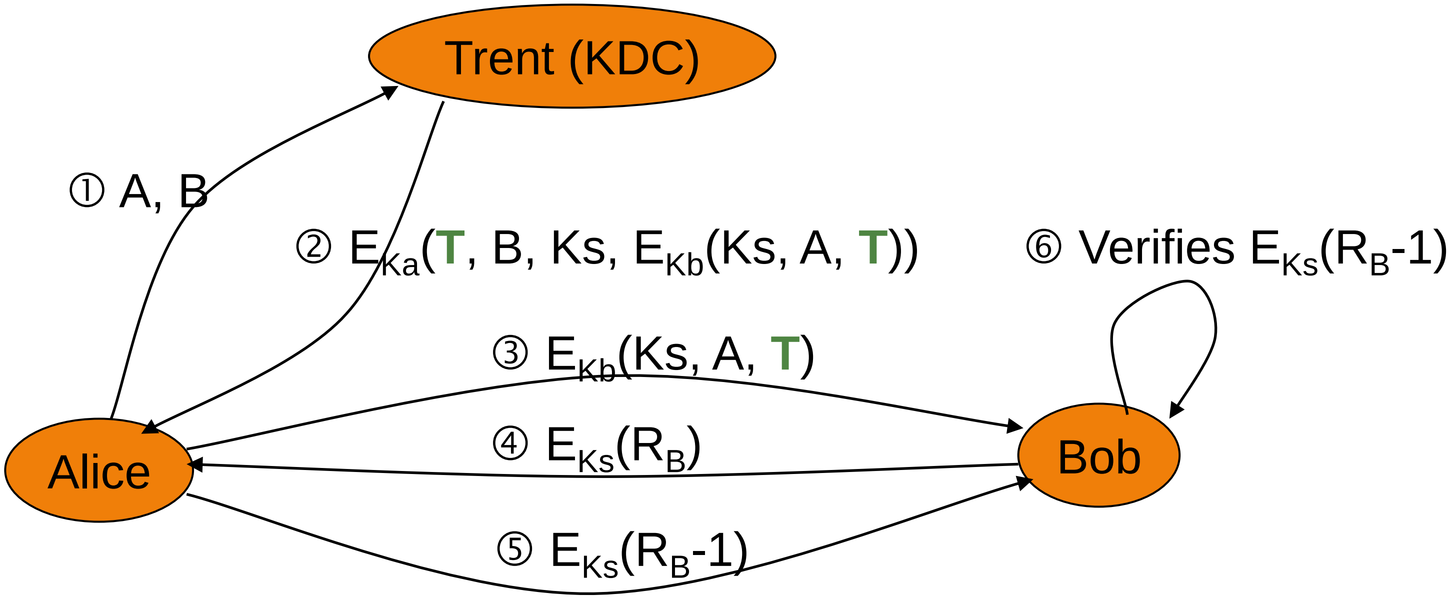
- Ra, Rb – nonces
- Ka – key shared between A and KDC
- Kb – same for B



- Can message ③ be sent directly by Trent to Bob?
- What are the messages ④ and ⑤ used for?
- What happens if someone can obtain/discover a session key?

