

# Information Security Education : Building Secure Applications

*Summer Workshop on Distributed Computing,  
Networking and Security with Applications*

Joochan Lee  
jlee@cs.ucf.edu

School of Computer Science  
University of Central Florida



# Course Outline

## ■ Part I

- Fundamentals of contemporary cryptography and its application to building secure distributed applications

## ■ Part II

- Developing and understanding of practical software attack methodologies and exploitation techniques. This will be used to lay the foundation for a discussion of secure coding practices and current challenges in the areas of malicious code detection and prevention.



# JAVA Security

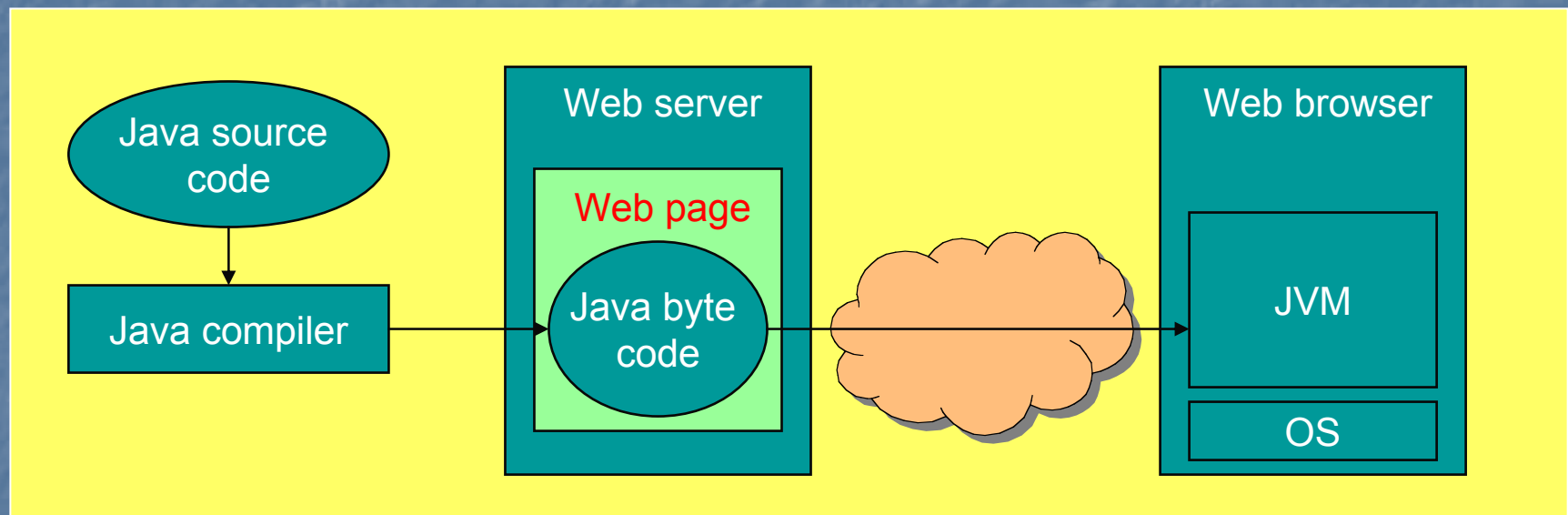
- Topics
  - Java Security
  - Java Sandbox
  - Java Cryptography Architecture (JCA)
  - Java Cryptography Extension (JCE)
  - Message Authentication and Digital Signature
  - Building Secure Applications
    - Symmetric Cryptography
    - Key Agreement
    - Signature Authentication and digital signature



# JVM in Web Browser

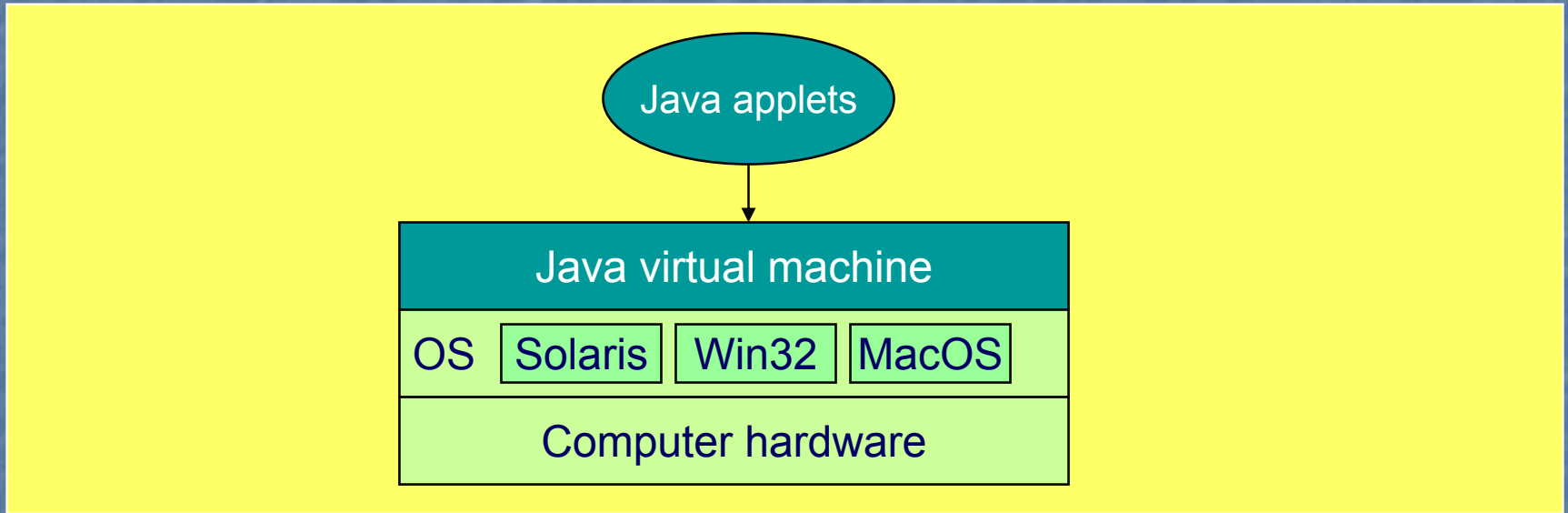
## ■ Java Security Concerns

- Java-enabled Web browsers allows Java code to be embedded in a Web page, downloaded across the net, and run on a local machine, security is a critical concern



# JVM (cont)

- Java applications run above JVM, which dynamically links in any needed Java classes



# JAVA Sandbox

## ■ Security Model for Java Applets

- Embodied by Java-enabled web browsers
- When a program is hosted on the computer, we want to provide an environment where the program can play but confine the program's play area in certain bounds (sandbox)
- Applets must be assumed to be **untrusted**, but **should not be completely restricted**
- Java security relies on three prongs of defense: **the Byte Code Verifier, the Class Loader, and the Security Manager**. Together, these three prongs perform load- and run-time checks to restrict file-system and network access, as well as access to browser internals.



# Class Loader

## ■ Applet Class Loader

- All objects belong to classes
- The Applet Class Loader determines when and how an applet can add classes to a running Java environment
- Ensure that important parts of the Java run-time environment are not replaced by code that an applet tries to install
- The Applet Class Loader, which is typically supplied by the browser vendor, loads all applets and the classes they reference. When an applet loads across the network, the Applet Class Loader receives the binary data and instantiates it as a new class



# Class Loader

- **Trusted class loader handles downloading**
  - Fetches a specified class (byte codes for class definitions) from a server and installs it in host's address space so that JVM can create objects from it
  - Also ensures that untrusted applets cannot interfere with other Java programs by isolating execution environments
- **A class loader is actually just another Java class, so a downloaded program can possibly contain its own class loaders**
  - But class loader must be trusted; a Java program is not allowed to create its own class loaders to circumvent normal class loading



# Byte Code Verifier

## ■ Byte Code

- Java uses platform-independent Java byte code in compilation
- Java byte code is "verified" before it can run
- Ensure that the byte code, which may or may not have been created by a Java compiler, plays by the rules
  - Byte code could well have been created by a "hostile compiler" that assembled byte code designed to crash the Java virtual machine



# Byte Code Verifier

- Checks a classfile for validity
  - When class files are loaded, byte code verifier checks whether downloaded class obeys certain security rules through a 4-pass verification process
  - Code only has valid instructions & register use
  - Code does not overflow/underflow stack
  - Does not convert data types illegally or forge pointers
  - Accesses objects as correct type
  - Method calls use correct number & types of arguments
  - References to other classes use legal names
  - Only classes downloaded from remote server are untrusted and must be checked; local classes are assumed to be trusted
- Goal is to prevent access to underlying machine
  - via forged pointers, crashes, undefined states

# Security Manager

## ■ Security Manager

- Security manager performs various runtime checks while JVM is instantiating objects and executing objects' methods
- Perform run-time checks on "dangerous" methods
  - Code in the Java library consults the Security Manager whenever a dangerous operation is about to be attempted
  - The Security Manager is given a chance to veto the operation by generating a Security Exception
  - Decisions made by the Security Manager take into account which Class Loader loaded the requesting class. Built-in classes are given more privilege than classes that have been loaded over the net
- Prevents objects from unauthorized access to host's resources



# Security Manager

## ■ Usually denied:

- Read/write local files
- Delete or rename files
- Create or list a directory, list file properties
- Reading of some system properties (eg, user name)
- A program can create a socket only to IP address where class files came from, eg, to prevent attacks on other machines
- Manipulating JVM, eg, replacing security manager after one has been installed, or creating a class loader
- Make Java interpreter exit
- Create a socket for listening on a port, eg, to prevent spoofing a service on the host
- Run other programs on the host



# Java Cryptography Architecture (JCA)

- **Java Cryptography Architecture (JCA)**
  - Composed of a number of classes in the **java.security** package and its sub-packages
  - Provides generic APIs for functions for various **security services**
  - Classes include
    - MessageDigest
    - Signature
    - KeypairGenerator
    - KeyFactory
    - CertificateFactory
    - KeyStore
    - AlgorithmParameters
    - AlgorithmParameterGenerator
    - SecureRandom



# Java Cryptography Architecture (JCA)

- **Cryptographic Service Provider (Provider)**
  - A collection of implementations of various algorithms
  - Sun J2VM is shipped with at least one provider (SUN provider)
    - `sun.security.provider.sun`
    - Includes implementations of the following algorithms
      - MD5 message digest
      - SHA-1 message digest
      - DSA digital certificate signing and verifying
      - DSA key pair generation
      - DSA key conversation
      - X.509 certificate creation
      - Proprietary keystore parameters
      - DSA algorithm parameters generation
      - DSA algorithm parameters
      - Proprietary random number generation



# Java Cryptography Architecture (JCA)

- JDK 1.3 ships with a second provider RSAJCA
  - `com.sun.rsajca.Provider`
  - Provides
    - RSA key pair generation
    - RSA key conversion
    - RSA signature using SHA-1 or MD5 message digest
- You or VM can specify the provider to be used



# Java Cryptography Extension (JCE)

- Java Cryptography Extension (JCE)
  - An extension to JCA, **javax.crypto** package and its sub-packages
  - Adds simple encryption and decryption APIs
  - Classes include
    - Cipher
    - KeyAgreement
    - KeyGenerator
    - Mac
    - SecretKey
    - SecretKeyFactory



# Java Cryptography Extension (JCE)

- Supported Algorithms by SUN's JCE
  - DES
  - Triple DES
  - Blowfish
  - Diffie-Hellman
- Unfortunately, RSA is not included
  - Another third party extension, Bouncy Castle JCE, provides a larger collection of cryptographic algorithms; Blowfish, DES, IDEA, RC2-6, AES, RSA, MD2, MD5, SHA-1, DSA,...
  - <http://www.bouncycastle.org>  
open source Apache style, most complete, freely available JCE provider



