Kerberos and Active Directory—symmetric cryptography in practice

COSC412

Learning objectives

- Understand the function of Kerberos
- Explain how symmetric cryptography supports the operation of Kerberos
- Summarise the relationship between Kerberos and Microsoft Active Directory

Motivation for Kerberos

- 1983: Project Athena (MIT + DEC + IBM)
  - Support campus-wide distributed computing
  - Particular emphasis on educational use
- Project Athena created many significant technologies:
  - Thin clients, X Windows, IM, directory services, …
- Athena terminals were widely dispersed, physically
  - Had to handle large numbers of users with different privileges

The goals of Kerberos

- Provide a consistent way to authenticate to different services. Moreover, provide **single sign-on** across them

- Facilitate mutually secure interactions between clients and servers

- Operate securely over untrusted networks
  - (So keep an eye out for how shared secrets are established)

Kerberos in context

- MIT developed it: public release in late 1980s

- Kerberos is very widely supported at the OS level
  - macOS, Linux, Windows, *BSD, Solaris, *etc*.
  - Used for Apple's "Back to My Mac" and other parts of iCloud

- Since Windows 2000, Kerberos is the means of authenticating to Windows domains
  - Crucial component of Microsoft Active Directory

History of Kerberos versions

- Versions 1–3: internal to MIT. It was for Project Athena…
- Version 4: released in late 1980s
  - Uses 56-bit DES… so you definitely shouldn’t use it anymore
  - … and also has protocol weaknesses (encryption oracle)
  - USA classified it as auxiliary military technology
- Version 5: released 1993; 2005 [RFC 4120]
  - Allows negotiation of encryption algorithms

Difficulties in using Kerberos

- Tickets have timestamps—requires synchronised clocks
- Key Distribution Centre (KDC):
  - Single point of failure within the distributed system
  - The KDC itself can be clustered …
  - … but clients need to be able to reach it!
  - Further: successfully breaking into KDC breaks **all** security
- Keys may be tied to hostnames… not so useful today

How does Kerberos use cryptography?

- Kerberos works with symmetric key cryptography
  - Can also use asymmetric key cryptography
- Where's the shared secret?
  - Actually, Kerberos uses many pairs of shared secrets
- Kerberos provides authorisation via **tickets**
  - You can show what **A** says about you to **B**
  - **A** and **B** don't need to communicate directly
  - Instead your ticket includes digital authenticated declarations

Kerberos architecture

- For **authentication** purposes (infrequent)
  - Client—the software that users control
  - Authentication Server (AS)
    - … part of the Key Distribution Centre (KDC)

- For service **authorisation** purposes (frequent)
  - Ticket Granting Service (TGS)
    - … also part of the Key Distribution Centre (KDC)
  - Service Server (SS)—the target system user enacts privileges on

Kerberos service use: four phases

- **1** User proves identity to their console
  - *e.g.*, using a password, smart-card, biometrics, *etc*.
- **2** Client contacts authentication service (AS)
  - Single sign-on done; **authentication** complete
  - Client receives a ticket granting ticket (TGT)
- **3** Client requests service authorisation (TGS)
  - Client receives a service ticket (ST)
- **4** Client contacts service server (SS)
  - **Authorised** to access service by the ST

Discussing authentication phases

- User proves identity (to AS) through the use of long-term, secure credentials
  - AS interacts with KDC's database to acquire TGT
  - Session key also established

- TGT allows user to make authenticated requests of the TGS without using the long-term secure credentials
  - This is a key point of single sign on (SSO)

Discussing authorisation phases

- User presents TGT to TGS
  - This shares the session key
  - TGS sends back service ticket
- TGS and the target service also share a secret
  - So TGS can ‘tunnel’ a message to the service via the user
- Key point: service tickets have a lifetime
  - … thus client can cache them locally

Compare Kerberos to SSH public-key auth.

