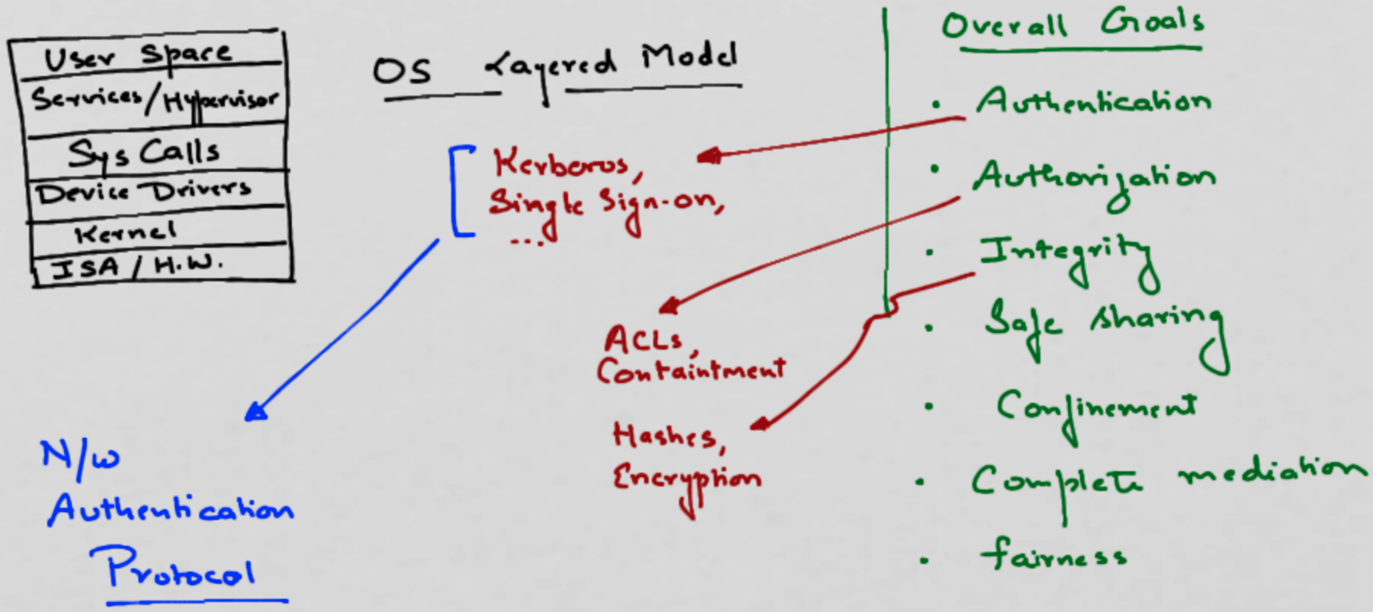
OS Security



User Space
Services / Hypervisor
Sys Calls
Device Drivers
Kernel
ISA / H.W.

OS Layered Model

Kerberos, Single Sign-on, ...

