Operating Systems Winter 2009

Module 17 Authentication / Authorization / Security

Mark Zbikowski
Gary Kimura

Terminology I: the entities

- Principals – who is acting?
  - User / Process Creator
  - Code Author
- Objects – what is that principal acting on?
  - File
  - Network connection
- Rights – what actions might you take?
  - Read
  - Write
- Familiar Windows file system example:
  - Guest / user / CSE451
  - read / write / append / enumerate

Terminology II: the activities

- Authentication – who are you?
  - identifying principals (users / programs)
- Authorization – what are you allowed to do?
  - determining what access users and programs have to specific objects
- Auditing – what happened
  - record what users and programs are doing for later analysis / prosecution

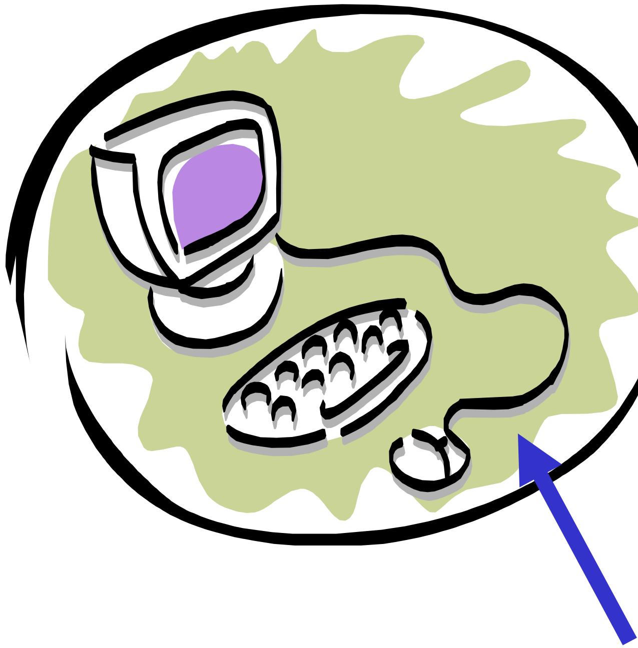
Authentication

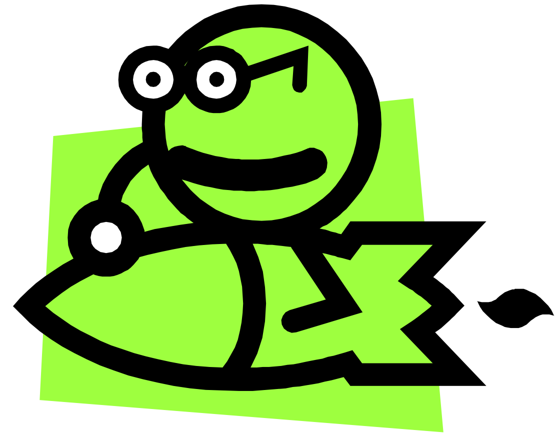
- How does the provider of a secure service know who it's talking with?
  - Example: WinLogon

- We'll start with the local case (the keyboard is attached to the machine you want to login to)

- Then we'll look at a distributed system

Local Login

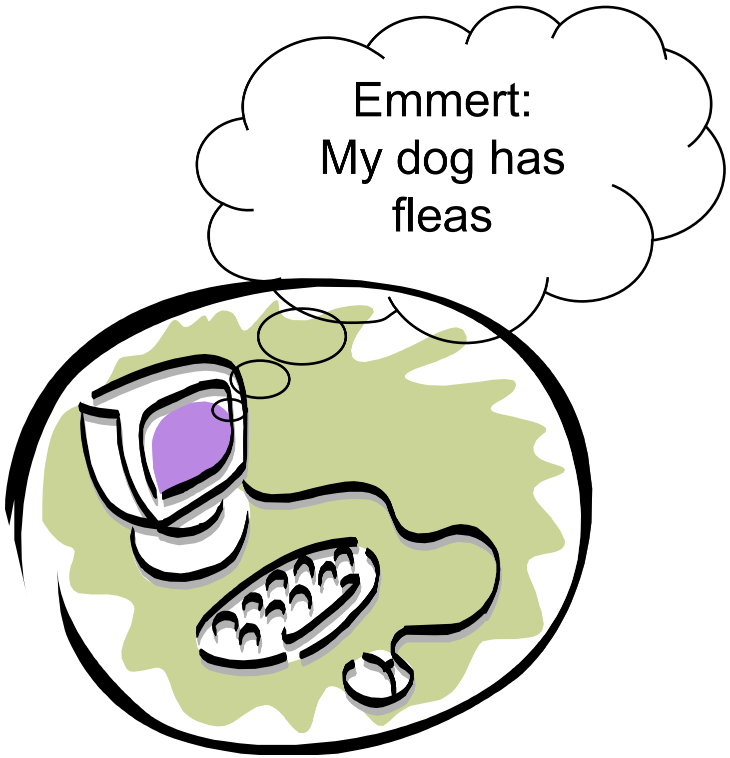




(“Local” ⇒ this connection is assumed secure)

How does the OS know that I’m ‘emmert’?

