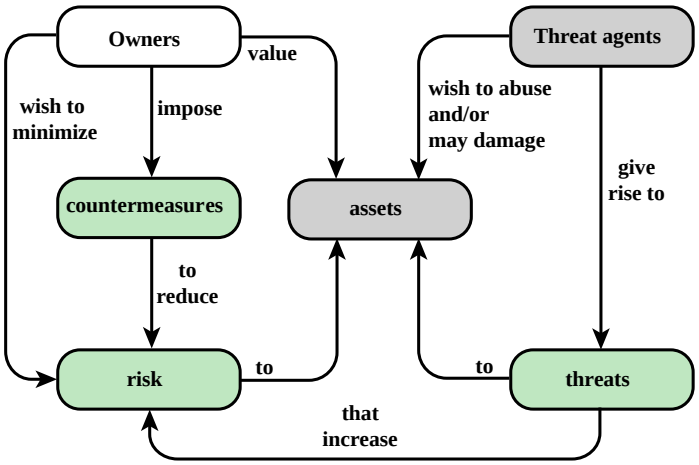
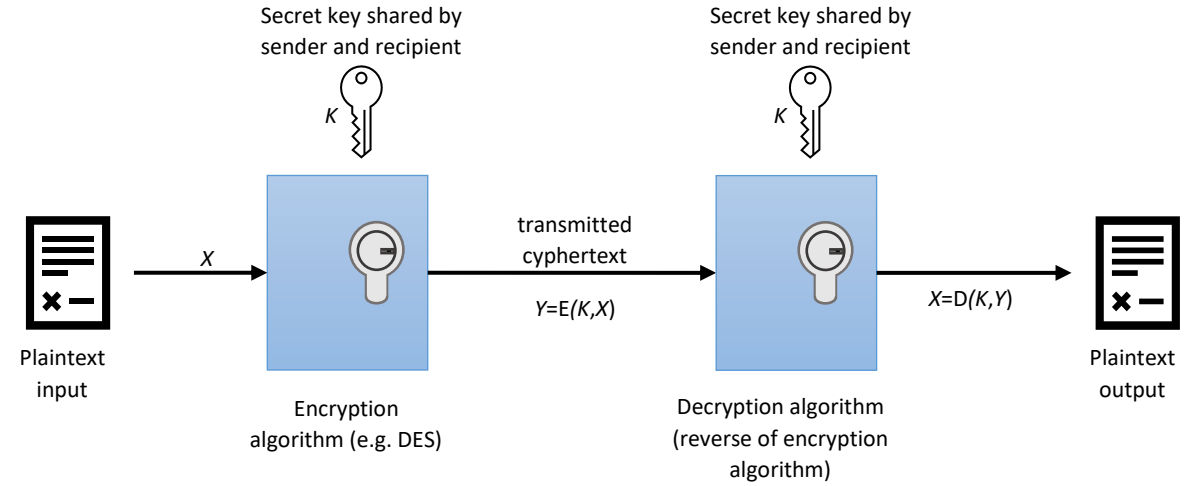
DSS – preparation for the oral exam

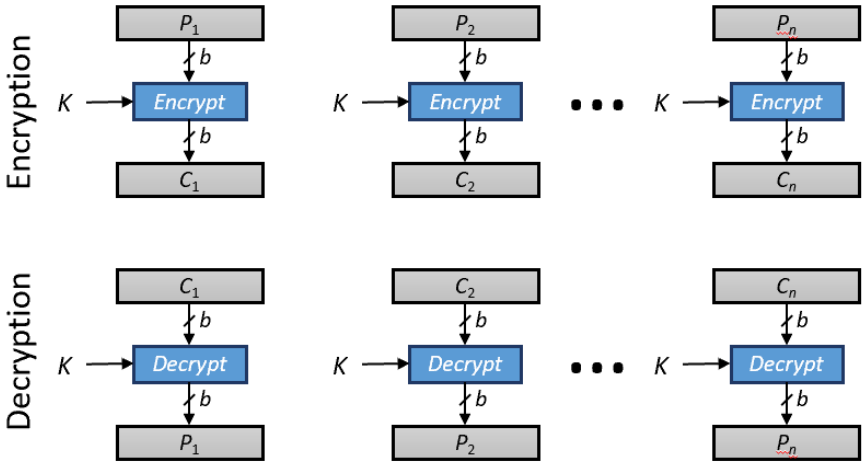
I may start the oral exam by asking you to comment/explain one of the following schemas or by asking you to answer to one of the following questions.

Part 1

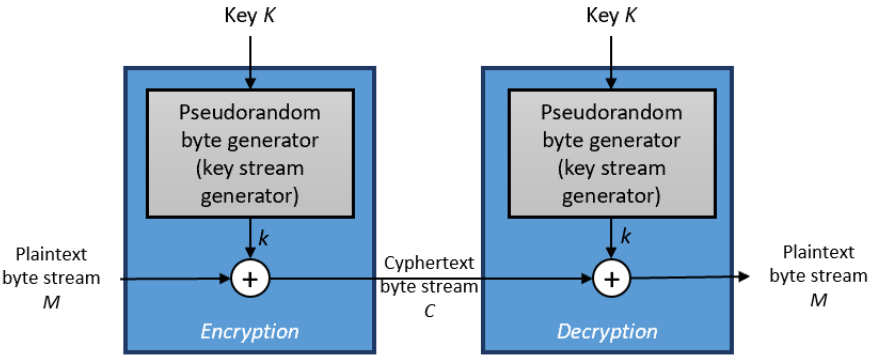
- What is the CIA? Explain its key security concepts.
- Define threats, attacks and assets; define the concepts of attack surface and defense in depth and provide relevant examples.
- Present and explain the properties of one-way hash functions
- Explain the meaning of "multifactor authentication" and provide relevant examples
- Discuss the methodologies of password cracking, explain the concepts of dictionary attack and of rainbow table attack and explain the role of the salt in the Unix password file.
- Discuss the different methods for biometric authentication
- Discuss the token-based authentication
- Explain the challenge-response protocol for remote user authentication
- Define Discretionary access control, Role-based accesso control, Attribute-based access control and give relevant examples
- Explain the differences between access control matrix, lists of capabilities and access control lists
- Explain the basic model of Unix for access control
- Discuss advantages and disadvantages of RBAC and ABAC
- Discuss methods to implement complex passwords policy and proactive password checking
- Explain how does RBAC can be implemented



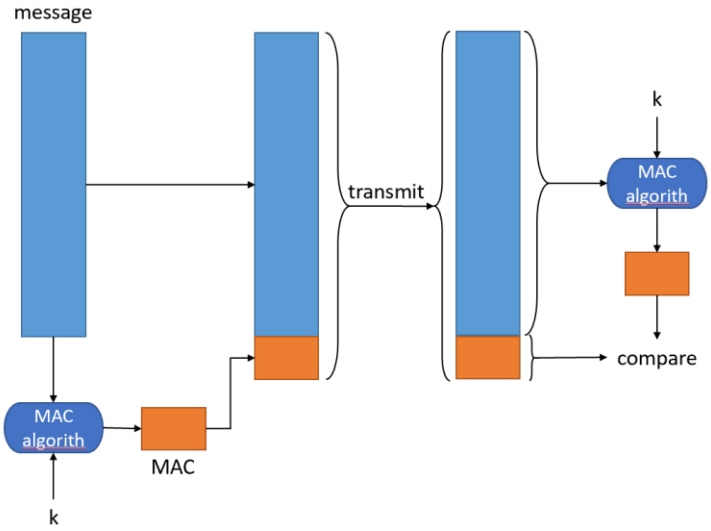


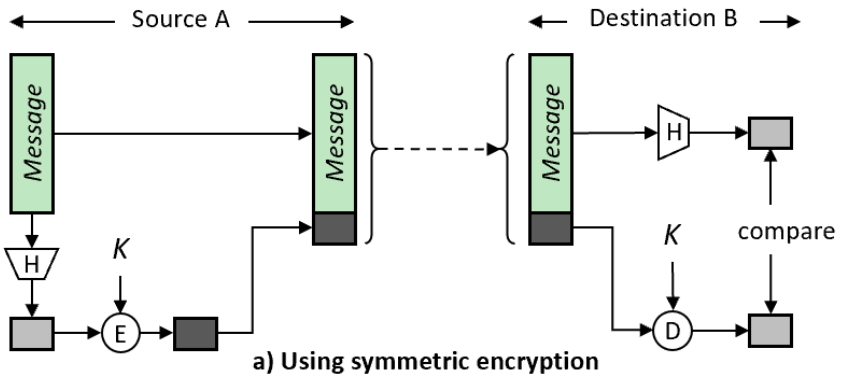


Block cipher encryption (electronic codebook mode)

