

**2 PROTECTION OVERVIEW BASIC**

**3 UNIX PAGE/SEGMENT PROTECTION**

**4 UNIX PROTECTION DOMAIN SWITCH - Setuid Program**

**5 UNIX PROCESS FILE PROTECTION**

**6 MULTI-LEVEL SECURITY**

**7 UNIX UNAUTHORIZED ENTRY PROTECTION - Password**

**8 CHANGING PASSWORDS-1 WAY FUNCTIONS**

**9 CRYPTOGRAPHY 1 PRIVATE KEY**

**10 CRYPTOGRAPHY 2 PUBLIC KEY**

**11 PUBLIC KEY CRYPTOGRAPHY RSA ALGORITHM AND EXAMPLE**

**12 MESSAGES AND KEY DISTRIBUTION**

### Objects To Be Protected

Resources, ex. [Files, Processes, Memory Areas, Pipes, Devices]

|      |                         | F1      | F2  | F3                                     | F4     | F5     | TP1 | Sg1       | D2           |     | Dn |  |  |  |  |
|------|-------------------------|---------|-----|--|--------|--------|-----|-----------|--------------|-----|----|--|--|--|--|
| User | Password Authentication | Domains | D1  | $\begin{matrix} O,x \\ w \end{matrix}$ |        | $x$    | $r$ |           |              |     |    | <span style="background-color: #90EE90;">switch</span> |  |  |  |
|      |                         |         | D2  | $x$                                    |        | $O\ x$ |     | $r$       |              |     |    |  |  |  |  |
|      |                         |         | D3  |  | $O\ x$ | $w\ r$ |     | $O\ w\ r$ | $x\ x$       | $r$ |    |  |  |  | <span style="background-color: #90EE90;">switch</span> |
|      |                         |         | ... |  |        |        | ... |           |              |     | .. |  |  |  |  |
|      |                         |         | Dn  | $C$                                    | $w$    |        |     | $r$       | $O\ x\ O\ x$ |     |    |  |  |  |  |

**Protection Matrix(Sparse)**

Users /Processes/Procedures Operate In Protection Domains

**\*Basic Domain Rights To Files**

**x** = execute

**w** = write

**r** = read

**O** = owner (can give any right in its column to other domains)

**C** = control (can remove or add any right to an object in its column) ex. Dn has complete control over D1

switch = domain switch: gives Di access to Dj

Allowed [Disallowed] Operations (Rights Privileges)

**D1:** F1, $O,w,x$ ; F3, $x$ ; F4, $r$ ; TP1, $r$ ; D2; $in/out$   
**D2:** F1, $x$ ; F3, $O,x$ ; TP1, $r$   
 ...  
**D3:** F2,  $O,w,r$ ; F4, $O,w,r,x$ ; SEG1, $r$   
 ...  
 Domain **CAPABILITY** Lists  
 (Compact Representation)

**F1:** D1, $O,w,x$ ; D2, $x$ .  
**F2:** D3, $O,w,r$ ; Dn, $w$ .  
 ...  
**F5:** D3, $O$ ; All except D1,  $r$   
 ...  
 Object **ACCESS** List  
 (Compact Representation)

Objects Are Secure (Safe from Dis-allowed Use) If they are Properly Protected.

There is a database, distributed throughout the computer hardware and Software which effectively contains the Rights of Subjects to the Objects. This is kept in Capability and/or Access Control Lists

In UNIX the Owner of a file is the only one that can determine who has access to that file. If there is a domain which has a C for a file it too can determine who has access to that file.

*\*Basic* because one can imagine much more complex rules-ex. F1 can be used for a time,  $t$  (weeks), by  $x$ ,  $y$  and  $z$  for read-only but by  $a$ ,  $b$ , provided  $x$  nor  $y$  have used in within the previous 2 days. Prof Minsky's Law based Interactions.

### ATTACKS

The "request for password screen" normally put up by the system can be mimiced by a malicious program which can thus get the password. That is, one needs confidence that one is actually in contact with machine, as well as the machine's need to to know that the user is legitimate.

An Attack By Mimic-ing the System Asking for Password

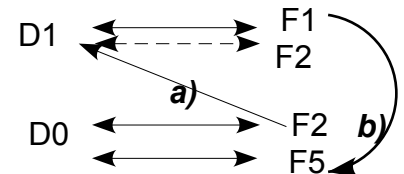
*The Plan*

a) D0 gives permission and induces D1 to execute and own F2, a program originally owned by D0 (by advertising its desirability maybe it contains a game)

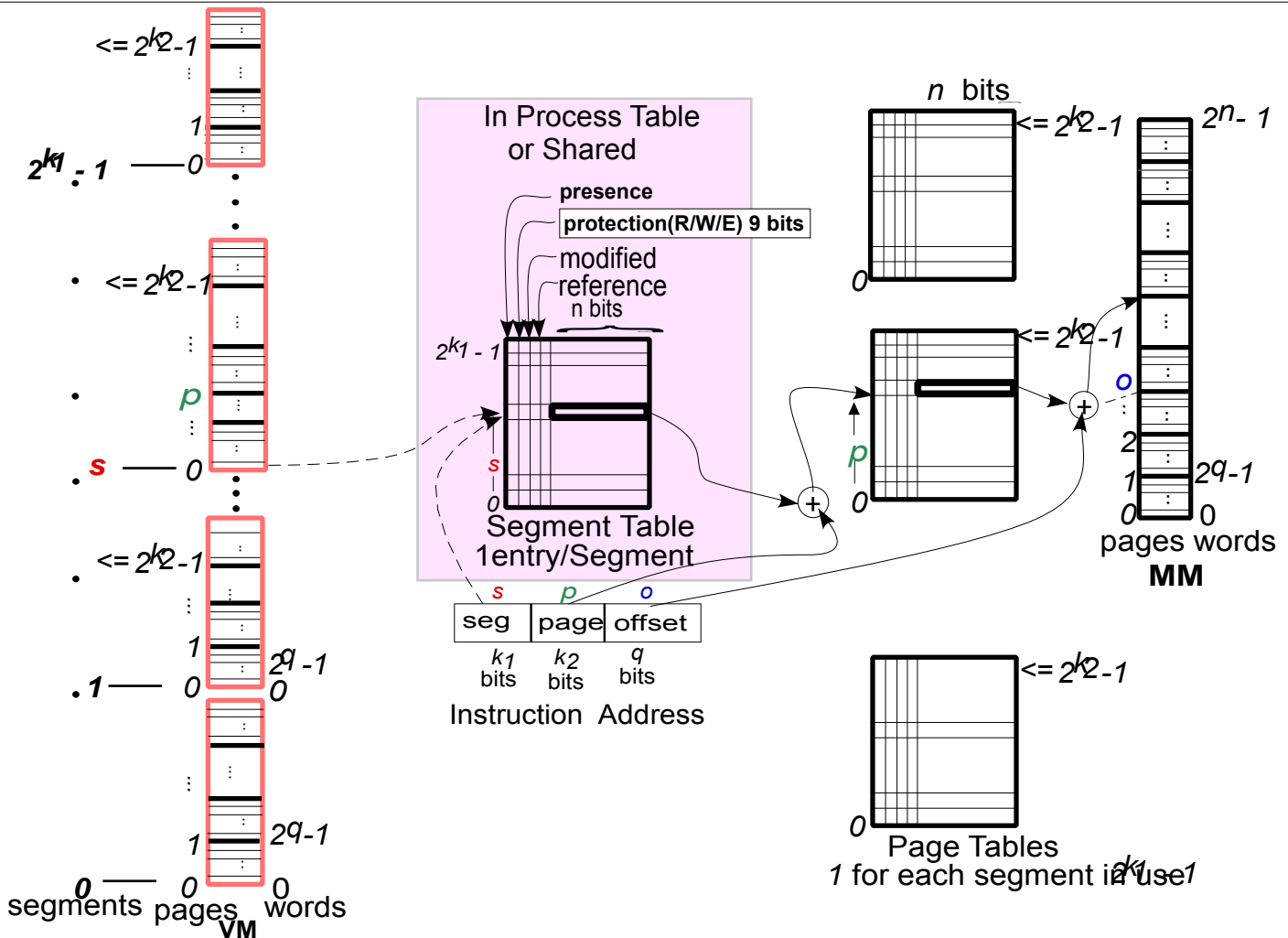
b) D1 runs program in F2 which has access to D1's and D0's files and D0 now copies F1 into F5

