OS Security

COMP 3361: Operating Systems I
Winter 2015
http://www.cs.du.edu/3361

The Security Problem

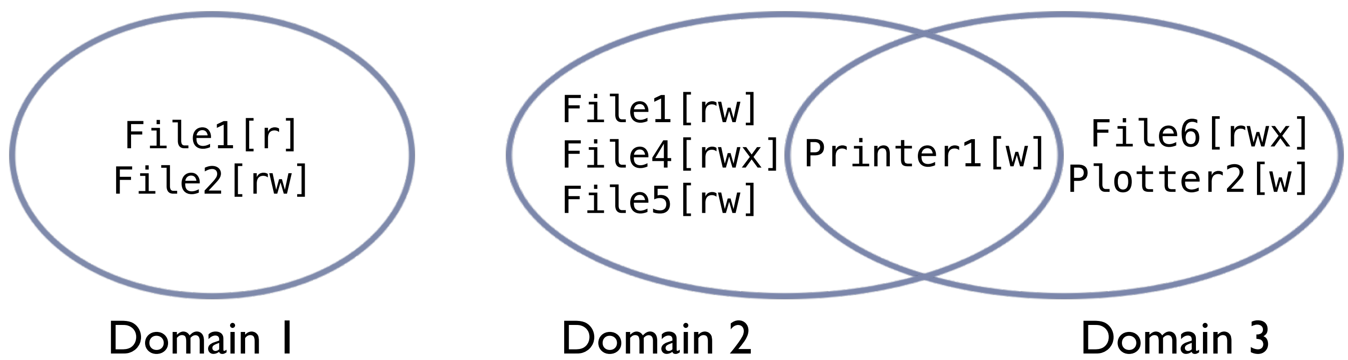
- Computers consist of a collection of objects, hardware or software
- Each object has a **unique name** and can be accessed through a well-defined **set of operations**
- **Security problem:** ensure that each object is accessed correctly and only by those processes that are allowed to do so
  - system is secure if resources used and accessed as intended under all circumstances

Terms

- An **intruder** (cracker) attempts to breach security
- A **threat** is a potential security violation, arising from the existence of some **vulnerability** in the system
- An **attack** is an attempt to breach security, typically by **exploit**ing a vulnerability
- For a system to be secure, preserve
  - **confidentiality:** release of data to unauthorized users never occurs
  - **integrity:** unauthorized users should not be able to modify any data
  - **availability:** the system should be serving the purpose it was designed for

Protection Domains

- Access-right = <*object-name*, *rights-set*>
  - rights-set is a subset of all valid operations that can be performed on the object
- A **domain** is a **set of access-rights**
  - rights assigned based of principle of least privilege



- Every process runs in some protection domain

4 Domain Implementation (UNIX)

- Domain = *user-id*
- Domain switch accomplished via file system
  - each file has associated with it a domain bit (setuid bit)
  - when file is executed and setuid = on, then user-id is set to owner of the file being executed
  - when execution completes user-id is reset
- Domain switch accomplished via passwords
  - **su** command temporarily switches to another user's domain when other domain's password is provided
- Domain switching via commands
  - **sudo** command prefix executes specified command in another domain (if original domain has privilege or password given)

Protection Matrix

- Rows represent domains
- Columns represent objects

	File1	File2	File3	File4	File5	File6	Printer1	Plotter2
D_1	R	RW						
D_2			R	RWX	RW		W	
D_3						RWX	W	W

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in cell (i,j)

6 Domain Switching in Matrix

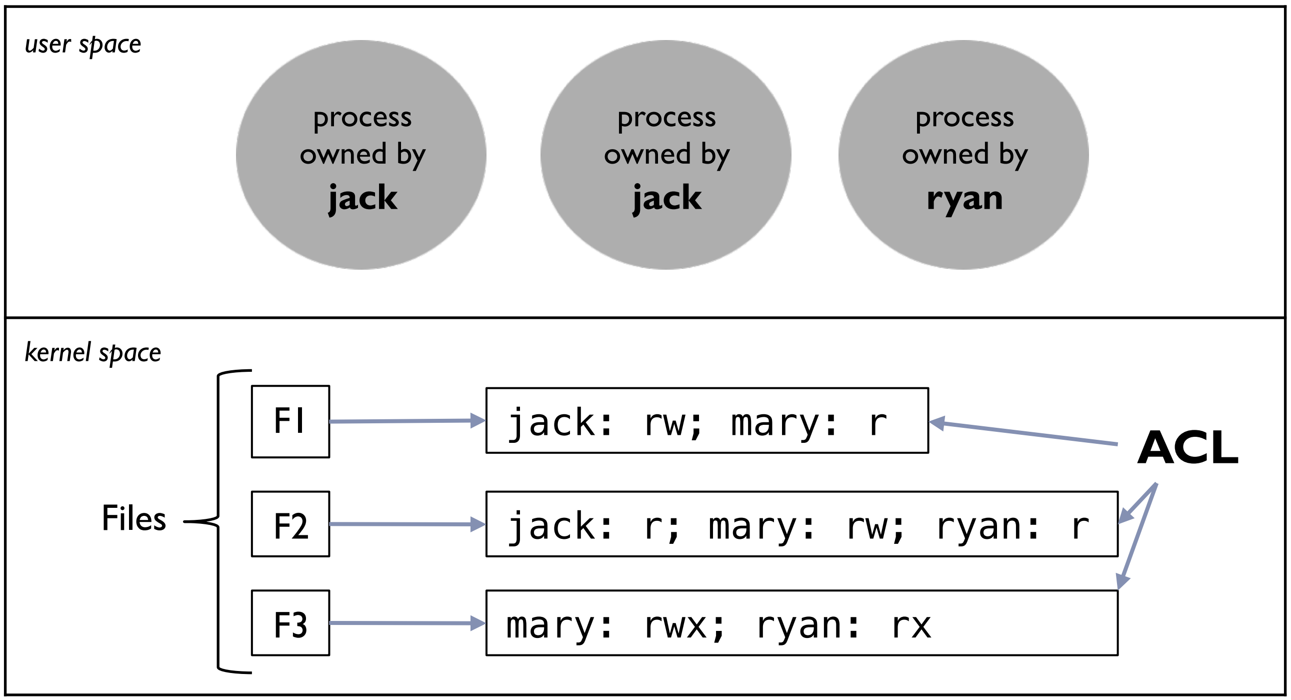
- Domains are also objects, with the *enter* operation

	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	D_1	D_2	D_3
D_1	R	RW								Enter	
D_2			R	RWX	RW		W		Enter		
D_3						RWX	W	W			

- D_1 can switch to D_2, but not D_3.