Shared Secret



The shared secret is typically a password, but it could be something else:

- Retina scan
- A key

Simple Enough

- This seems pretty trivial
- Like pretty much all aspects of security, there are perhaps unexpected complications
- As an introduction to this, let’s look at briefly at the history of password use

Storing passwords

- CTSS (1962): password file {user name, user identifier, password}

Bob, 14, “12.14.52”
David, 15, “allison”
Mary, 16, “!ofotc2n”

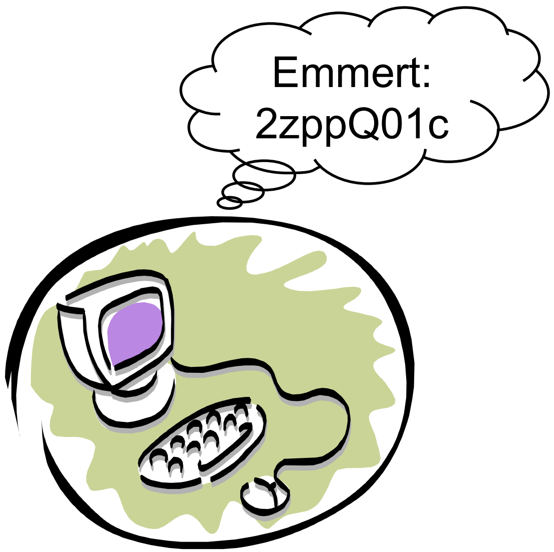
If a bad guy gets hold of the password file, you’re in deep trouble

– **Any** flaw in the system that compromises the password file compromises all accounts!

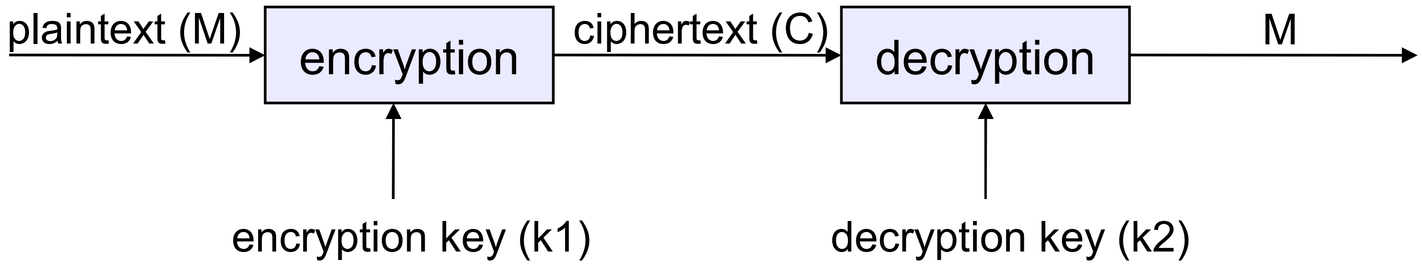
Two Choices

1. Make sure there are no flaws in the system (ha!)
2. Render knowledge of the password file useless

Unix (1974): store encrypted forms of the passwords



Aside on Encryption



- Encryption: takes a key and plaintext and creates ciphertext: $E_{k1}(M) = C$
- Decryption: takes ciphertext and a key and recovers plaintext: $D_{k2}(C) = M$

- Symmetric algorithms (aka secret-key aka shared secret algorithms):
  - k1 = k2 (or can get k2 from k1)
- Public-Key Algorithms
  - decryption key (k2) cannot be calculated from encryption key (k1)
  - encryption key can be made public!
    - encryption key = “public key”, decryption key = “private key”

- Computational requirements:
  - Deducing M from $E_k(M)$ is “really hard”
  - Computing $E_k(M)$ and $D_k(C)$ is efficient

Unix Password File

- Encrypt passwords with passwords

K=[alison]$_{\text{allison}}$

Bob: 14: S6Uu0cYDVdTAk
David: 15: J2ZI4ndBL6X.M
Mary: 16: VW2bqvTalBJKg

- David's password, "allison," is encrypted using itself as the key and stored in that form.
- Password supplied by user is encrypted with itself as key, and result compared to stored result.
- "No problem if someone steals the file"
- Also no need to secure a key

Windows Passwords

- NTLM – run user name and password through “secure hash”: SHA4, MD4/5 to map to 128-bit “digest”. “Cryptographically secure”
- Store user name and digest.
- Lose the password file, no problem
  - Uh, er, with large enough input buffer algorithms exist to create a fake password that has same hash. Solution: limit input buffer size. Sorta ok…

The Dictionary Attack

- Encrypt many (all) possible password strings offline, and store results in a dictionary
  - I may not be able to invert any particular password, but the odds are very high I can invert one or more
- 26 letters used, 7 letters long
  - 8 billion passwords (33 bits)
  - Generating 100,000/second requires 22 hours
- But most people's passwords are not random sequences of letters!
  - girlfriend's/boyfriend's/spouse's/dog's name/words in the dictionary
- Dictionary attacks have traditionally been incredibly easy

Making it harder

- Using symbols and numbers and longer passwords
  - 95 characters, 14 characters long
  - 10^{27} passwords = 91 bits
  - Checking 100,000/second breaks in 10^{14} years
- Require frequent changing of passwords
  - guards against loaning it out, writing it down, etc.
  - Avoid algorithmic passwords or recycling from long list
    - Microsoft retains last 18 passwords. Sorta stops "ThisIsMy1stPassword", "ThisIsMy2ndPassword"…

Do longer passwords work?

- People can’t remember 14-character strings of random characters
- People write down difficult passwords
- People give out passwords to strangers
- Passwords can show up on disk
- If you are forced to change your password periodically, you probably choose an even dumber one
  - “feb04” “mar04” “apr04”
- How do we handle this in CSE?