**Stolen!**

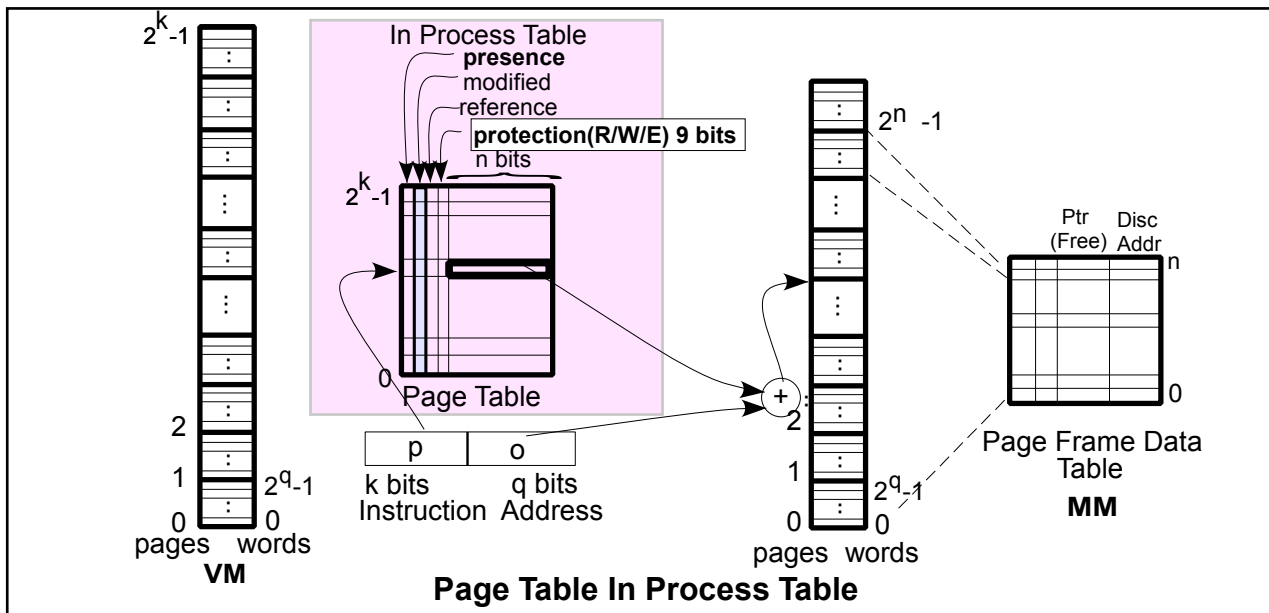
An Attack By Giving Ownership **Trojan Horse**



## PROTECTION OVERVIEW BASIC



**Shared Segment Table-One Accessible To All Processes (IBM 801)**



**Page Table In Process Table**

If the Segment or Page Table is in the Process Table it determines the **CAPABILITIES** of that Process, other Processes cannot reach it because of its location. In this case the 3 **protection** bits can indicate whether the page is Write, Read, Execute or any combination of these.

If the Segment (or page Table) is shared it contains an **ACCESS** list in its **protection** bits. It could have 9 protection bits three for Access Rights of the owner and 3 for the Group, and 3 for Every one else.

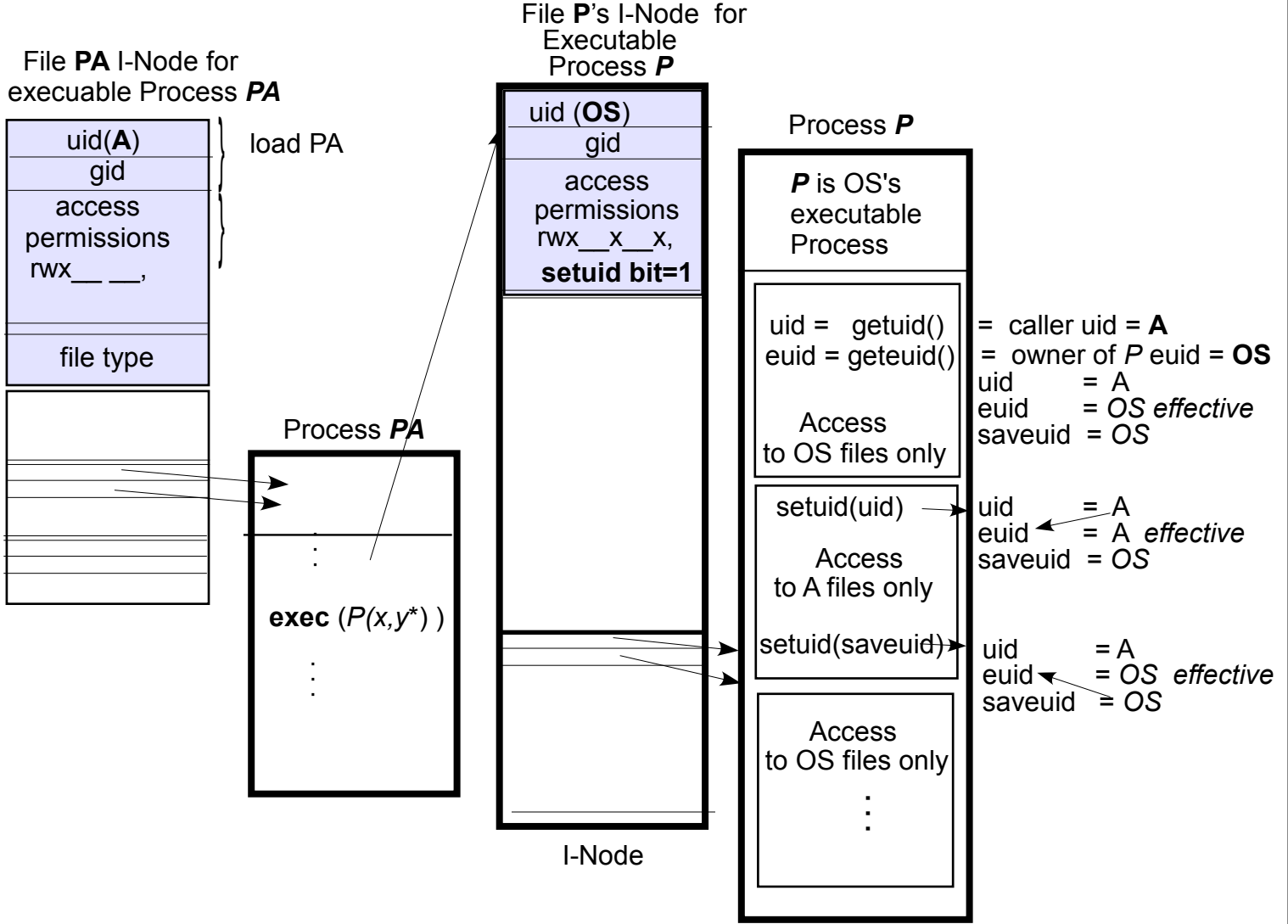
**UNIX PAGE/SEGMENT PROTECTION**

**Purpose:** In order to give access to U's files in a OS process P when P is called by user U.

**Example: Printing:**

OS program P is an executable program for Printing. which, of course, wants exclusive access to the Printer. On the other hand P must print files owned by users, say A, user that is to print. A must have access to P so that P can get access to their files to be printed. This requires a context switch while running P. The setuid permission bit being set to 1 allows such a context switch.