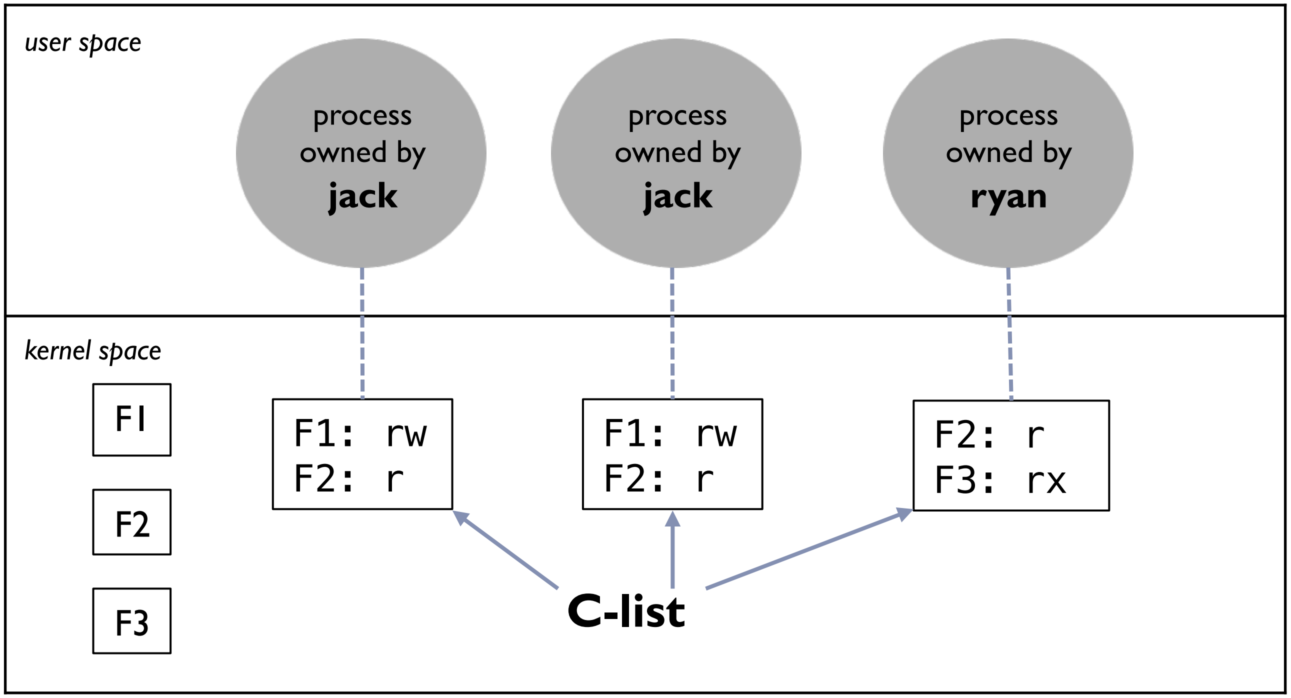
Access Control List Implementation

- **Access control list (ACL)** for objects
  - each column implemented as an access list for one object
  - resulting <u>per-object</u> list consists of ordered pairs <*domain, rights-set*> defining all domains with non-empty set of access rights for the object



Capability List Implementation

- **Capability list (C-list)** for domains
  - each process has a capability list
  - capability list for a process is list of objects together with operations allowed on them



9 Comparison of Implementations

- Rights check for an operation
  - ACL: check through long list of (domain,rights)
  - C-list: only check capability list of process
- Revocation: remove a right for an object
  - ACL: search access list of object and remove entries corresponding to the right
    - can be selective, e.g. "remove write access in file x for domain y"
  - C-List: search all C-lists for object with the particular right, and then remove
    - selective removal is difficult

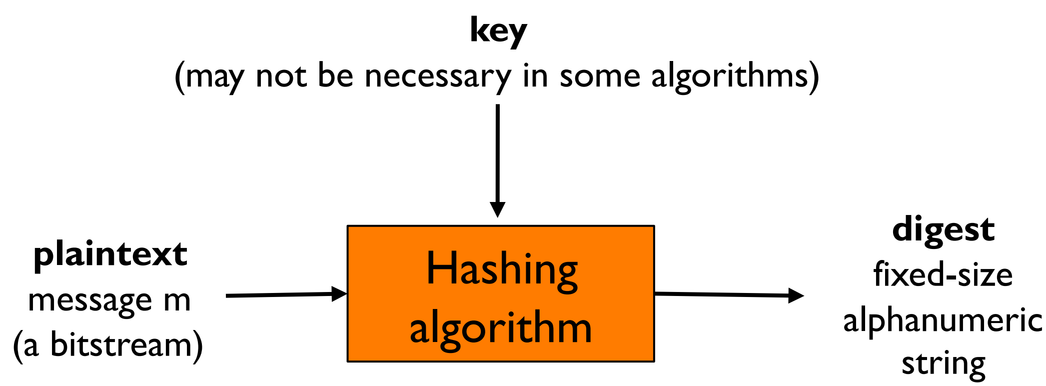
10

Cryptography as a Security Tool

- Means to constrain potential senders (sources) and/or receivers (destinations) of messages
  - based on secrets (keys)
  - enables
    - receipt only by certain destination
    - confirmation of source