# Programming Project #1 : Symmetric Encryption/Decryption

## ■ Objectives

- Create a JAVA Applet that can encrypt/decrypt with multiple symmetric cryptographic algorithms
- The options for the encryption/decryption
  - DES, TripleDES, Blowfish, and S-DES (**students need to implement S-DES themselves**)
  - With or without password options
- Take an input from the textbox and encrypt/decrypt it
- Use the security manager to control the Applet



# JAVA Applet Structure

## Cryptographic Algorithms

- S-DES
- DES
- TripleDES
- Blowfish

## Password

- W/o Password
- With Password



plaintext

Hello World

ciphertext

+w#aws@\_1fs

Encryption

Decryption



# Programming Project #2 : Secure Chatting Program Using a Diffie-Hellman Key Agreement

## ■ Diffie-Hellman Key Exchange

- To enable two users to exchange a key securely that can then be used for subsequent encryption of messages
- Algorithm itself is limited to the exchange of the keys

## ■ Mission

- Develop a client/server socket programs that implements Diffie-Hellman key agreement method



# Diffie-Hellman Key Exchange

User A

Generate  
random  $X_A < q$ ;  
Calculate  
 $Y_A = \alpha^{X_A} \text{ mod } q$

Calculate  
 $K = (Y_B)^{X_A} \text{ mod } q$

User B

Generate  
random  $X_B < q$ ;  
Calculate  
 $Y_B = \alpha^{X_B} \text{ mod } q$ ;  
Calculate  
 $K = (Y_A)^{X_B} \text{ mod } q$

$Y_A$

$Y_B$

Need to have  $\alpha$  and  $q$  in advance  
Use of discrete logarithm



# Public-Key Cryptosystems

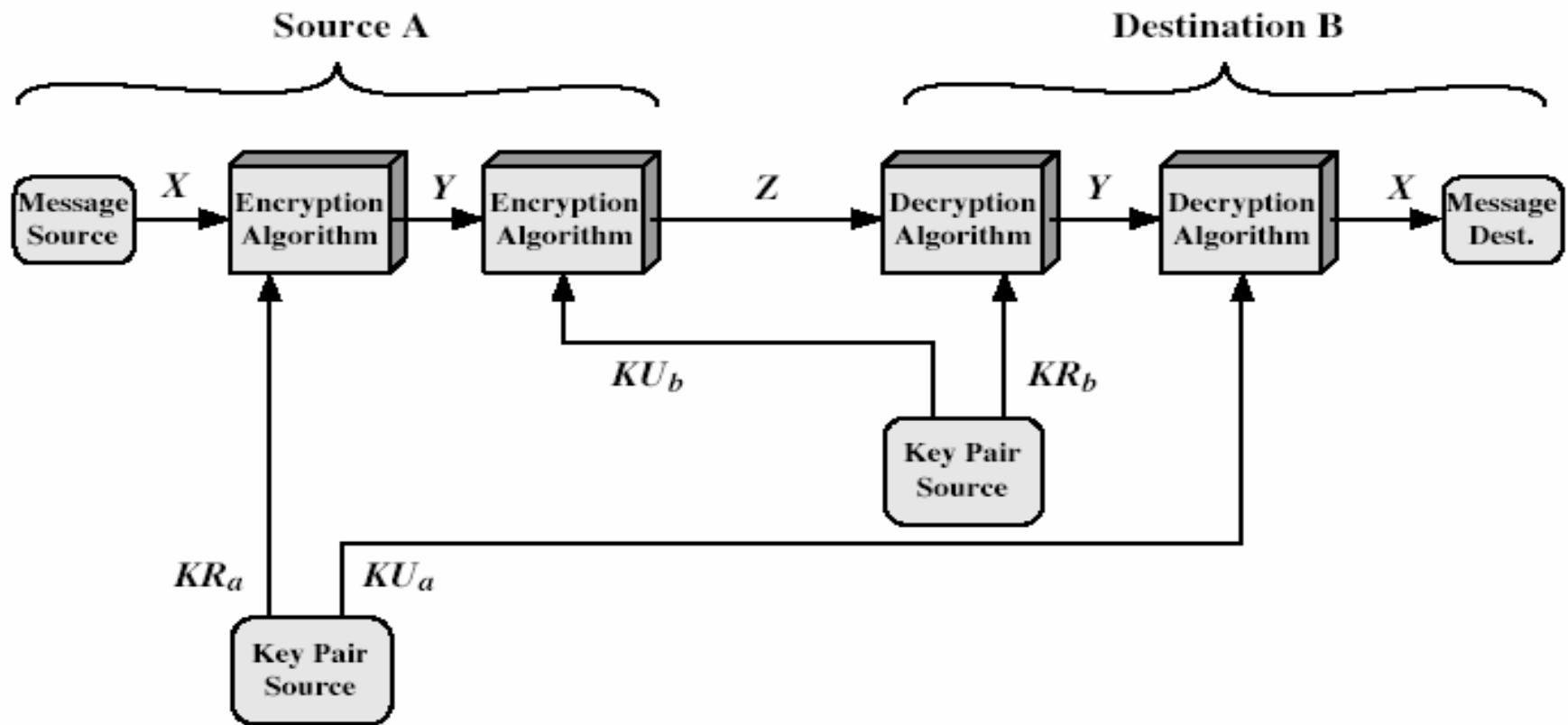


Figure 9.4 Public-Key Cryptosystem: Secrecy and Authentication



# Confidentiality vs. Authentication

## ■ Confidentiality

- The protection of data from unauthorized disclosure
- Encryption → protection against passive attack

## ■ Authentication

- The assurance that the communicating entity is the one that it claims to be
- Requirements - must be able to verify that:
  - Message came from the apparent source or author
  - Contents have not been altered
  - It was sent at a certain time or sequence.
- Protection against active attack (falsification of data and transactions)

## ■ Message authentication is concerned with:

- Message authentication: a procedure to verify that received messages come from the alleged source and have not been altered.
- Protects two parties who exchanges messages from any third party.
- Trust between two communicating parties

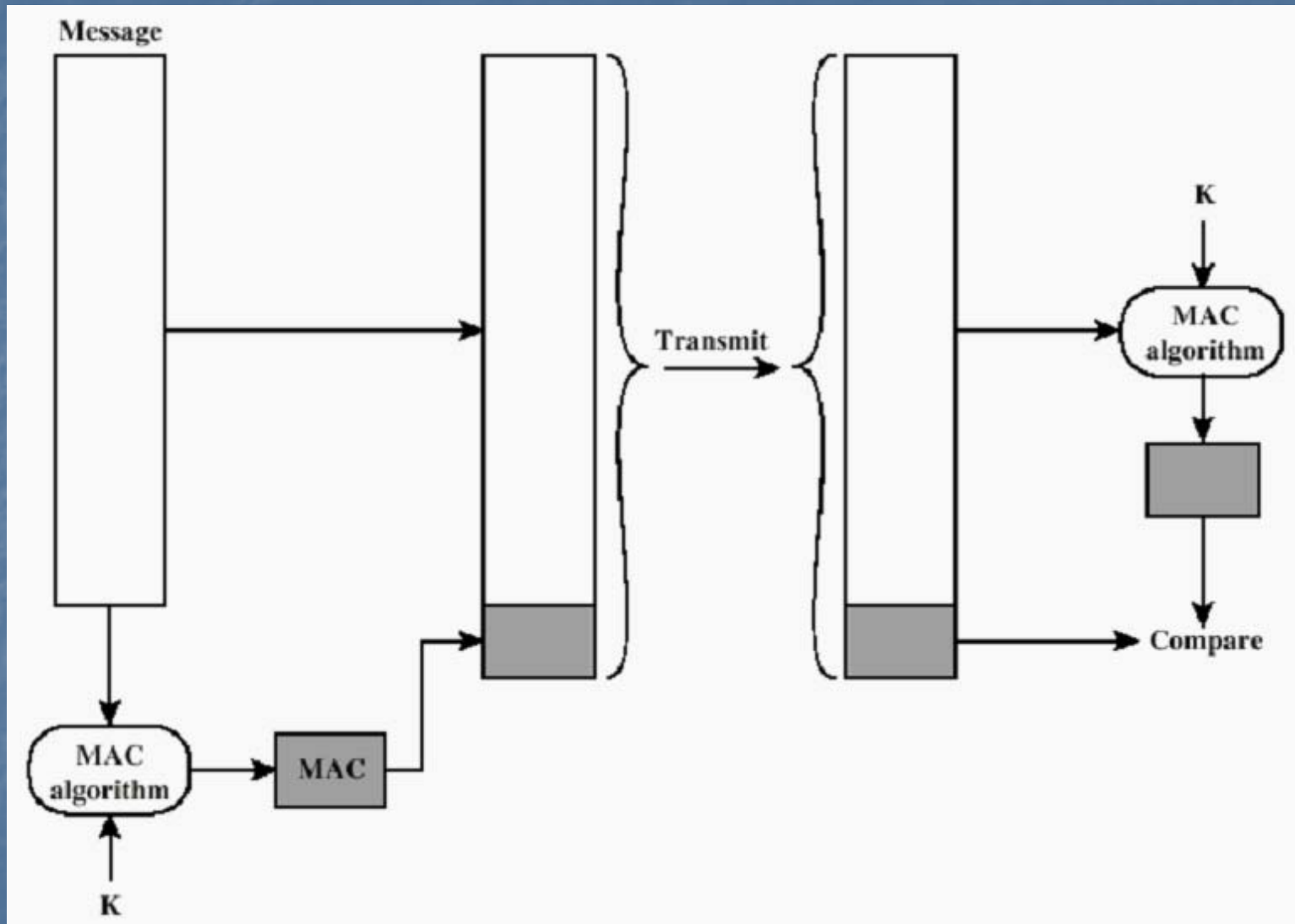


# Message Authentication Code (MAC)

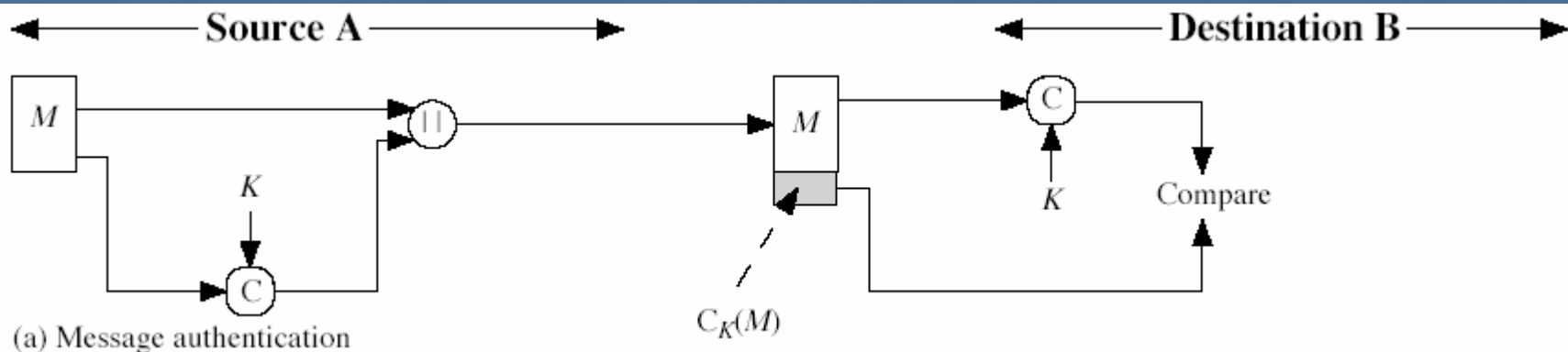
- Generated by an algorithm that creates a small fixed-sized block
  - Depending on both **message** and some **key**
  - Like encryption though **it need not be reversible**
    - Mac function is a many-to-one function
- Appended to a message as a **signature**
- Receiver performs same computation on message and checks it matches the MAC
- Provides assurance that message is unaltered and comes from the sender



