Kerberos

Developed by the Massachusetts Institute of Technology (MIT)
Implements an authentication service for a whole (local) network:

- User logs into workstation with name and password (weak)
- The workstation establishes authenticated connections
- Secret key based
- Uses KDC
  → „*Single Sign On*“ network

Assumptions:

- The network is insecure
- The KDC is trusted

Keys in Kerberos

Master Keys

- K_{KDC}: KDC's master key
  KDC's own secret key, known only to the KDC; whoever has this key can encrypt the KDC database
- K_A: master key of a principal A (DES key)
  The KDC shares a secret master key with each principal

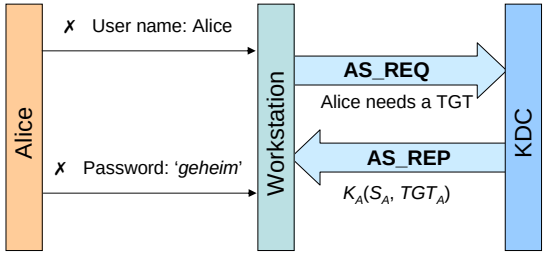
Session Keys

- S_{WS}: a session key
  The KDC invents session keys for communication with a workstation (WS), it is valid only during the actual session
- K_{AB}: a shared session key
  he KDC invents shared session keys for communications between principals A and B
- Shared keys are distributed using *Tickets* and TGTs (*Ticket-Granting Tickets*)

Login via Password

A human user (Alice) wants to establish a secret connection via a workstation to a principal in the network:

- Alice logs in, her WS sends an AS_REQ (Authentication Server Request) to the KDC
- The KDC responds with an AS_REP (Authentication Server Reply), encrypted with K_A
- Alice types in only a password (not a strong key), her WS converts Alice's password into the key K_A and decrypts a session key S_A and a *TGT*
- The WS forgets about K_A

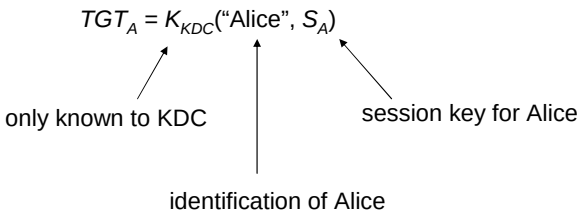


- Creates key S_A
- Finds Alices master key K_A
- $\boldsymbol{TGT_A = K_{KDC}(\text{“Alice”}, S_A)}$

Ticket-Granting Ticket

Ticket-Granting Ticket: *a ticket to get tickets*

- It is advantageous to have no volatile data on the KDC
- KDC does not store the session key
- Instead, the WS uses the TGT for ticket requests
- The TGT contains all data needed by the KDC to identify Alice and encrypt the reply with the suitable session key



Tickets and Ticket-Granting Tickets

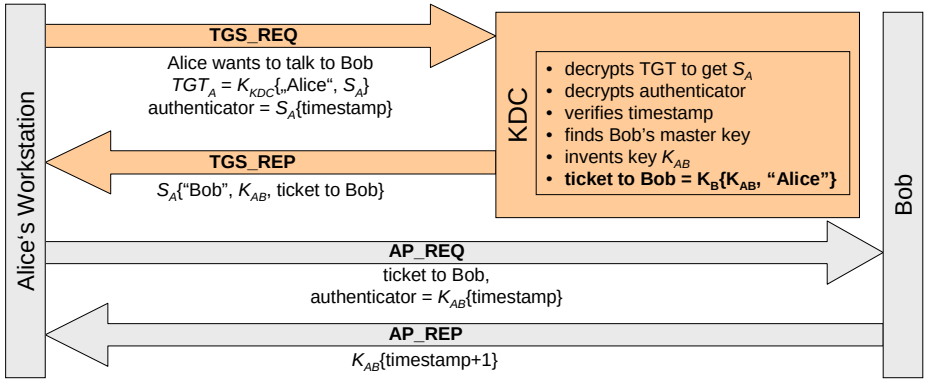
Drawbacks of the login protocol:

- It is easy to obtain data for an offline password guessing attack: simply send: "Alice needs a TGT" to the KDC
  - Try getting K_A by guessing passwords till you can decrypt the KDC's response
  - Remedy in Kerberos V5: the user has to prove his identity, by sending a pre-authentication
- Double encryption of the TGT:
  - $K_A(S_A, TGT_A) = K_A(S_A, K_{KDC}$("Alice", S_A))
  - Offers no security benefit, but needs computational effort
- If a principal changes its master key, already distributed tickets will become unusable (since they are still encrypted with the old key)
  - Thus: assign version numbers with keys and include them in tickets and TGTs
  - New keys get a new version number
  - Principals remember several old key versions
  - Tickets expire after about 21 hours, keys must not be remembered longer than that

Tickets and Ticket-Granting Tickets

Kerberos uses the Needham-Schroeder protocol for authentication, but:

- Timestamps are used instead of nonces
- The TGT and the session key are used instead of Alice's master key



TGS_REQ: Ticket-Granting-Service Request
TGS_REP: Ticket-Granting-Service Reply
AP_REQ: Application Request
AP_REP: Application Reply

Replicated KDCs

Problem: The KDC is a *bottleneck*

- If the KDC is down, it will not be possible to access remote resources (*single-point-of-failure*)
- If the KDC is overloaded, the whole network performance will be affected

Solution: *Replicated KDCs*

- Multiple, interchangeable KDCs
- All share the same master KDC key
- Have identical databases
- Use of one *master copy* to keep all KDCs identical
- All updates are done only on this master copy
- All other KDCs are *read-only slaves* which update from the master copy (periodically or initiated by a human)
- If the master copy fails, no new entries can be created, but authentication of principals still is possible with read-only KDCs

Replicated KDCs - Updates

An update consists on inserting, deleting, or changing a database entry

- A database entry is of the format: < *principal*, *name*, K_{KDC}(*key*)>

Update the database of a slave KDC:

- Transmission is done in the clear
  - An attacker may learn the names of the resources by eavesdropping
  - But all keys are encrypted with the master key of the KDC
  - An attacker can get no use from eavesdropping
- To prevent an attacker from re-arranging the data, it is transmitted using the Kerberos integrity protection
  - Database replay attack is prevented, as the integrity protocol includes a timestamp

Realms

Remaining problem with replicated KDCs: consider several companies, banks, governments, ... in a big network:

- Whoever manages the KDC can access all user master keys
- It is hard to find an organisation to manage the KDC that anybody would trust
- Replicated KDCs are physically located at the different stakeholders' sites, and all of them need to be secure and trusted by all stakeholders

Solution: split network into *realms*

- Each realm has its own trusted master KDC database
- KDCs in the same realm are equivalent
- KDCs of different realms are different:
  - Different KDC master key
  - Different principals (and also keys)

Interrealm Authentication

Alice, located in realm *X*, wants to talk to Bob located in a different realm, say *Y*.

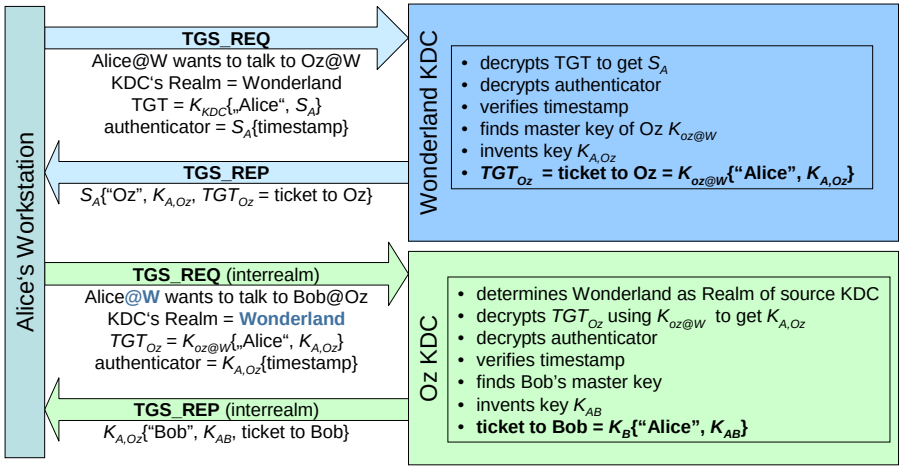
How to authenticate Alice to Bob?