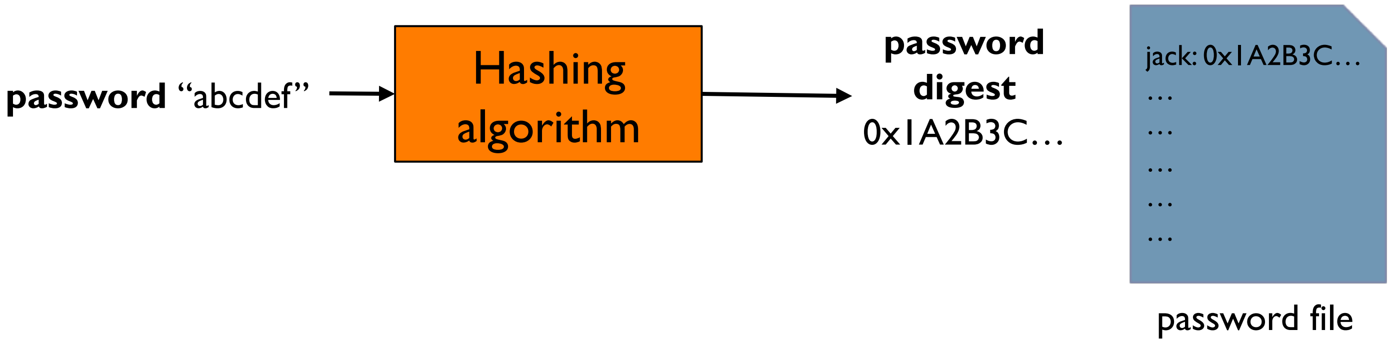
11 Cryptographic Hash

for message fingerprinting

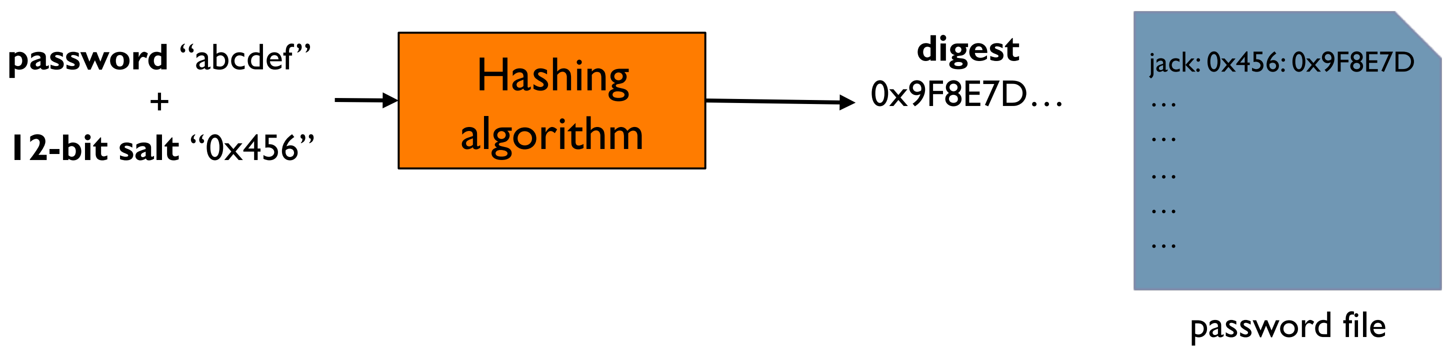


Obtaining the message from the digest is not possible even after knowing the algorithm and the key

Usage Example: Password Storage



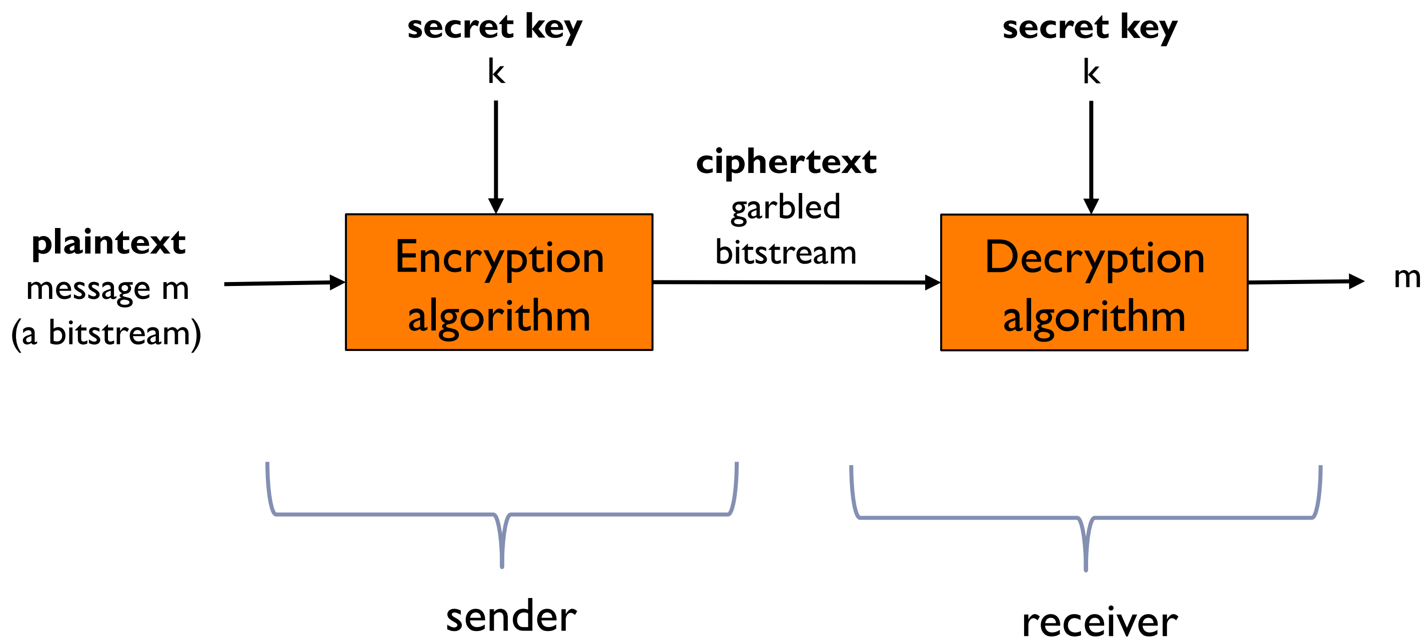
(attacker can pre-compute digest for possible passwords and then compare)



(2^{12} more pre-computations necessary)