- Kerberos: authorisation + authentication
  - Supports delegation (share ticket); fine-grained access control
- SSH key-pair is typically about your identity
  - Can create keys for services, but they they lose link to user ID
- Security model different. Compromised host?
  - Using public-key SSH, host learns your private key
  - Using Kerberos? Less bad: root trust in KDC; also, tickets expire

Accessing a service: SSH

```
# (Visit COSC412 resources page for more information.)
# On your computer run:
git clone https://altitude.otago.ac.nz/cosc412/demo-vm
cd cosc412-demo; vagrant up; vagrant ssh
# Then, after SSHing to the VM, run:
. /vagrant/bash-vars.sh
/vagrant/kerberos/setup-kerberos.sh
```

- Let's add a user 'testme', password 'testme'

```
: ~$; sudo adduser testme
…
```

- SSH using a password (first time will check fingerprint)

```
: ~$; ssh testme@ubuntu-xenial.testdomain whoami
testme@ubuntu-xenial.testdomain's password:
testme
```

- (Note: If you have already run `kinit`, you can remove your existing tickets by using `kdestroy`)

Kerberos in practice: first steps

- First, look at Kerberos from client's view

```
: ~$; klist
klist: No ticket file: /tmp/krb5cc_1000
: ~$; kinit testme
testme@TESTDOMAIN's Password:
: ~$; klist
Credentials cache: FILE:/tmp/krb5cc_1000
        Principal: testme@TESTDOMAIN

  Issued                Expires               Principal
Aug  8 11:47:14 2020  Aug  8 21:47:05 2020  krbtgt/TESTDOMAIN@TESTDOMAIN
```

- Kerberos 5 tickets: `name/instance@REALM`
  - Note that default domain was preconfigured
  - TESTDOMAIN typically related to DNS domain

Accessing an SSH service with Kerberos

- SSH using Kerberos

```
: ~$; ssh testme@ubuntu-xenial.testdomain whoami
testme
```

- In more detail:

```
: ~$; ssh -v testme@ubuntu-xenial.testdomain whoami
OpenSSH_7.2p2 Ubuntu-4ubuntu2.8, OpenSSL 1.0.2g  1 Mar 2016
…
debug1: Host 'ubuntu-xenial.testdomain' is known and matches the ECDSA host key.
…
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,password
…
debug1: Next authentication method: gssapi-with-mic
debug1: Authentication succeeded (gssapi-with-mic).
Authenticated to ubuntu-xenial.testdomain ([127.0.1.1]:22).
…
debug1: Sending command: whoami
…
```

Kerberos is a GSSAPI implementation

Having SSHed, look at our tickets

- Requesting SSH has cached tickets for us:

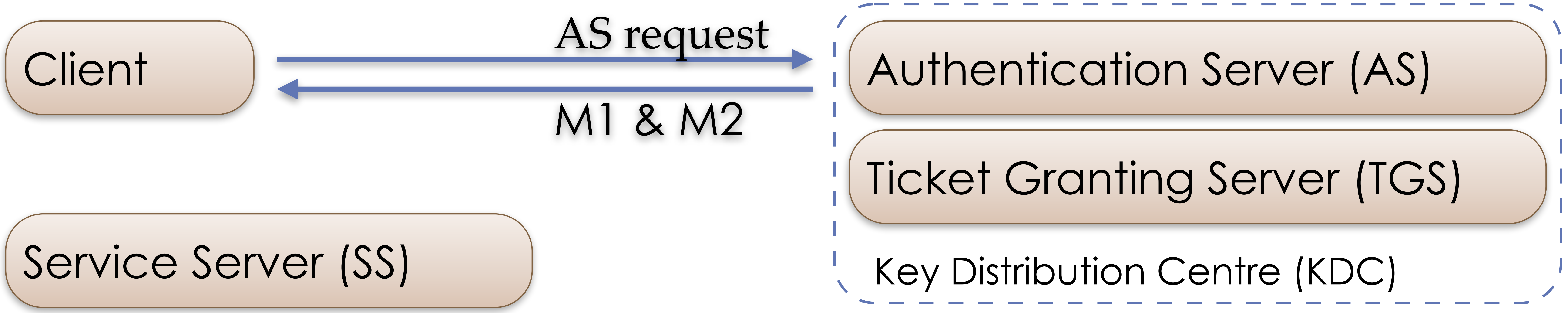
```
: ~$; klist
Credentials cache: FILE:/tmp/krb5cc_1000
        Principal: testme@TESTDOMAIN

  Issued                Expires               Principal
Aug  8 11:47:14 2020  Aug  8 21:47:05 2020  krbtgt/TESTDOMAIN@TESTDOMAIN
Aug  8 11:47:40 2020  Aug  8 21:47:05 2020  host/ubuntu-xenial.testdomain@
Aug  8 11:47:40 2020  Aug  8 21:47:05 2020  host/ubuntu-xenial.testdomain@TESTDOMAIN
```