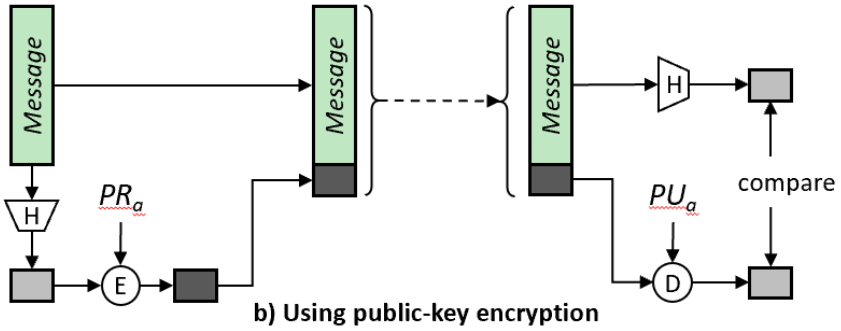


Stream encryption

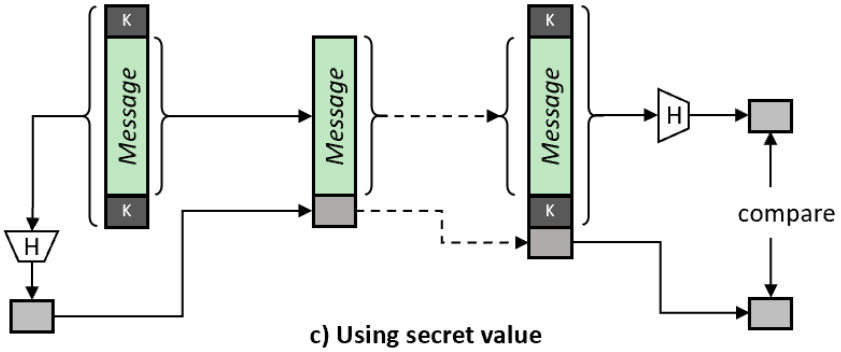




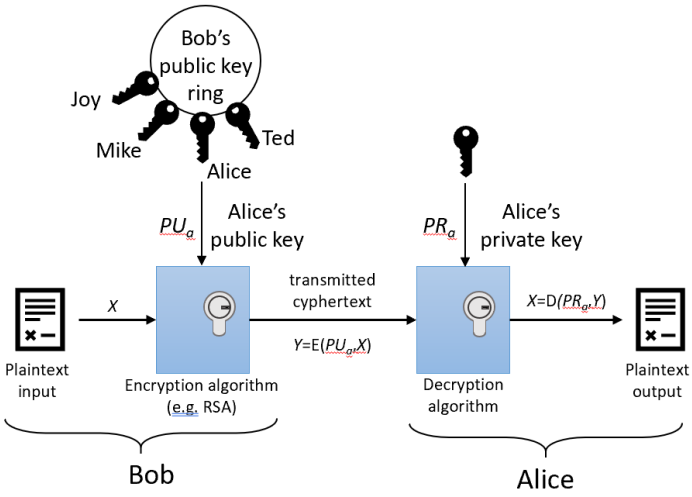
a) Using symmetric encryption



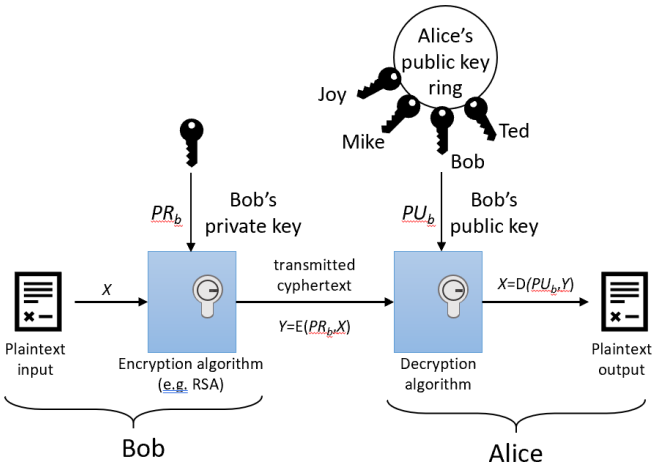
b) Using public-key encryption



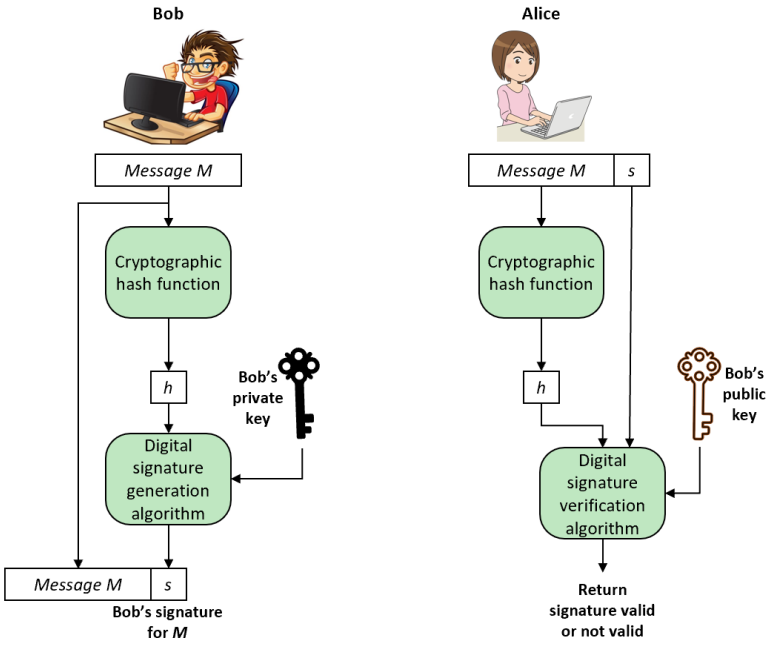
c) Using secret value



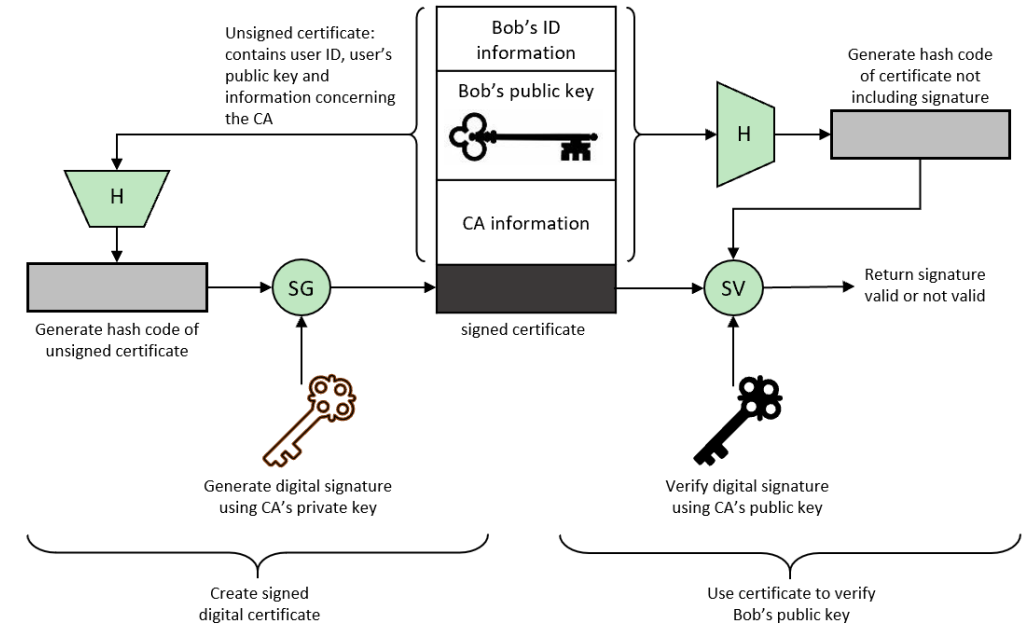
Encryption with public key

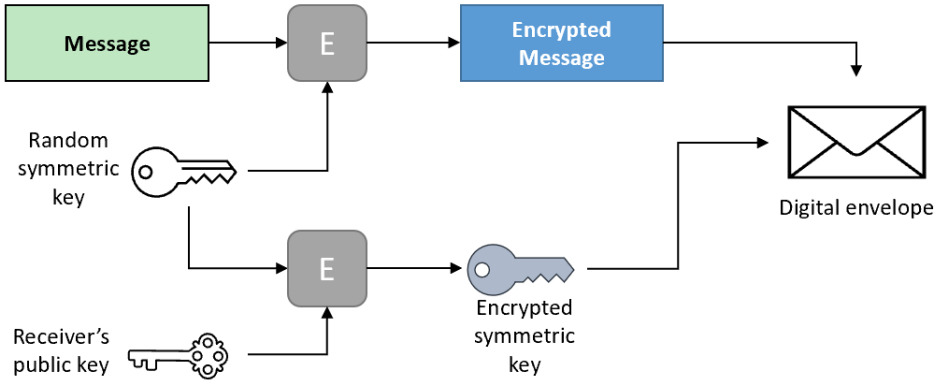


Encryption with private key

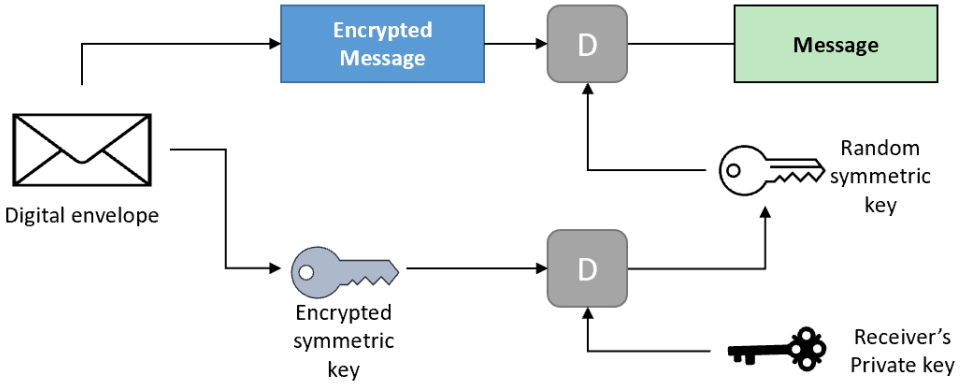


a) Bob signs a message **b) Alice verifies the signature**

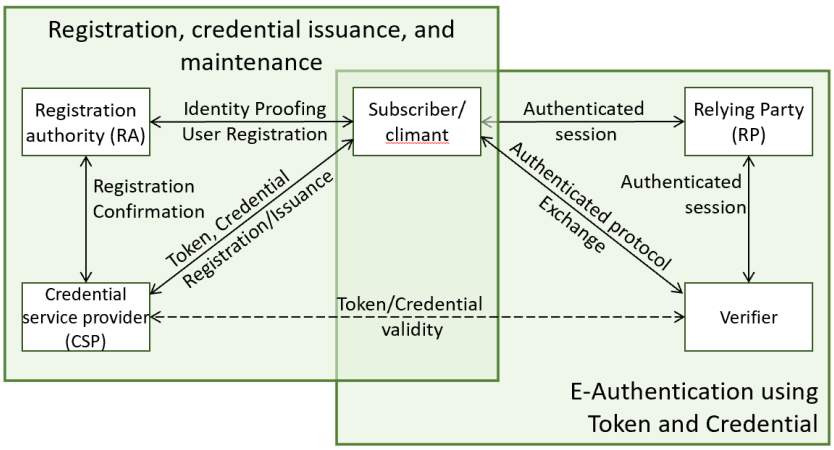




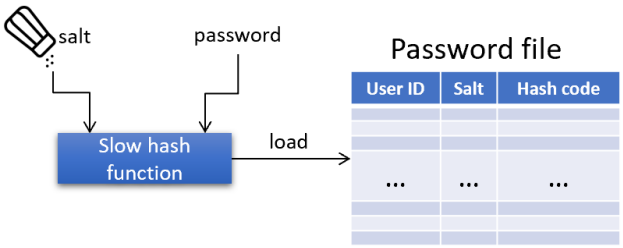
a) Creation of a digital envelope



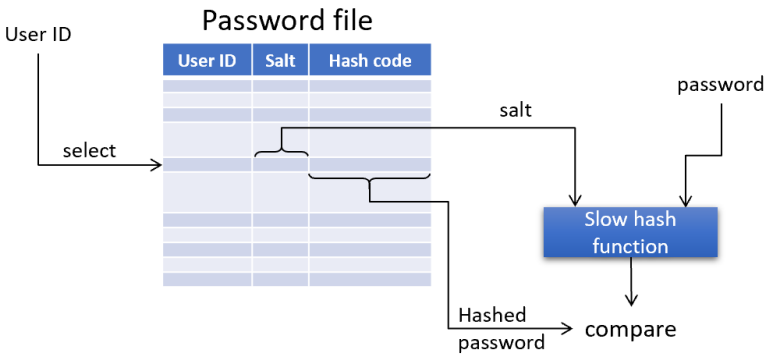
b) Opening a digital envelope



The NIST SP 800-63-3 E-authentication architectural model



a) Loading a new password



b) Verifying a password

U, user —*U*→ *r*: random number, *h*(), *f*(): functions

←*r*, *h*(), *f*()

P' (user password); *r'* = *r*