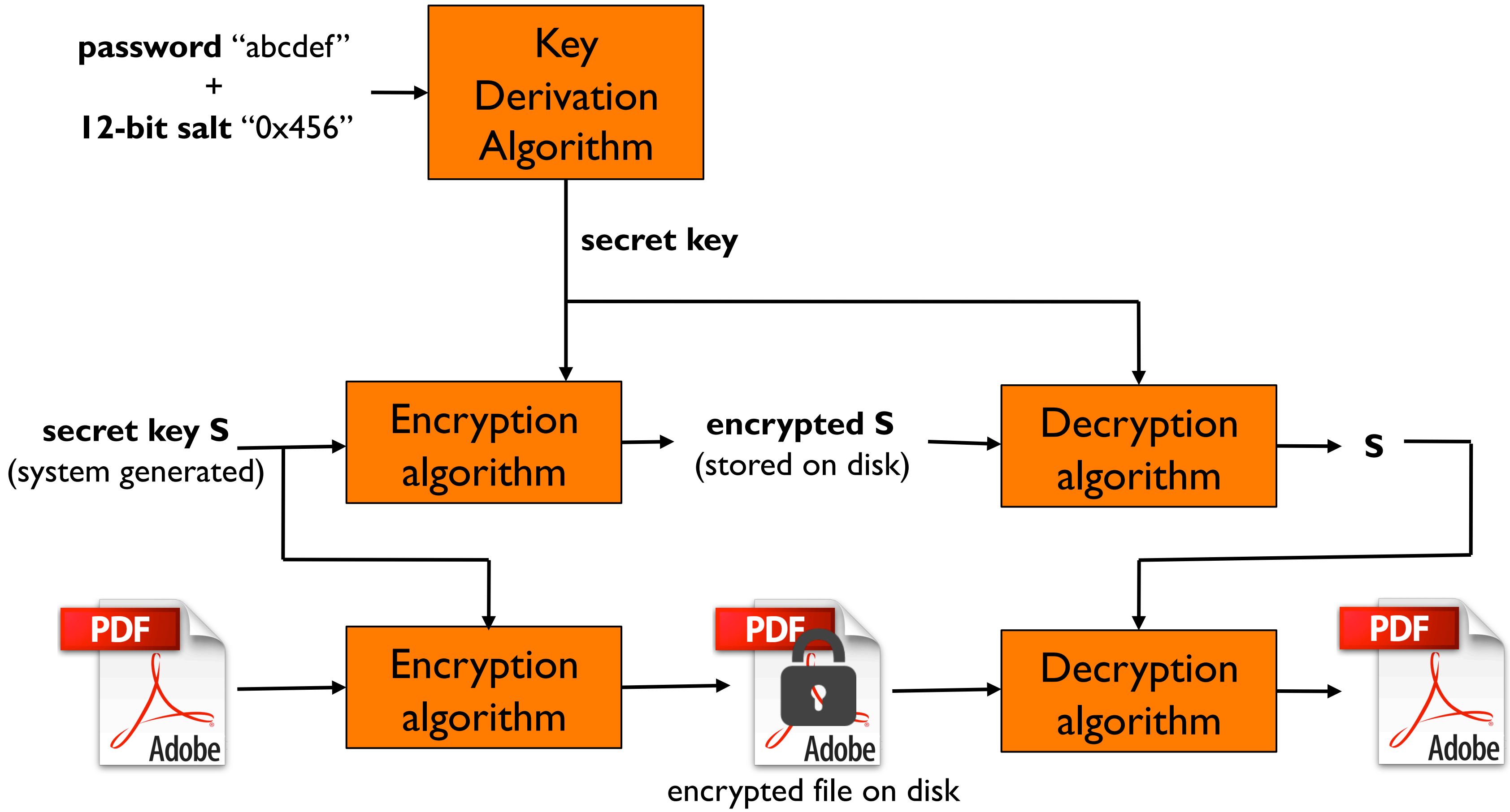
Secret-Key (Symmetric) Encryption

constrain who sends and who receives



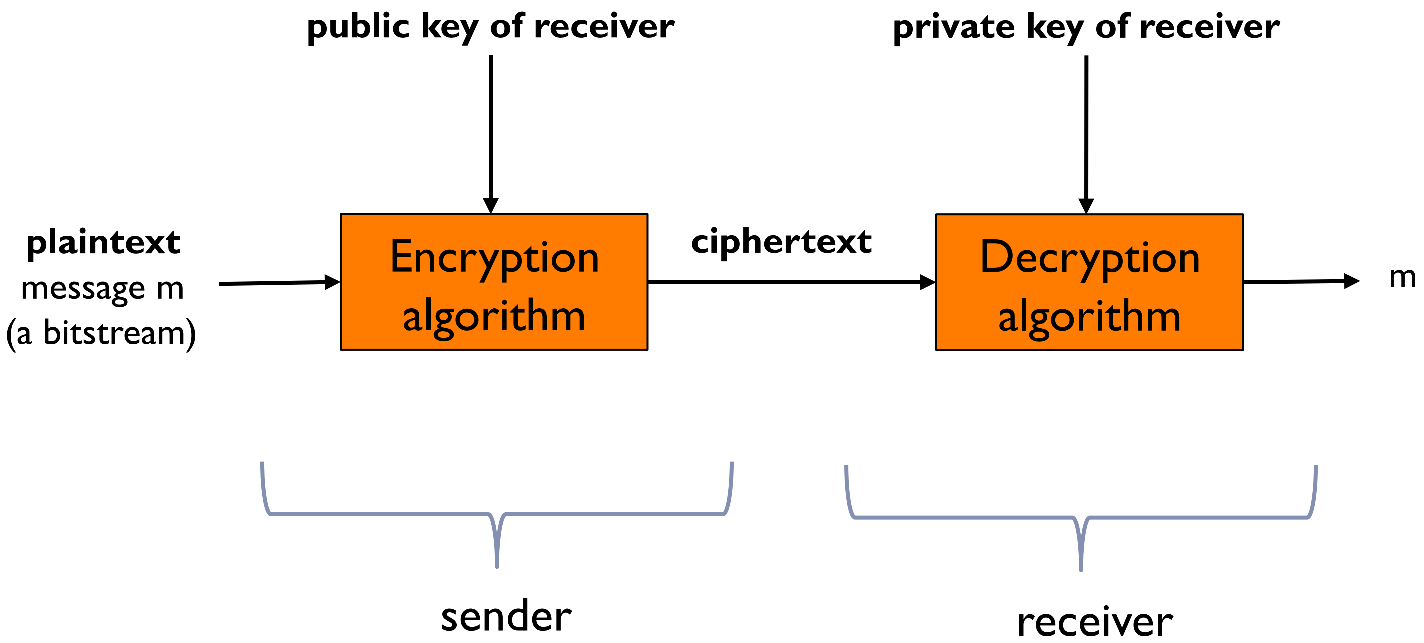
attacker knows the algorithms and can see ciphertext;
obtaining the secret key from the ciphertext is computationally infeasible