  - (Note: the middle ticket is due to my dodgy hack to avoid setting up DNS. It shouldn't be there but doesn't break things.)

- SSH to `ubuntu-xenial` can use cache; no TGS comms.

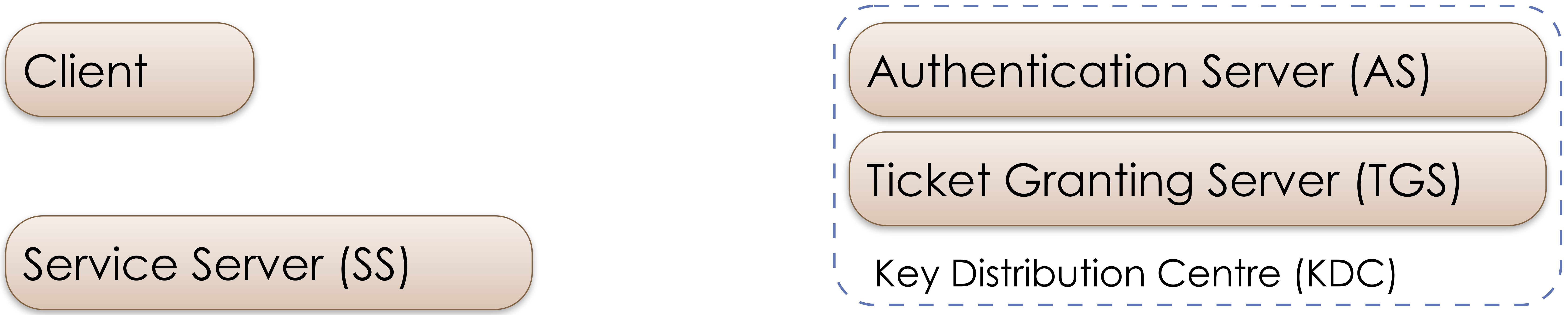
Kerberos steps in detail

- Client requests services for user from AS (no creds. sent)
- AS checks for valid user, and if so sends:
  - Message **M1**: $\{K\_session[client \leftrightarrow TGS]\}_{K[client \leftrightarrow user]}$
    - I am using notation $\{D\}_K$ for D encrypted with key K.
  - Message **M2**: $\{TicketGrantingTicket\}_{K[AS \leftrightarrow TGS]}$
    - Includes client ID, ticket validity, K_session[client↔TGS]



