Computer Systems Security
CSE 628A

Pramod Subramanyan
Indian Institute of Technology Kanpur

ADMINISTRIVIA

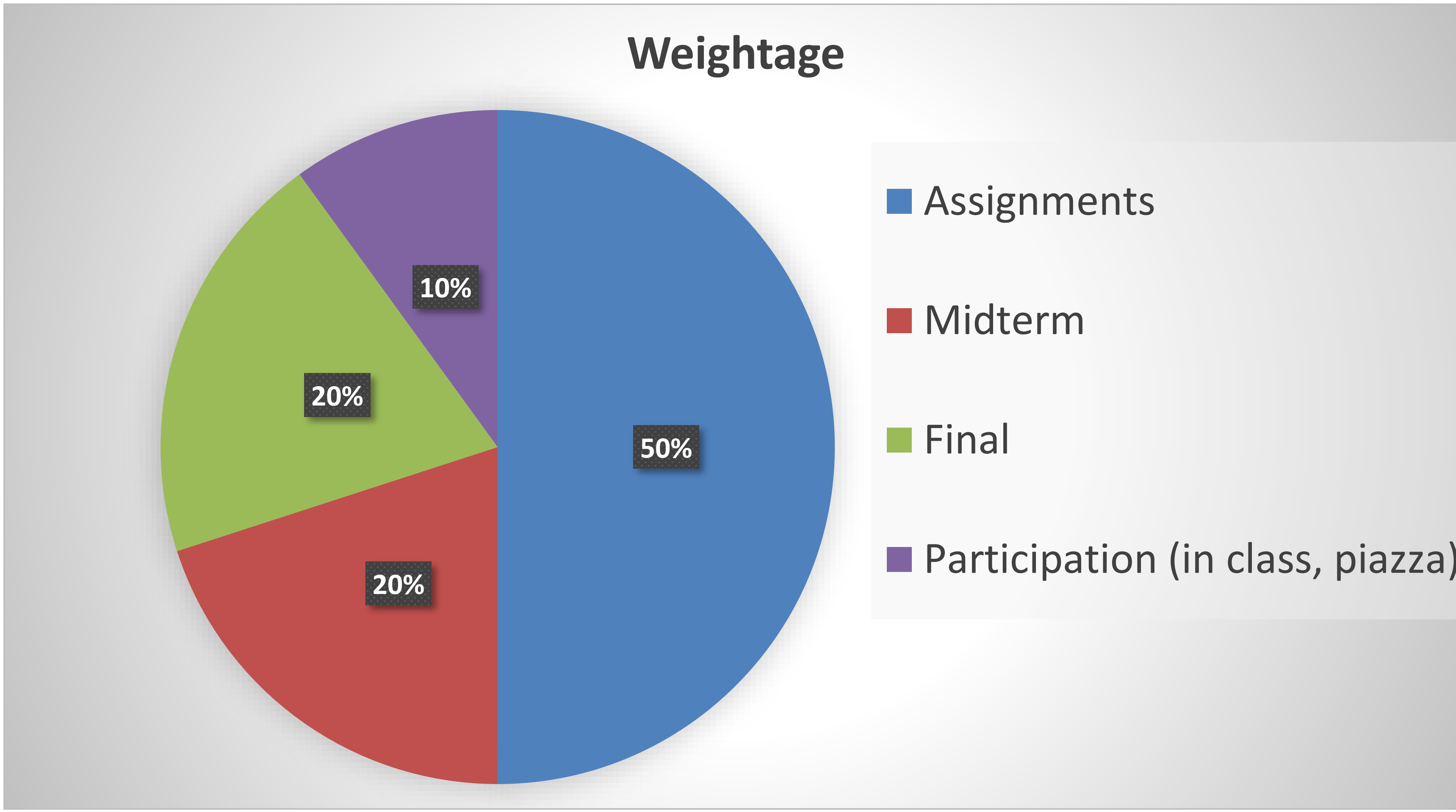
Team

- Instructor: Pramod Subramanyan (spramod@cse)
- Teaching Assistants:
  - Bidya Sarkar (bidya@)
  - Dixit Kumar (dixit@)
  - Hariom (hariom@)
  - Mayank Rawat (mayankr@)
  - M. Jeevan Kumar (jeevank@)
  - Nirjhar Roy (nirjhar@)
  - Krishna Kumar Tayal (ktayal@)
  - Supriya Suresh (ssuresh@)

Links

- Piazza signup link:
  - piazza.com/iitk.ac.in/firstsemester2019/cs628
- Moodle course:
  - We will add you by the end of the week
  - Assignments will be posted on moodle
- Course webpage:
  - Slides and readings will be posted under schedule
  - https://web.cse.iitk.ac.in/users/spramod/courses/cs628-2019-I/