N/w Authentication Protocol

ACLs, Containment

Hashes, Encryption

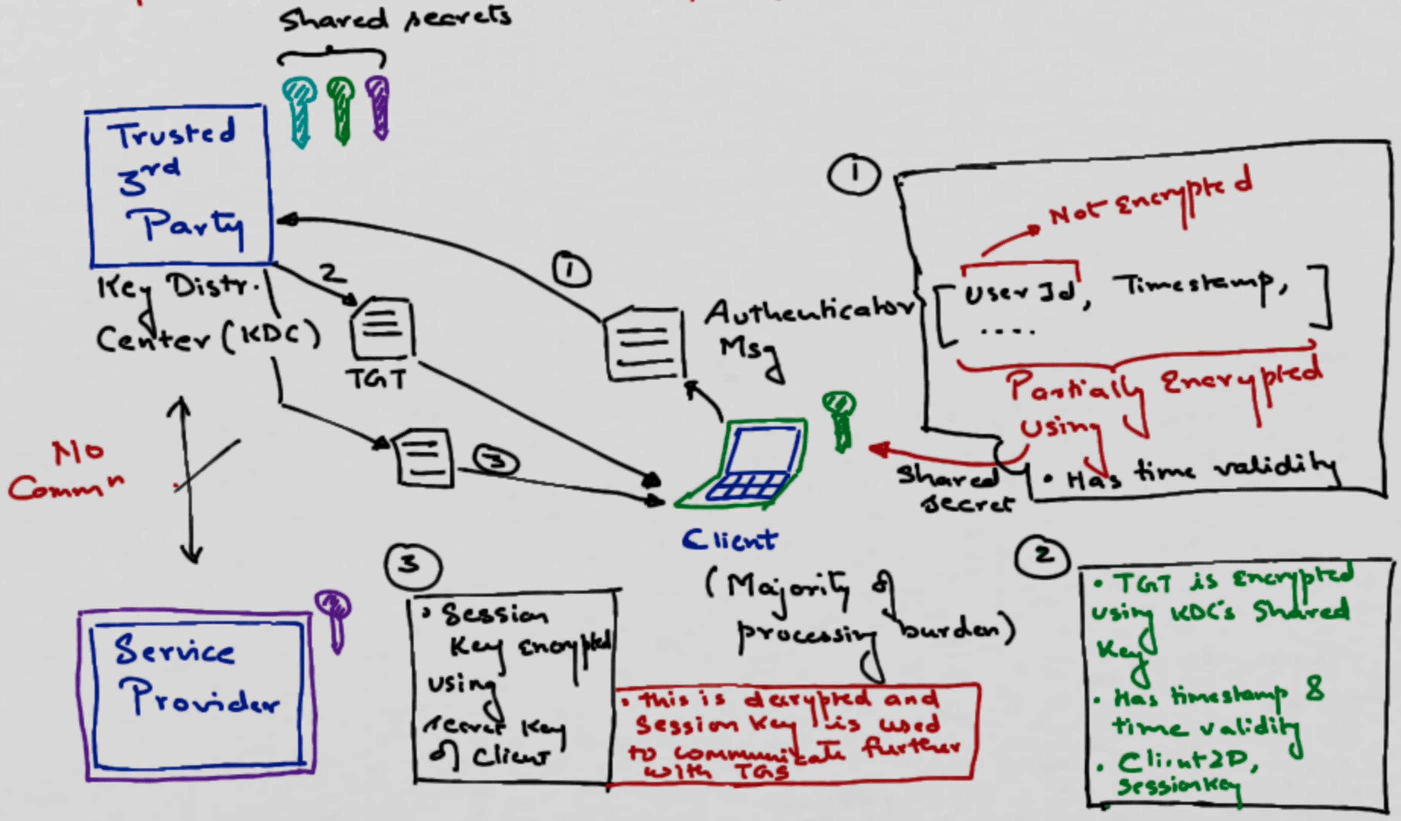
Overall Goals

- Authentication
- Authorization
- Integrity
- Safe sharing
- Confinement
- Complete mediation
- fairness
-

Kerberos

→ Builds on Symmetric Key cryptography (not limited to it though)

→ Requires a trusted third party



1

[User Id, Timestamp, ...]

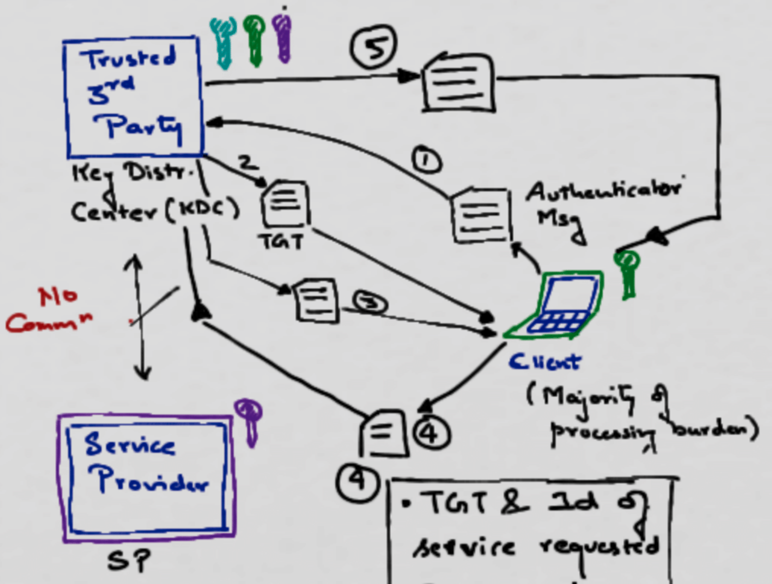
- User Id: Not Encrypted
- Partially Encrypted using Shared secret
- Has time validity