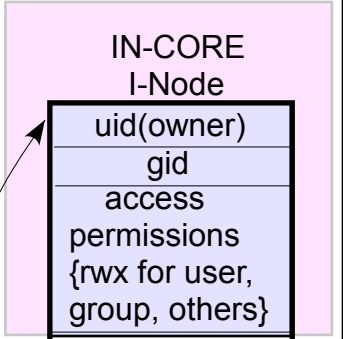
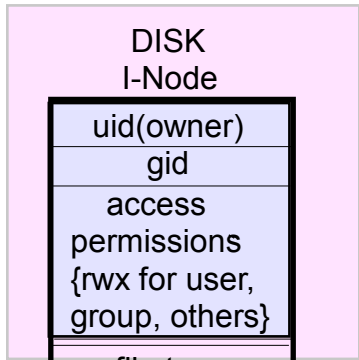
| File | File Owner | Permissions                  |
|------|------------|------------------------------|
| P    | OS         | [r,w,,x, setuid (= switch);] |
| PA   | A          | [r,w,x]                      |



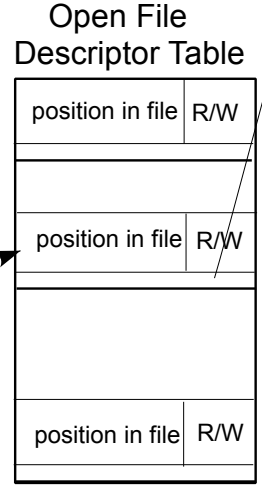
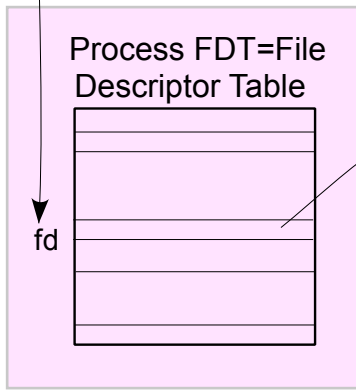
Basic idea setuid(X) X determines which id, uid or saveuid is effective = (euid)

PA owned by A does an exec on OS's process P. OS allows this exec call to go through for A with rwx\_x\_x,bits in I-node.of file P. P is a setuid file, OS can allow A to access some of OS's files and also some of A's files in process P of file P.

**UNIX PROTECTION DOMAIN SWITCH - Setuid Program**

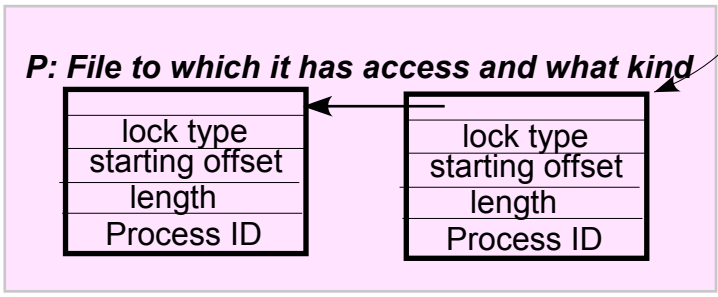


```
fd = open(path_filename, (read,write,read&write))
fd = create(path_filename, mode)
q = read(fd,buffer,nbytes)
```



File **CAPABILITIES** Protection  
Each Process Has Its Own FDT

File **ACCESS** Protection  
*F: accesses allowed*



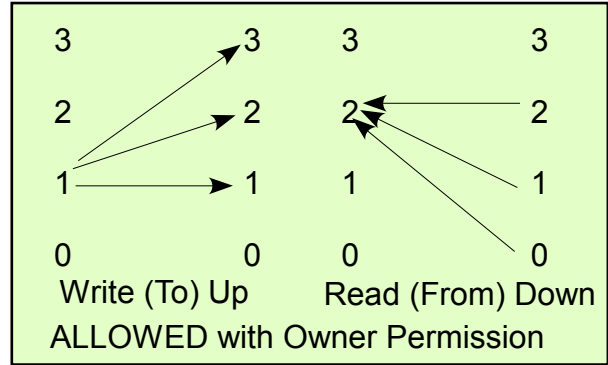
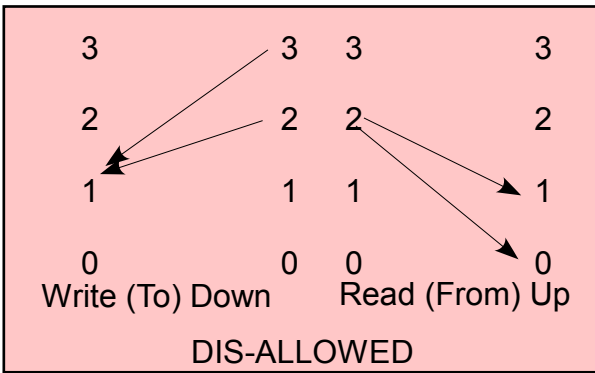
```
lock(fd,startbyte, length, (readshared, writeexcl), (block,rtrnstaatus))
```

File **ACCESS** Protection

**UNIX PROCESS FILE PROTECTION**

## Multilevel Security: of Files & Owners

ex., Unclassified 0, Confidential 1, Secret 2, Top Secret 3, Level Applies to Users and Files