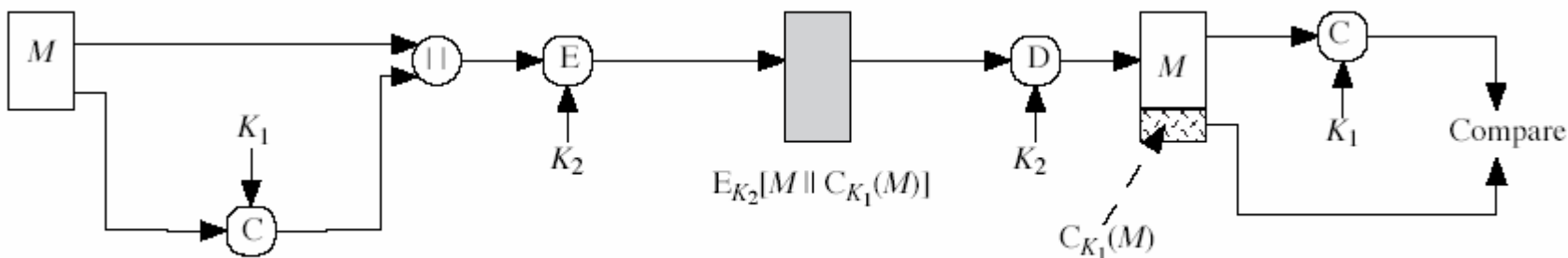
# Message Authentication Code (MAC)



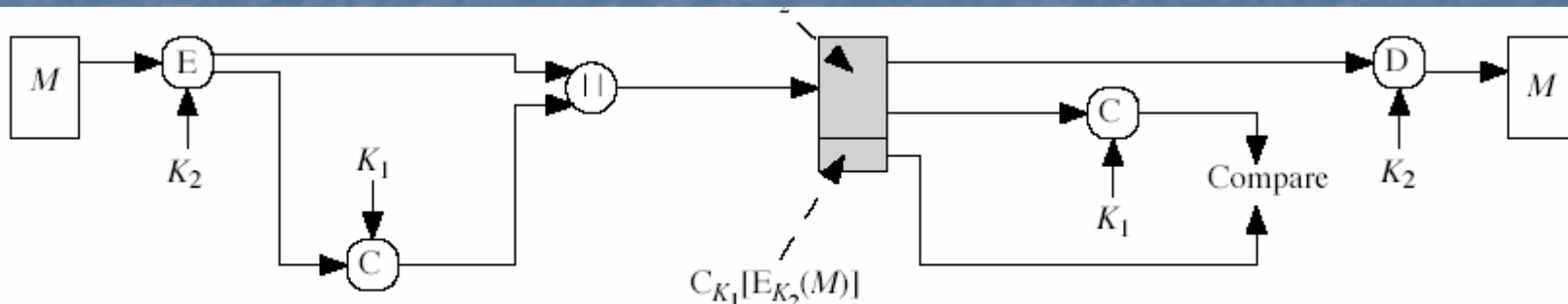
# Message Authentication Code (MAC)



(a) Message authentication



(b) Message authentication and confidentiality; authentication tied to plaintext **MAC is calculated on plaintext**



(c) Message authentication and confidentiality; authentication tied to ciphertext **MAC is calculated on ciphertext**



# Using Symmetric Ciphers for MACs

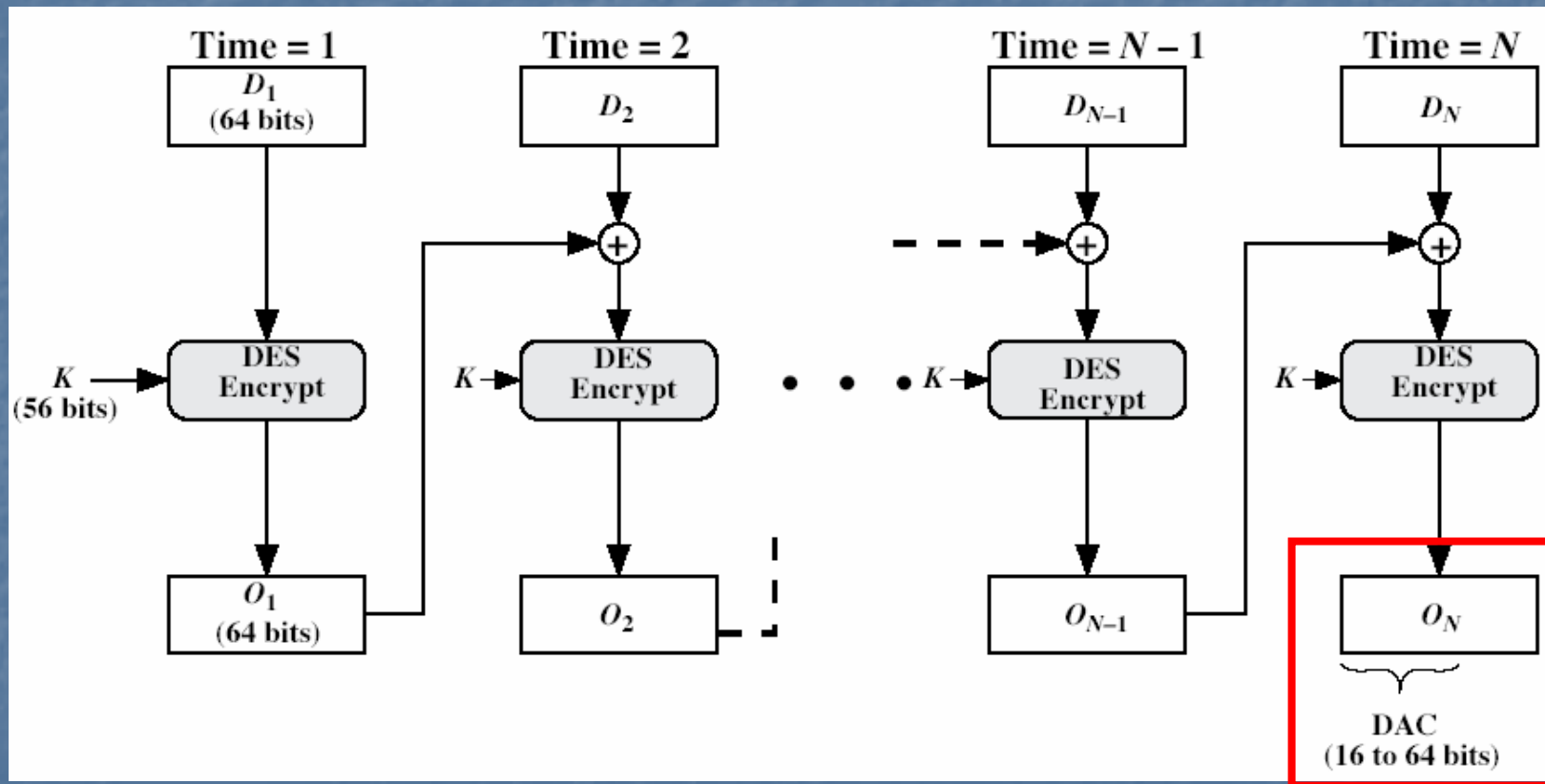
- Can use any block cipher chaining mode and use final block as a MAC
- **Data Authentication Algorithm (DAA)** is a widely used MAC based on DES-CBC
  - using IV=0 and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final block as the MAC
    - or the leftmost M bits ( $16 \leq M \leq 64$ ) of final block
- But final MAC is now too small for security



# Message Authentication Code Based on DES

Data Authentication Algorithm: NIST FIPS PUB 113, ANSI standard (X9.17)

DES Cipher Block Chaining mode is used to generate the encrypted message and MAC



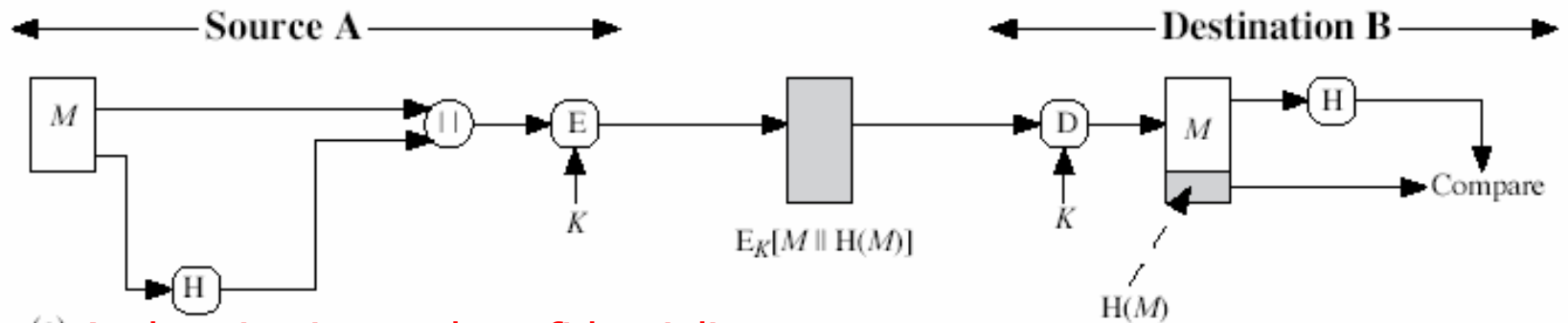
DAC is either entire block  $O_N$  or the left most  $M$  bits of the block ( $16 \leq M \leq 64$ )

# One-way Hash Function

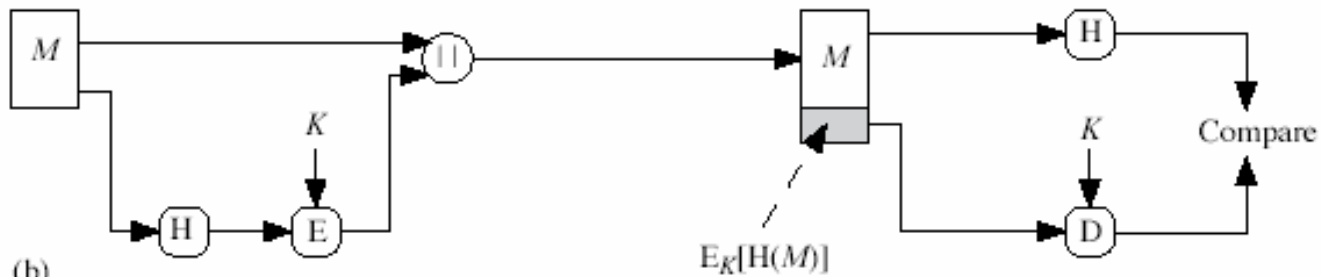
- Condenses arbitrary message to a fixed size
- usually assume that the hash function is public and not keyed
  - MAC takes a key as input while hash function **does not**
- Hash used to detect changes to message
- The hash code is referred to as a **message digest** or **hash value**
- Most often to create a digital signature