- A KDC can be registered as principal in several other realms
- Assume KDC-*Y* is registered at KDC-*X*: they share a key $\boldsymbol{K_{Y@X}}$
- Shared keys for different realms are different: $\boldsymbol{K_{Y@X} \neq K_{y@z}}$,
- If KDC-*Y* receives a ticket generated by a KDC of another realm, it needs to know the source realm in order to use the right key for decryption

  → The source realm is included in the TGS_REQ message

Interrealm Authentication

Alice (in Wonderland) wants to authenticate to Bob (in Oz)



Now Alice (her workstation) knows everything needed to communicate with Bob

Interrealm Authentication

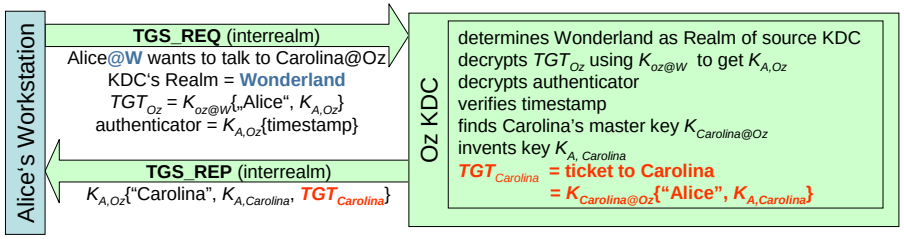
Kerberos V4 does *not allow* to follow a chain of realms

Suppose:

- Realm "Wonderland" and "Oz" share a key $K_{Oz@W}$
- Realm "Oz" and "Carolina" share a $K_{Corolina@Oz,}$
- The realms "Wonderland" and "Carolina" do not share a key

Assume now:

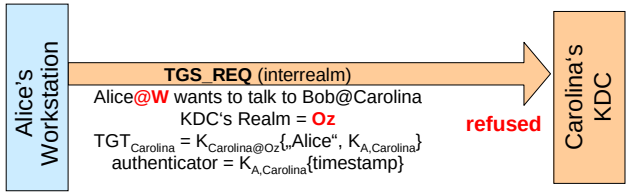
- Bob is located in "Carolina", but Alice contacts – as above – the KDC of „Oz", because there is no shared key between „Wonderland" and „Carolina"
- The „Oz" KDC could invent a ticket to „Carolina" the same way and pass it back to Alice



Interrealm Authentication

Next, Alice contacts Carolina's KDC to ask for a ticket to Bob:

- The attempt to obtain a ticket to Bob@Carolina from the KDC of Carolina will fail, due to mismatching realms
- "Alice's home realm" *is not equal* to the entry "KDC's realm" in the TGS_REQ:



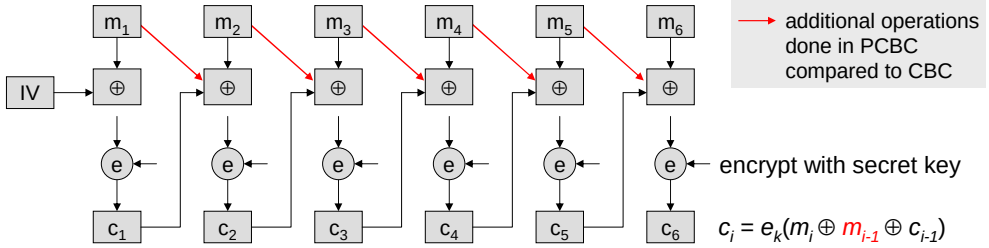
- A principal can only use TGTs *originating from its home KDC* to ask for a ticket at any other KDC. TGTs originating from realms other than the home realm of the requesting principal are refused
- Otherwise, a malicious KDC could not only impersonate its own principals, but also those of any other realm, when it is (or pretends to be!) a connecting realm (by simple generating a suitable TGT)