2

- TGT is Encrypted using KDC's Shared Key
- Has timestamp & time validity
- ClientID, SessionKey

3

- Session Key encrypted using secret Key of Client
- this is decrypted and Session Key is used to communicate further with TGS



4

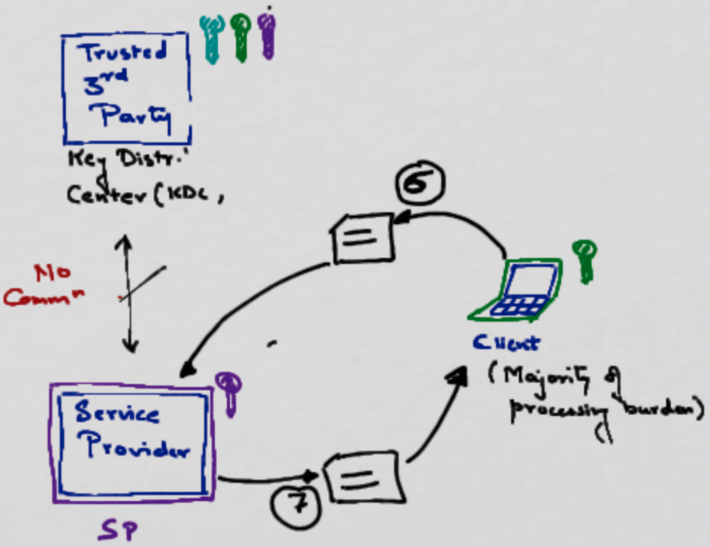
- TGT & Id of service requested
- Decoded by TGS using its Key
- Client Id & timestamp encrypted using session Key

5

- If checks of authentication & authorization succeed
- Sends a tkt & session Key for client-SP communication
- Client-SP session Key

Encrypted with SP's secret Key

Encrypted with Client-KDC session Key



6

- tkt & session key encrypted with SP's secret key
- New authenticator msg (client id, timestamp) encrypted using session key

7

- decrypts, gets session key
- decrypts authenticator & compare the client ID in tkt & authenticator msg
- sends timestamp in authenticator msg to authenticate itself to client

Drawbacks ?

→ Clocks not synchronized ? DoS ?

→ KDC → single pt. of failure ?