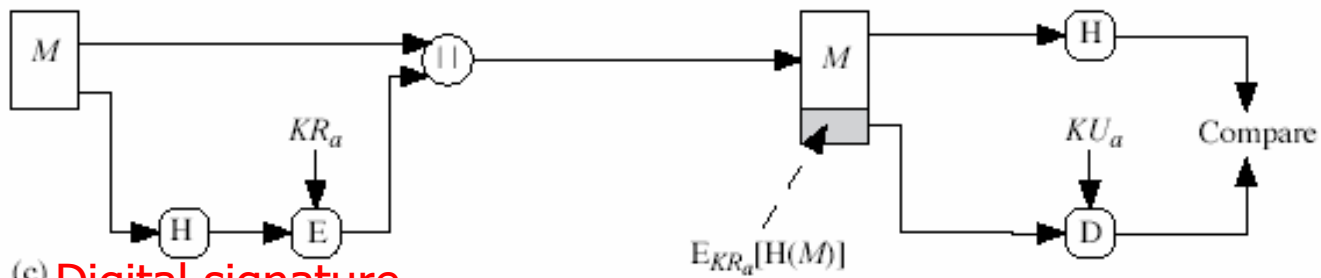
# One-way Hash Function



(a) Authentication and confidentiality



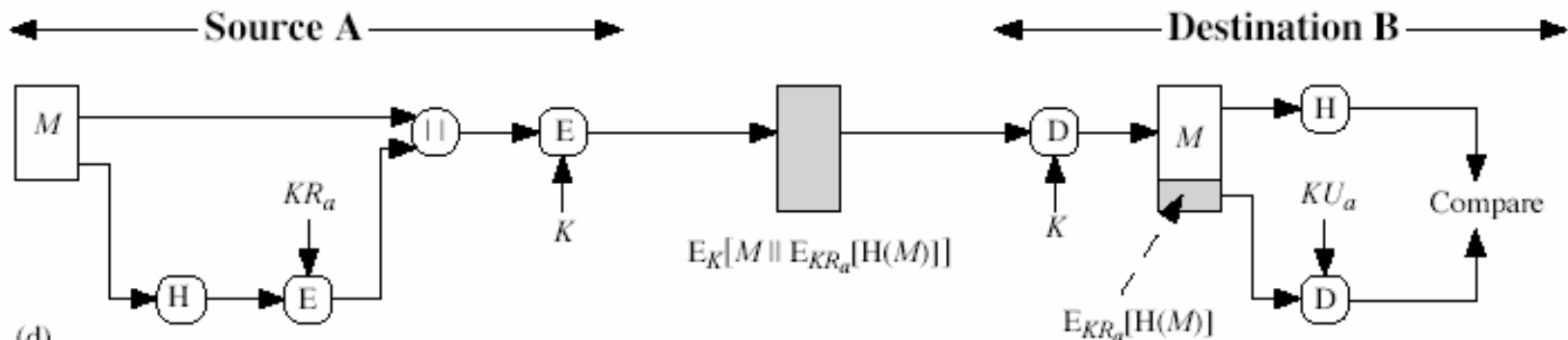
(b) Authentication



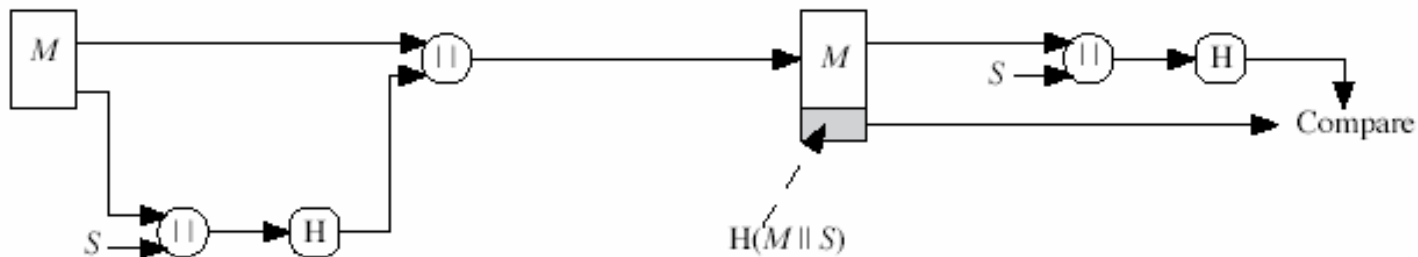
(c) Digital signature



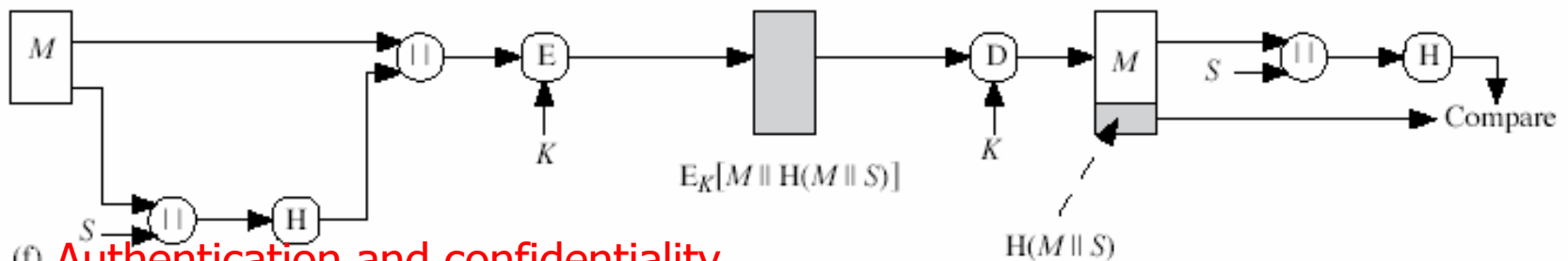
# One-way Hash Function



(d) Digital signature and confidentiality



(e) Authentication (using a secret value  $s$ )



(f) Authentication and confidentiality



# Why Digital Signature?

- What is the difference between message authentication and digital signature?
  - Message authentication does not protect two parties against each other.
  - What if there is no complete trust between sender and receiver?
- **Examples** (Bob sends an authenticated message to Alice)
  - Alice may forge a message and claim that it came from Bob by simply creating a message and append an authentication code using the key Bob and Alice share.
  - Bob can deny sending the message since it's possible for Alice to forge a message.



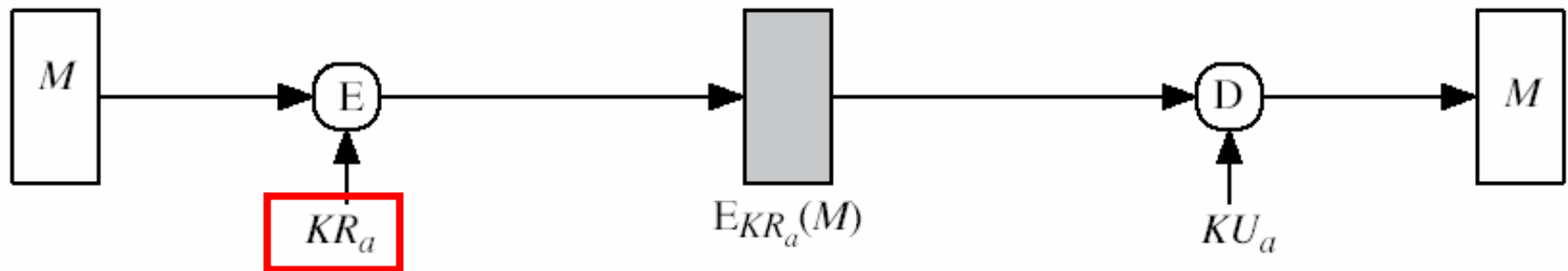
# Digital Signature

- Digital signatures provide the ability to:
  - verify author, date & time of signature
  - authenticate message contents
  - be verified by third parties to resolve disputes
- Hence, include authentication function with additional capabilities

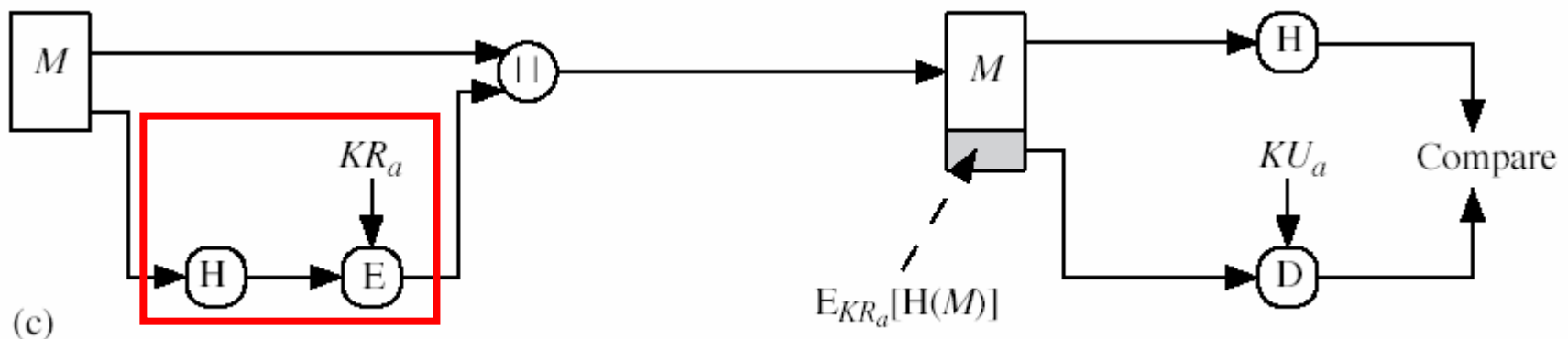


# Direct Digital Signatures

- Involve only sender & receiver
- Assumed receiver has sender's public-key
- Digital signature made by sender signing entire message or hash with private-key



(c) Public-key encryption: authentication and signature

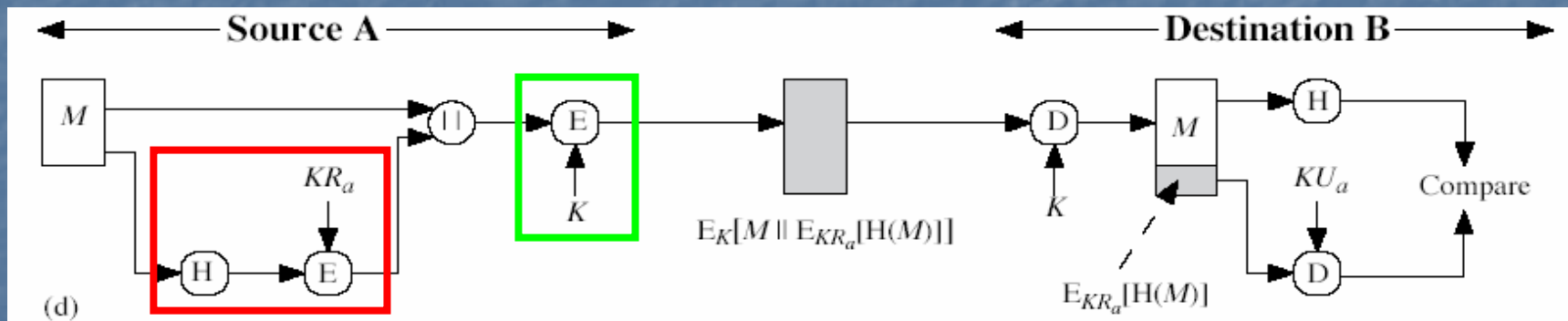
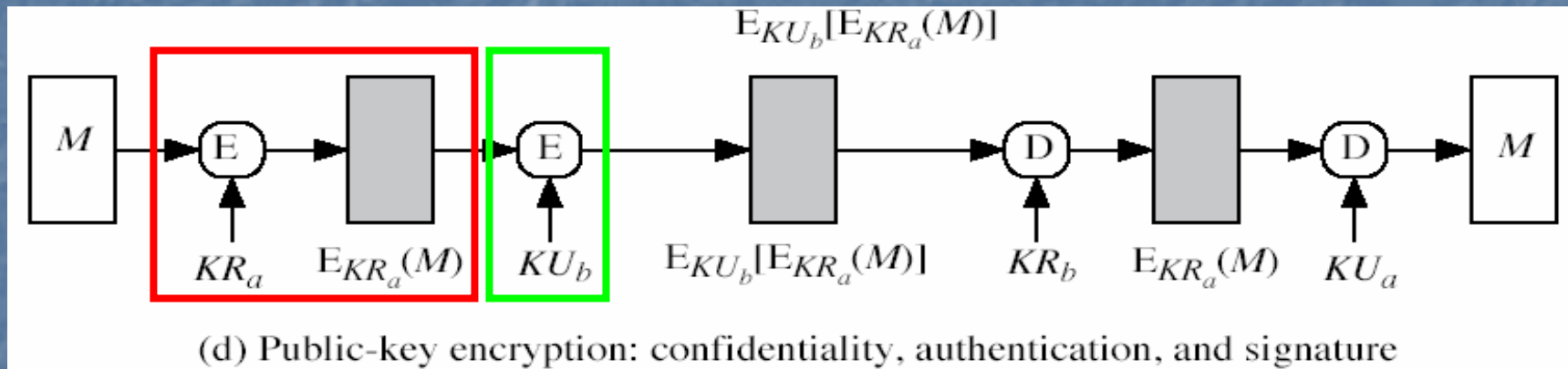