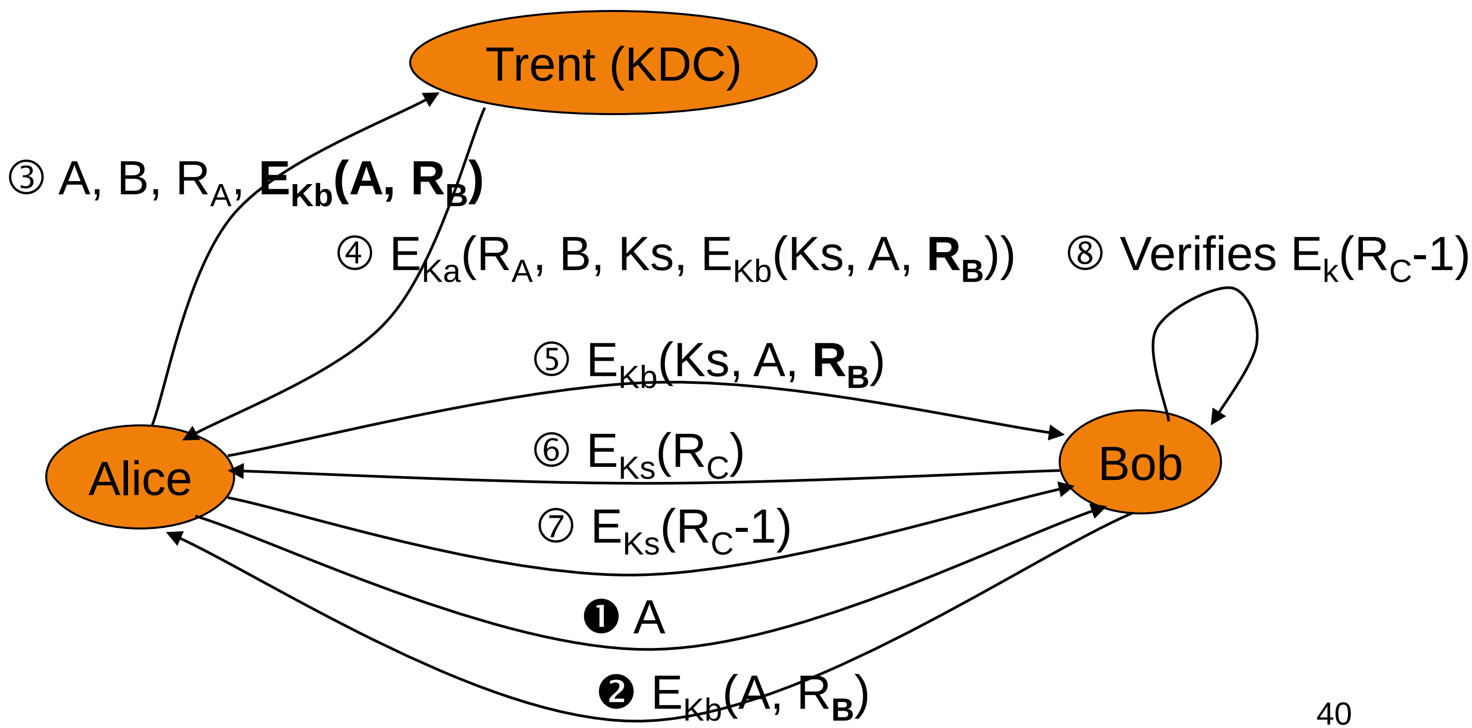
NS with timestamps

- Modification proposed by Dorothy Denning
- Bob accepts message ③ only if it comes within the timeframe
- The time interval to share the session key is limited; no nonces
- Needs clock synchronization



NS revisited

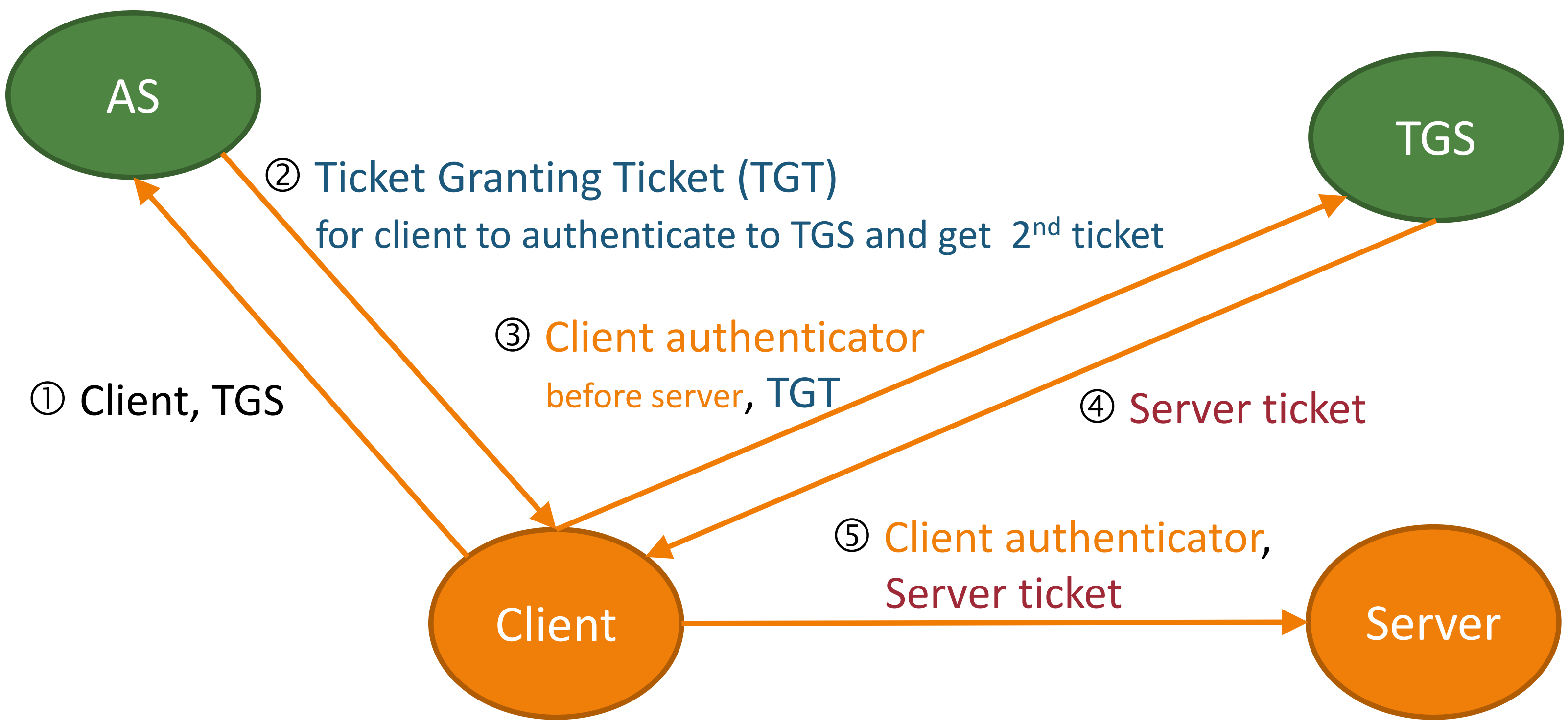
- Modification proposed by Needham & Schroeder
  - Uses nonces to validate the freshness of connection request from A
- Does not need clock synchronization