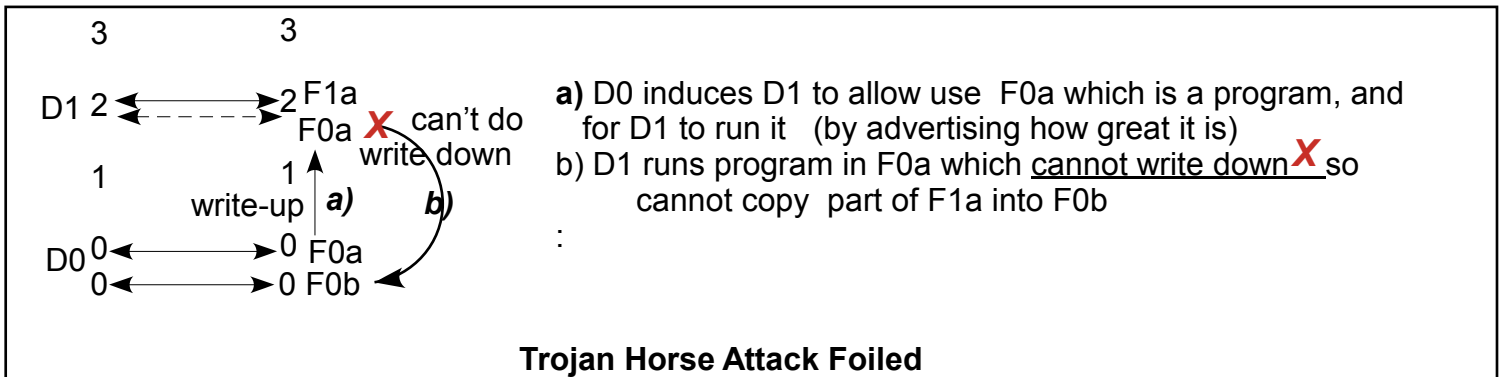
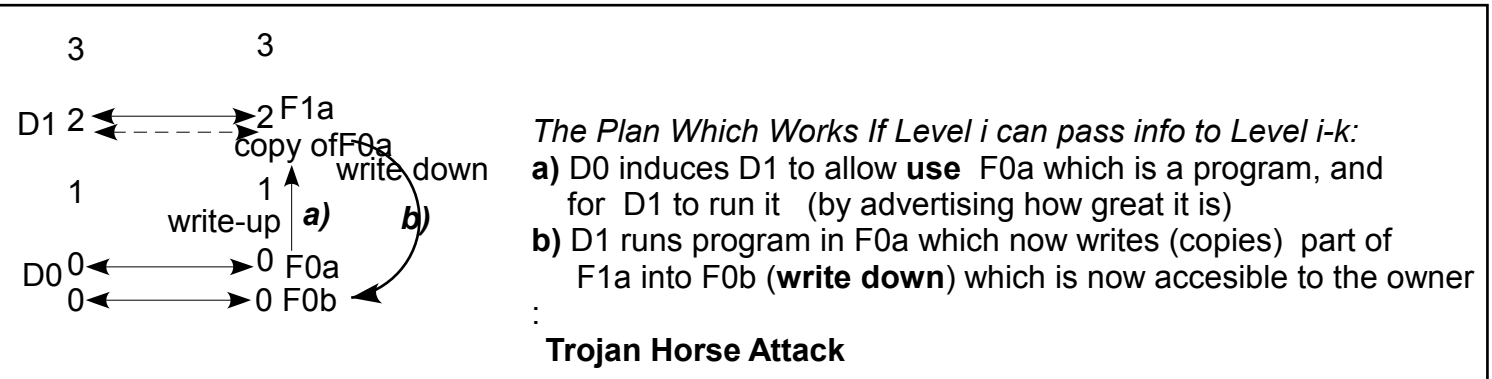
**Rule1:** *No Read From Up:* Level  $j$  cannot read anything at level  $j+k$ ,  $k > 0$

*No Write To Down:* Level  $j$  cannot write anything at level  $j-k$ ,  $k > 0$

It follows that:

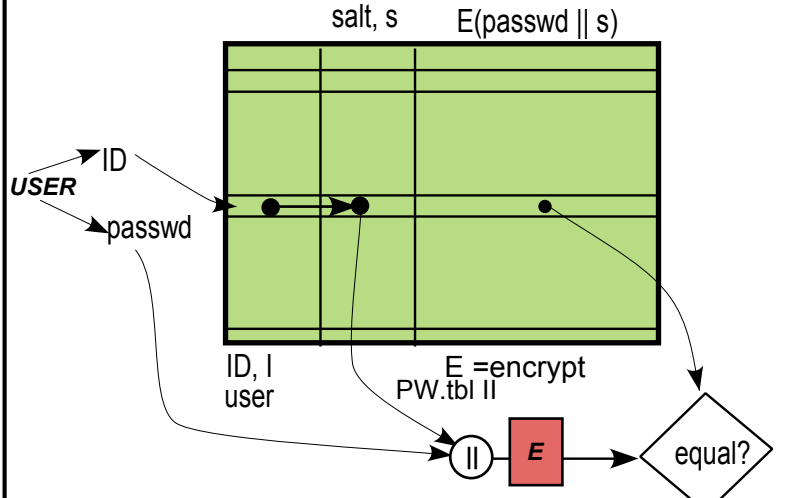
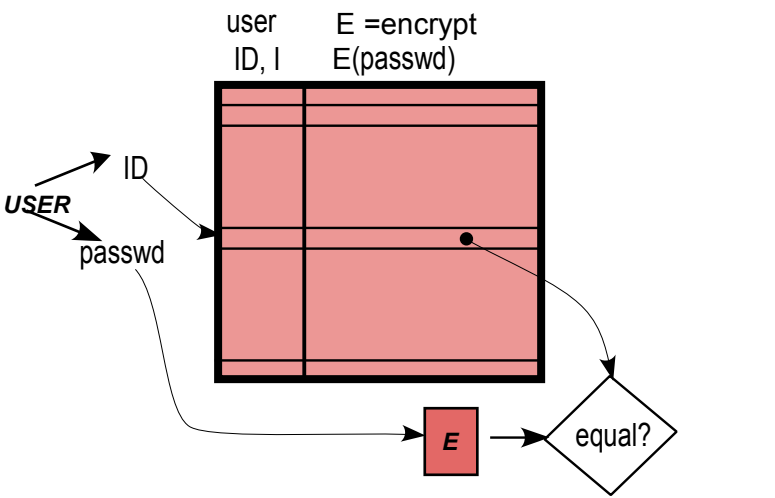
Result = **Transfer of Information-Up only:**

One can read from below write above transfer information only from Level  $j$  to level  $j+k$ ,  $k \geq 0$ .



This approach requires having access to the level of every owner and then for each write or read on a file knowledge of that files number will determine what can be done.

# UNIX PASSWORD ALGORITHMS



Initial UNIX-Password Table, PW.tbl1

UNIX-Password Check With Salt, PW.tbl2

Given ID, encryption of password and Encryption Algorithm, E all known. Password is unknown

Given ID, encryption of password, Encryption Algorithm, S= Salt all known, Password unknown

## Break In: Plan I

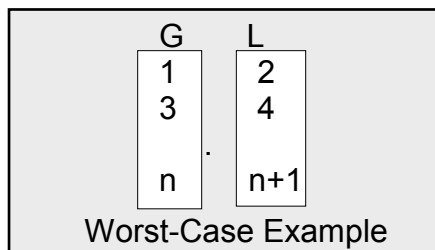
## Break In Plan Ia

Make a list of guessed passwords. Apply E to each. Sort the resultant list, array, G[N]. Sort encrypted entries in PW.tbl 1, to form list L[M], O(NlogN+MlogM). Now go through G and L comparing entries with the last one put in the output. Each comparison puts an entry in the O(N+M) size table, T).

Make a list of guessed passwords- Salt combinations Apply E to each. Sort the resultant list, array, G'[NM]. Sort encrypted entries in PW.tbl 2, to form list L'[M], O(NMlogN + MlogM). Now search G'[NM] and L for a match (an i and j such that G'[i] = L[j] ). In the worst case one would have to compare each entry in L with an entry in G, with complexity O(NM+M) .

$f(G,i,L,j) = f(G,i+1,L,j)$  if  $G[i] < L[j]$  entry in T  
 $f(G,i,L,j) = f(G,i,L,j+1)$  if  $G[i] > L[j]$  entry in T  
 $f(G,i,L,j) = G[i] (= L[j])$  if  $G[i] = L[j]$  entry in T

$f(G,i,L,j) = f(G,i+1,L,j)$  if  $G[i] < L[j]$   
 $f(G,i,L,j) = f(G,i,L,j+1)$  if  $G[i] > L[j]$   
 $f(G,i,L,j) = G[i] (= L[j])$  if  $G[i] = L[j]$



**This requires much more computation than Plan I and so discourages this attack.**

This is very efficient way to discover a Password, as long as the Password of any user will do

(Notice that even when a password, P, salt, S, combination is whose encoding matches an entry, E, in the password table is found the salt in the table,  $S_T$  may not = S. That is  $E(P, S)$  may not equal  $E(?, S_T)$ )