(c)



# Direct Digital Signatures

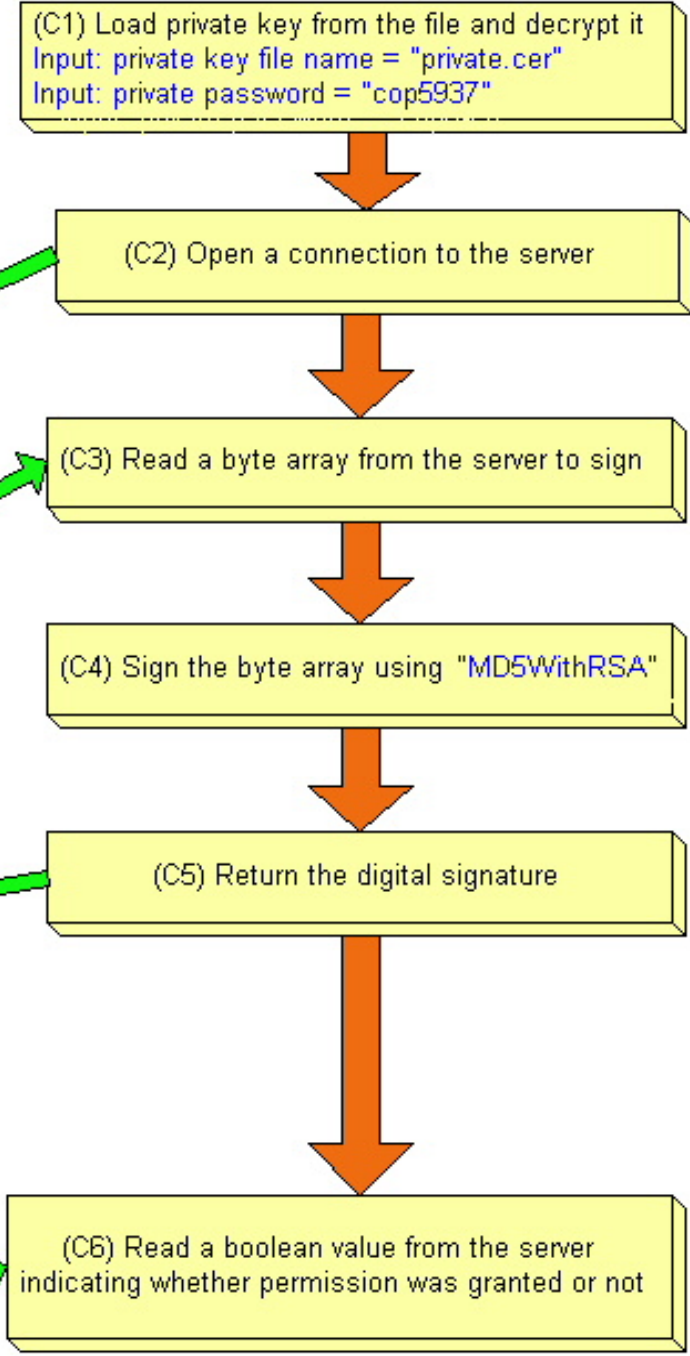
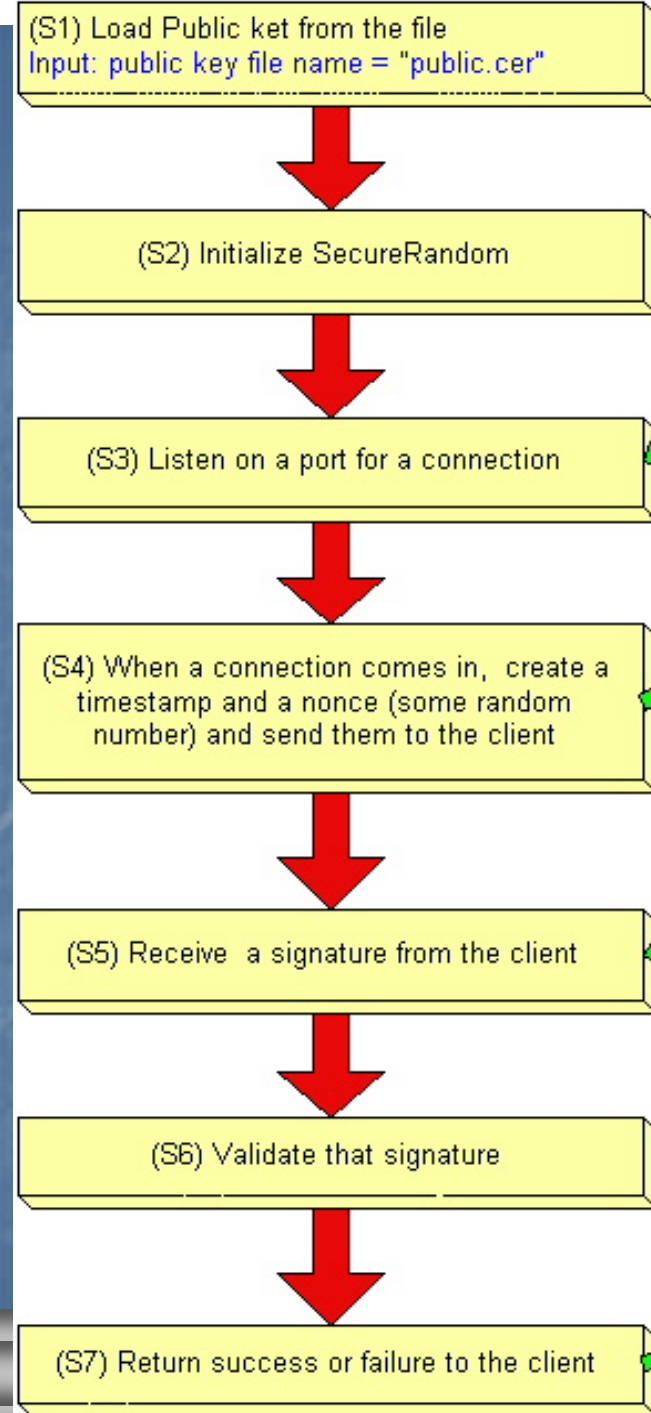
- Confidentiality can be provided by further encrypting using receivers public-key or a shared secret key
- Important that sign first then encrypt message & signature



# Programming Project #3 : Signature Authentication Server using Digital Signature

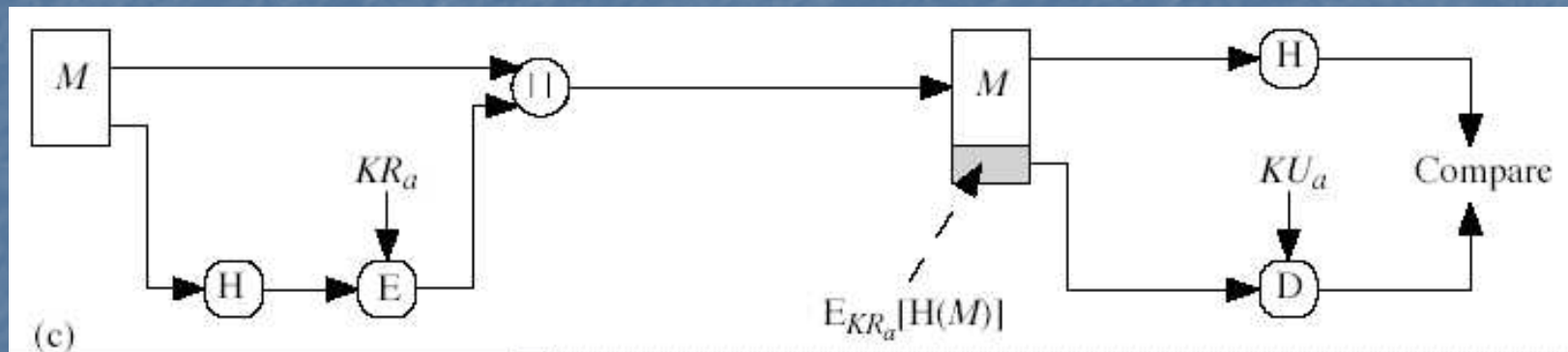
- Objectives
  - The server reads a client's public key from the file and authenticates users over a network connection using the client's public key information.





# Authentication and Signature

- Message Authentication
  - (S6) Validate that signature using client's public key with "MD5WithRSA" option.



# Software Security



# Reverse Code Engineering

- **What Is Reverse Code Engineering (RCE)?**
  - Seeking to understand and modify the behavior of a program for which you have no source code or documentation other than a compiled binary.
- **Why Reverse Code Engineering?**
  - Learn how software security might be compromised and how to better protect one's own software.
  - Improve software debugging skills
  - Understand how programs interact with the operating system.
  - Ability to modify / repair "legacy code"
  - Skills to reverse engineer and disable "mal-ware" (viruses, trojans, spy-ware, etc.)



# Code Reversing Skills

- Using your brain
  - Research
  - Pattern Recognition
- Using your tools
  - Debuggers
  - Disassemblers
  - Hex Editors
- Know your target
  - The instruction set
  - Common file formats
  - Native API
  - Standard error codes
- Intuition “Zen”
  - Anti-patterns? “Normal” vs. “Abnormal”?
- RCE Methodology
  - Putting it all together

# Pattern Recognition

- We are searching the program code for a proverbial “needle in the haystack”.
  - A program disassembly may be 20,000 pages and a million lines long!
- We are viewing the program code through a proverbial “toilet paper tube” (ie. with a very narrow field of view).
  - We may only be interested in 5 out of those 20,000 pages!
- The goal is NOT to understand every line.
  - The goal is to screen out “noise”!
  - We want to recognize patterns and define “normal” behavior to orient ourselves in the code and narrow focus.



# Defining & Recognizing Normal Behavior

- High level vs. low level code patterns...i.e. recognizing...
  - Function calls
  - Structure references
  - "If" statements
  - "Switch" statements
  - "While" loops
  - "For" loops
- Windows patterns...i.e. recognizing...
  - Standard windows startup code
  - WinMain & WndProc
  - Windows API functions
  - Windows Messages & Message Handlers



# Recognizing Function Calls (cdecl)

Review: Caller pushes function parameters on the stack from right to left; caller balances the stack.