f(*r'*,*h*(*P'*))→ **if** *f*(*r'*,*h*(*P'*)) = *f*(*r*,*h*(*P*(*U*))) **then** yes **else** no

←yes/no

U, user —*U*→ *r*: random number, *h*(), *f*(): functions

←*r*, *h*(), *f*()

P' →*W'* (Password *P'* to passcode *W'* via token); *r'* = *r*

f(*r'*,*h*(*W'*))→ **if** *f*(*r'*,*h*(*W'*)) = *f*(*r*,*h*(*W*(*U*))) **then** yes **else** no

←yes/no

U, user —*U*→ *r*: random number *E*(): encryption function

←*r*, *E*()

B'→*BT'* biometric; *D'* biometric device; *r'* = *r*

E(*r'*, *D'*, *BT'*)→ *E*$^{-1}$(*E*(*r'*, *D'*, *BT'*)) → (*r'*, *D'*, *BT'*) **if** *r*=*r'* and *D'*=*D* and *BT'*=*BT*(*U*) **then** yes **else** no

←yes/no

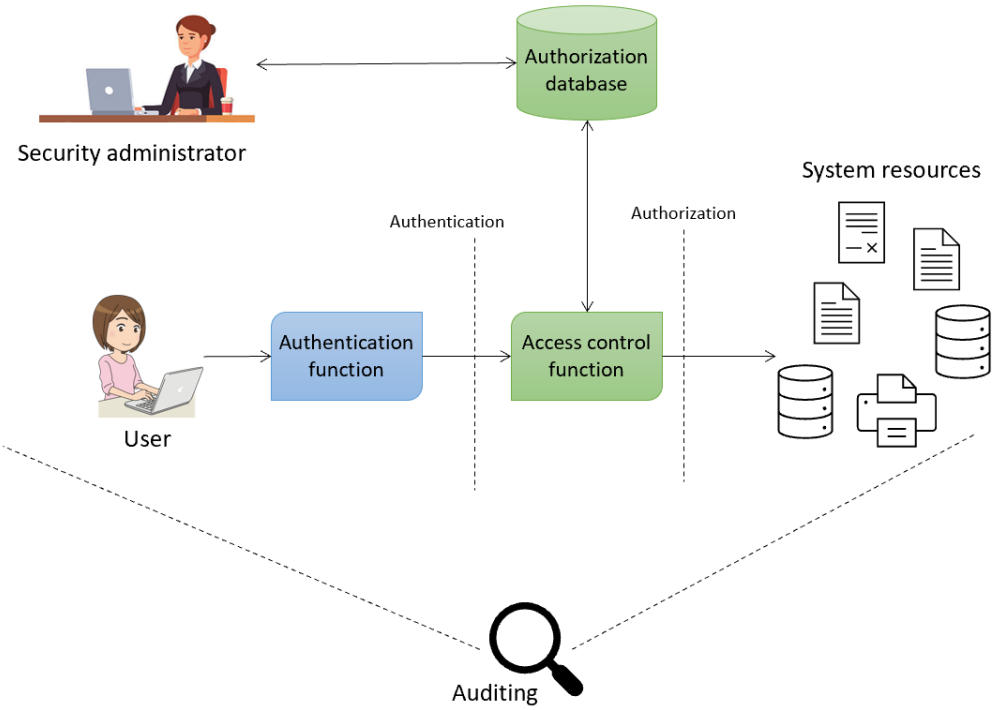
U, user —*U*→ *r*: random number *x*: random sequence challenge *E*(): function

←*r*,*x*, *E*()

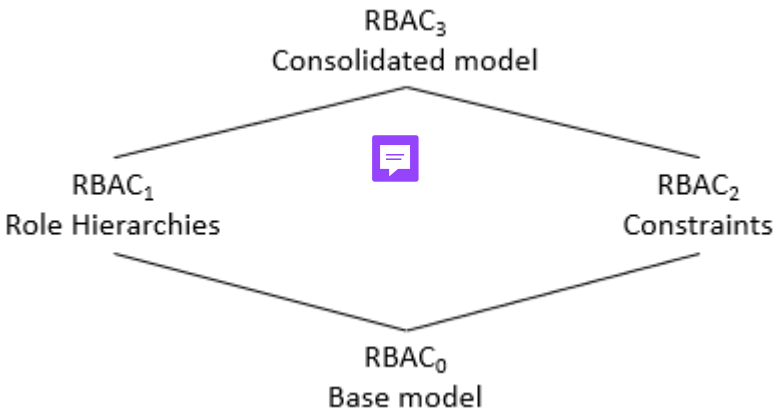
B',*x'*→*BS*(*x'*); *r'* = *r*

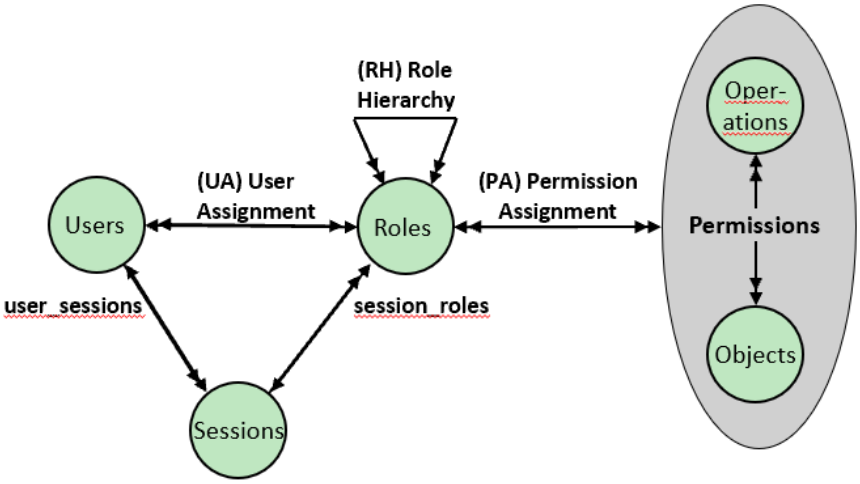
E(*r'*, *BS'*(*x'*))→ *E*$^{-1}$(*E*(*r'*, *BS*(*x'*))) → (*r'*, *BS'*(*x'*)) **if** *r*=*r'* and *BS'*(x')=*BT*(*U*) **then** yes **else** no