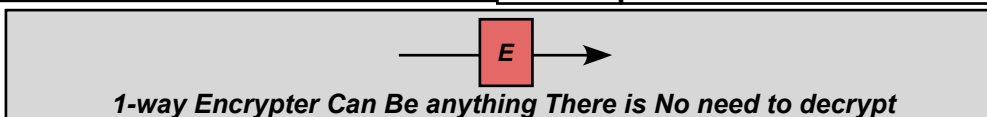
## 2 Break In:Plan II

## 2' Break In Plan IIa

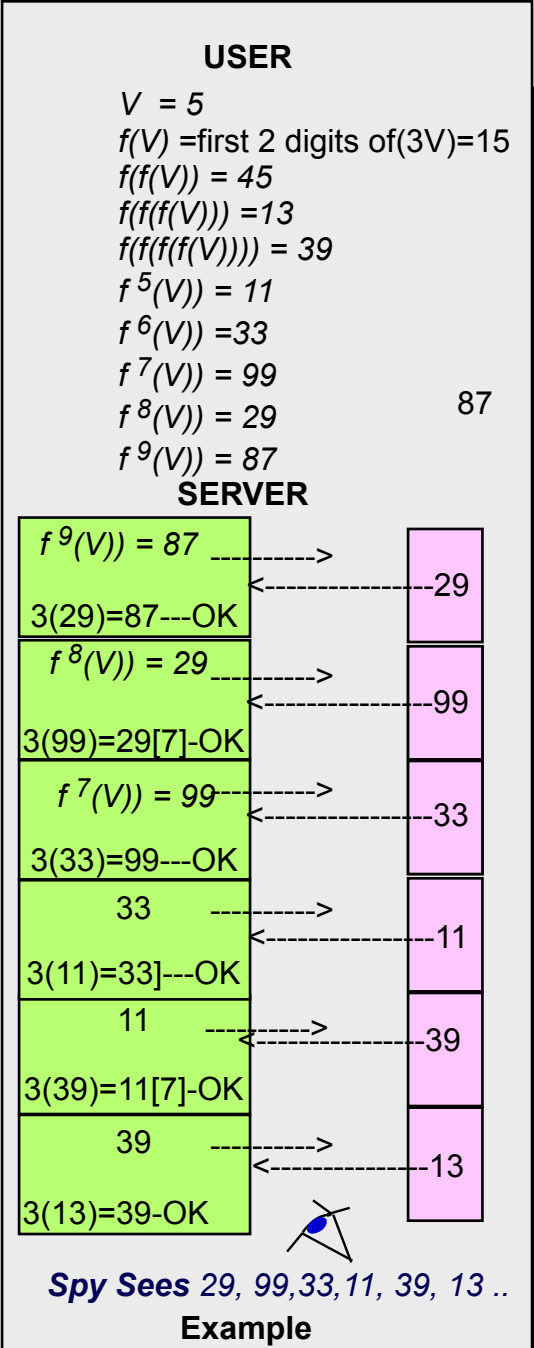
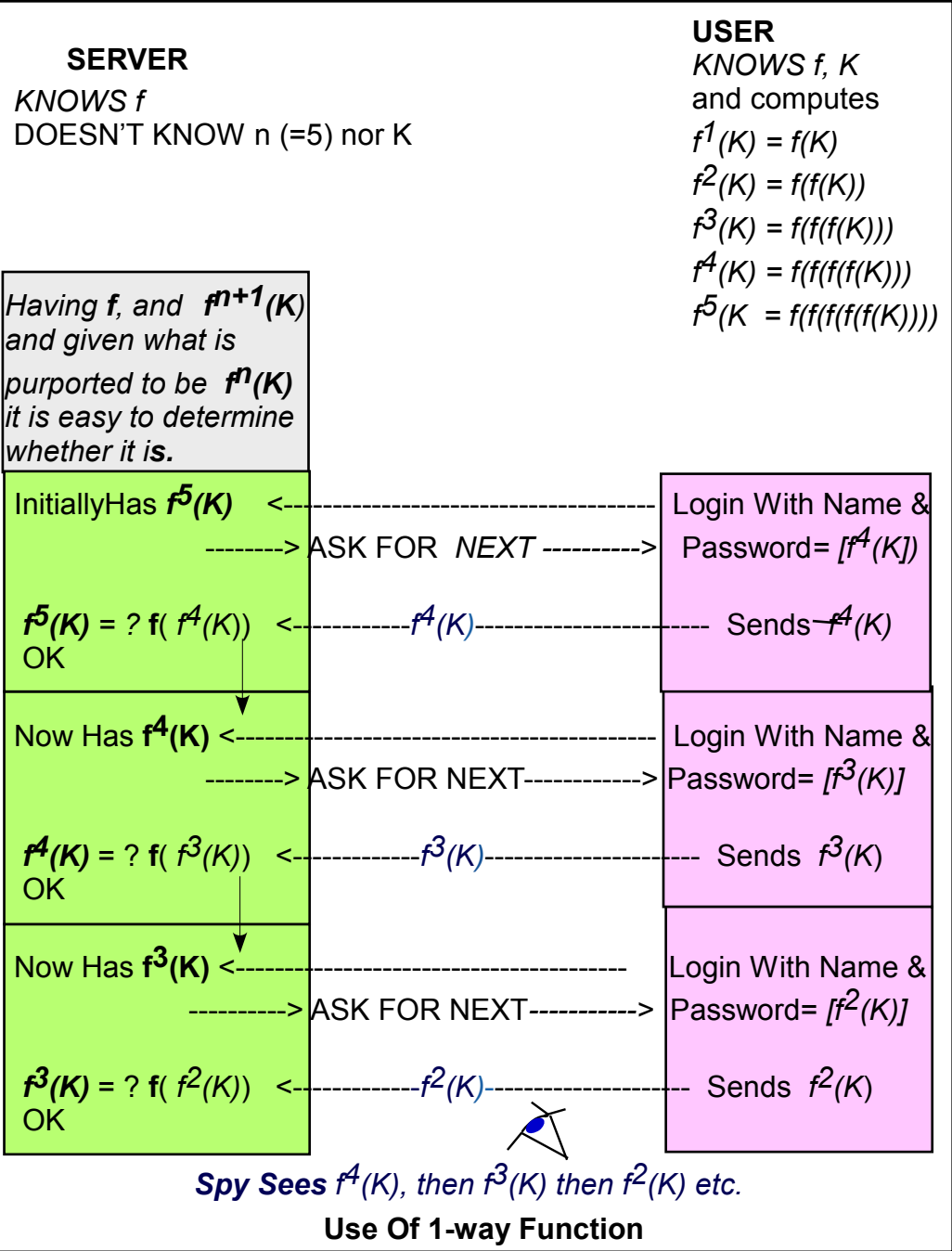
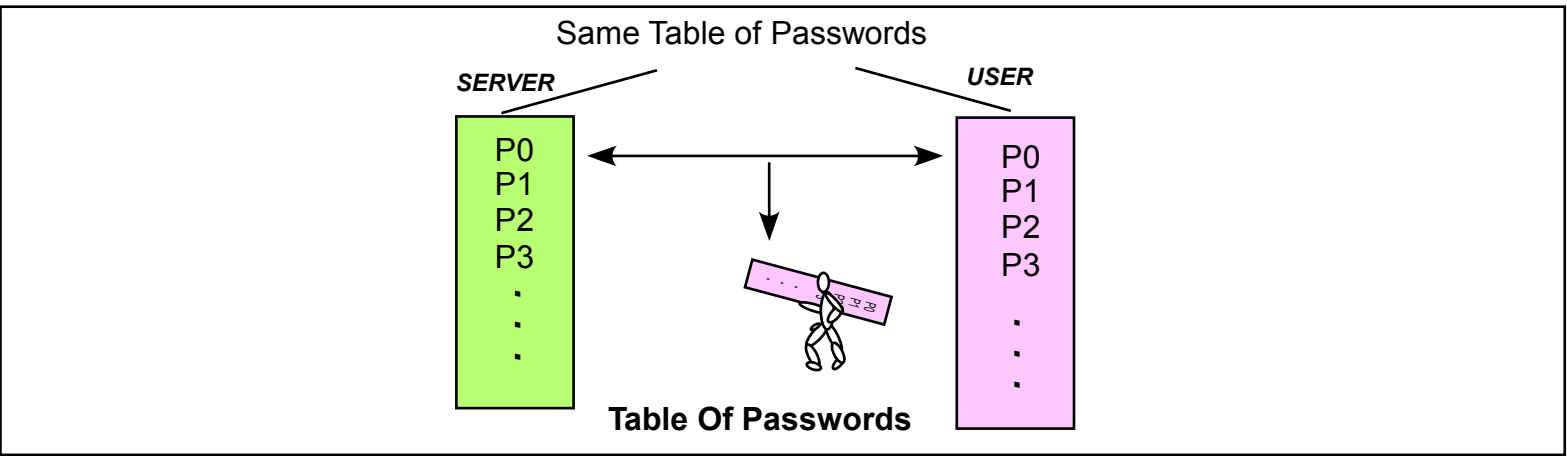
Make a list of guessed passwords for one User that might be used by this User, Encrypt each and sort-result L[P] Compare each member of L[P] with the encrypted password of the selected user. This is a much more directed search than above If, however, it were done for each of K users individually its complexity would be  $O(KN)$ , thus increasing rapidly with K