## ■ Assembly

### ■ Function Invocation:

```
■ .text:00401049      push    2
■ .text:0040104B      push    1
■ .text:0040104D      call   sub_401166
■ .text:00401052      add     esp, 8
■ .text:00401055      mov     [ebp+var_4], eax
```

### ■ Function Body:

```
■ .text:00401166 sub_401166  proc near
■ .text:00401166 arg_0      = dword ptr 8
■ .text:00401166
■ .text:00401166      push   ebp
■ .text:00401167      mov    ebp, esp
■ .text:00401169      mov    eax, [ebp+arg_0]
■ .text:0040116C      mov    [ebp+arg_0], eax
■ .text:0040116F      mov    ecx, [ebp+arg_0]
■ .text:00401172      add    ecx, 1
■ .text:00401175      mov    [ebp+arg_0], ecx
■ .text:00401178      mov    eax, [ebp+arg_0]
■ .text:0040117B      pop    ebp
■ .text:0040117C      retn
■ .text:0040117C sub_401166  endp
```

## ■ C

### ■ Function Invocation:

```
■ result = test_cdecl(1,2);
```

### ■ Function Body:

```
■ int __cdecl test_cdecl( int t, int u )
■ {
■     t = t++;
■     return t;
■ }
```

# Recognizing Function Calls (stdcall)

Review: Caller pushes function parameters on the stack from right to left; callee balances the stack.

## ■ Assembly

### ■ Function Invocation:

```
■ .text:0040103D      push  2
■ .text:0040103F      push  1
■ .text:00401041      call  sub_40114D
■ .text:00401046      mov   [ebp+var_4], eax
```

### ■ Function Body:

```
■ .text:0040114D sub_40114D  proc near
■ .text:0040114D arg_0      = dword ptr 8
■ .text:0040114D
■ .text:0040114D      push  ebp
■ .text:0040114E      mov   ebp, esp
■ .text:00401150      mov   eax, [ebp+arg_0]
■ .text:00401153      mov   [ebp+arg_0], eax
■ .text:00401156      mov   ecx, [ebp+arg_0]
■ .text:00401159      add   ecx, 1
■ .text:0040115C      mov   [ebp+arg_0], ecx
■ .text:0040115F      mov   eax, [ebp+arg_0]
■ .text:00401162      pop   ebp
■ .text:00401163      retn  8
■ .text:00401163 sub_40114D  endp
```

## ■ C

### ■ Function Invocation:

```
■ result = test_stdcall(1,2);
```

### ■ Function Body:

```
■ int __stdcall test_stdcall( int t, int u )
■ {
■     t = t++;
■     return t;
■ }
```



# Recognizing Function Calls (fastcall)

Review: Caller places first 2 parameters in registers ecx and edx and pushes remaining function parameters on the stack from right to left; callee balances the stack.

## ■ Assembly

### ■ Function Invocation:

- .text:00401058 push 3
- .text:0040105A mov edx, 2
- .text:0040105F mov ecx, 1
- .text:00401064 call sub\_40117D
- .text:00401069 mov [ebp+var\_4], eax

### ■ Function Body:

- .text:0040117D sub\_40117D proc near
- .text:0040117D var\_C = dword ptr -0Ch
- .text:0040117D var\_8 = dword ptr -8
- .text:0040117D var\_4 = dword ptr -4
- .text:0040117D arg\_0 = dword ptr 8
- .text:0040117D push ebp
- .text:0040117E mov ebp, esp
- .text:00401180 sub esp, 0Ch
- .text:00401183 mov [ebp+var\_C], edx
- .text:00401186 mov [ebp+var\_8], ecx
- .text:00401189 mov eax, [ebp+var\_8]
- .text:0040118C add eax, [ebp+var\_C]
- .text:0040118F add eax, [ebp+arg\_0]
- .text:00401192 mov [ebp+var\_4], eax
- .text:00401195 mov eax, [ebp+var\_4]
- .text:00401198 mov esp, ebp
- .text:0040119A pop ebp
- .text:0040119B retn 4
- .text:0040119B sub\_40117D endp

## ■ C

### ■ Function Invocation:

- result = test\_fastcall(1,2,3);

### ■ Function Body:

- int \_\_fastcall test\_fastcall( int t, int u, int v )
- {
- int x = t + u + v;
- return x;
- }



# Recognizing An "If" Statement

## ■ Assembly

```
■ .text:0040108A      mov     ecx, [ebp+var_90]
■ .text:00401090      and     ecx, 0FFh
■ .text:00401096      cmp     ecx, 1
■ .text:00401099      jnz    short loc_4010A4
■ .text:0040109B      mov     byte ptr [ebp+var_90], 0
■ .text:004010A2      jmp     short loc_4010AB
■ .text:004010A4 loc_4010A4:  ; CODE XREF
■ .text:004010A4      mov     byte ptr [ebp+var_90], 1
■ .text:004010AB
```

## ■ C

```
■ if( test == true)
■     test = false;
■ else
■     test = true;
```



# Recognizing A "While" Loop

## ■ Assembly

```
■ .text:004010F7 loc_4010F7 ; CODE XREF
■ .text:004010F7      cmp    [ebp+var_94], 0Ah
■ .text:004010FE      jge   short loc_401111
■ .text:00401100      mov   eax, [ebp+var_94]
■ .text:00401106      add   eax, 1
■ .text:00401109      mov   [ebp+var_94], eax
■ .text:0040110F      jmp   short loc_4010F7
```

## ■ C

```
■ int i = 0;
■ while( i < 10)
■ {
■     i++;
■ };
```



# Recognizing A "For" Loop

## ■ Assembly

```
■ .text:0040111B      mov     [ebp+var_98], 0
■ .text:00401125      jmp     short loc_401136
■ .text:00401127
■ .text:00401127 loc_401127:      ; CODE XREF
■ .text:00401127      mov     ecx, [ebp+var_98]
■ .text:0040112D      add     ecx, 1
■ .text:00401130      mov     [ebp+var_98], ecx
■ .text:00401136
■ .text:00401136 loc_401136:      ; CODE XREF
■ .text:00401136      cmp     [ebp+var_98], 0Ah
■ .text:0040113D      jge     short loc_401144
■ .text:0040113F      nop
■ .text:00401140      nop
■ .text:00401141      nop
■ .text:00401142      jmp     short loc_401127
■ .text:00401144 loc_401144:      ; CODE XREF
```

## ■ C

```
■ int j = 0;
■ for( j = 0; j < 10; j++)
■ {
■     __asm
■     {
■         nop
■         nop
■         nop
■     };
■ };
```



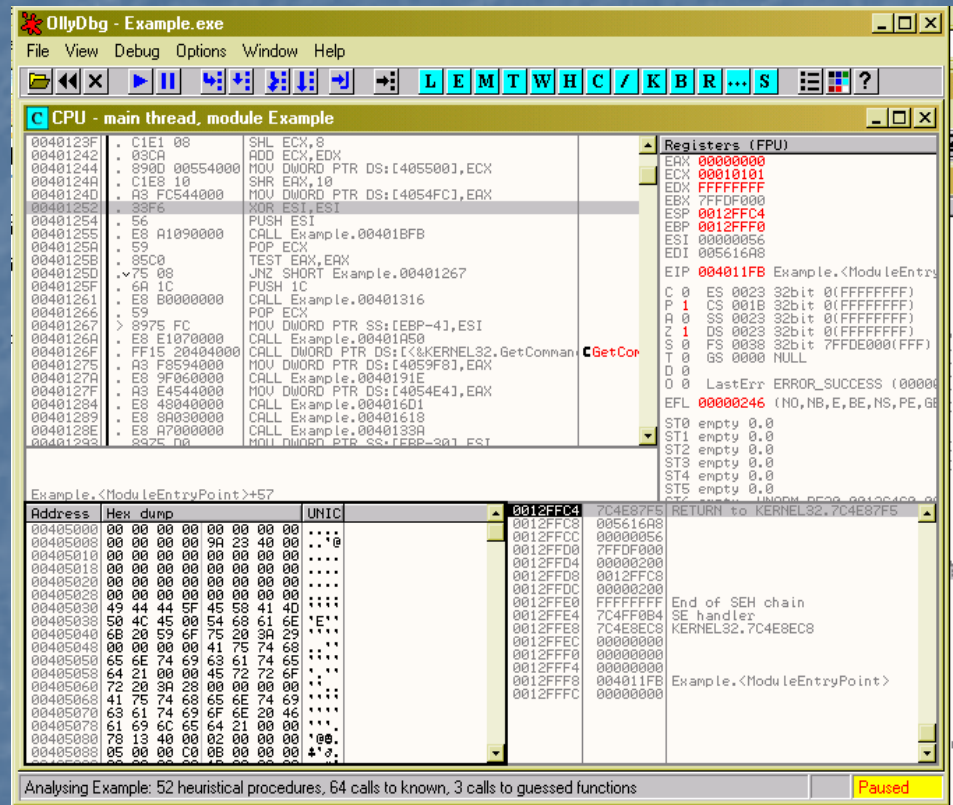
# The Tools Of The Trade

- Disassemblers
- Debuggers
- Hex Editors
- PE Editors
- Registry, File, and API Monitoring Tools
- Resource Hackers



# Debugger

- OllyDbg
- Used for real time or “live” examination of running code using breakpoints or single step tracing.
- Dynamic view of memory, registers, and the system stack.







# Other Tools

- **PE Editor** – used to modify physical structure / attributes of files conforming to the “portable executable” format specification (ie. .exe, .sys, and .dll) files.
- **Real-Time Monitoring Tools** – tools used to dynamically monitor system activity including registry access, file access, and sequence of system calls.
- **Resource Hackers** – used to modify GUI resources stored in the executable file (icons, images, cursors, and strings)



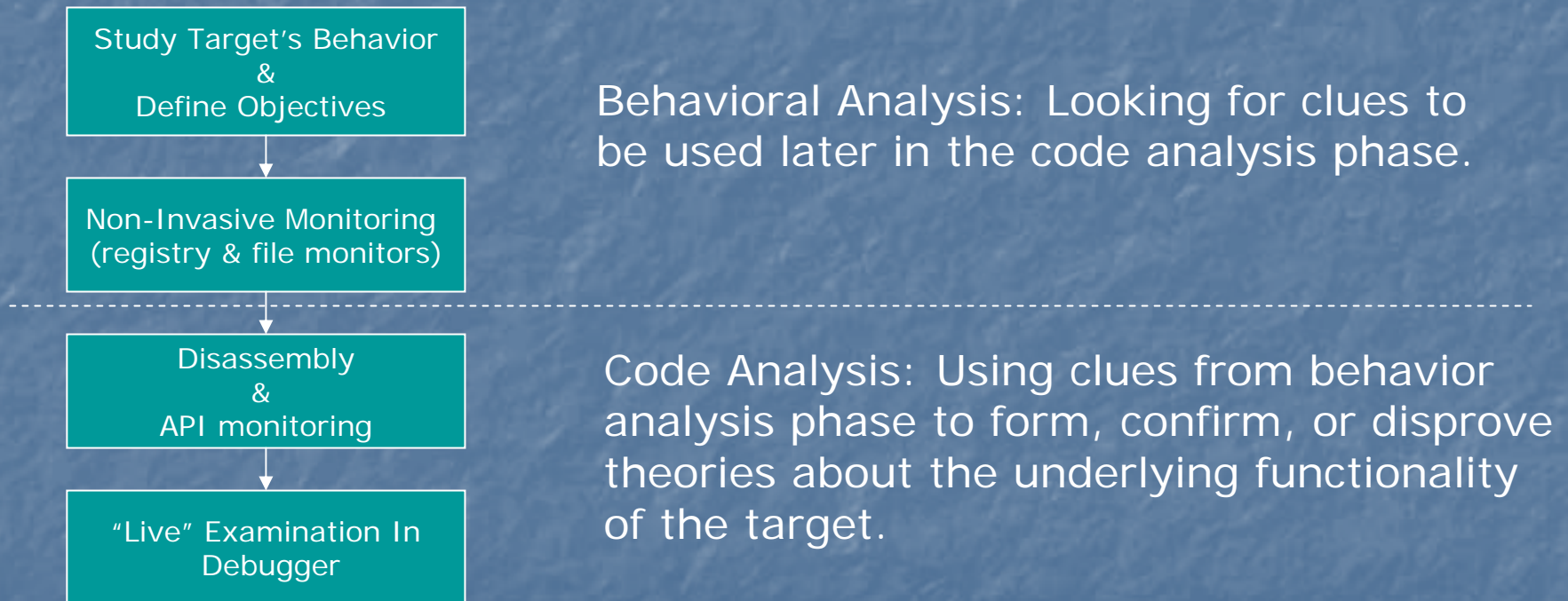
# Intuition: “Zen”

- Antipatterns?
  - What is “abnormal”?
- Sensing the “BIG” picture
  - Graph visualization between functional components
    - What type of patterns or relationships among components do you expect to see in a control flow or call graph?
  - Cross references between functional components
    - How many times do you expect to see the target functions referenced (or not referenced).