Privacy and Integrity

After authentication the communication is:

- either in clear text,
- or integrity protected only (message digest, done with a mathematically questionable so-called *"modified Jueneman checksum"*),
- or privacy and integrity protected (DES encryption in modified CBC mode):
  - Modified CBC: *Plaintext Cipher Block Chaining (PCBC)*
  - The unmodified CBC provides for privacy
  - PCBC claims to additionally provide integrity
  - But: the combined privacy and integrity protection proved to be difficult and was not fully provided in Kerberos V4

Plaintext Cipher Block Chaining



Difference to CBC:

- CBC: modification of c_i will garble only m_i and m_{i+1}
- PCBC: modification of c_i will garble all following: m_i, m_{i+1}, ..., m_n

Purpose: the last part of the message, m_n, decrypts properly (and thus can be used to transfer integrity check information) only if the message was not changed

- With CBC this assumption does not hold, therefore PCBC was introduced
- But: some problems assumed, e.g. easier to break the key than in CBC

Kerberos V5

Today: Kerberos version 5 – same functionality as version4, but:

- *More features and flexibility*
  e.g. delegation of rights, ASN.1 message format description, realm chaining
- *Fewer restrictions*
  e.g. longer addresses, long life tickets
- *Optimisations*
  e.g. more algorithms for privacy and integrity
- But… also more overhead!

Delegation

Sometimes it is necessary to give someone else access to things you are authorized to access:

- Alice runs a batch job during night on the WS of her colleague Bob
- This batch job needs to access files located on a remote WS in the network, or needs to log in into a third WS (e.g. Carol's WS)
- To do so, Bob's WS needs authorization from Alice

Therefore: *delegation* - give someone else access to things you are authorized to access

- limited tin time
- limited in scope (subset of resources)

In Kerberos, Alice delegates rights to Bob by allowing Bob to impersonate Alice to

- ➢the KDC and/or
- ➢other principals

- Thus Alice in some sense "passes on her identity"

Delegation

Delegation is done by sending tickets (e.g. a "ticket to Carol") or even the TGT to Bob

- Kerberos V4: network layer address of Alice is included in TGT and tickets
  → Tickets cannot be used by Bob
- Kerberos V5: Alice can request tickets ("*proxy tickets*") and TGTs containing a network layer address different from her own (e.g. Bob's address)
  → Even multiple or no address can be specified
  → "no addresses" = ticket usable from any address
  → Delegation possible to certain users or user groups

Delegation can be controlled using *flags* in the TGT:

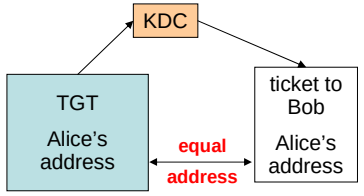
- `forwardable` – this TGT can be forwarded, i.e. it can be exchanged by a TGT with a different address
- `proxiable` – with this TGT its possible to obtain a ticket including a different address
- `forwarded` – this TGT originates from another address

Additionally, flags are defined for notifying the status of a ticket:

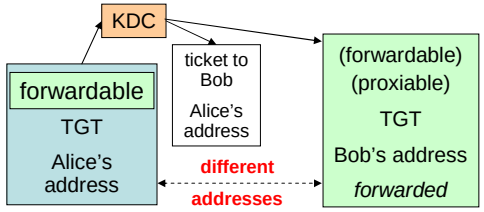
- `forwarded` – this ticket originates from a forwarded TGT
- `proxy` – this tickets was generated with a different address than the originating TGT