Make a list of guessed passwords for one User that might be used by this User, Encrypt each together with Users salt (given in table) and sort-result L[P] Compare each member of L[P] with the encrypted password of the selected user. This is a much more directed search than above If it were done for each of K users individually its cost would increase very much faster  $O(KN)$

**This requires the same order of computation as was required without the salt..**

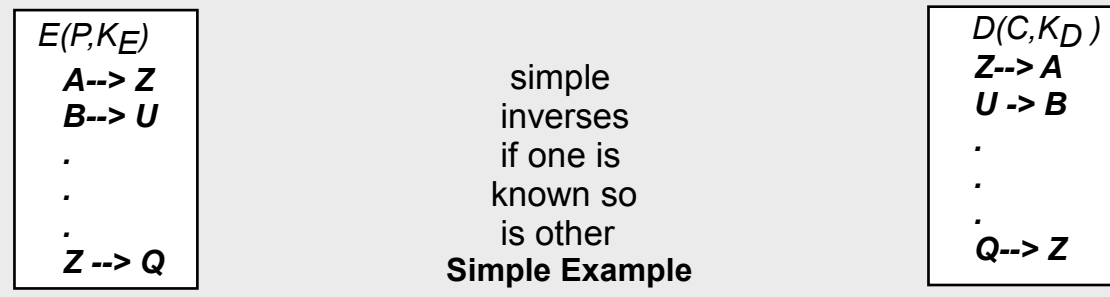
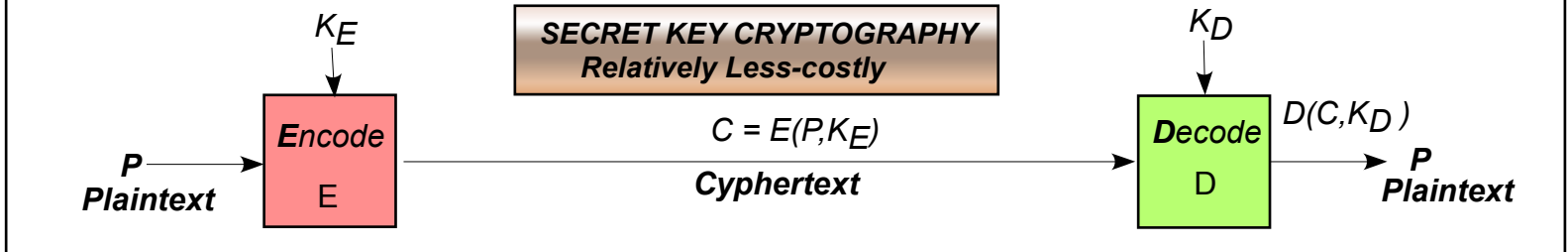


# UNIX UNAUTHORIZED ENTRY PROTECTION - Password



Protection When Repeated Password Use Is Visible

## CHANGING PASSWORDS-1 WAY FUNCTIONS



|   | Plaintext                         |                      |
|---|-----------------------------------|----------------------|
| $E(P, K_E)$   | C    A    B    A                  | $D(C, K_D)$          |
| 1 Convert text to binary number<br><i>letter--&gt;binary number</i>     | 0 1 1 / 0 0 1 / 0 1 0 / 0 0 1     | 5 binary to text     |
| 2 Break into groups, ( $G_j$ s-size 2)<br><i>group in twos</i>          | 0 1   1 0   0 1   0 1   0 0   0 1 | 4 Group              |
| 3 Permute within each $G_j$<br><i>interchange in alternate group</i>    | 1 0   1 0   1 0   0 1   0 0   0 1 | 3 Un-Permute Groups  |
| 4 Make substitutions to each $G_j$<br><i>subst: x -&gt; x + 1 mod 4</i> | 1 1   1 1   1 1   1 0   0 1   1 0 | 2 Undo substitutions |
| 5 Permute groups<br><i>rotate groups</i>                                | 1 0   1 1   1 1   1 1   1 0   0 1 | 1 Undo Group Permute |

$$E(\langle C \ A \ B \ A \rangle, K_E) = D(\langle 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1 \rangle, K_D)$$

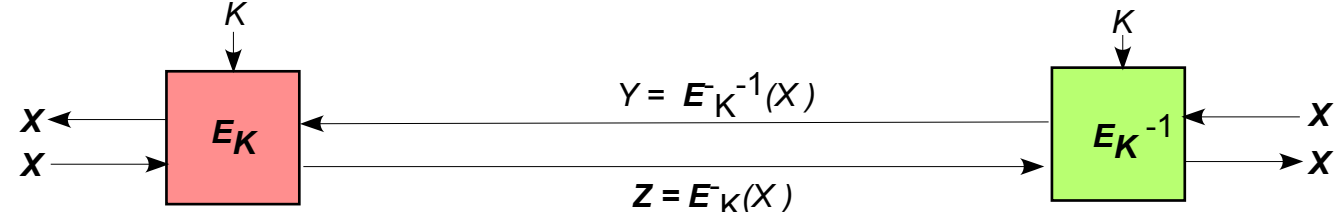
**More Complex Example**

Such codes are often breakable using 1 letter, 2 letter etc. frequencies ex. in order: e, t, o, a...; th, in, ..

Notice that when two entities, A and B, communicate secretly they use a key only known to them. Therefore the communication message is protected from all but these two and also the source of the message can be identified without loss of secrecy. If we count  $K_E$  and  $K_D$  as the same key then there is one key for every pair of agents who wish to keep their communications secret from all others except that pair. So there must be  $\binom{N}{2} = N(N-1)/2$  secret keys.