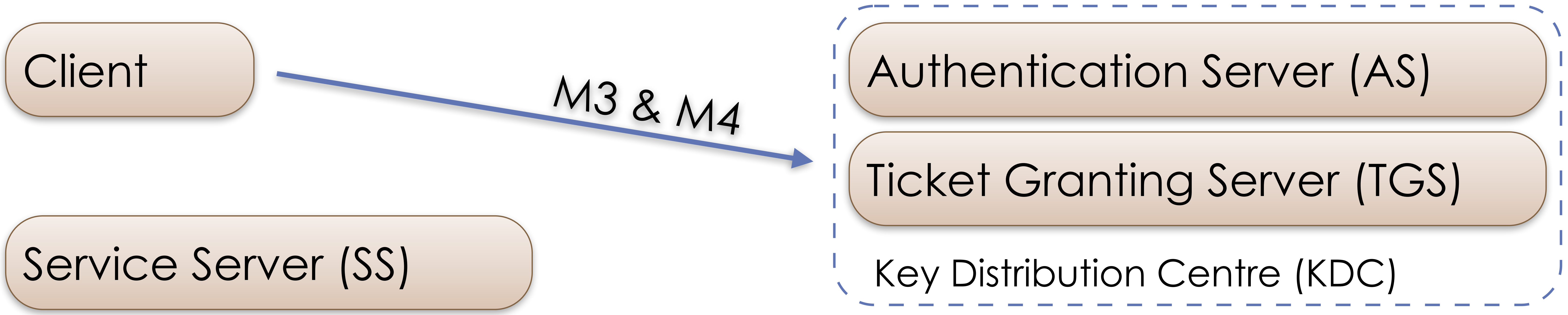
Kerberos steps in detail

- Client decrypts **M1** using key generated from user having authenticated
  - (user authentication failure means client can't decrypt **M1**)
  - Client gets K_session[client↔TGS]
  - Client can't decrypt **M2**, and doesn't need to
- Client can now actually authenticate to TGS



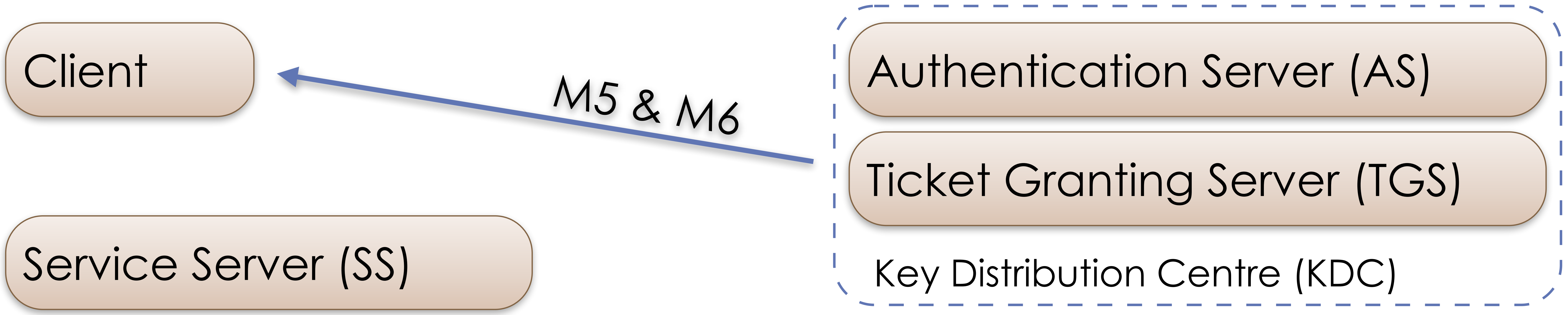
Kerberos steps in detail

- To use a service, client sends 2 messages to the TGS:
  - **M3**: {**M2**, serviceID}
  - **M4**: {clientID, timestamp}$_{K\_session[client \leftrightarrow TGS]}$
  - **M4** is called an “authenticator”
- TGS retrieves **M2** from **M3**; decrypts **M2** (TGS & AS share a key)
  - TGS now has K_session[client↔TGS]