Attack Models

- Besides the problems already mentioned that obviously remain (people give out their passwords / write them down / key loggers / …), there may be other clever attacks that we haven't thought of

- Attack Model: when reasoning about the security of a mechanism, we typically need to carefully describe what kinds of attacks we're thinking of
  - helps us reason about what vulnerabilities still remain

Example 1: Login spoofers

- Login spoofers are a specialized class of Trojan horses
  - Attacker runs a program that presents a screen identical to the login screen and walks away from the machine
  - Victim types password and gets a message saying “password incorrect, try again”
- Can be circumvented by requiring an operation that unprivileged programs cannot perform
  - E.g., start login sequence with a key combination user programs cannot catch, CTRL+ALT+DEL on Windows
- False fronts have been used repeatedly to steal bank ATM passwords!

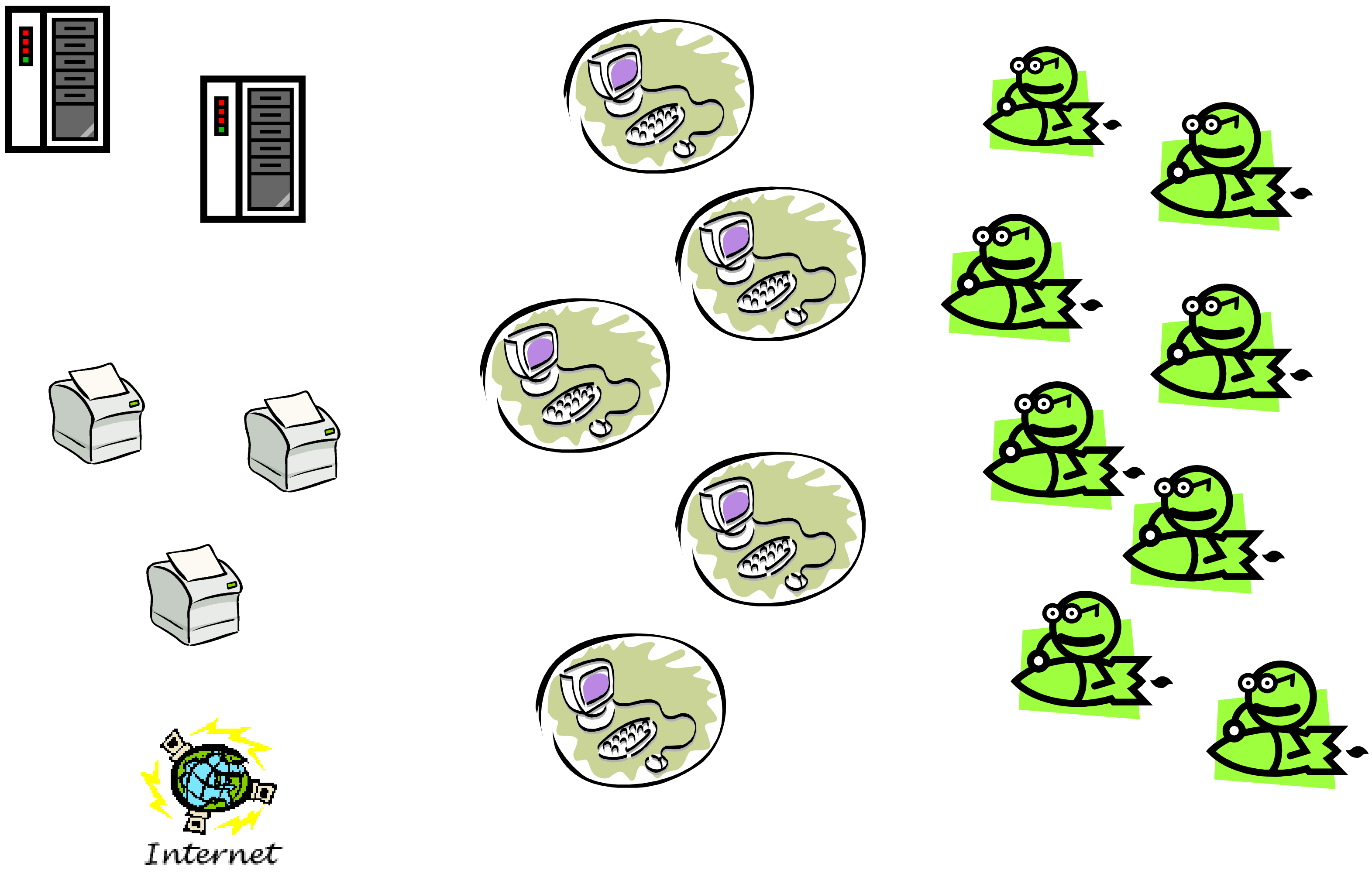
Example 2: Page faults as a signal

- VMS (early 80's) password checking flaw
  - password checking algorithm:

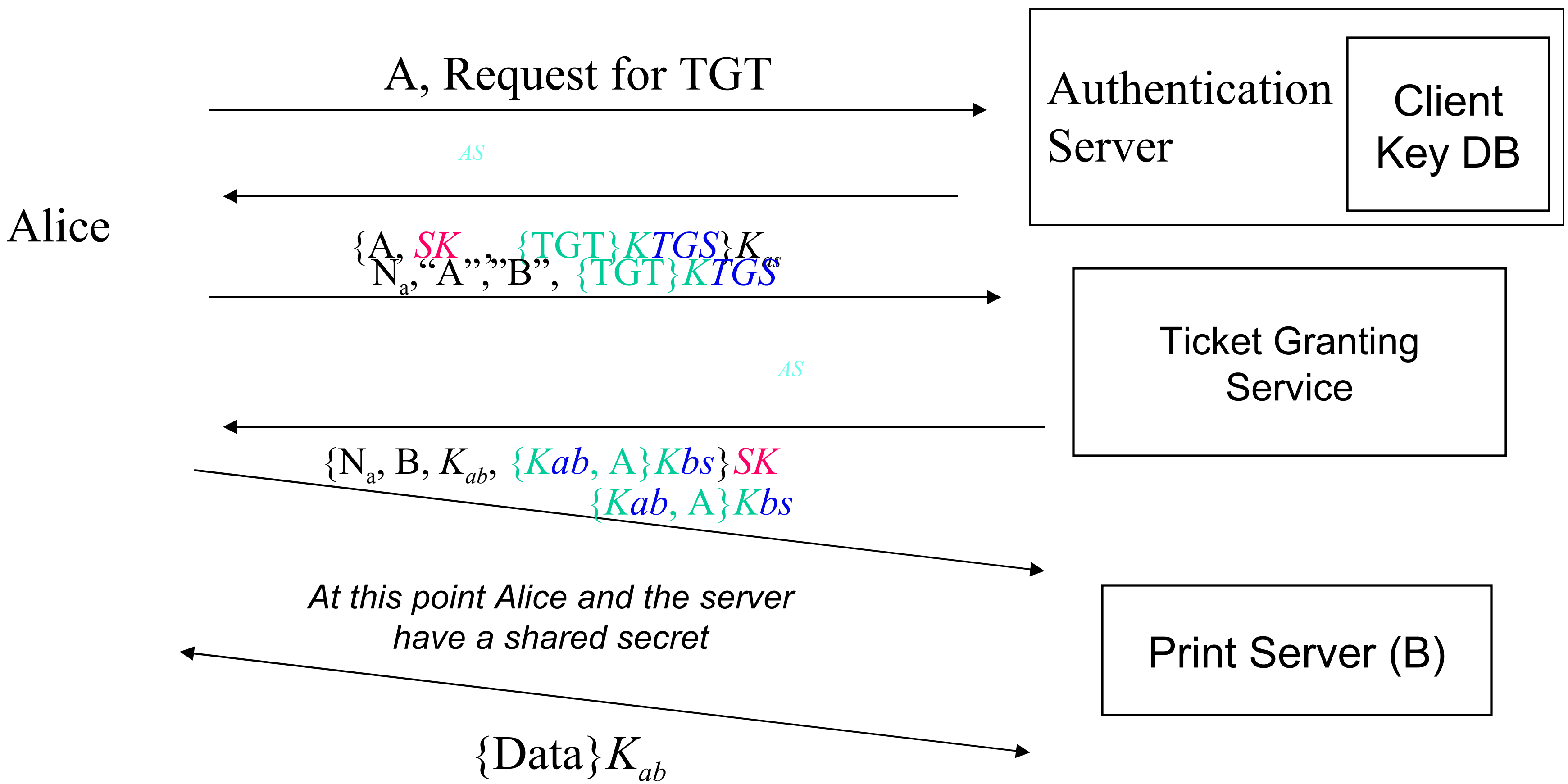
```
for (I=0; I<password.length( ); I++) {
    if password[I] == supplied_password[I]
        return false;
}
return true;
```

  - can you see the problem?
    - hint: think about virtual memory…
    - another hint: think about page faults…
    - final hint: who controls where in memory supplied_password lives?

Distributed Authentication (Single Domain)



Kerberos


A, Request for TGT
Authentication Server
Client Key DB
Alice
{A, SK, {TGT}KTGS}K
N, "A", "B", {TGT}KTGS
Ticket Granting Service
{N, B, K, {Kab, A}Kbs}SK
{Kab, A}Kbs
At this point Alice and the server have a shared secret
Print Server (B)
{Data}K

Trust Relationships

- Both Alice and the server must trust the Kerberos servers ("trusted third party")
- This architecture is essentially what Microsoft passport is:

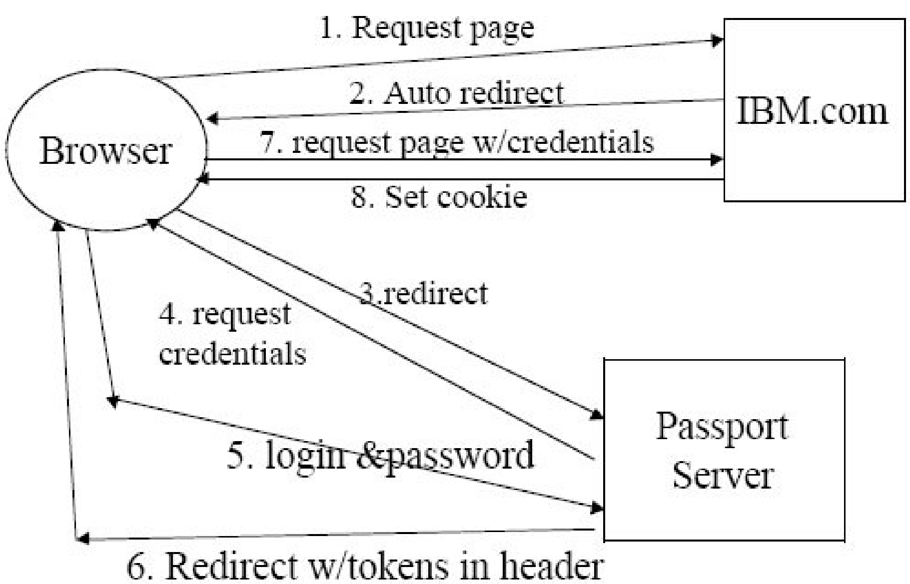
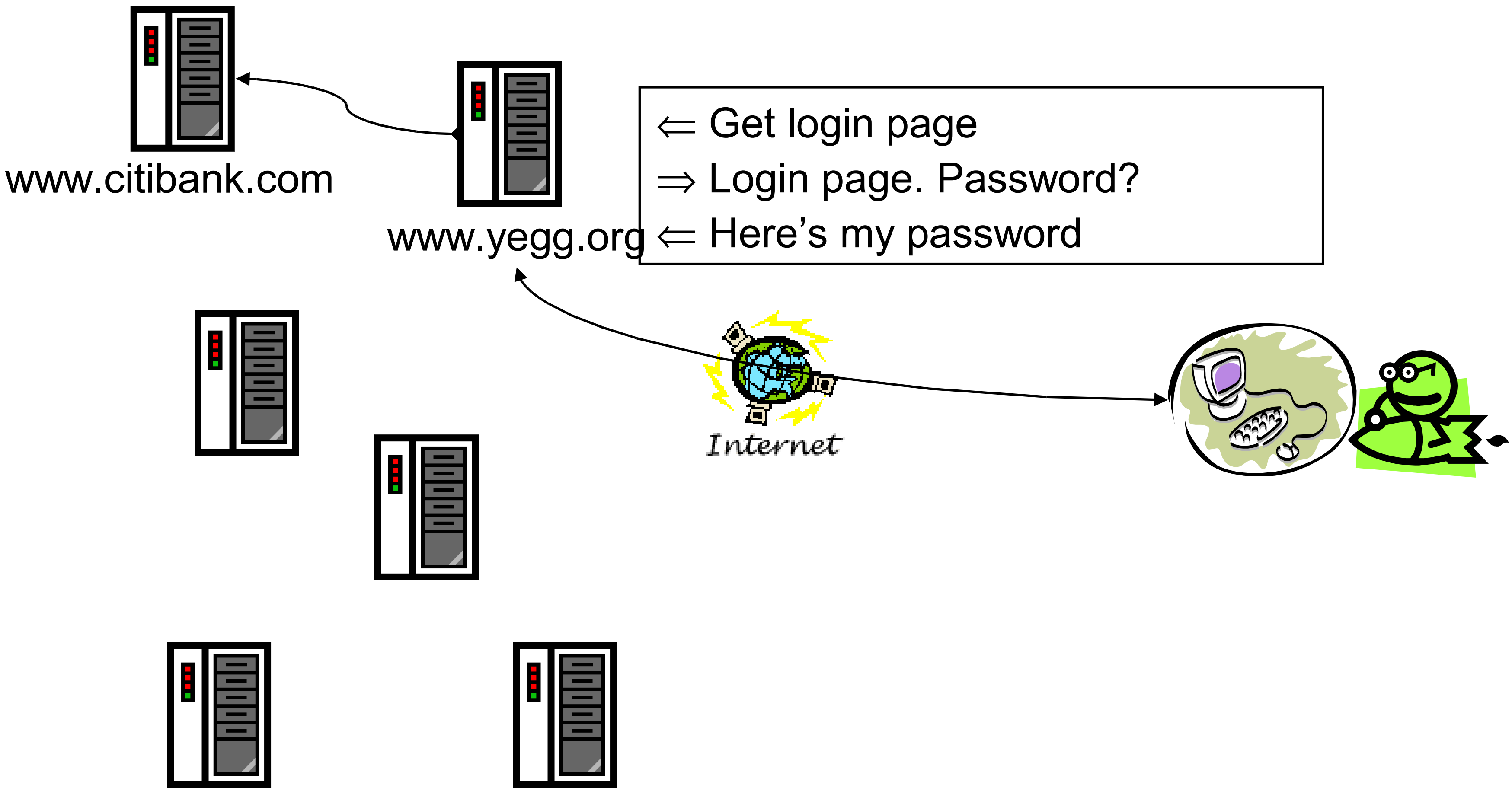


Figure 1. The Passport architecture.

Distributed Authentication at World Scale

- Bill Gates wants to login to his Citibank account to move $10 from savings to checking
- Both Bill and Citibank are worried:
  - Citibank:
    - How do I know that I'm talking with Bill?
    - Does Bill have $10 in his savings account?
    - …
  - Bill:
    - How do I know that I'm talking with Citibank?

Man in the Middle Attack

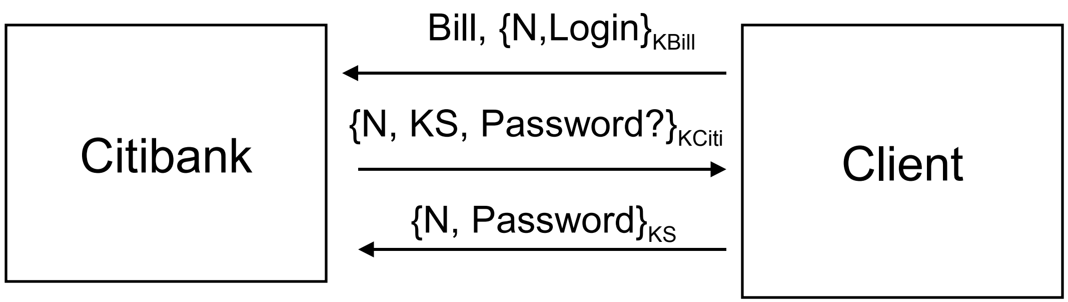


Authentication Solutions

- Citibank authenticating Bill
  - This is just a client accessing a server. Citibank can use shared secrets.
    - Bill has to use some secret communicated out-of-band (e.g., ATM PIN number) to create a shared secret for online access.

- Bill authenticating Citibank
  - Could shared secret work for the bank to authenticate itself to the client?
    - …
  - In the end, we rely on a trusted third party (just like Kerberos, but implemented differently)

Why not this?