←yes/no

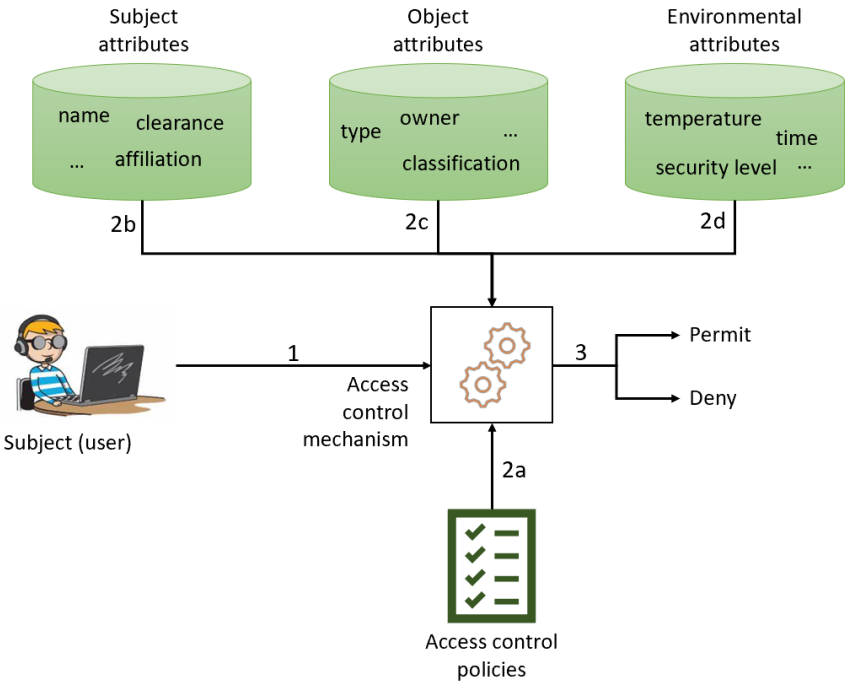


Rule	Command (by S_0)	Authorization	Operation
R1	**transfer** $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ **to** S, X	'α^*' in $A[S_0, X]$	store $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ in $A[S, X]$
R2	**grant** $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ **to** S, X	'owner' in $A[S_0, X]$	store $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	delete α in $A[S, X]$
R4	$w \leftarrow$ **read** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	copy $A[S, X]$ into w
R5	**create object** X	none	add column for X to A; store 'owner' in $A[S_0, X]$
R6	**destroy object** X	'owner' in $A[S_0, X]$	delete column for X from A
R7	**create subject** S	none	add row for S to A; execute **create object** S; store 'owner' in $A[S_0, S]$; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_0, S]$	delete row for S from A; execute **destroy object** S


RBAC3
Consolidated model
RBAC1
Role Hierarchies
RBAC2
Constraints
RBAC0
Base model


(RH) Role Hierarchy
(UA) User Assignment
(PA) Permission Assignment
Users
Roles
Oper-ations
Permissions
Objects
user_sessions
session_roles
Sessions

RBAC models


Subject attributes
name
clearance
...
affiliation
Object attributes
type
owner
...
classification
Environmental attributes
temperature
time
security level
...
2b
2c
2d
1
Access control mechanism
3
Permit
Deny
Subject (user)
2a
Access control policies

Part 2

- Explain the need for security in databases
- Explain what is an SQL-injection attack and what are its “avenues”. Provide an example of an SQL-injection attack
- Discuss advanced persistent threats
- Define a virus and a worm and discuss their differences.
- Explain the purpose and the methodologies for intrusion detection

Figure 5.1

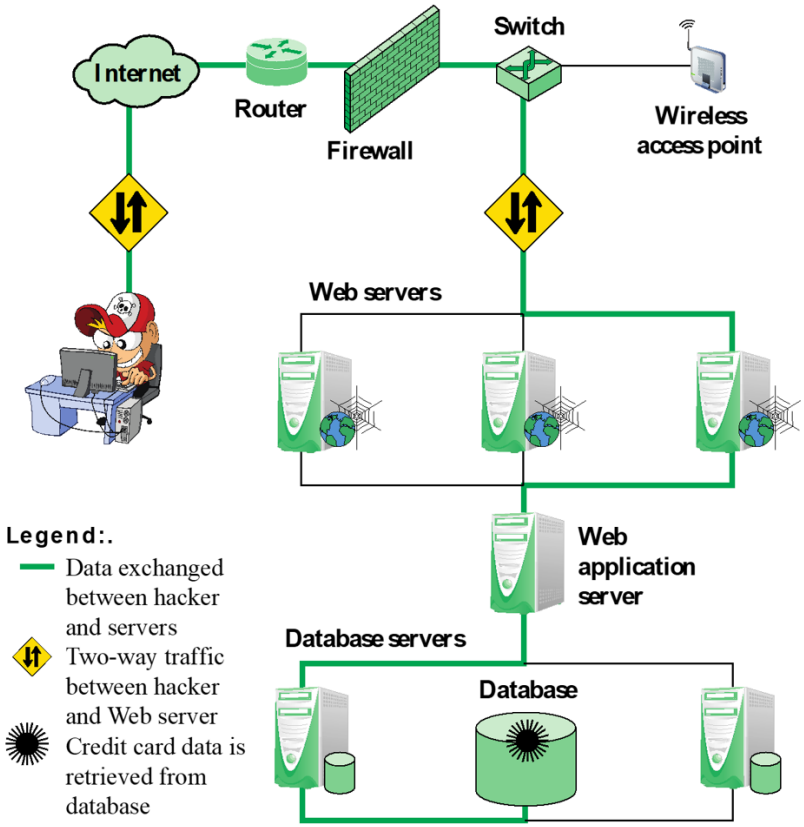


Figure 5.2

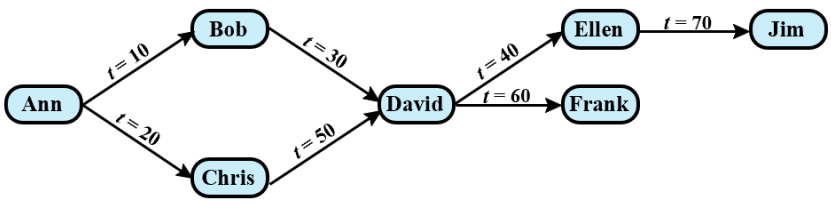


Figure 5.3

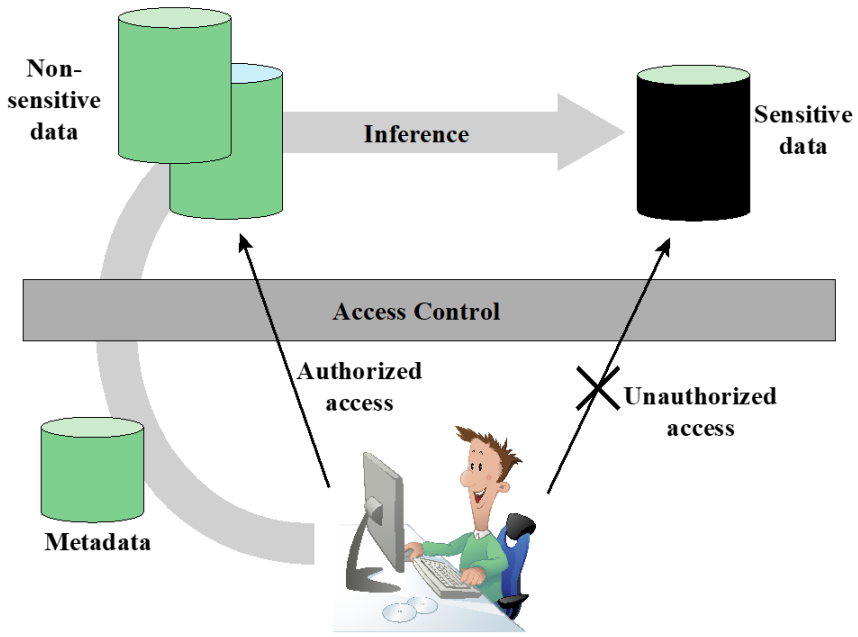


Figure 5.4 – inference example

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware
Cake pan	online only	12.99	housewares
Shower/tub cleaner	in-store/online	11.99	housewares
Rolling pin	in-store/online	10.99	housewares

(a) Inventory table

Availability	Cost ($)
in-store/online	7.99
online only	5.49
in-store/online	104.99

Item	Department
Shelf support	hardware
Lid support	hardware
Decorative chain	hardware

(b) Two views

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware

(c) Table derived from combining query answers

Figure 5.5

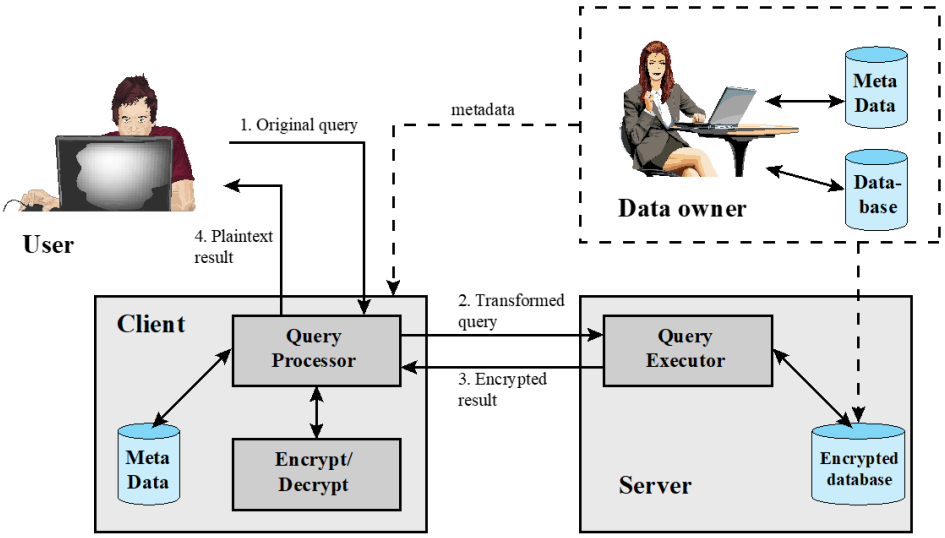


Figure 5.6 - range queries in encrypted DB

Employee table

Eid	Ename	Salary	Addr	Did
23	Tom	70K	Maple	45
860	Mary	60K	Main	83
320	John	50K	River	50
875	Jerry	55K	Hopewell	92