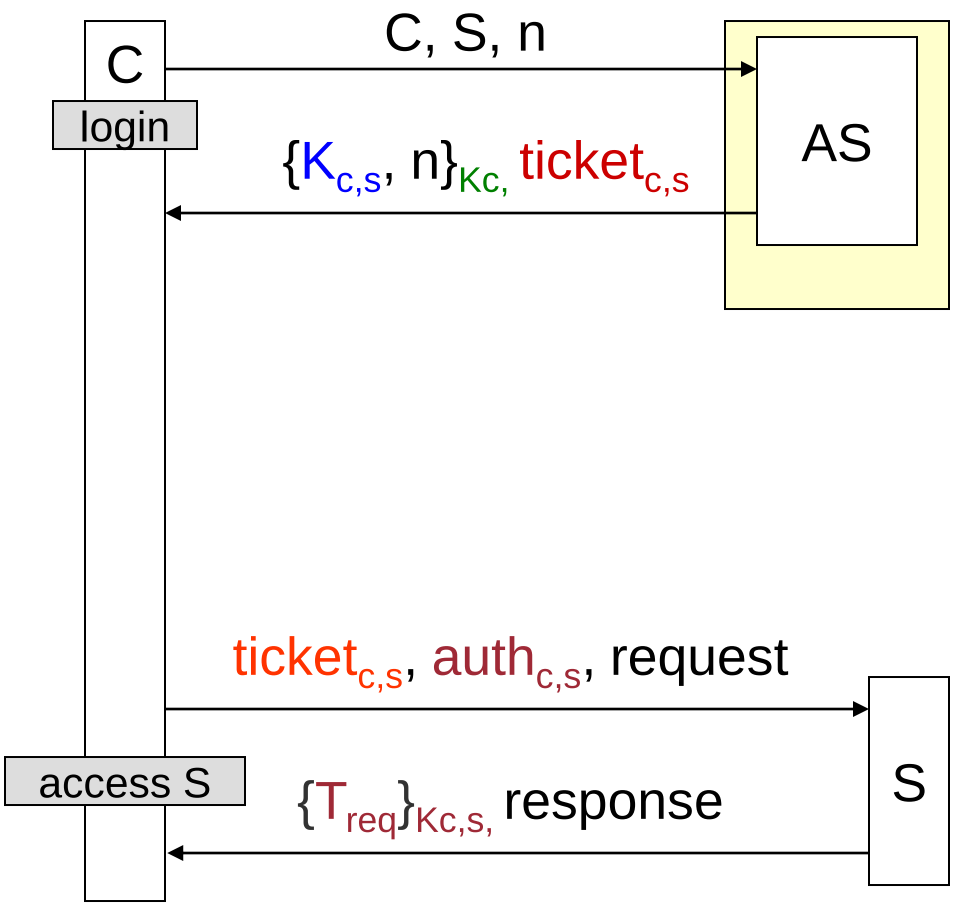
Kerberos

- Based on Needham-Schroeder with timestamps
  - Solves problem of NS requiring using Ka for every contact with KDC
  - … and Ka typically obtained from a password provided by the user
- Ticket Granting Service (TGS)
  - Provides time-limited credentials (tickets) for several services/servers
- Authentication Service (AS)
  - Allows client to login in Kerberos
  - Each client has a shared key with AS derived from password
- Kerberos operates in organizational realms / security domains
  - Multi-realms possible if realms cooperate
- Communication over TCP/IP

Kerberos overview



Kerberos (AS only)