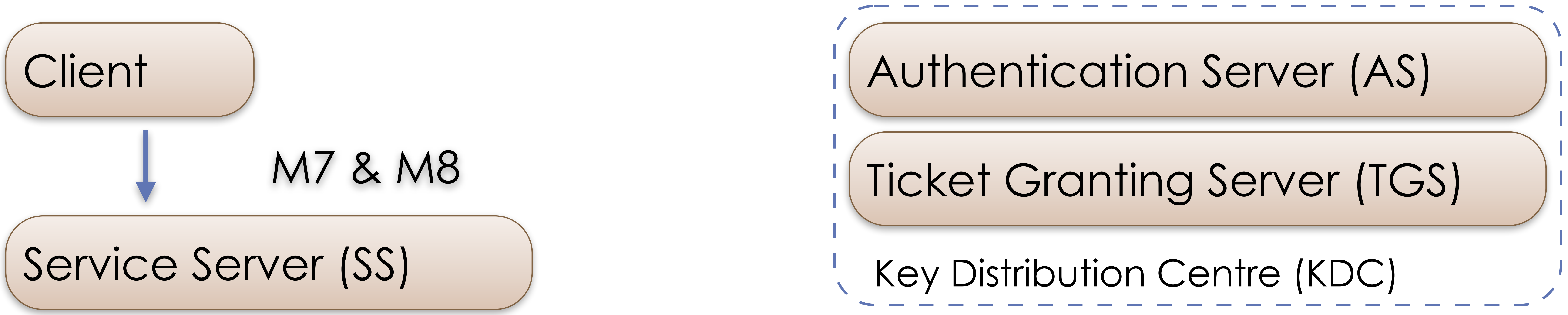
Kerberos steps in detail

- TGS decrypts **M4** (the authenticator); sends:
  - **M5**: {ClientToServerTicket}$_{K[TGS \leftrightarrow server]}$
    - Ticket has client ID, validity, K_session[client↔server]
  - **M6**: {K_session[client↔server]}$_{K\_session[client \leftrightarrow TGS]}$
- Client can now make authorisation request of service



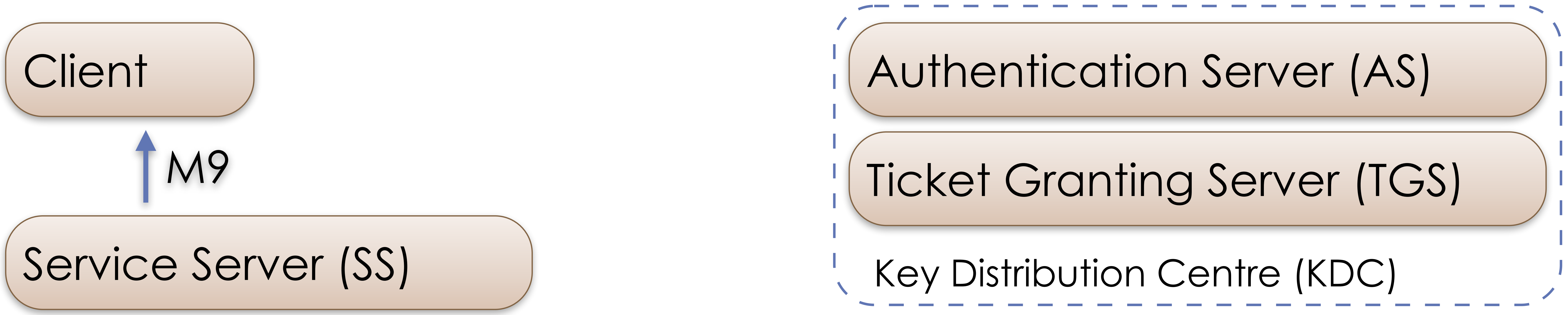
Kerberos steps in detail

- Client sends Service Server (SS) two messages:
  - **M7**: {**M5**}
  - **M8**: {clientID, timestamp}$_{K\_session[client \leftrightarrow server]}$
- SS decrypts **M7**; gets K_session[client↔server]
  - SS then decrypts **M8** (which is an authenticator)



Kerberos steps in detail

- If SS is satisfied with client's authenticator (**M8**), it sends the following message to the client:
  - **M9**: {timestamp[from **M8**]+1}$_{K\_session[client \leftrightarrow server]}$
- Client checks updated timestamp within **M9**
  - Client can trust server and can start issuing requests