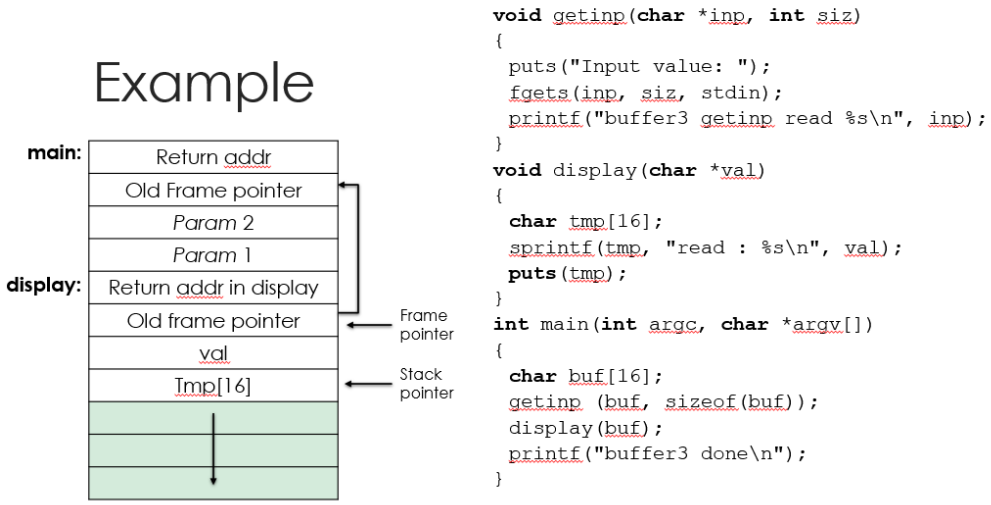
Encrypted employee table with indexes

E(k,B)	I(Eid)	I(Ename)	I(salary)	I(Addr)	I(Did)
11010011101000011010...	1	10	3	7	4
11101010010001011010...	5	7	2	7	8
000001110100110101001...	2	5	1	9	5
100111110111010000101...	5	5	2	4	9

Part 3

- Explain the purpose of the shellcode in a buffer overflow attack and explain its main functionalities.
- Discuss the following defenses against stack overflow: random canary, Stackshield and Return Address Defender, stack space randomization, guard pages, executable address space protection
- Explain the relationship between software security, quality and reliability
- Discuss the best practices for defense programming
- Explain the concept of operating system hardening and its main steps
- Explain the following protection methods: system call filtering, sandbox, code signing, compile-based/language-based protection.
- Discuss the security concerns about virtualization.

Fig. 6.1




```
void getinp(char *inp, int siz)
{
  puts("Input value: ");
  fgets(inp, siz, stdin);
  printf("buffer3 getinp read %s\n", inp);
}
void display(char *val)
{
  char tmp[16];
  sprintf(tmp, "read : %s\n", val);
  puts(tmp);
}
int main(int argc, char *argv[])
{
  char buf[16];
  getinp (buf, sizeof(buf));
  display(buf);
  printf("buffer3 done\n");
}
```

Fig. 6.2

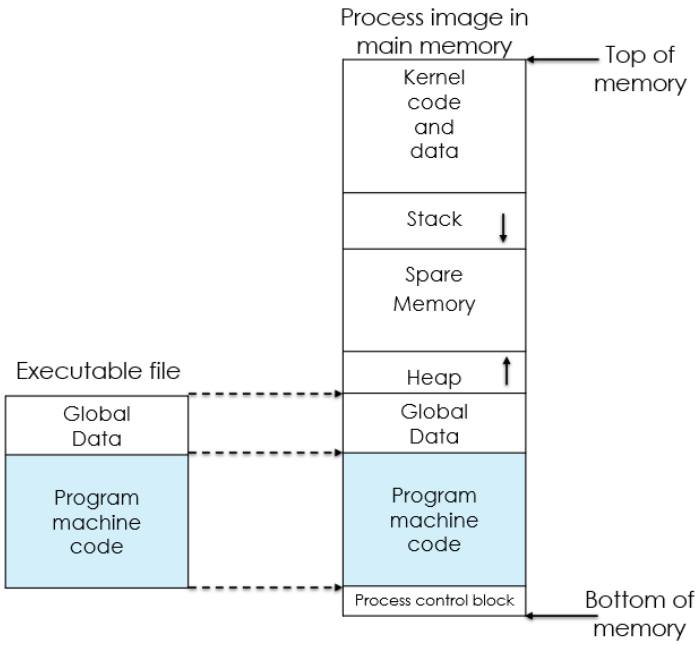


Figure 6.3

```
/* record type to allocate on heap */
typedef struct chunk {
    char inp[64];
        /* vulnerable input buffer */
    void (*process)(char *);
        /* pointer to function to
           process inp */
} chunk_t;

void showlen(char *buf)
{
    int len;
    len = strlen(buf);
    printf("buffer5 read %d chars\n", len);
}
```

```
int main(int argc, char *argv[])
{
    chunk_t *next;
    setbuf(stdin, NULL);
    next = malloc(sizeof(chunk_t));
    next->process = showlen;
    printf("Enter value: ");
    gets(next->inp);
    next->process(next->inp);
    printf("buffer5 done\n");
}
```

Figure 6.4

```
/* global static data, targeted for attack
*/
struct chunk {
    char inp[64]; /* input buffer */
    void (*process)(char *);
    /* pointer to function to process it */
} chunk;

void showlen(char *buf)
{
    int len;
    len = strlen(buf);
    printf("buffer5 read %d chars\n", len);
}
```

```
int main(int argc, char *argv[])
{
    setbuf(stdin, NULL);
    chunk.process = showlen;
    printf("Enter value: ");
    gets(chunk.inp);
    chunk.process(chunk.inp);
    printf("buffer6 done\n");
}
```

Figure 6.5

```
int main(int argc, char *argv[]) {
  int valid = FALSE;
  char str1[8];
  char str2[8];

  next_tag(str1);  // puts a valid string in str1, say "START"
  gets(str2);      // reads str2 from stdin
  if (strncmp(str1, str2, 8) == 0)
    valid = TRUE;
  printf("buffer1: str1(%s), str2(%s), valid(%d)\n",
                   str1, str2, valid);
}
```