Usage Example: Encrypting Files



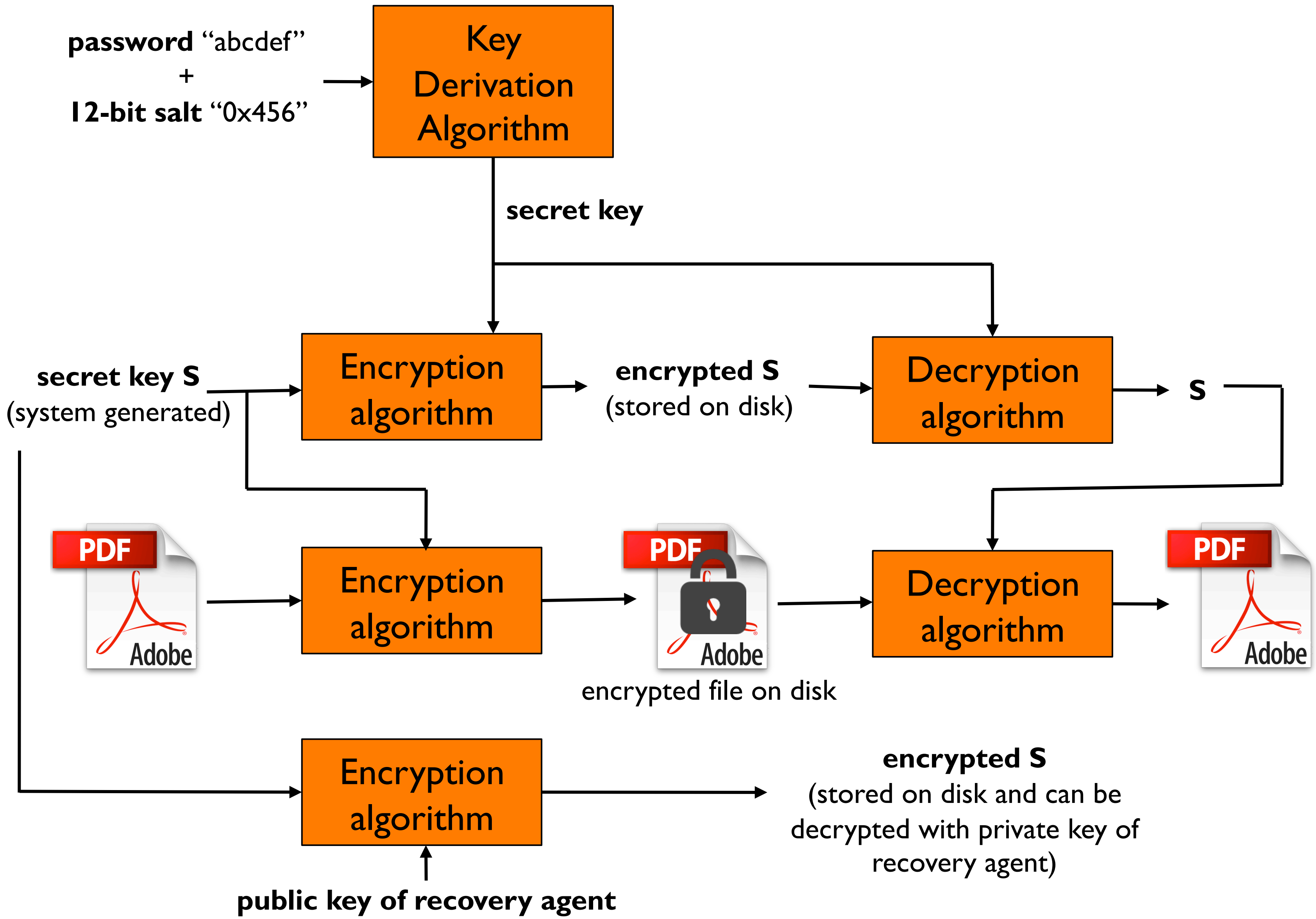
Asymmetric Encryption

constrain who receives



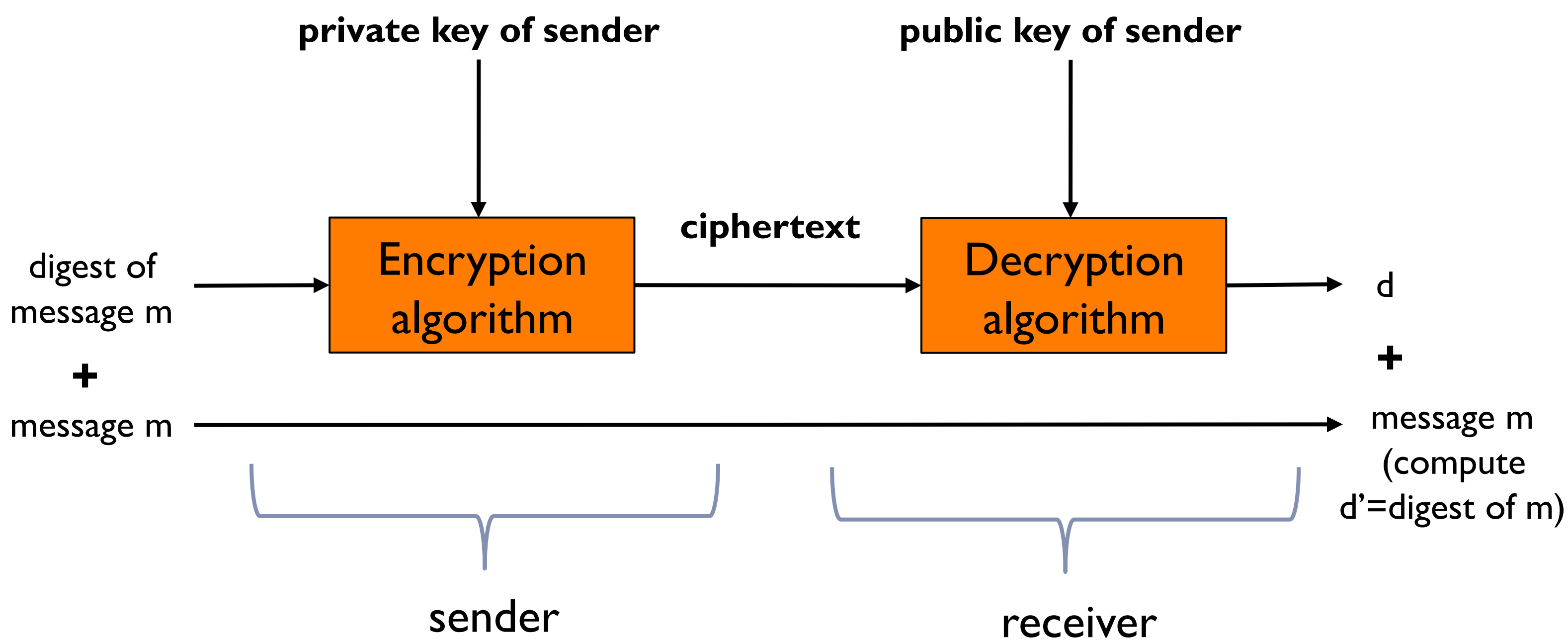
attacker knows the algorithms, public key, and can see ciphertext; obtaining the private key from the ciphertext and the public key is computationally infeasible