Different Flag Settings in a TGT

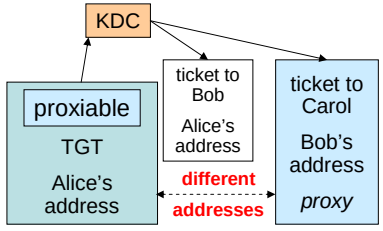
1. No delegation



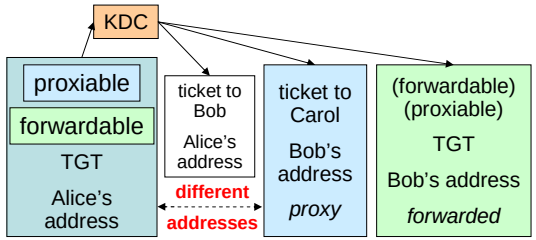
2. TGT can be forwarded



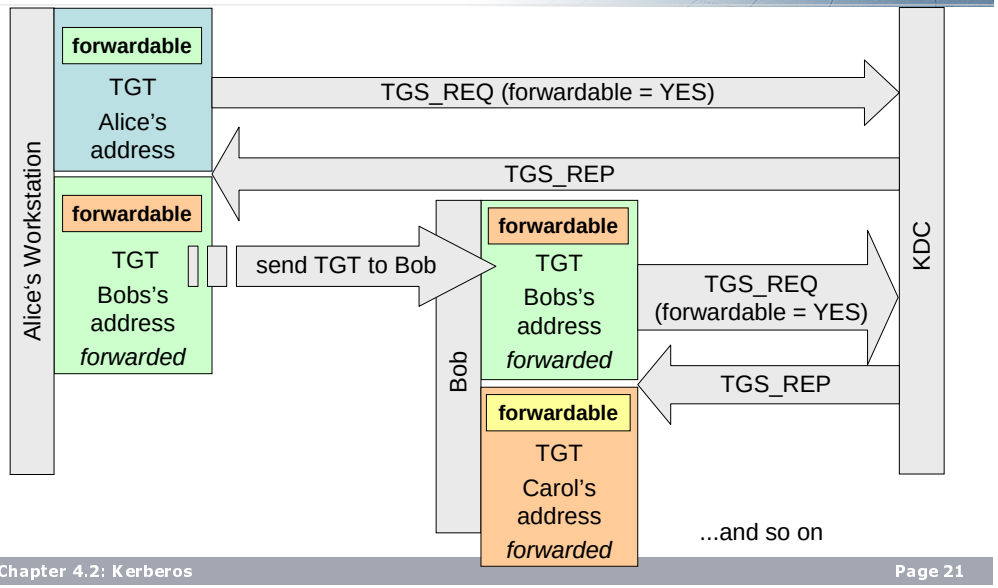
3. Obtain ticket with different address



4. Combination of 2 and 3



Delegation – TGT Forwarding



Delegation

When Alice requests a forwarded TGT, she can specify the desired settings of the "forwardable" and "proxiable" flags. Using these flags in a TGT...

- ... the KDC can control the delegation rights of clients (with higher priority than Alice)
- ... Alice can control the delegation rights of the principal the delegation is given to

Alice can limit the delegation in 3 different ways:

- By using the "forwardable" and "proxiable" flags
- By giving no TGT to Bob, but only "proxy tickets" for the required services
- By using the so-called AUTHORIZATION-DATA field in a request, which...
  - ... is given by Alice when requesting a TGT or ticket,
  - ... is added to the TGT or ticket, or
  - ... is not interpreted by the KDC, but is instead application-specific

Furthermore the applications are involved in the delegation:

- By using the forwarded and proxy flags, when deciding what access to allow
- By interpreting the AUTHORIZATION-DATA field

→ This results in a very flexible, but also very confusing access control

Long Life Tickets

Lifetime of tickets in Kerberos V4:

- Four bytes start time, one byte life time (units of 5 minutes)
- Results in approx. 21 hours maximum life

Ticket lifetime in Kerberos V5:

- ASN.1 defined lifetime quantity of 17 bytes with granularity of 1 second
- Lifetime is practically unlimited: end time ≤ 31 dec 9999
- But: l*ong life time → higher security risk*
- Solution: *renewable tickets*