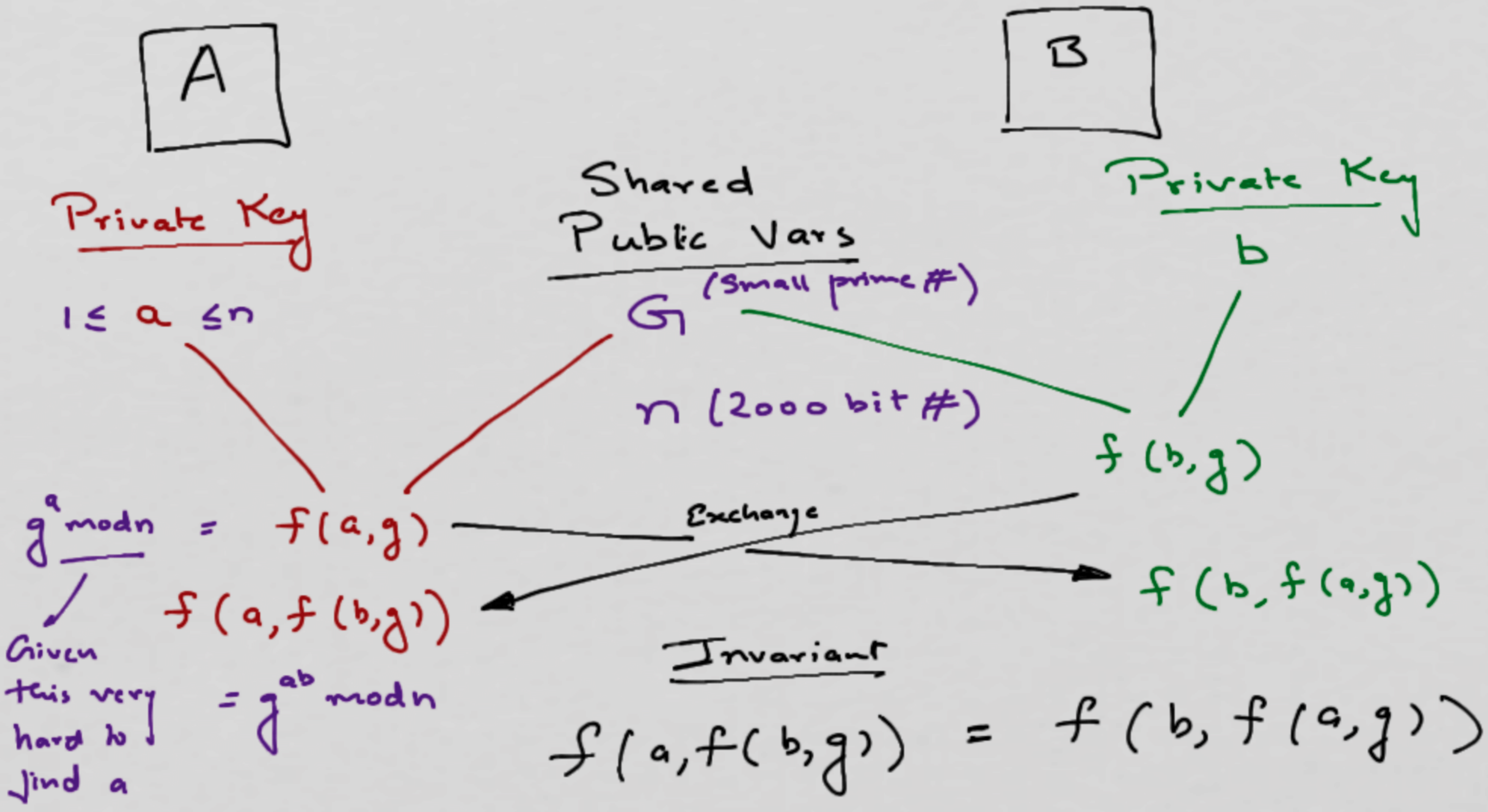
→ Each n/w service (by the same provider) requires all of this commn

→ Can't connect with 3rd party untrusted service providers

Vulnerabilities ?

→ KDC implementation (Windows 2014 patch)

Diffie–Hellman Key Exchange [based on modular arithmetic]



Invariant

$$f(a,f(b,g)) = f(b,f(a,g))$$

OS Code, Runtime Security

- Privelge Escalation
- Code Injection

⋮

```
#define SZ 256
int main (....){
  char buf[SZ];
   if (argc < 2)
       return -1;
   Else
      strcpy (buf, argv[1]);
   return 0;
}
```

Cond^n

→ Knowledge that Vulnerability exists

→ understanding of process's memory map

Sol^n

→ Compile time defense

→ Runtime defense

Compile-time :

a) Choose a high-level language
b) safe-coding practices
c) add^{n} code in standard libraries,

Runtime :

a) Stack canaries, Shadow stack
b) ASLR, DEP, SEHOP
c) CFI

Information-flow leaks

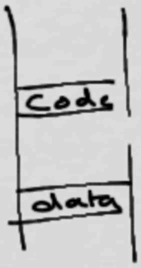
Bell-LaPadula Model

→ fixed security class of actors & objects

→ infer flow leaks based on properties

- Subject at a given security level can read objs only at a level equal or lower than its own level
- Subj. can't write to an object at a lower security level
- Includes Access Control Matrix [Graham-Denning Model]

Data Execution Prevention



Code Injection by modifying data.

Sol^n

Disallow code to execute in data space

H/w Sol^n

- → Mark all mem locations in process mem as non-Executable unless the location explicitly contains code
- → granularity : per-Virtual memory page basis
- → Operation: bit in a page-table entry (NX or XD)

S/w Sol^n

- → Limited to only system-specific binaries

Address Space Layout Randomisation

→ Randomises the layout of stack & heap

→ Make it more difficult for jump to malicious code (by toying with EIP, ESP, etc.]

by ROP → jumping all the way to memory protection API & bypassing it → thus, rendering DEP ineffective.

Sandboxing

[Executing untrusted prgs]

→ restricted access to n/w , isolated memory , etc.

→ Eg: mem isolation for each process

→ Eg: virtualization technology (Emulate & restrict) (Subtle diff. with sandboxing)