16 Usage Example: Recoverable Enc. Files



Digital Signature

establish authenticity of sender



message m is not tampered with during transit if d = d';
if d = d', then sender is also authenticated

Key Distribution

- Delivery of symmetric keys is a huge challenge
  - sometimes done out-of-band
- Asymmetric keys can proliferate
  - public keys are no secret
  - even asymmetric key distribution needs care
    - man-in-the-middle attack

Digital Certificates

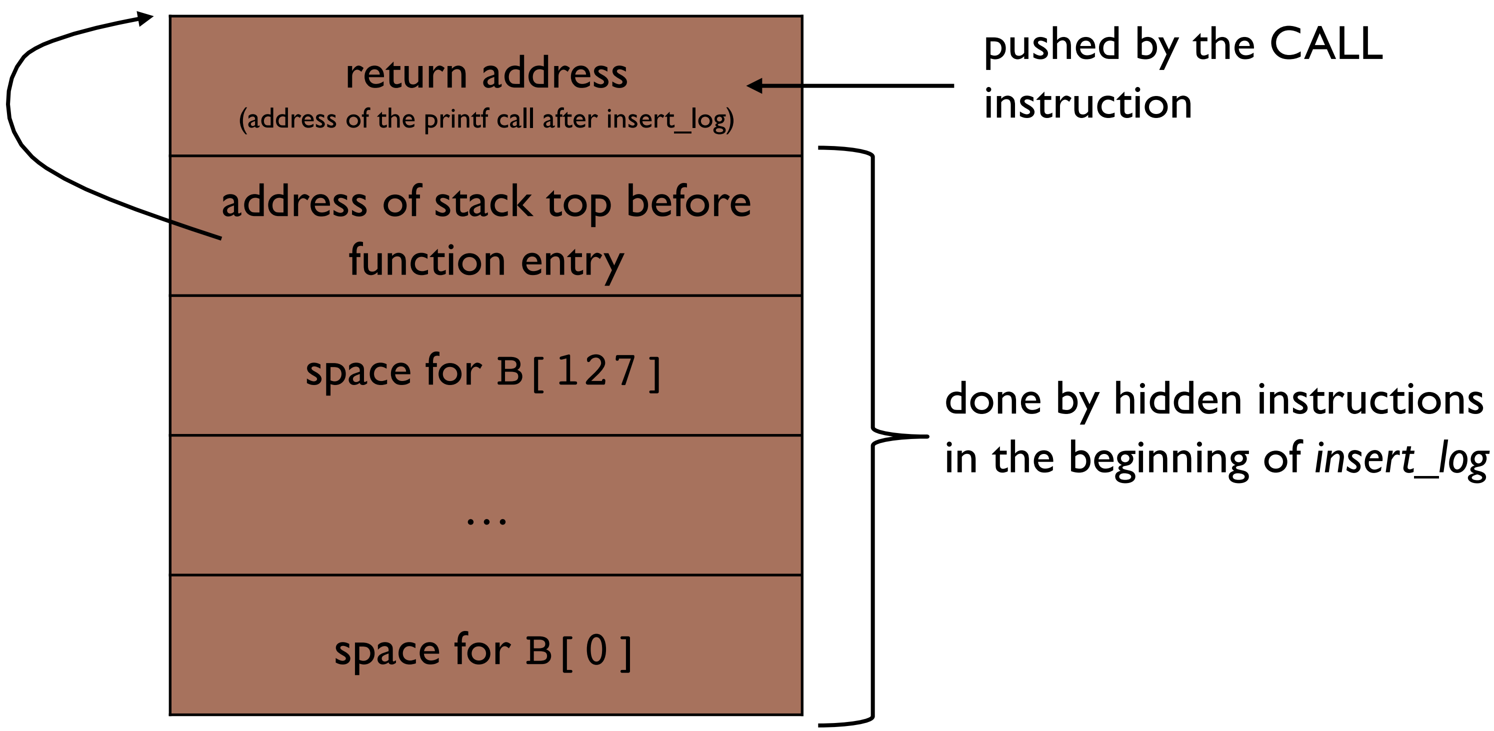
- Proof of who or what owns a public key
- Trusted party receives proof of identification from user and certifies that public key belongs to the user
  - public key digitally signed by trusted party
    - user's public key encrypted (signed) with trusted party's private key
    - also known as a **digital certificate**
  - how to know signature is legitimate?
- Certificate authorities are trusted parties – their public keys are included with web browser distributions
  - they vouch for other authorities via digitally signing their keys, and so on