Renewing tickets

- The lifetime of a ticket is specified by *start time*, *end time*, *authtime* (time when Alice received her initial TGT), *renew-till* (necessary for renewable tickets)
- To renew a ticket it has to be presented to the KDC
  - ➢ The KDC then changes the end-time, if the ticket is still renewable (renew-till time)
  - ➢ If Alice is ever late renewing a ticket, the KDC will refuse to renew it

Privacy and Integrity

Cryptographic algorithms in Kerberos V4 uses

- DES with PCBC for privacy and integrity
  → PCBC not protects against cipher block exchange
- "Modified Jueneman checksum" for integrity only
  → mathematically questionable (though never publicly broken yet)

Kerberos V5 allows for *integrity* protection using the following MICs:

- *rsa-md5-des* (required)
- *des-mac* (required)
- des-mac-k (required)
- rsa-md4-des (optional)
- rsa-md4-des-k (optional)

Comments:

- "md5" means usage of MD5, algorithms containg "md4" are older version of that using MD4
- Algorithms ending with "-k" are old versions with a different key *k*

rsa-md5-des

rsa-md5-des has nothing to do with RSA, but with RSADSI, a company owning rights to MD5

MIC calculation

1. Choose "*Confounder*" = random number, length 64 Bit
2. *X* = [*Confounder* | *message*], *message* has variable length
3. MD = MD5(*X*), length 128 Bit
4. $K' = K_{AB} \oplus \text{F0F0F0F0F0F0F0}_{16}$
5. *Y* = [*Confounder* | MD], length 192 Bit
6. MIC = *K'*(*Y*), length 192 Bit, encrypt in CBC mode using IV = 0

MIC verification

1. Calculate *K'*
2. Decrypt MIC = [*Confounder'* | MD'] using *K'*
3. *X'* = [*Confounder'* | *message'*]
4. if MD' = MD5(*X'*) then *message'* = *message*

→ success

des-mac

dec-mac is similar to rsa-md5-des; main difference: step 3

MIC calculation

1. Choose "*Confounder*" = random number, length 64 Bit
2. *X* = [*Confounder* | *message*], *message* has variable length
3. *Residue* = $K_{AB}(X)$, length 64 Bit, encrypt in CBC mode using IV = 0
4. $K' = K_{AB} \oplus \text{F0F0F0F0F0F0F0}_{16}$
5. *Y* = [*Confounder* | *Residue*], length 128 Bit
6. MIC = *K'*(*Y*), length 128 Bit, encrypt in CBC mode using IV = 0

MIC verification

1. Calculate *K'*
2. Decrypt MIC = [*Confounder'* | *Residue*'] using *K'*
3. *X'* = [*Confounder'* | *message'*]
4. if *Residue*' = $K_{AB}(X')$ then *message'* = *message*

→ success

Privacy and Integrity

For privacy and integrity protection, Kerberos V5 uses the following algorithms:

- des-cbc-crc (MIC = CRC-32, length 32 Bit)
- des-cbc-md4 (MIC = MD4, length 128 Bit)
- *des-cbc-md5* (MIC = MD5, length 128 Bit)

All algorithms do the following:

1. Choose *Confounder* = random number (64 Bit)
2. **X** = [*Confounder* | zeros (length of MIC) | message],
3. Y = [*Confounder* | **MIC(X)** | message],
4. Add padding (64-bit chunks)
5. Encrypt the result using DES in CBC mode with IV = 0

Interrealm Authentication

In Kerberos V4, principals in realm *A* can authenticate with principals in realm *B* only if KDC-*A* is registered as principal in realm *B*

→ large registration effort

Interrealm authentication in Kerberos V5:

- Allow to go through series of realms
- But: if one KDC in the chain is not trusted, the whole authentication can not be trusted
  - List all traversed KDCs in the message headers, so that no involved KDC can avoid to be listed
  - It is the client's decision if he trusts all traversed KDCs or not
- Principle:
  - Arrange realms in a tree structure
  - The tree structure often emerges from present address structures (e.g. Internet domains)
  - Possibly allow additional shortcuts (cross links)