	C	S	AS
C			
S	$K_{c,s}$		
AS	K_c	K_s	

$$ticket_{x,y} = \{x, y, T_1, T_2, K_{x,y}\}_{Ky}$$

$$auth_{x,y} = \{x, T_{req}\}_{Kx,y}$$

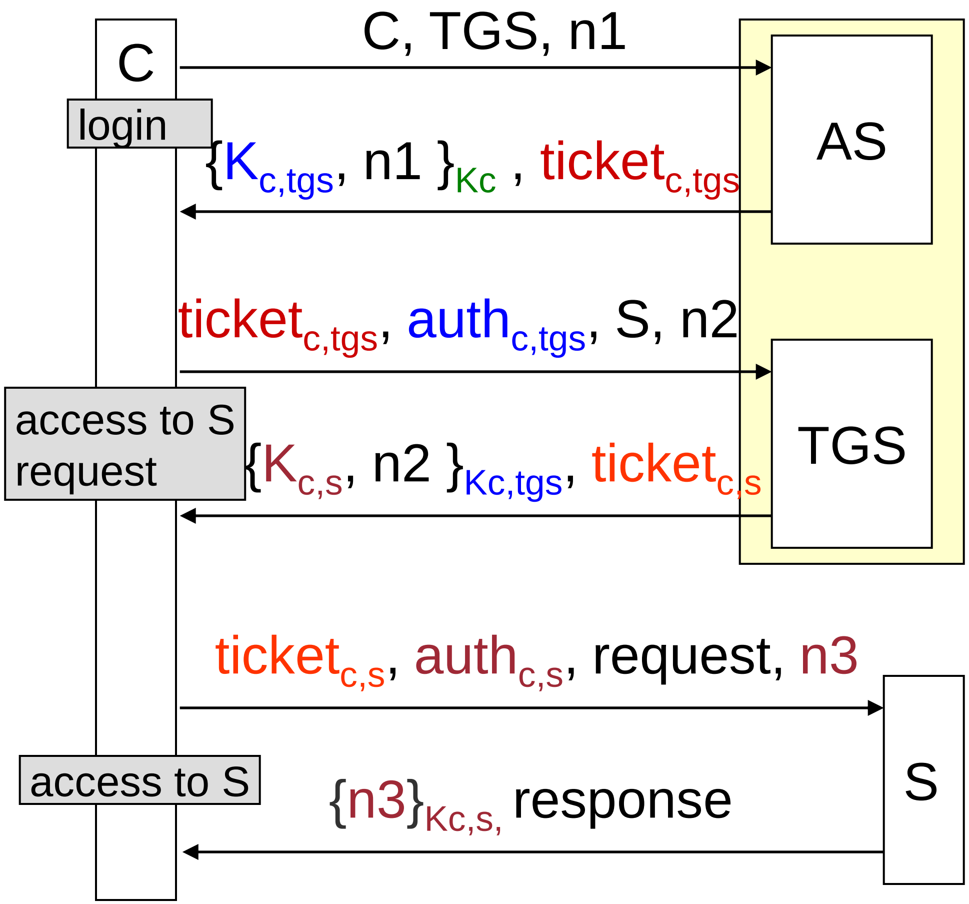
What is inside a ticket?

- $\text{ticket}_{x,y} = \{ x, y, T_1, T_2, K_{x,y} \}_{Ky}$
  - X – client identifier
  - Y – server identifier
  - Timestamps
    - T1 – beginning of validity period
    - T2 – end of validity period
    - To avoid reuse of old tickets (implies clock synchronization)
  - Kx,y – session key value
  - Information ciphered with server key

What is inside an authenticator?

- $auth_{x,y} = \{ x, T_{req} \}_{Kx,y}$
  - X – client identifier
  - Treq – timestamp of request
    - To avoid resending of old request (also imples clock synchronization)
  - Information ciphered with session key

Kerberos V5



	C	S	TGS	AS
C				
S	$K_{c,s}$			
TGS	$K_{c,tgs}$	K_s		
AS	K_c		K_{tgs}	

$$ticket_{x,y} = \{x, y, T_1, T_2, K_{x,y}\}_{Ky}$$

$$auth_{x,y} = \{x, T_{req}\}_{Kx,y}$$

Why separate AS and TGS?

- Separate keys of users from keys of services
- Separate authentication function (AS) from authorization function (TGS)
  - Distribute load between servers
- Minimize use of Kc
  - Used only on login
- Allow composition of TGS servers
  - To access other realms

Roadmap

- Introduction
- Generation and manual distribution
- Distribution with shared values
- Distribution without shared values
- Distribution with third parties
- **Key renewal**

Renewal of keys

- Renewal methods
  - Using KEK keys to distribute new session keys
  - Using thrusted third parties
    - Example: distribution of keys in Kerberos
- Perfect Forward Secrecy
  - Key renewal *per se* does not assure Perfect Forward Secrecy
    - Although it can, if Diffie-Hellman is used with *ephemeral* private values

Summary

- Introduction
- Generation and manual distribution
- Distribution with shared values
- Distribution without shared values
- Distribution with third parties
- Key renewal