# RCE Methodology: Putting It All Together

- A “top down” approach:



# Modifying a compiled binary...

## □ **Code Injection via "Inline Patching"**

- Inserting or replacing the instructions in a compiled program with new assembly instructions. The inline patch may also require rerouting code flow to and from an external injection site in the file.



## Static Vs. Dynamic Code Injection

- **Static Code Injection** is performed on the disk image (i.e., the .exe file) of a program using a Hex Editor like *Hacker's View (HIEW)*.
  - Changes to the file system resulting from static code injection can be detected using integrity checkers like "Trip Wire".
- **Dynamic Code Injection** is performed at runtime after the program image has been loaded into memory.
  - The program file remains unchanged.
  - Defeats integrity checkers.



# Basic Inline Patching

- Replacing instructions with a patch of the same length.
- Replacing the operand of an existing instruction (ie. the destination of an existing jmp or call).
- Replacing instructions with a patch of a different length.



## EXAMPLE: Replacing instructions with a patch of the same length.

Before Patch

Address	Opcodes	Instructions
00400000	034A	Inst1
00400002	6B328021	Inst2 op1, op2
00400006	846A23	Inst3 op1

Patch Address = 00400000 Replace Inst 1 with Inst 4 (053B)

After Patch

Address	Opcodes	Instructions
00400000	053B	Inst4
00400002	6B328021	Inst2 op1, op2
00400006	846A23	Inst3 op1

# EXAMPLE: Replacing instructions with a patch of a different length.

Must maintain existing instruction boundaries... If the patch is smaller than the last overwritten instruction, remaining. Bytes in that instruction must be padded with nop instructions (0x90).

Before  
Patch

Address	Opcodes	Instructions
00400000	034A	Inst1
00400002	6B328021	Inst2 op1, op2
00400006	846A23	Inst3 op1

Patch Address = 00400000 Replace Inst 1 with Inst 5 (053BA2)

After  
Patch

Address	Opcodes	Instructions
00400000	053B	Inst5
00400002	A2 90 90 90	nop nop nop
00400006	846A23	Inst3 op1



# RCE of a Simplest Authentication Program

- Comparison of the password  
if (strcmp(password entered, reference password))
- Assumption
  - The password program isn't ciphered and stored in the program
  - Compiler puts the initial variables in the data segment (.data)
  - Note that modern operating systems prohibit modifying the code segment (.text)
- Tools
  - DUMPBIN : supplied with MS Visual Studio
  - Hex-Editors : QVIEW, HIEW



```
#include <stdio.h>
#include <string.h>
```

```
#define PASSWORD_SIZE 100
#define PASSWORD "myGOODpassword\n"
```

```
int main()
{
    int count=0;
    char buff[PASSWORD_SIZE];

    for (;;) {
        printf("Enter Password : ");
        fgets(&buff[0], PASSWORD_SIZE, stdin);
        if (strcmp (&buff[0], PASSWORD))
            printf("Wrong Password\n");
        else
            break;
        if (++count>3) return -1;
    }
    printf("Password OK\n");
}
```

## Code Example 1: Simple Authentication



# Searching for the Password

- Use dumpbin to find the clues to the password

```
jlee@sony2 ~/Program/Hacker/1_SimpleAuthentication/Release
$ dumpbin /RAWDATA:BYTES /SECTION:.data 1_SimpleAuthentication.exe > .data
```

```
jlee@sony2 ~/Program/Hacker/1_SimpleAuthentication/Release
$ more .data
Microsoft (R) COFF Binary File Dumper Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

Dump of file 1_SimpleAuthentication.exe

File Type: EXECUTABLE IMAGE

SECTION HEADER #3
.data name
3E64 virtual size
7000 virtual address
3000 size of raw data
7000 file pointer to raw data
0 file pointer to relocation table
0 file pointer to line numbers
0 number of relocations
0 number of line numbers
C0000040 flags
Initialized Data
Read Write

RAW DATA #3
00407000: 00 00 00 00 00 00 00 00 00 00 00 00 47 11 40 00 .....G.e.
00407010: 84 49 40 00 00 00 00 00 00 00 00 00 EC 11 40 00 ,IE.....i.e.
00407020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407030: 50 61 73 73 77 6F 72 64 20 4F 4B 0A 00 00 00 00 Password OK....
00407040: 57 72 6F 6E 67 20 50 61 73 73 77 6F 72 64 08 00 Wrong Password..
00407050: 6D 79 47 4F 4F 44 70 61 73 73 77 6F 72 64 0A 00 myGOODpassword..
00407060: 45 6E 74 65 72 20 50 61 73 73 77 6F 72 64 20 3H Enter rassword :
00407070: 20 00 00 00 00 00 00 00 60 9E 40 00 00 00 00 00 .....`ze.....
00407080: 60 9E 40 00 01 01 00 00 00 00 00 00 00 00 00 00 .....`ze.....
00407090: 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....`ze.....
```

```

C:\Program\Hacker\1_SimpleAuthentication\Release
DATA~1  ↓PR  00000000  0  -----  61275 || Hiew DEMO (c)SEN
Microsoft (R) COFF Binary File Dumper Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

Dump of file 1_SimpleAuthentication.exe
File Type: EXECUTABLE IMAGE

SECTION HEADER #3
.data name
3E64 virtual size
7000 virtual address
3000 size of raw data
7000 file pointer to raw data
0 file pointer to relocation table
0 file pointer to line numbers
0 number of relocations
0 number of line numbers
C0000040 flags
Initialized Data
Read Write

RAW DATA #3
00407000: 00 00 00 00 00 00 00 00 00 00 00 00 47 11 40 00 .....G.e
00407010: 84 49 40 00 00 00 00 00 00 00 00 00 EC 11 40 00 äIE.....w.e
00407020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407030: 50 61 73 73 77 6F 72 64 20 4F 4B 0A 00 00 00 00 Password OK.....
00407040: 57 72 6F 6F 67 20 50 61 73 73 77 6F 72 64 00 00 Wrong Password
00407050: 6D 79 47 4F 4F 44 70 61 73 73 77 6F 72 64 0A 00 noGOODpassword..
00407060: 45 6E 74 65 72 20 50 61 73 73 77 6F 72 64 20 3A Enter Password :
00407070: 20 00 00 00 00 00 00 00 60 9E 40 00 00 00 00 00 .....Re
00407080: 60 9E 40 00 01 01 00 00 00 00 00 00 00 00 00 00 .....Re
00407090: 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
004070A0: 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 .....
004070B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
004070C0: 00 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00 .....
004070D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
004070E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
004070F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00407180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
1|help 2|map 3 4|mode 5 6|LnFeed 7|Search 8|Lat 9|Files 10|Quit

```

HIEX  
Text Mode

# Searching for the Password

- The compiler has selected a too prominent place for a password
- Hide it in a different place, in a different section
  - Try a new section named `“.kpnc”`



```

#include <stdio.h>
#include <string.h>

#define PASSWORD_SIZE 100
#define PASSWORD "myGOODpassword\n"

#pragma data_seg(".kpnc")
char passwd[] = PASSWORD;
#pragma data_seg()

int main()
{
    int count=0;

    char buff[PASSWORD_SIZE] = "";

    for (;;) {
        printf("Enter Password : ");
        fgets(&buff[0], PASSWORD_SIZE, stdin);

        if (strcmp (&buff[0], &passwd[0]))
            printf("Wrong Password\n");
        else
            break;

        if (++count>3) return -1;
    }

    printf("Password OK\n");
}

```

Use of a new section for a  
password  
.kpnc section



```
jlee@sony2 ~/Program/Hacker/2_SimpleAuthentication/Release
$ dumpbin /RAWDATA:BYTES /SECTION:.kpnc 2_SimpleAuthentication.exe > .kpnc
```

```
- /Program/Hacker/2_SimpleAuthentication/Release
KPNCT1  JFR  00000000  0  -----  682 || Hiew DEMO (c)SEN
Microsoft (R) COFF Binary File Dumper Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

Dump of file 2_SimpleAuthentication.exe
File Type: EXECUTABLE IMAGE

SECTION HEADER #4
.kpnc name
 10 virtual size
B000 virtual address
1000 size of raw data
A000 file pointer to raw data
 0 file pointer to relocation table
 0 file pointer to line numbers
 0 number of relocations
 0 number of line numbers
C0000040 flags
  Initialized Data
  Read Write

RAW DATA #4
0040B000: 6D 79 47 4F 4F 44 70 61 73 73 77 6F 72 64 0A 00 myGOODpassword..

Summary
 1000 .kpnc
```

.kpnc section contains the password

Still the possibility of an automated search for text string in a binary file



# Improved Solution

- In the previous step, we found the password and need to use it every time we start the program
  - Hack the program so that no password is requested
- Available Tools
  - Debugger, disassembler
- Disassembler
  - There are many disassemblers with different capabilities
  - We start with the **dumpbin** utility



# Improved Solution

- Disassemble the code section

```
jlee@sony2 ~/Program/Hacker/1_SimpleAuthentication/Release
$ dumpbin /Section:.text /DISASM 1_SimpleAuthentication.exe > .code

jlee@sony2 ~/Program/Hacker/1_SimpleAuthentication/Release
$ ls -al
total 808
drwxrwxrwx+ 2 jlee None 0 Jun 17 17:38 .
drwxrwxrwx+ 4 jlee None 0 Jun 17 17:36 ..
-rw-r--r-- 1 jlee None 397976 Jun 17 17:54 .code
-rw-r--r-- 1 jlee None 61275 Jun 17 17:09 .data
-rw-r--r-- 1 jlee None 61275 Jun 17 17:38 .kopc
-rwxpwxpwx 1 jlee None 40960 Jun 14 22:21 1_SimpleAuthentication.exe
-rwx-----+ 1 jlee None 220020 Jun 14 22:20 1_SimpleAuthentication.pch
-rwxpwxpwx 1 jlee None 1486 Jun 14 22:21 SimpleAuthentication.obj
-rwxpwxpwx 1 jlee None 33792 Jun 14 22:21 uc60.idb
```