Citibank
Client
Bill, {N,Login}KBill
{N, KS, Password?}KCiti
{N, Password}KS

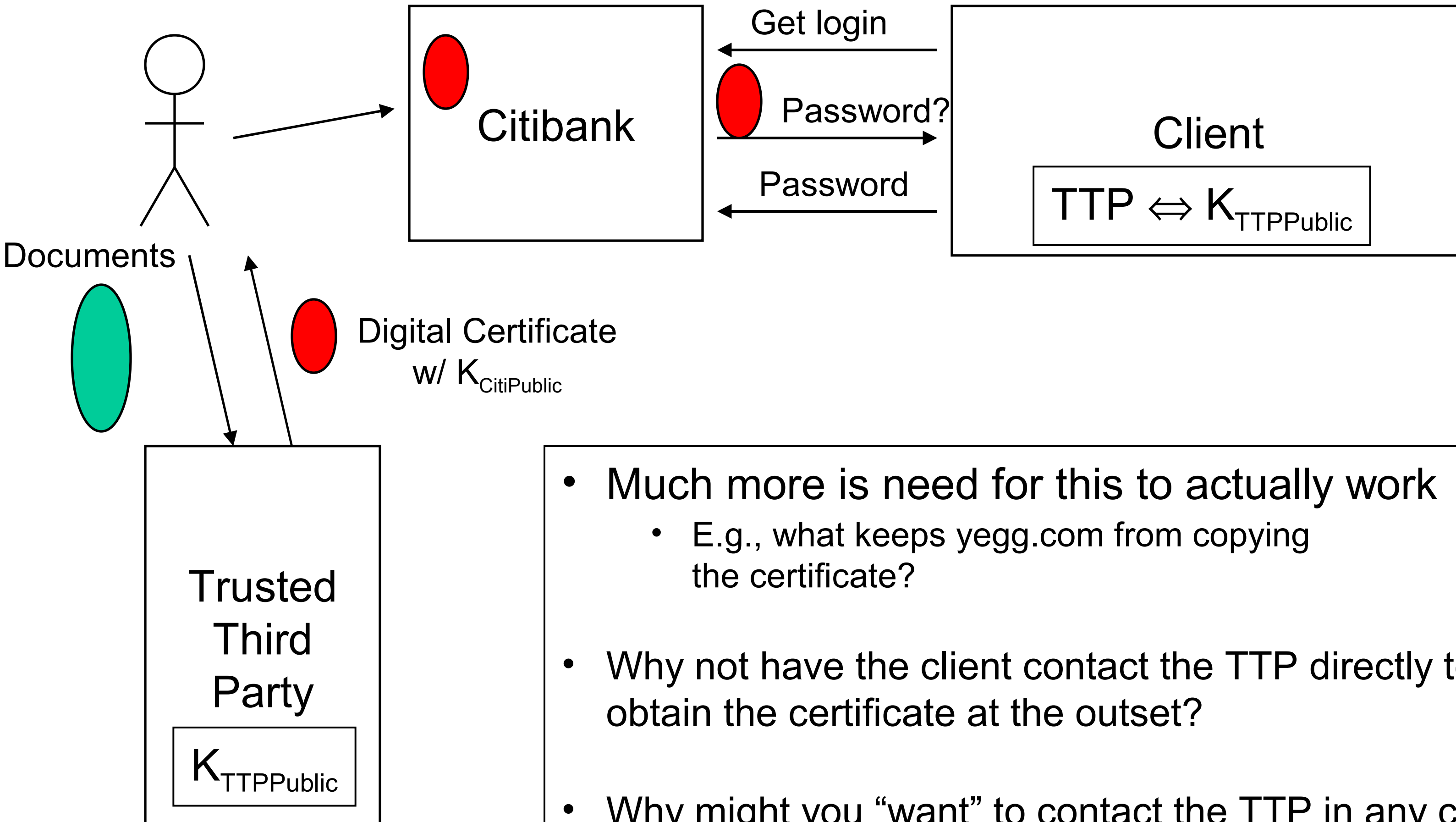
Public Key Encryption

- Key pairs, KPublic / KPrivate
  - $\{\{M\}_{KPublic}\}_{KPrivate} = \{\{M\}_{KPrivate}\}_{KPublic} = M$
    - Each key is the decryption key for the other used as an encryption key
  - It is computationally infeasible to deduce KPrivate from KPublic
    - You can distribute KPublic freely