Examining the ticket we requested

- We can get more information from `klist`
  - **Addressless**: can use with NAT, *etc*.
  - Note that **encryption type** is negotiable

```
: ~$; klist -v
Credentials cache: FILE:/tmp/krb5cc_1000
        Principal: testme@TESTDOMAIN
    Cache version: 4

Server: krbtgt/TESTDOMAIN@TESTDOMAIN
Client: testme@TESTDOMAIN
Ticket etype: aes256-cts-hmac-sha1-96, kvno 1
Ticket length: 317
Auth time:  Aug  8 11:47:14 2020
End time:   Aug  8 21:47:05 2020
Ticket flags: enc-pa-rep, pre-authent, initial, proxiable, forwardable
Addresses: addressless
```

Cryptography in the ticket

- Ticket etype: `aes256-cts-hmac-sha1-96`
  - (also includes key version number '1')

- AES block cipher with 256-bit key
  - Using cipher-text stealing (CTS)
  - CTS allows non-block-length data to be handled

- Hash Message Authentication Code: SHA1?
  - SHA1 is 160-bits… the 96 just means it is truncated to fit

Kerberos tickets: encrypted fields

Field Name	Description
Flags	Options regarding how & when ticket can be used (more later).
Key	Session key for (en/de)crypting client/server communications.
Client Realm	Realm from which the ticket was requested.
Client Name	Name of the requestor.
Transited	List Kerberos realms that participated in cross-realm client auth.
Authentication Time	Timestamp from when client first received TGT. TGS copies this time to service tickets.
Start Time	Ticket is valid after this time.
End Time	Ticket is not valid after this time.
Renew Till	(Optional) A "RENEWABLE" ticket (more later) can be renewed until this time.
Client Address	(Optional) A list of addresses from which the ticket can be used.
Authorisation-Data	(Optional) Authorisation data relating to the client: not interpreted by KDC. • MIT Kerberos uses the field for access restrictions. • Microsoft Kerberos uses field to store SIDs (user + their groups).

Kerberos flags

Flag	Description
FORWARDABLE	**(TGT only)** TGS is instructed that it can issue new TGTs with different network addresses, when a client shows this TGT.
FORWARDED	**TGT:** indicates this TGT was forwarded. **Non-TGT:** shows that a ticket was issued from a forwarded TGT.
PROXIABLE	**(TGT only)** TGS is instructed that it can issue tickets with network addresses different from the TGT's.
PROXY	Ticket's address is different from that of the TGT that authorised it.
MAY-POSTDATE	**(TGT only)** TGS is instructed that postdated tickets are OK.
POSTDATED	Records that this ticket was postdated when issued.
INVALID	Services must have KDC validate this ticket before it's used (*e.g.*, a postdated ticket that hasn't yet reached its start time).
RENEWABLE	If "End Time" has passed, but "Renew Till" has not, the KDC can issue a new ticket without requiring re-authentication.
INITIAL	Ticket not issued based on TGT, *e.g.*, part of initial AS interaction.
PRE-AUTHENT	The KDC authenticated the client before issuing a ticket. The evidence may be within this ticket (*e.g.*, an authenticator).
HW-AUTHENT	Special-purpose hardware device was used for authentication.

Kerberos administration: first steps

- Let's rewind: see how Kerberos was set up
  - On VM we first have to install packages
  - Also hack `/etc/hosts` so we don't need DNS
- In terms of actual Kerberos administration:
  - First the database needs to be initialised:

```
sudo kadmin -l init --realm-max-ticket-life=unlimited \
   --realm-max-renewable-life=unlimited TESTDOMAIN
```

  - Note that the `-l` option means we use a local Kerberos database, rather than remote admin.