Buffer-Overflow Condition

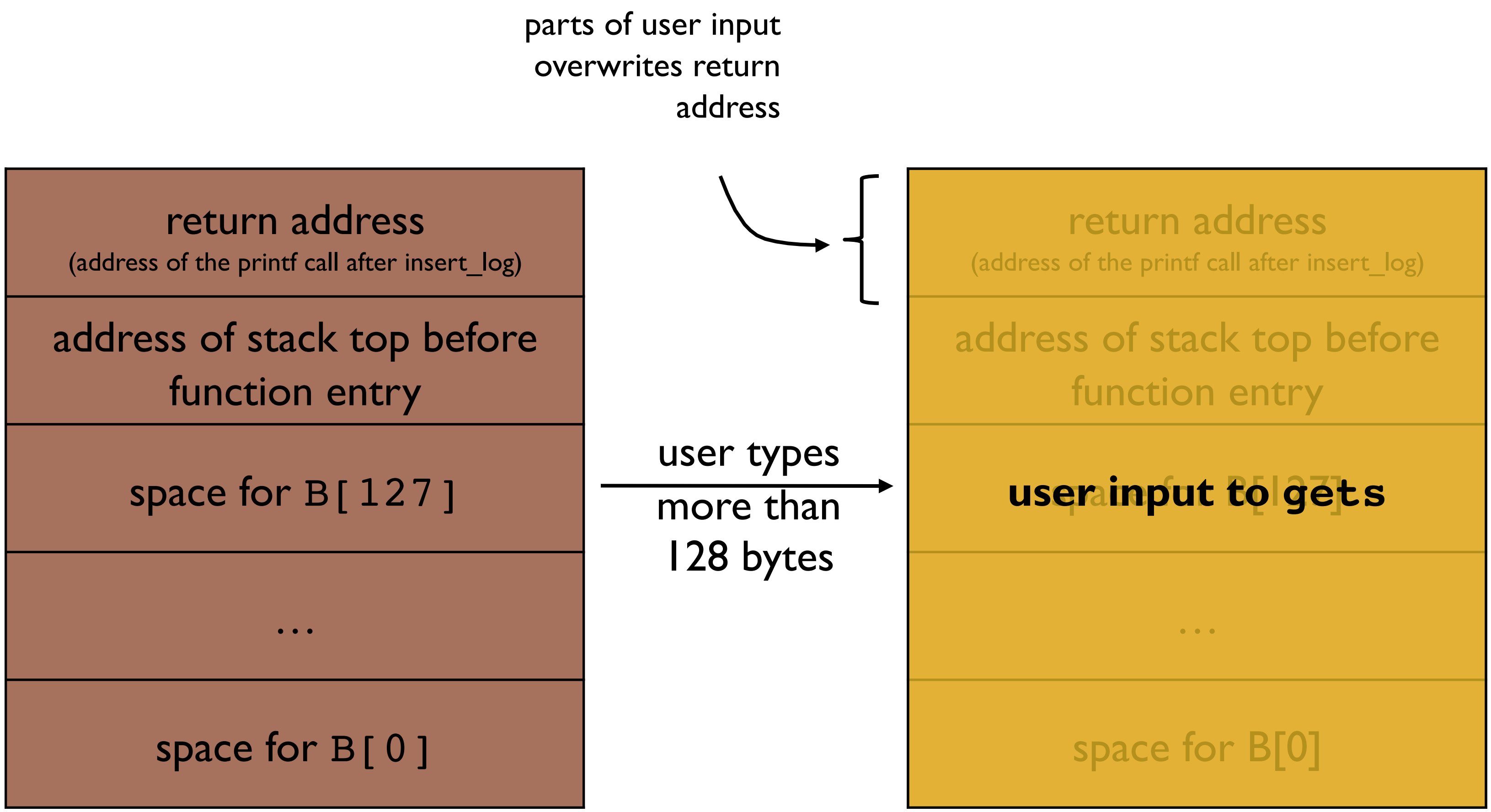
```
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf(“Begin logging…\n”);
  insert_log();
  printf(“End logging…\n”);
}

void insert_log() {
  char B[128];
  printf(“Enter log message:”);
  gets(B);
  writeLog(B);
}
```