- Note that the size is usually much larger than the original code
- Need to focus on the useful codes rather than the entire code



# Improved Solution

## ■ Clues

- We don't know where the procedure to match passwords is located and how it works
- However, assert that one of its arguments is a pointer to the reference password



# Example: Finding Functions and Parameters

- Parameters of a function

```
#include <stdio.h>
#include <string.h>
#define PASSWORD "password"
int myFunction(char *a, int b);
int main()
{
    int x;
    x = myFunction(PASSWORD, 2);
}
int myFunction(char *a, int b)
{
    return 1;
}
```





# Example: Finding Functions and Parameters

- Inline functions (**intrinsic function**)
  - In certain high-level programming languages, such as FORTRAN, a function that is part of the language. The compiler automatically links intrinsic functions to the program without any additional effort on the programmer's part. Programs that use intrinsic functions are faster because they do not have the overhead of function calls, but they may be larger due to the additional code generated
  - In Microsoft C++, a library function, such as **strcmp** or **strcpy**, that has an intrinsic form. Use of the `#pragma intrinsic` compiler directive generates these functions as inline code rather than as function calls
    - No function calls for `strcmp`



```

RAW DATA #3
00407000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4A 1B 40 00 .....J.e.
00407010: A4 34 40 00 00 00 00 00 00 00 00 00 00 00 EF 1B 40 00 *4e.....i.e.
00407020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00407030: 50 61 73 73 77 6F 72 64 20 4F 4B 0A 00 00 00 00 Password OK.....
00407040: 57 72 6F 6E 67 20 50 61 73 73 77 6F 72 64 08 00 Wrong Password..
00407050: 6D 79 47 4F 4F 44 70 61 73 73 77 6F 72 64 0A 00 myGOODpassword..
00407060: 45 6E 74 65 72 20 50 61 73 73 77 6F 72 64 20 3A Enter Password :
00407070: 20 00 00 00 11 10 10 00 01 00 00 00 20 61 10 00 .....a.e.
00407080: 18 61 40 00 00 00 00 00 40 7E 40 00 00 00 00 00 .....e...
00407090: 40 7E 40 00 01 01 00 00 00 00 00 00 00 00 00 00 .....e...
004070A0: 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
004070B0: 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 .....

```

.data section  
407030h : "Password OK"  
407040h : "Wrong Password"  
407050h : "myGOODpassword"  
407060h : "Enter Password"

### XOR—Logical Exclusive OR

Opcode	Instruction	Description
34 <i>ib</i>	XOR AL, <i>imm8</i>	AL XOR <i>imm8</i>
35 <i>iw</i>	XOR AX, <i>imm16</i>	AX XOR <i>imm16</i>
35 <i>id</i>	XOR EAX, <i>imm32</i>	EAX XOR <i>imm32</i>
80 <i>/6 ib</i>	XOR <i>r/m8</i> , <i>imm8</i>	<i>r/m8</i> XOR <i>imm8</i>
81 <i>/6 iw</i>	XOR <i>r/m16</i> , <i>imm16</i>	<i>r/m16</i> XOR <i>imm16</i>
81 <i>/6 id</i>	XOR <i>r/m32</i> , <i>imm32</i>	<i>r/m32</i> XOR <i>imm32</i>
83 <i>/6 ib</i>	XOR <i>r/m16</i> , <i>imm8</i>	<i>r/m16</i> XOR <i>imm8</i> (sign-extended)
83 <i>/6 iw</i>	XOR <i>r/m32</i> , <i>imm8</i>	<i>r/m32</i> XOR <i>imm8</i> (sign-extended)
30 <i>lr</i>	XOR <i>r/m8</i> , <i>r8</i>	<i>r/m8</i> XOR <i>r8</i>
31 <i>lr</i>	XOR <i>r/m16</i> , <i>r16</i>	<i>r/m16</i> XOR <i>r16</i>
31 <i>lr</i>	XOR <i>r/m32</i> , <i>r32</i>	<i>r/m32</i> XOR <i>r32</i>
32 <i>lr</i>	XOR <i>r8</i> , <i>r/m8</i>	<i>r8</i> XOR <i>r/m8</i>
33 <i>lr</i>	XOR <i>r16</i> , <i>r/m16</i>	<i>r16</i> XOR <i>r/m16</i>
33 <i>lr</i>	XOR <i>r32</i> , <i>r/m32</i>	<i>r32</i> XOR <i>r/m32</i>

xor eax, eax



```

33 0040101D: 50          push     eax
34 0040101E: E8 CD 00 00 00 call    004010F0
35 00401023: 83 C4 10      add     esp,10h
36 00401026: BE 50 70 40 00 mov     esi,407050h
37 0040102B: 8D 11 21 00    lea    eax,[esp+80h]
38 0040102F: 8A 10          mov     dl,byte ptr [eax]
39 00401031: 8A 1E          mov     bl,byte ptr [esi]
40 00401033: 8A CA          mov     cl,dl
41 00401035: 3A D3          cmp     dl,bl
42 00401037: 75 1E          jne    00401057
43 00401039: 84 C9          test    cl,cl
44 0040103B: 74 16          je     00401053
45 0040103D: 8A 50 01      mov     dl,byte ptr [eax+1]
46 00401040: 8A 5E 01      mov     bl,byte ptr [esi+1]
47 00401043: 8A CA          mov     cl,dl
48 00401045: 3A D3          cmp     dl,bl
49 00401047: 75 0E          jne    00401057
50 00401049: 83 C0 02      add     eax,2
51 0040104C: 83 C6 02      add     esi,2
52 0040104F: 84 C9          test    cl,cl
53 00401051: 75 DC          jne    0040102F
54 00401053: 33 C0          xor     eax,eax
55 00401055: EB 05          jmp     0040105C
56 00401057: 1B C0          sbb    eax,eax
57 00401059: 83 D8 FF      sbb    eax,0FFFFFFFh
58 0040105C: 85 C0          test    eax,eax
59 0040105E: 74 6B          je     004010CB
60 00401060: 68 40 70 40 00 push   407040h
61 00401065: E8 96 01 00 00 call    00401200
62 0040106A: 83 C4 04      add     esp,4

```

**.code section**  
 Note that 401026h uses 407050h as a parameter

Test results depend on it

Lead to a correct password

Lead to an incorrect password

```

94 004010C2: 1B C0          sbb    eax,eax
95 004010C4: 83 D8 FF      sbb    eax,0FFFFFFFh
96 004010C7: 85 C0          test   eax,eax
97 004010C9: 75 95          jne    00401060
98 004010CB: 68 30 70 40 00 push   407030h
99 004010D0: E8 2B 01 00 00 call    00401200
100 004010D5: 83 C4 04      add     esp,4
101 004010D8: 5F           pop     edi
102 004010D9: 5E           pop     esi

```

**.code section**  
 Note that 4010CBh uses 407030h as a parameter



# Observation

```
58 0040105C: 85 C0          test     eax,eax
59 0040105E: 74 6B          je      004010CB
60 00401060: 68 40 70 40 00  push   407040h
```

- 0040105C : test eax, eax
  - Tests if **eax** is equal to zero
  - If it's zero, then jumps to the correct output message display function
  - Otherwise, if eax is non-zero, it leads to a wrong output message
- What if we change **je** it to **jne** ?
  - All the incorrect passwords except for the correct one will be accepted
- What if we change **test** it to **xor** ?
  - **xor eax, eax** is always zero. Therefore, any password will be accepted



# Modify the Code!

```
00401030: 10 8A 1E 8A CA 3A D3 75 1E 84 C9 74 16 8A 50 01  >èÀè::UuÄärt-èP@
00401040: 8A 5E 01 8A CA 3A D3 75 0E 83 C0 02 83 C6 02 84  è^@è::UuÈÈLèèLèè
00401050: C9 75 DC 33 C0 EB 05 1B C0 83 D8 FF 35 C0 74 6B  ru_3'5è+Lâ+ 3Ltk
00401060: 68 40 70 40 00 E8 96 01 00 00 83 C4 04 47 83 FF  h0p0 00@ â-âGâ
00401070: 03 7F 6C 68 60 70 40 00 E8 83 01 00 00 68 78 70  ↓Δlh`p0 0â@ hxp
54 00401053: 33 C0          xor     eax,eax
55 00401055: EB 05          jmp     0040105C
56 00401057: 1B C0          sbb    eax,eax
57 00401059: 83 D8 FF      sbb    eax,0FFFFFFFh
58 0040105C: 85 C0          test   eax,eax
59 0040105E: 74 6B          je     0040106B
60 00401060: 68 40 70 40 00  push  407040h
61 00401065: E8 96 01 00 00  call  00401200
62 0040106A: 83 C4 04      add    esp,4
```

- Replace **test eax, eax** with **xor eax, eax**
  - Opcode for test : 85
  - Opcode for xor : 33
  - Use HIEW for changing the hex values of the binary executable

```
00001030: 10 8A 1E 8A CA 3A D3 75 1E 84 C9 74 16 8A 50 01  >èÀè::UuÄärt-èP@
00001040: 8A 5E 01 8A CA 3A D3 75 0E 83 C0 02 83 C6 02 84  è^@è::UuÈÈLèèLèè
00001050: C9 75 DC 33 C0 EB 05 1B C0 83 D8 FF 33 C0 74 6B  ru_3'5è+Lâ+ 3Ltk
00001060: 68 40 70 40 00 E8 96 01 00 00 83 C4 04 47 83 FF  h0p0 00@ â-âGâ
00001070: 03 7F 6C 68 60 70 40 00 E8 83 01 00 00 68 78 70  ↓Δlh`p0 0â@ hxp
00001080: 40 00 8D 44 24 14 6A 64 50 E8 62 00 00 00 83 C4  è ìD$ŋjdp0b â-
0040104F: 84 C9          test   cl,cl
00401051: 75 DC          jne    0040102F
00401053: 33 C0          xor     eax,eax
00401055: EB 05          jmp     0040105C
00401057: 1B C0          sbb    eax,eax
00401059: 83 D8 FF      sbb    eax,0FFFFFFFh
0040105C: 33 C0          xor     eax,eax
0040105E: 74 6B          je     0040106B
00401060: 68 40 70 40 00  push  407040h
00401065: E8 96 01 00 00  call  00401200
0040106A: 83 C4 04      add    esp,4
0040106D: 47            inc    edi
```

# After Modification of the Code

```
jlee@sony2 ~/Program/Hacker/1_SimpleAuthentication/Release
$ ./1_SimpleAuthentication.exe
Enter Password : asd
Password OK

jlee@sony2 ~/Program/Hacker/1_SimpleAuthentication/Release
$ ./1_SimpleAuthentication.exe
Enter Password : s
Password OK

jlee@sony2 ~/Program/Hacker/1_SimpleAuthentication/Release
$ ./1_SimpleAuthentication.exe
Enter Password : sd
Password OK

jlee@sony2 ~/Program/Hacker/1_SimpleAuthentication/Release
$ ./1_SimpleAuthentication.exe
Enter Password : myGOODpassword
Password OK
```

```
jlee@sony2 ~/Program/Hacker/1_SimpleAuthentication/Release
$ /cygdrive/c/WINDOWS/system32/fc 1_SimpleAuthentication.exe 1_SimpleAuthentication_Modified.exe
Comparing files 1_SimpleAuthentication.exe and 1_SIMPLEAUTHENTICATION_MODIFIED.EXE
0000105C: 85 33
```

Shows the difference between two binary executables (test vs. xor)



# Assignment #1

- A simple serial number based authentication scheme.
- Tasks
  - Modify the program so that it bypasses the authentication mechanism (ie. accepts any serial).
  - Find the correct serial.