Basic buffer overflow C code

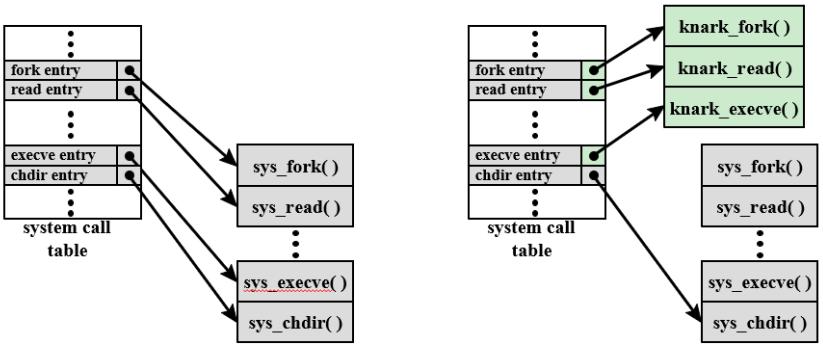
```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

Basic buffer overflow example runs

Memory Address	Before gets(str2)	After gets(str2)	Contains Value of
....	....	....	
bffffbf4	34fcffbf 4...	34fcffbf 3...	argv
bffffbf0	01000000	01000000	argc
bffffbec	c6bd0340 ...@	c6bd0340 ...@	return addr
bffffbe8	08fcffbf	08fcffbf	old base ptr
bffffbe4	00000000	01000000	valid
bffffbe0	80640140 .d.@	00640140 .d.@	
bffffbdc	54001540 T..@	4e505554 NPUT	str1[4-7]
bffffbd8	53544152 STAR	42414449 BADI	str1[0-3]
bffffbd4	00850408	4e505554 NPUT	str2[4-7]
bffffbd0	30561540 0V.@	42414449 BADI	str2[0-3]
....	....	....	

Stack grows in this way

Figure 6.6



(a) Normal kernel memory layout (b) After knark install

System Call table modification by rootkit

Figure 6.7

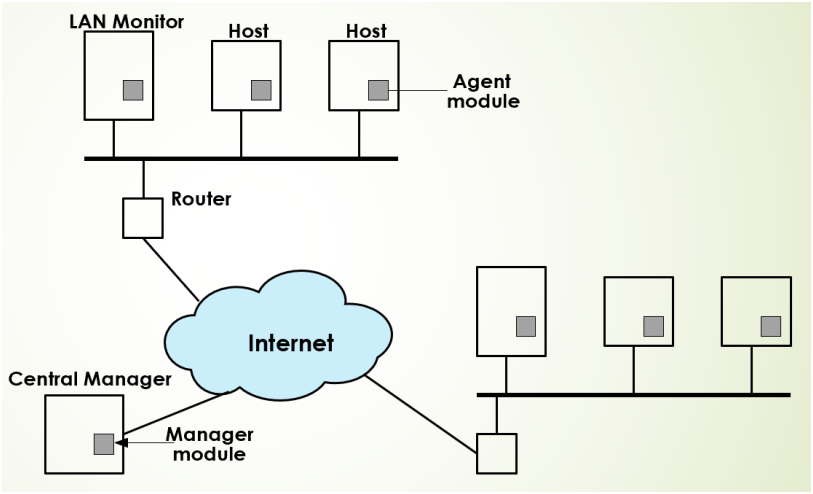


Figure 6.8

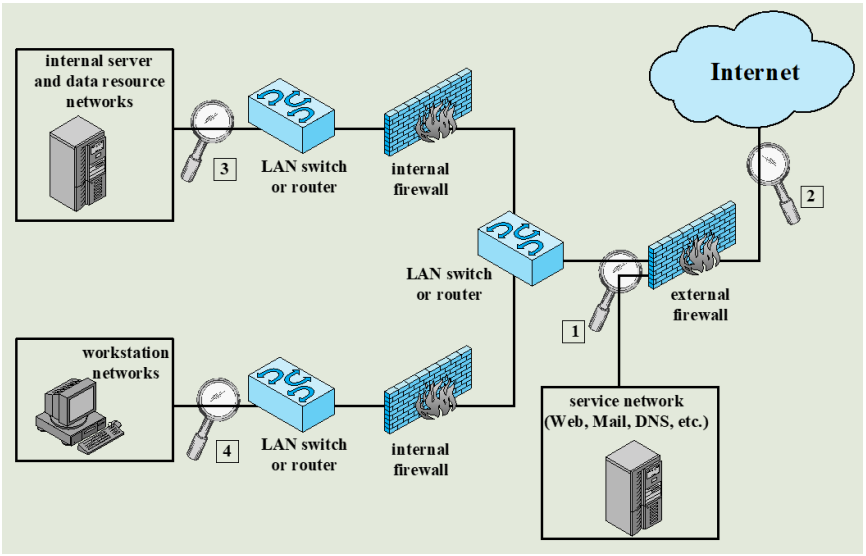


Figure 6.9

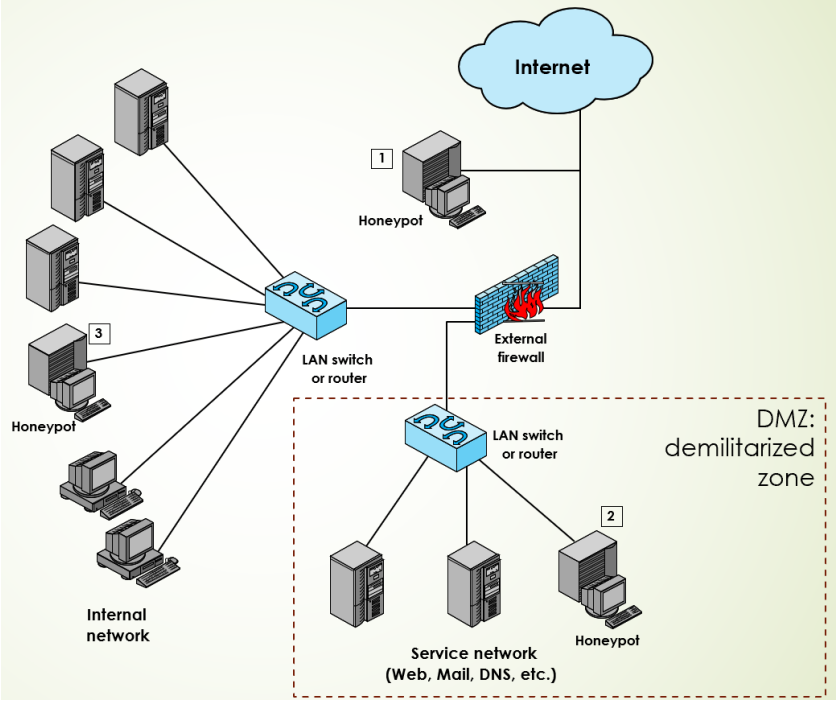


Figure 10.1

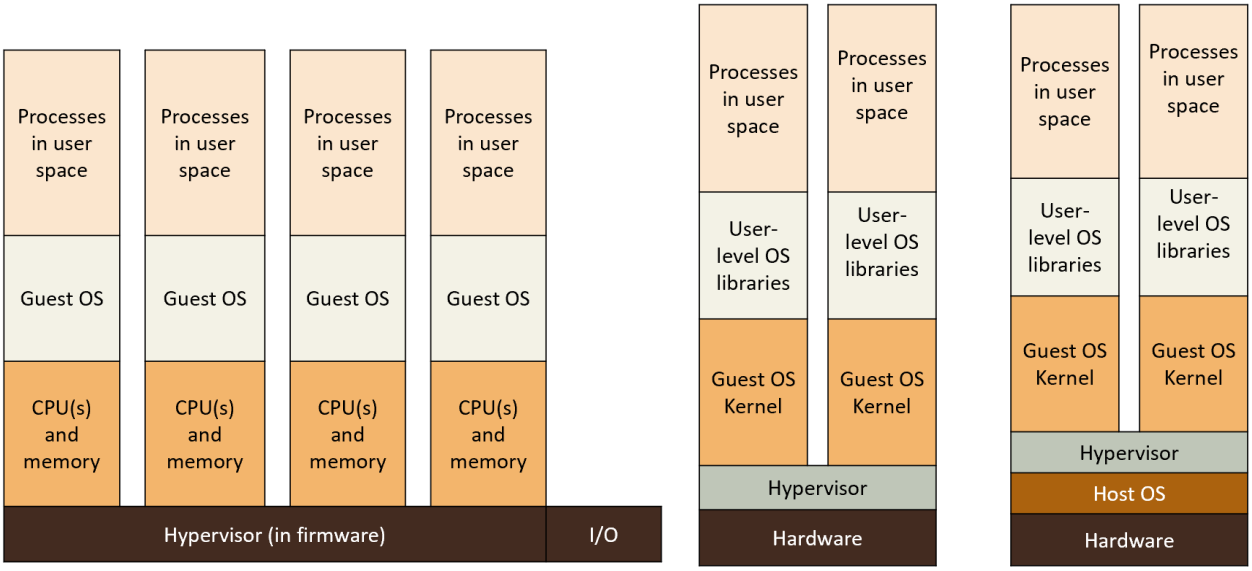


Figure10.2

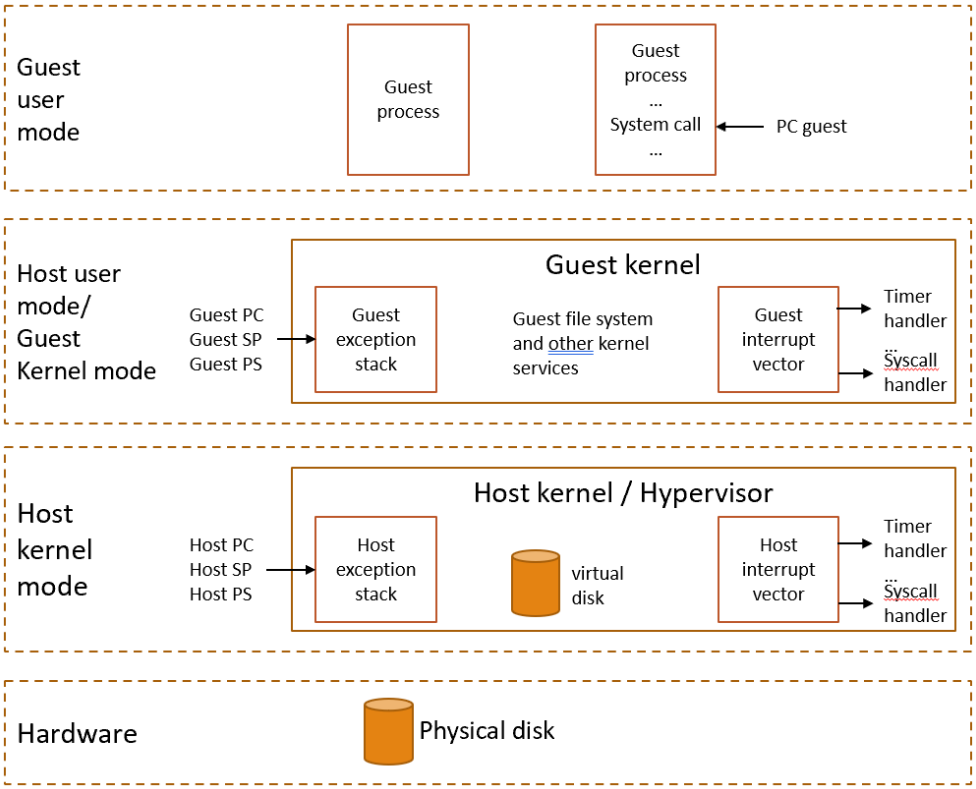


Figure 10.3

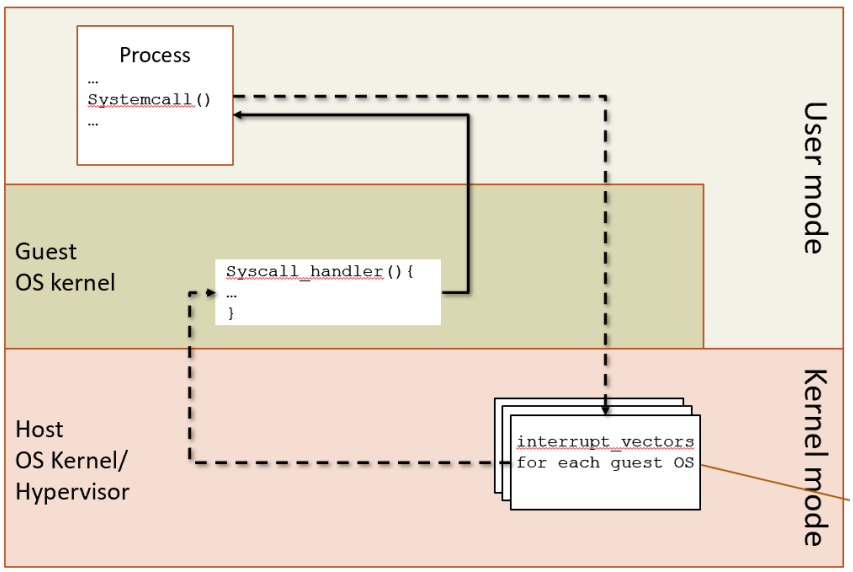


Figure 10.4