- $\{M\}_{KPublic}$ can be decrypted only by the holder of the private key

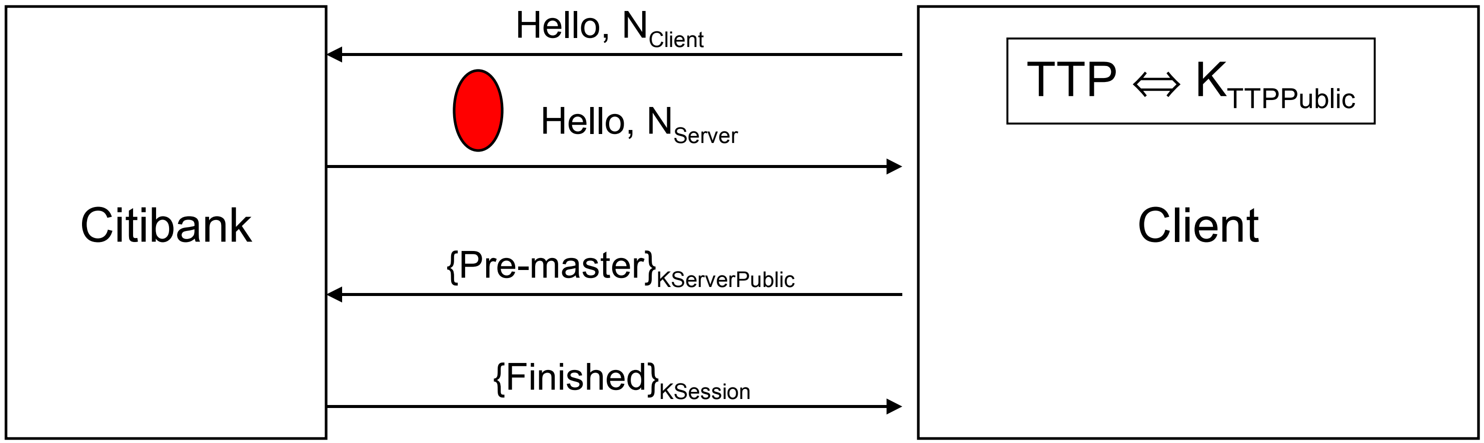
- $\{M\}_{KPrivate}$ can be created only by the holder of the private key
  - “Signing”

Authentication by Certificate: Basic Idea



- Much more is need for this to actually work
  - E.g., what keeps yegg.com from copying the certificate?
- Why not have the client contact the TTP directly to obtain the certificate at the outset?
- Why might you “want” to contact the TTP in any case?

Client/Server Communication: ssl (tls)



Notes:

1. Master/session key determined independently by both client and server as:

   $F(N_{client}, N_{server}, \text{Pre-master})$

2. I've taken some liberties to simplify the explanation… (cf. CSE 461)

The Larger Security Problem

- Integrity
  My data should be protected against modification by malicious parties
  - "Modification" includes deletion

- Privacy
  My data should not be disclosed without my consent

- Both issues have become much more complicated in the last decade
  - Attackers exploit bugs/weaknesses accessible through the net
  - We all run third-party code

Spyware

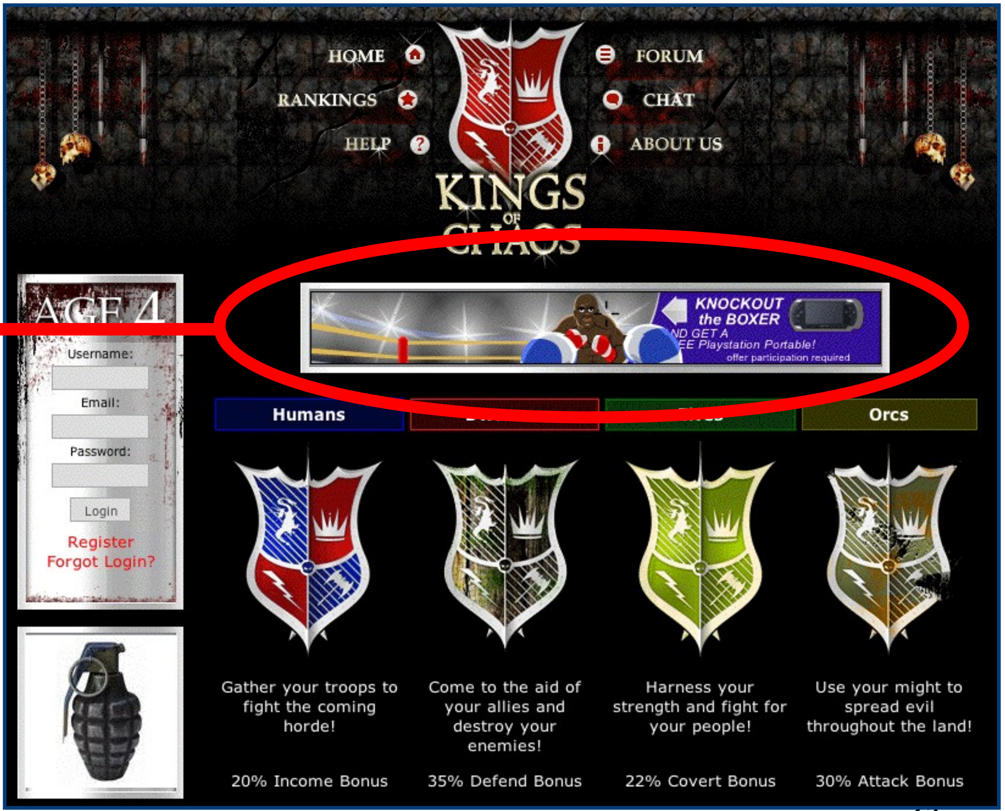
- Software that is installed that collects information and reports it to third party
  - key logger, adware, browser hijacker, …
- Installed one of two ways
  - piggybacked on software you choose to download
  - “drive-by” download
    - your web browser has vulnerabilities
    - web server can exploit by sending you bad web content
- Estimates
  - majority (50-90%) of Internet-connected PCs have it
  - 1 in 20 executables on the Web have it
  - about 0.5% of Web pages attack you with drive-by-downloads

kingsofchaos.com

- A benign web site for an online game
  - earns revenue from ad networks by showing banners
  - but, it relinquishes control of the ad content

banner ad from adworldnetwork.com (a legitimate ad network)

inline javascript loads HTML from ad provider



Incident

- kingsofchaos.com was given this “ad content”

```
<script type="text/javascript">document.write(‘ \u003c\u0062\
u006f\u0064\u0079\u0020\u006f\u006e\u0055\u006f\u0077\u0050\u
006f\u0070\u0075\u0070\u0028\u0029\u003b\u0073\u0068\u006f\u0
077\u0048\u0069 …etc.
```

- This “ad” ultimately:
  - bombarded the user with pop-up ads
  - hijacked the user’s homepage
  - exploited an IE vulnerability to install spyware

What’s going on?