Administration: adding a service

- Establish shared KDC↔service secret

```
sudo kadmin -l add --random-key --max-ticket-life='1 day' --max-renewable-life='1 week' \
   --expiration-time=never --pw-expiration-time=never --attributes='' --policy='default' \
   host/ubuntu-xenial.testdomain
sudo kadmin -l ext_keytab host/ubuntu-xenial.testdomain
```

- Here we cheated a bit with the `/etc/krb5.keytab` …
  - …for the demo, KDC and SSH are sharing the same keytab file!
  - Otherwise, we would copy the keytab file to the SSH server

```
: ~$; sudo ktutil list
FILE:/etc/krb5.keytab:

Vno  Type                     Principal                                  Aliases
  1  aes256-cts-hmac-sha1-96  host/ubuntu-xenial.testdomain@TESTDOMAIN
  1  des3-cbc-sha1            host/ubuntu-xenial.testdomain@TESTDOMAIN
  1  arcfour-hmac-md5         host/ubuntu-xenial.testdomain@TESTDOMAIN
```

Administration: adding a user

- Add a user principal to the local database:

```
sudo kadmin -l add --password='password' --max-ticket-life='1 day' --max-renewable-life='1 week' \
   --expiration-time=never --pw-expiration-time=never --attributes='' --policy='default' testme
```

  - Note: `testme` principal was added before the Linux user had been created, in the earlier demo

- Conventions link principle names and users
  - e.g. SSH's GSSAPI accepts login if you hold ticket: `host/`**`FQDN@REALM`** (bold indicates a variable)

Distributed Kerberos authorisation

- Many other principals created in our VM:
  - `krbtgt/TESTDOMAIN@TESTDOMAIN`
  - `kadmin/changepw@TESTDOMAIN`
  - `kadmin/admin@TESTDOMAIN`
  - `changepw/kerberos@TESTDOMAIN`
  - `kadmin/hprop@TESTDOMAIN`
  - `WELLKNOWN/ANONYMOUS@TESTDOMAIN`
- Allows for password changing, *etc*.

Microsoft Active Directory

- Combines LDAP, Kerberos and dynamic DNS
  - Facilitates almost entirely point-and-click setup of complex distributed infrastructure
- Lightweight Directory Access Protocol (LDAP) manages hierarchical directory of principals and group privileges
- Dynamic DNS allows clients to join domains from non-fixed infrastructure

Microsoft AD in context

- Microsoft's underlying Kerberos is standard:
  - Interoperation with other OSs is well supported
- Has some implementation-specific behaviour:
  - *e.g.*, password changing protocols added by MS (not in MIT)
- Samba from version 4.0 onwards allows Linux to act as an Active Directory Domain Controller
  - Many open source AD "drop-in" replacements are available

Kerberos cross-realm authentication

- For example, allow service tickets in B.REALM.ORG to be issued for principles from A.REALM.ORG
  - Add to A.REALM.ORG and B.REALM.ORG the special principal `krbtgt/B.REALM.ORG@A.REALM.ORG`
  - Principals need the same key, encryption type, etc.
- For two-way trust, also add to both KDCs principal
  - `krbtgt/A.REALM.ORG@B.REALM.ORG`

- Services ask for other realm's TGT from local TGS

Kerberos cross-realm authentication

- Microsoft AD incorporates similar concepts thoroughly
  - Cross-tree trust support, *e.g.*, for company mergers, *etc.*
  - Includes explicit Kerberos 5 realm trust
    - (complexities: MS ticket-equivalent has more fields)
- Cross-realm authentication killed Kerberos 4:
  - Attacker that controls one realm can fabricate principal names to align block-cipher blocks and have target realm help create forged tickets
  - Attacker can then authenticate as target's local users

In summary

- Described the motivation behind Kerberos, how it works, and its relationship to Microsoft Active Directory
- Indicated how symmetric cryptography fits within Kerberos systems, and its limitations
- Demonstrated how Kerberos can be configured to be used for both authentication, and for authorisation of service use