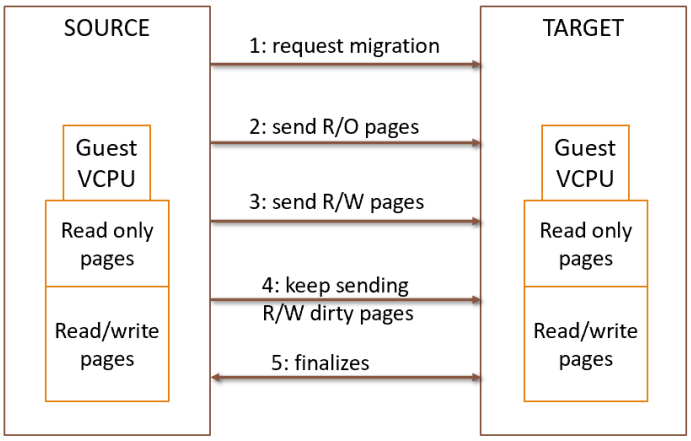
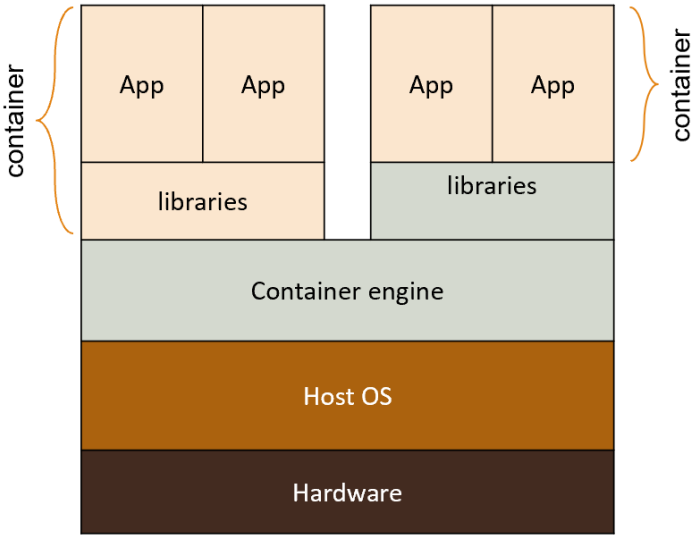


Figure 10.5



Part 4

- Explain the basic security model of Unix, the concept of user and group, and the permissions. Explain the use of the sticky bit, SetUID, SetGID
- Explain the potential vulnerability due to the use of SetUID
- Explain the meaning and use of chroot jail
- [not for A.A. 2024/25] Explain the security model of SELinux. Discuss the subjects and objects; the roles and domains and the inheritance
- Explain the security model of Windows. Discuss its discretionary access control its mandatory access control
- In Windows discuss the purpose of these components: Security reference monitor, Local security authority, Security account manager, Active directory
- Discuss the purpose of integrity levels in Windows
- Discuss the Byzantine Generals Problem
- Discuss the vulnerabilities of and attacks against blockchains

Figure 12.1

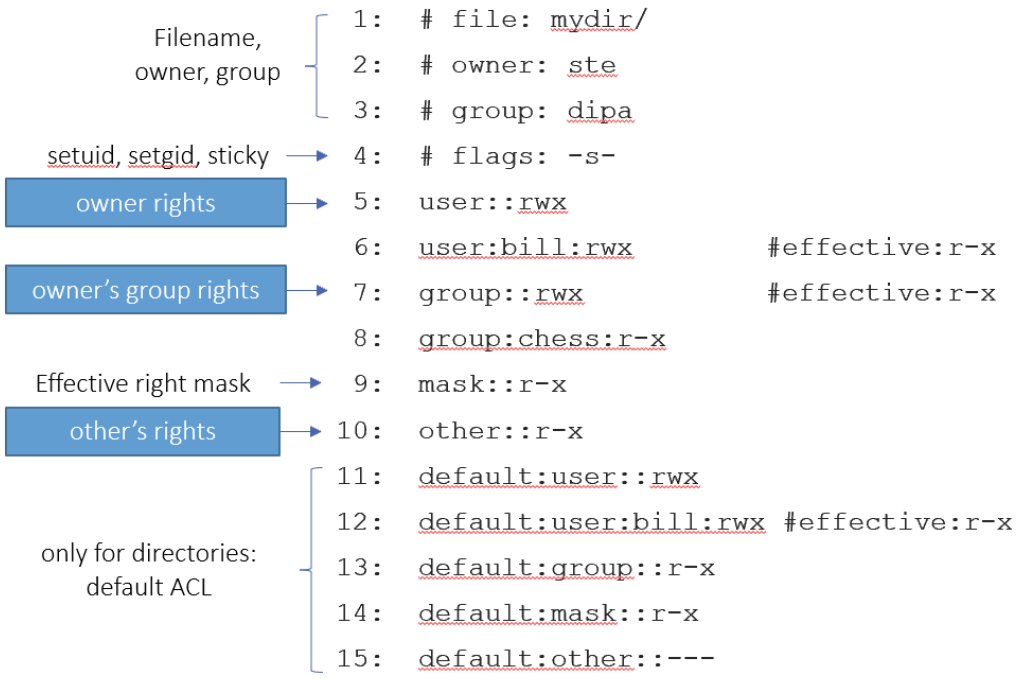


Fig. 13.1

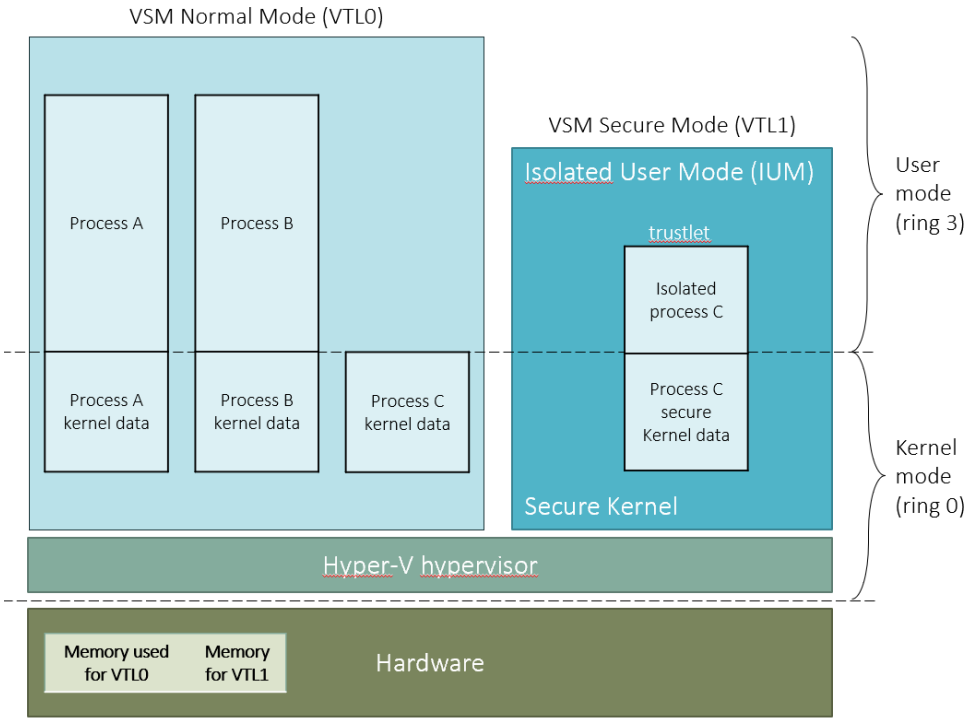


Figure 13.2

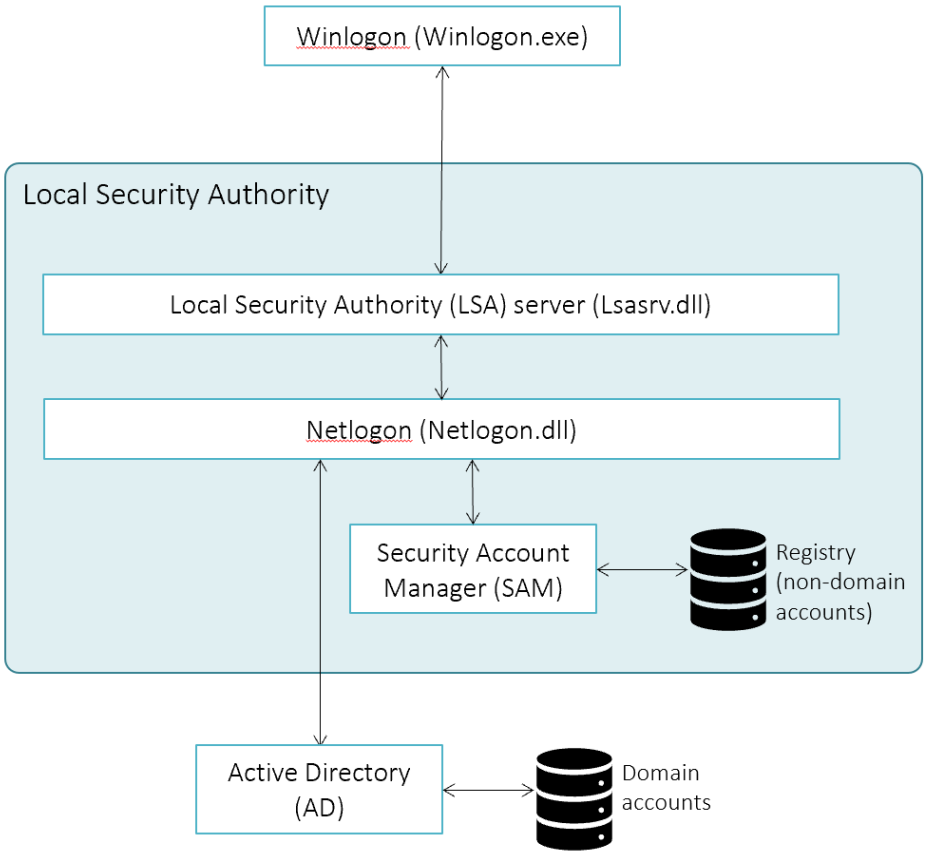


Figure 13.3 - Explain the concept of “privilege” in Windows and present some privileges that concern the file system