- The advertiser was an ex-email-spammer
- His goal:
  - **force** users to see ads from his servers
  - **draw revenue** from ad “affiliate programs”
    - Apparently earned several millions of dollars
- Why did he use spyware?
  - control PC and show ads even when not on the Web

Principle of Least Privilege

- Figure out exactly which capabilities a program needs to run, and grant it only those
  - start out by granting none
    - run program, and see where it breaks
    - add new privileges as needed.

- Unix: concept of root is not a good example of this
  - some programs need root just to get a small privilege
    - e.g., FTP daemon requires root:
      - to listen on network port < 1024
      - to change between user identities after authentication
    - but root also lets you read any file in filesystem

Principle of Complete Mediation

- Check **every** access to **every** object
  - in rare cases, can get away with less (caching)
    - but only if sure nothing relevant in environment has changed… and there is a lot that's relevant!
- A TLB caches access control information
  - page table entry protection bits
  - is this a violation of the principle?

Modern security problems

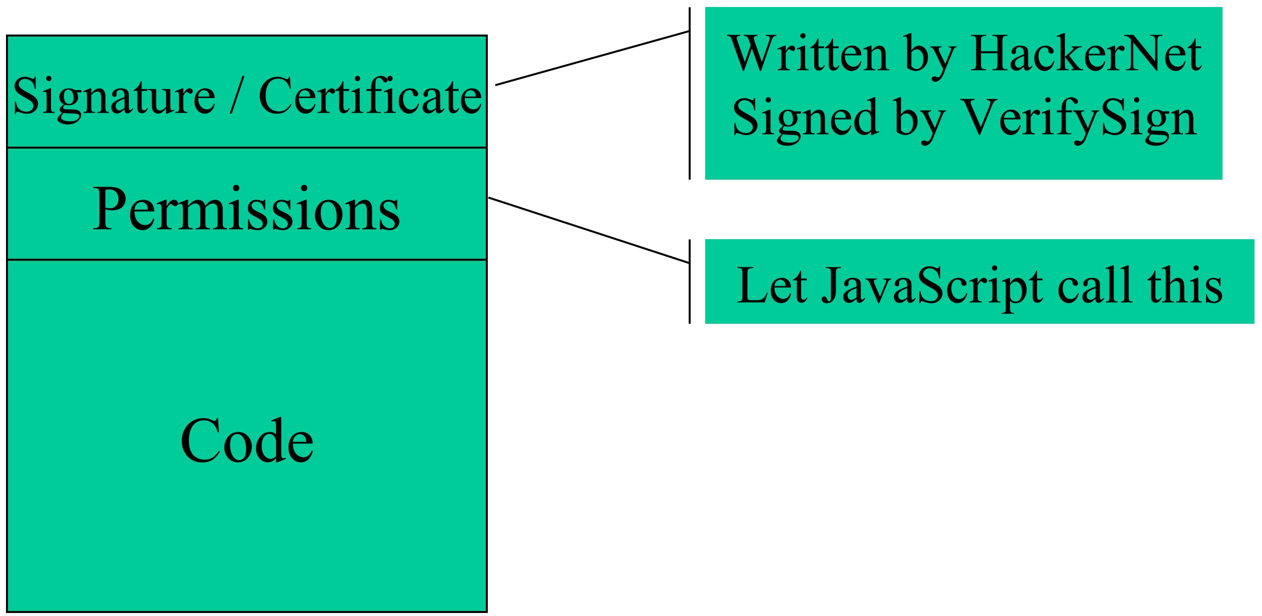
- Confinement
  - How do I run code that I don’t trust?
    - e.g., RealPlayer, Flash
  - How do I restrict the data it can communicate?
  - What if trusted code has bugs?
    - e.g., Internet Explorer
- Solutions
  - Restricted contexts – let the user divide their identity
  - ActiveX – make code writer identify self
  - Java – use a virtual machine that intercepts all calls
  - Binary rewriting – modify the program to force it to be safe

Restricted contexts

- Role-based access control (RBAC)
  - Add extra identity information to a process
    - e.g., both username and program name (mikesw:navigator)
  - Use both identities for access checks
    - add extra security checks at system calls that use program name
    - add extra ACLs on objects that grant/deny access to the program
  - Allows users to sub-class themselves for less-trusted programs
- `chroot`
- Browse in a VMWare machine

ActiveX

- All code comes with a public-key signature
- Code indicates what privileges it needs
- Web browser verifies certificate
- Once verified, code is completely trusted



Java / C#

- All problems are solved by a layer of indirection
  - All code runs on a virtual machine
  - Virtual machine tracks security permissions
  - Allows fancier access control models - allows stack walking

- Interposition using language VM doesn't work for other languages

- Virtual machines can be used with all languages
  - Run virtual machine for hardware
  - Inspect stack to determine *subject* for access checks

Binary rewriting

- Goal: enforce code safety by *embedding* checks in the code
- Solution:
  - Compute a mask of accessible addresses
  - Replace system calls with calls to special code

Original Code:

```
lw   $a0, 14($s4)
jal  ($s5)
move $a0, $v0
jal $printf
```

Rewritten Code:

```
and $t6,$s4,0x001fff0
lw   $a0, 14($t6)
and  $t6,$s5, 0x001fff0
jal  ($t6)
move $a0, $v0
jal $sfi_printf
```