Typically for Secret Key Cryptography (as well as other forms of Cryptography). Decrypting is the inverse of Encrypting that is:  $E(P, K_E) = E_K(P)$  and  $D(C, K_D) = E_K^{-1}(C)$  and assuming symmetry\* it follows that

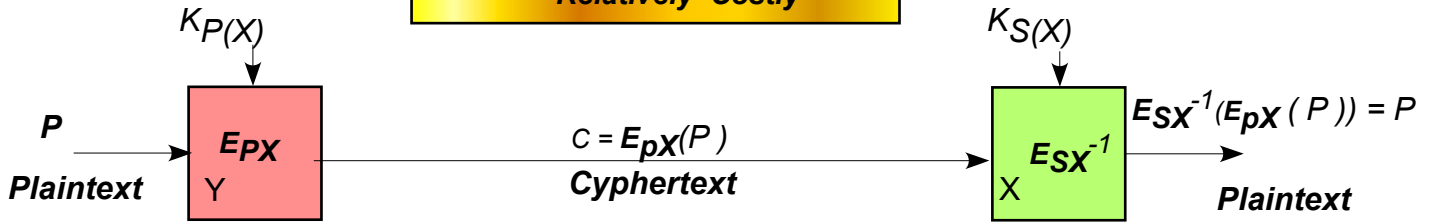
$$E_K^{-1}(E_K(X)) = E(E_K^{-1}(X))$$



\*Symmetry Ex.  $\sqrt{(a^2)} = a = (\sqrt{a})^2$

**CRYPTOGRAPHY 1 PRIVATE KEY**

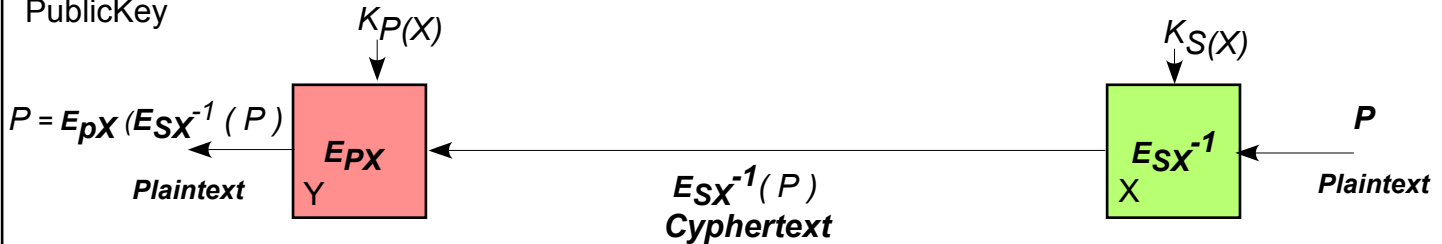
**PUBLIC KEY CRYPTOGRAPHY**  
Relatively Costly



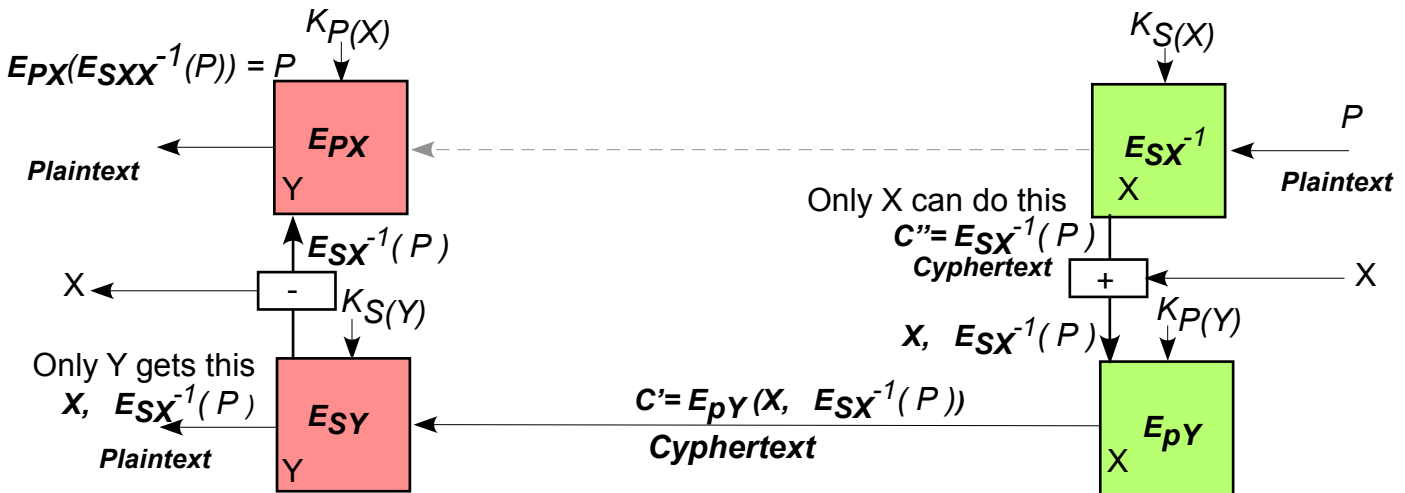
X has a public key  $K_{P(X)}$  which anyone can use to **Encode** (designated  $E_{pX}$ ) a message to X, and it has a secret key,  $K_{S(X)}$ , unavailable to any one else, which it uses to **Decode** (designated  $E_{sX}^{-1}$  to emphasis the inverse relation between **Encode** and **Decode**). Notice with this system, although only X can decode a message encoded with its public key it has no way of knowing who sent it since everyone has its public key. This problem can be remedied as shown below.

For N agents there must be N private and N public keys, total = 2N, to assure that any communication between two agents will be a secret from all other. **messages encoded with  $K_{P(X)}$**

Assuming that **symmetry**, i.e.  $E_{sX}^{-1}(E_{pX}(P)) = P = E_{pX}(E_{sX}^{-1}(P)) = P$  exists. One can use it to encode plaintext, P, at X and send it,  $E_{sX}^{-1}(P)$ , to Y. Now if Y can communicate with X it must have X's PublicKey



Inverse Path and Symmetry



Sending Information With Assurance of Origin (Signature based on Symmetry)

C. Crowley

One way to get Public Key Cryptography is for each party,  $p_j$ , to build a machine,  $E_j$ , which encodes Plaintext so that  $p_j$  alone can decode it with its own machine  $D_j$ . (The encoding can be one involving Secret Key cryptography-permutations, interchange, etc). Then any one who wants to send to  $p_j$  is given a machine  $E_j$ . There are also algorithms for doing so. The RSA algorithm for doing so is sketched next.

**CRYPTOGRAPHY 2 PUBLIC KEY**

# RivestShamirAdleman Public Key Encryption (RSA on Google)

|   |   |
|---|---|
| Choose two prime numbers $P$ and $Q$ , Let $PQ$ be $N$  | $P = 2$<br>$Q = 5$<br>$N = PQ = 10$   |
| Choose $E$ , such that $1 < E < N$ , and $E$ and $Z = (P-1)(Q-1)$ are relatively prime.   | $Z = (P-1)(Q-1) = 1 \times 4 = 4$<br>$1 < E < 10$ & $E$ , and $Z$ are relatively prime  |
| Choose $D$ such that $DE-1$ is divisible without remainder by $Z = (P-1)(Q-1)$ , i.e<br>$(DE-1) = X(P-1)(Q-1)$<br>$DE = X(P-1)(Q-1) + 1$<br>So Choose $D$ such that $[X(P-1)(Q-1) + 1] / E = \text{an integer}$       | $E = 3$<br>$D$ so that $DE(=3)-1$ divisible by $Z=4$<br>$D = 7$<br>$DE-1 = 20$  |
| Public Key $(N, E)$<br>Secret Key $(N, D)$<br>Encryption $T$ is the plaintext (an integer). It must be $< N$ ,<br>$C$ is the resultant ciphertex)<br>$C = T^E \text{ mod } N$<br>Decryption: $T = C^D \text{ mod } N$ | <b>Public Key <math>(N=10, E=3)</math></b><br><br><b>Secret Key <math>(N=10, D=7)</math></b><br><br>$[x^{n1} \text{ mod } 10]^{n2} \text{ mod } 10 = x$ |