```
PS C:\Users\stefa> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege name                Description                                        State
============================= ========================================== ========
SeShutdownPrivilege           System shutdown                            Disabled
SeChangeNotifyPrivilege       Ignore cross-checking                      Enabled
SeUndockPrivilege             Removing your computer from the housing    Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set             Disabled
SeTimeZonePrivilege           Changing the time zone                     Disabled
```

Figure 13.4 – explain the structure and purpose of a security descriptor. Discuss some examples of access rights concerning the file system

```
PS C:\Users\stefa> get-acl c:\Windows | Format-List

Path   : Microsoft.PowerShell.Core\FileSystem::C:\Windows
Owner  : NT SERVICE\TrustedInstaller
Group  : NT SERVICE\TrustedInstaller
Access : CREATOR OWNER                  Allow  268435456
         NT AUTHORITY\SYSTEM            Allow  268435456
         NT AUTHORITY\SYSTEM            Allow  Modify, Synchronize
         BUILTIN\Administrators         Allow  268435456
         BUILTIN\Administrators         Allow  Modify, Synchronize
         BUILTIN\Users                  Allow  -1610612736
         BUILTIN\Users                  Allow  ReadAndExecute, Synchronize
         NT SERVICE\TrustedInstaller    Allow  268435456
         NT SERVICE\TrustedInstaller    Allow  FullControl
```

Figure 13.5 – access token

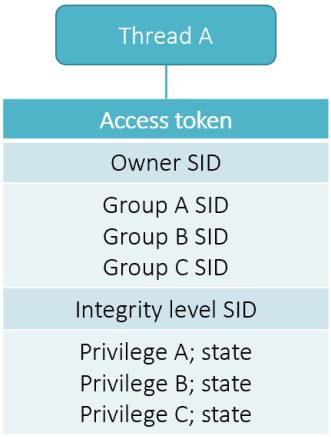


Figure 14.1

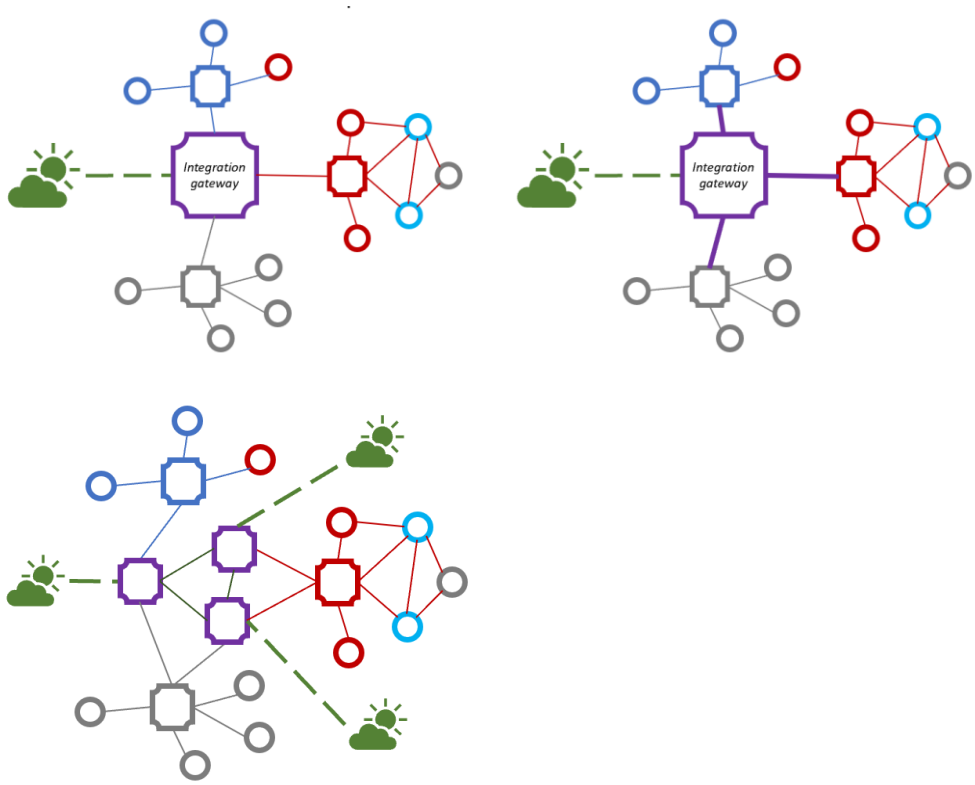


Figure 14.2

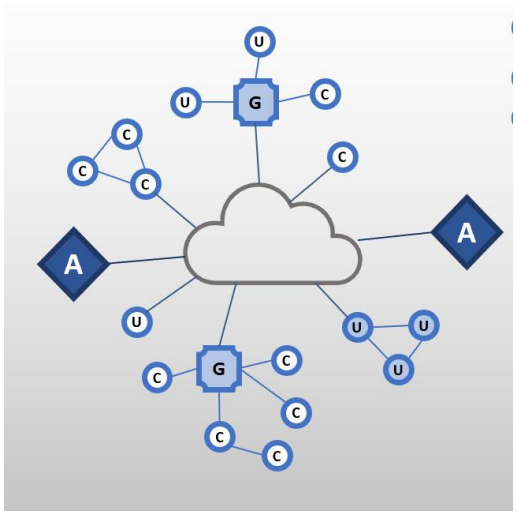


Figure 14.3

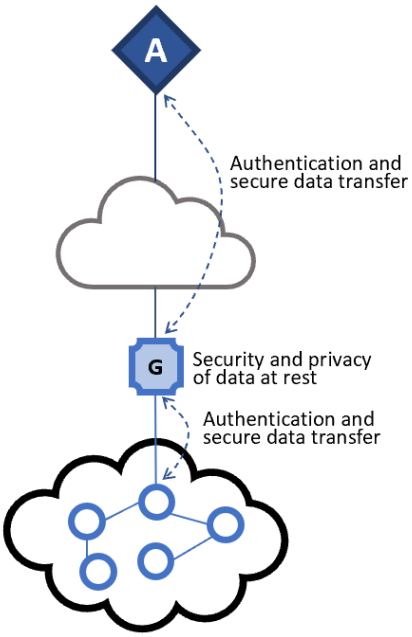


Figure 15.1

```
function create_block() {
    // executed by miner m_i when the
    // transaction pool is full, to
    // propose the append of a new block

// let B_i be the blockchain at miner m_i
b = new block;
b.transactions = get_transactions(pool);
while true do
   nounce=local-random-coin()
   b.pow=nounce;
   b.parent=last_block(B_i);
   if solve_cryptopuzzle(b) {
       broadcast(b);      //included itself
       break;
   }
}

function update(b) {
    // executed by miner M_i upon
    // reception of block b to append

  // let B_i be the blockchain at miner M_i
  if !check_validity(b) { reject b; return; }
  B_i = B_i ∪ b;
}
```

Figure 15.2

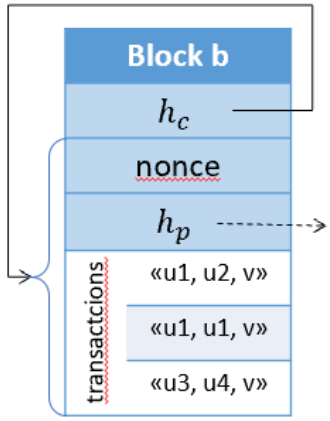


Figure 15.3 - Discuss the problem of forks in the blockchain

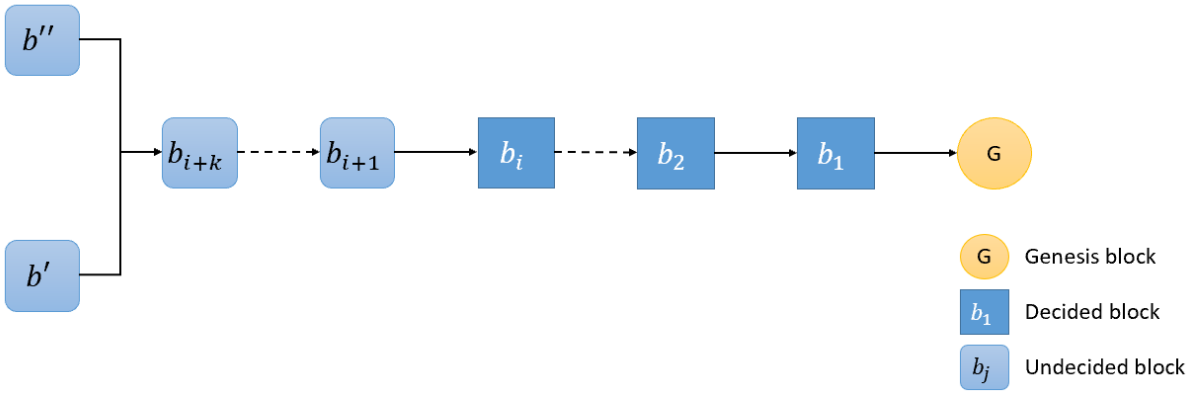


Figure 15.4