Layout of Stack

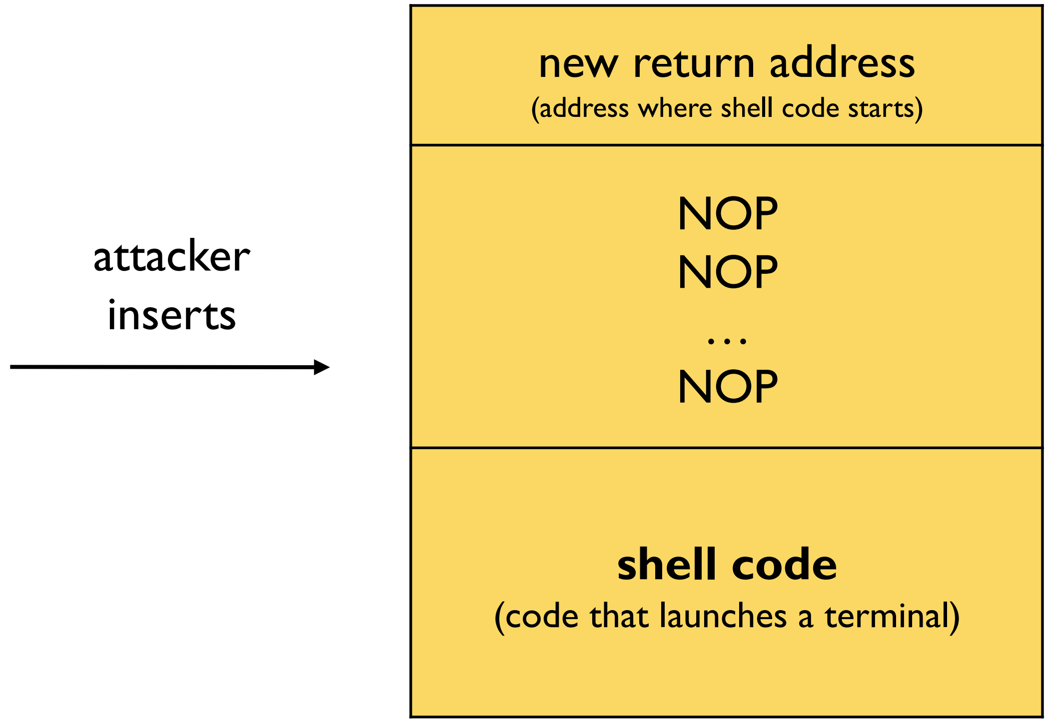
stack layout when insert_log begins



Buffer-Overflow



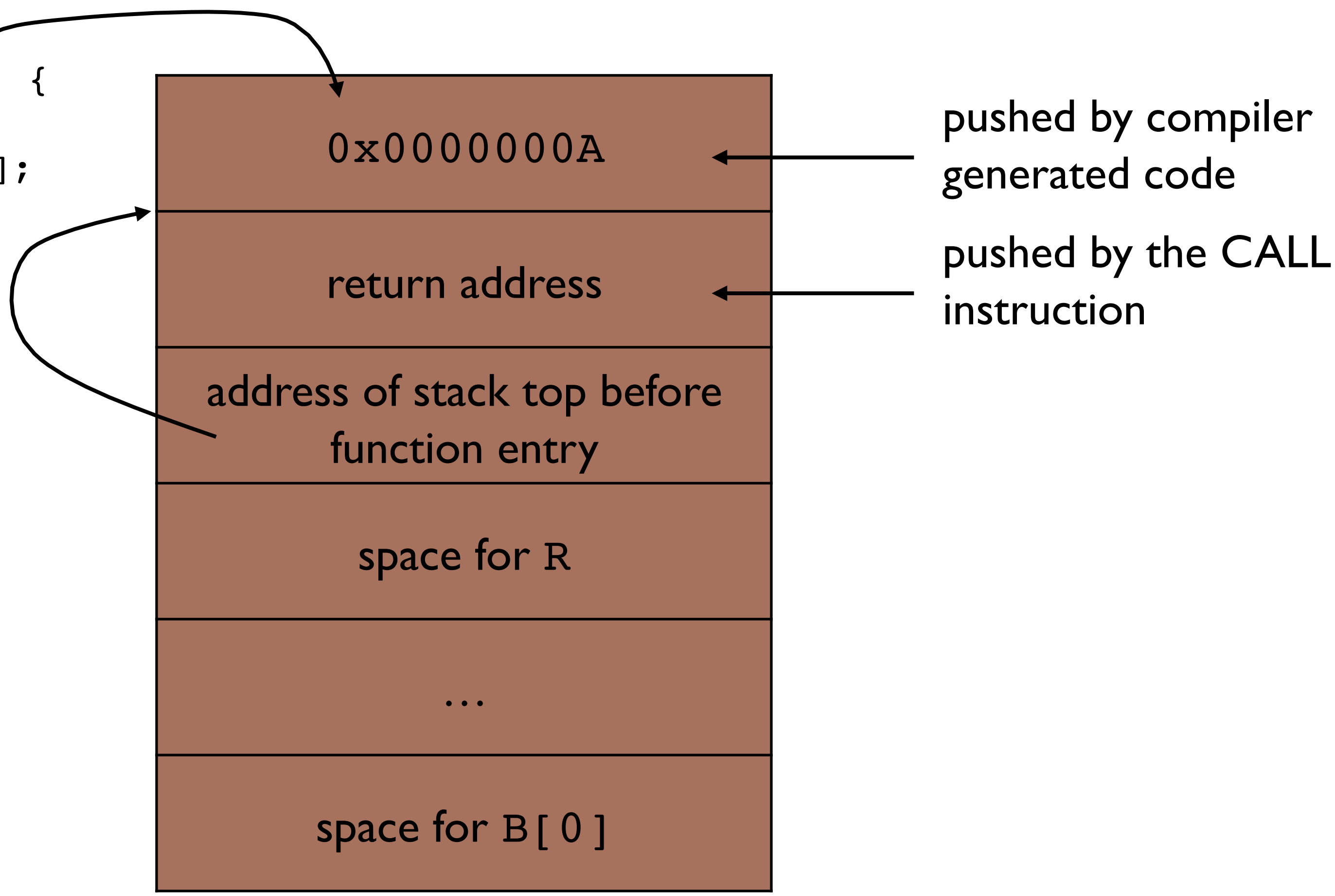
Stack Layout After gets()



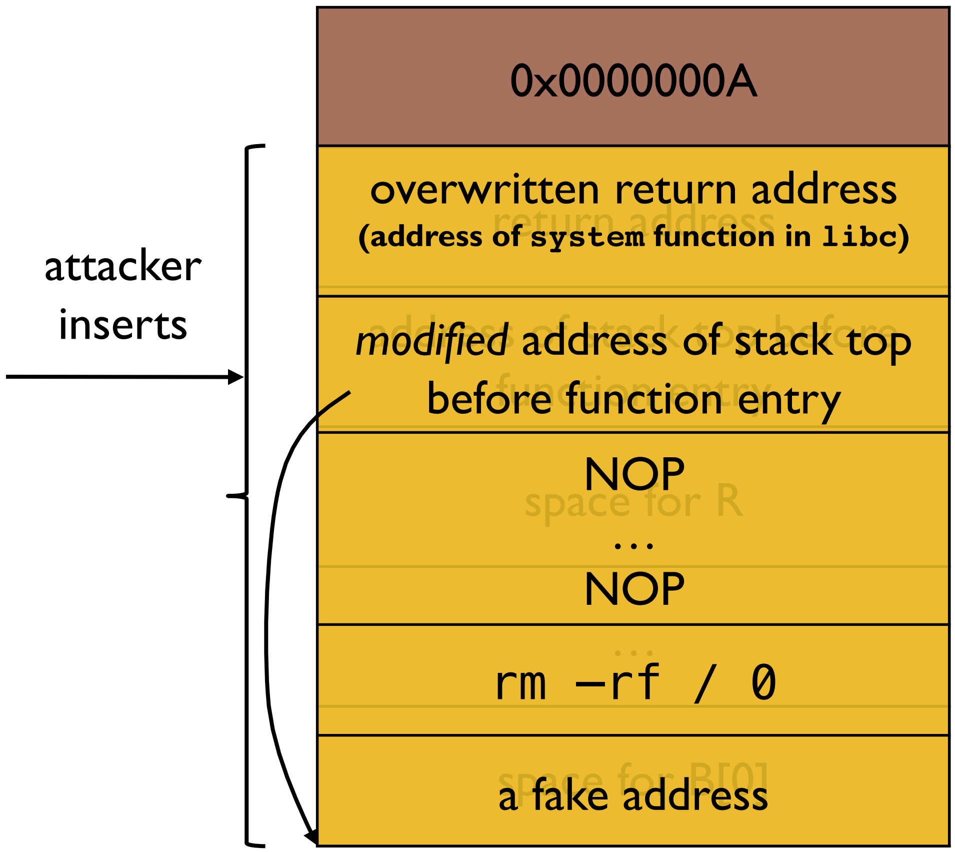
will not work if stack is set up to be data-only (non-executable)

Stack Layout with Function Arguments

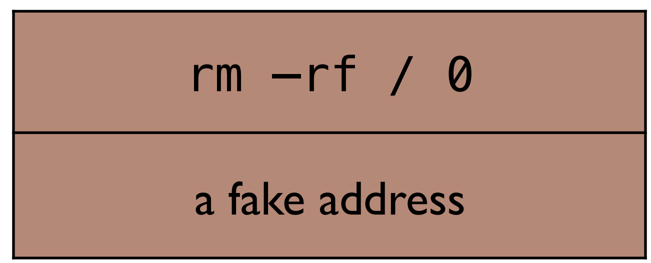
```
long foo(int N) {
     long R;
     char B[128];
     …
     gets(B);
     …
}


foo(0xA);
```



Code Reuse Attack



after returning from **foo**, stack is restored (but, stack pointer points to modified stack top) and we end up in the beginning of the **system** function in libc



stack as seen by the **system** function (exactly as it would look like if someone CALLed the function)

`system("rm –rf /")` means delete ALL files!

References

- Chapter 9.1, 9.2, 9.3, 9.5, 9.7.1, Modern Operating Systems, A. Tanenbaum and H. Bos, 4th Edition.