$N$  is known Breaking Requires finding factors  $P$  and  $Q$

$x^n \text{ mod } 10$

| $x \backslash n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|------------------|---|---|---|---|---|---|---|---|---|----|
| 1                | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  |
| 2                | 2 | 4 | 8 | 6 | 4 | 2 | 8 | 6 | 2 | 4  |
| 3                | 3 | 9 | 7 | 1 | 3 | 9 | 7 | 1 | 3 | 9  |
| 4                | 4 | 6 | 4 | 6 | 4 | 6 | 4 | 6 | 4 | 6  |
| 5                | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5  |
| 6                | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6  |
| 7                | 7 | 9 | 3 | 1 | 7 | 9 | 3 | 1 | 7 | 9  |
| 8                | 8 | 4 | 2 | 6 | 8 | 4 | 2 | 6 | 8 | 4  |
| 9                | 9 | 1 | 9 | 1 | 9 | 1 | 9 | 1 | 9 | 1  |
| 10               | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  |

*x = message are 1 through 9*

$7^3 \text{ mod } 10$

public key: message = plaintext  $^3$  = cyphertex

private key: cyphertex  $^7$  = message

sender message: 3 encodes  $3^3 \text{ mod } 10 = 27 \text{ mod } 10 = 7$

receiver decodes  $7^7 \text{ mod } 10 = 823543 \text{ mod } 10 = 3$

$8^3 \text{ mod } 10$

sender message: 8 encodes  $8^3 \text{ mod } 10 = 512 \text{ mod } 10 = 2$

receiver decodes  $2^7 \text{ mod } 10 = 128 \text{ mod } 10 = 8$

Example Public Key Cryptography

## PUBLIC KEY CRYPTOGRAPHY RSA ALGORITHM AND EXAMPLE

# SECRET KEY DISTRIBUTION From A to B

## I. Distribute Secret Key With Public Key Encryption

### II Choices

- 1 Physical Delivery
  - 1) From A to B.
  - 2) Third Party Administrator, C to A and B

### 2 Distribute Key $j$ encrypted with Key $j-1$

### 3 A (Host) wants to connect to (Host) B

A Request Session Key From Key Distribution Center (**KDC**)

By sending request To local Front End Processor, **FEP**, which passes it on to **KDC**.

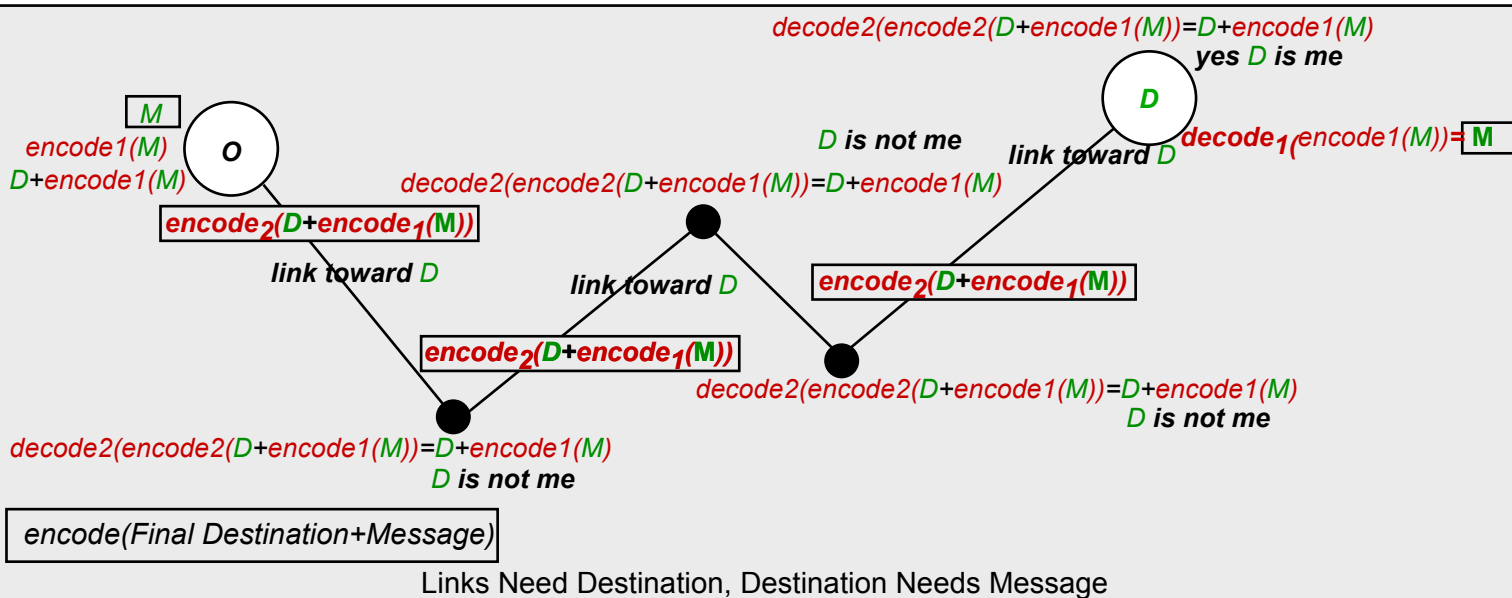
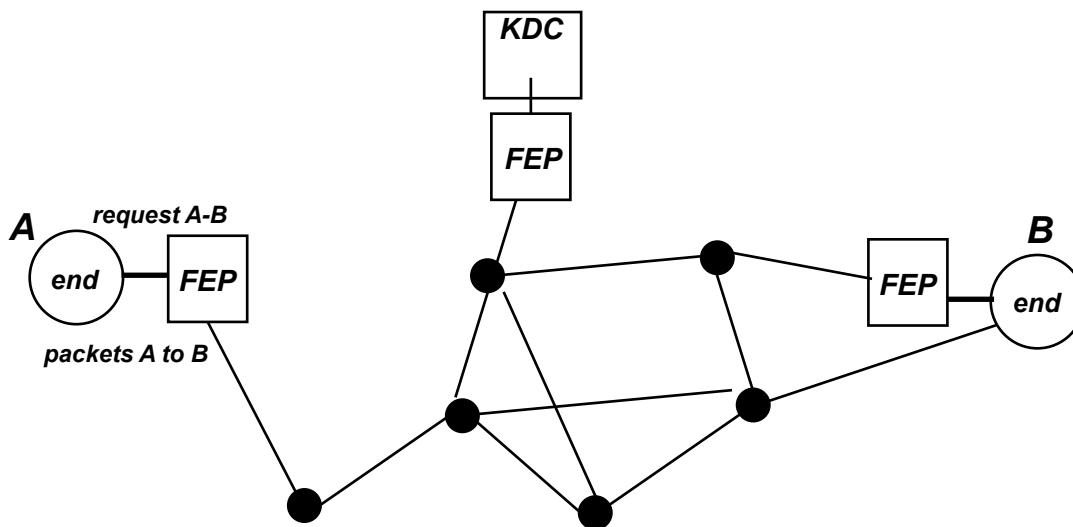
**KDC** determines if legitimate and if so it generates a One Time Session Key. and passes it both **A** and **B's FEP**.

Messages between **FEP<sub>j</sub>** and **KDC** are encrypted by a Master Key known only to them.

**Good But Costly**

**Link OK But  
End-to-End Awkward**

**Once Code Is Broken  
It is Broken Forever**



## SECRET KEY DISTRIBUTION

## MESSAGES AND KEY DISTRIBUTION