Grading



Weightage
10%
20%
50%
20%
Assignments
Midterm
Final
Participation (in class, piazza)

Syllabus

- Module 0: Introduction and Review (5%)
- Module 1: Software and Systems Security (35%)
- Module 2: Web Security (25%)
- Module 3: Network Security (25%)
- Module 4: Hardware Security (10%)

Expectations/Advice

1. Ensure you have the background knowledge
2. Come to the classes and participate
3. Study after each class
4. Post and answer questions on piazza
5. Please do the the readings
6. Start the homeworks early

In the long run, learning will pay-off over prioritizing grade maximization

Background/Preparation

- Computer organization (assembly language, TLBs, demand paging, privilege separation)
- Operating systems (processes, threads, heaps, stacks, page tables, permissions, etc.)
- Networks (IP, TCP, UDP, HTTP, DNS etc.)
- Read, write and understand programs; reason carefully about the difference between programmer intent and code behaviour

Module 0: Introduction

Context and Landscape

Acknowledgements

- Sandeep Shukla (IIT Kanpur)
- Arvind Narayanan (Princeton University)
- Dan Boneh (Stanford University)
- John C. Mitchell (Stanford University)
- Nicolai Zeldovich (MIT)
- Jungmin Park (Virginia Tech)
- Patrick Schaumont (Virginia Tech)
- Web Resources

The modern computer security problem

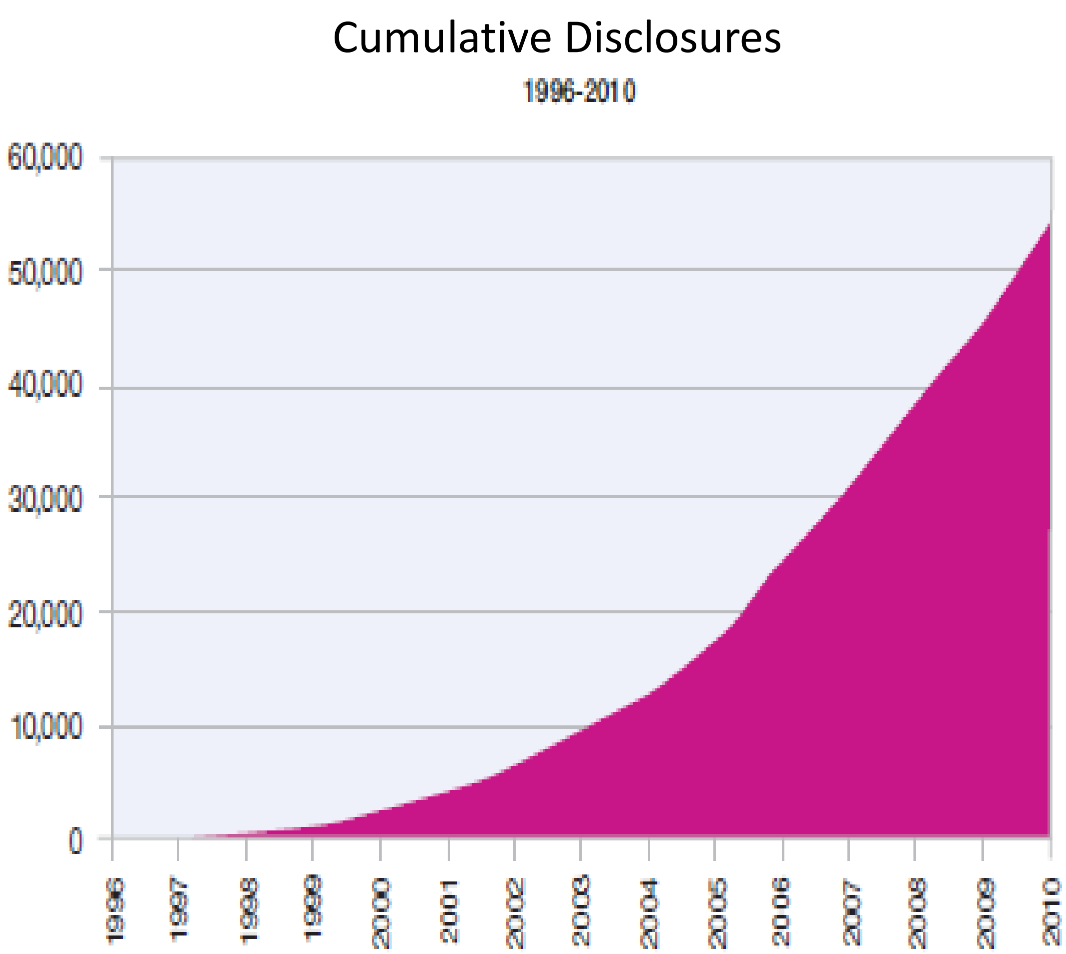
Two factors:

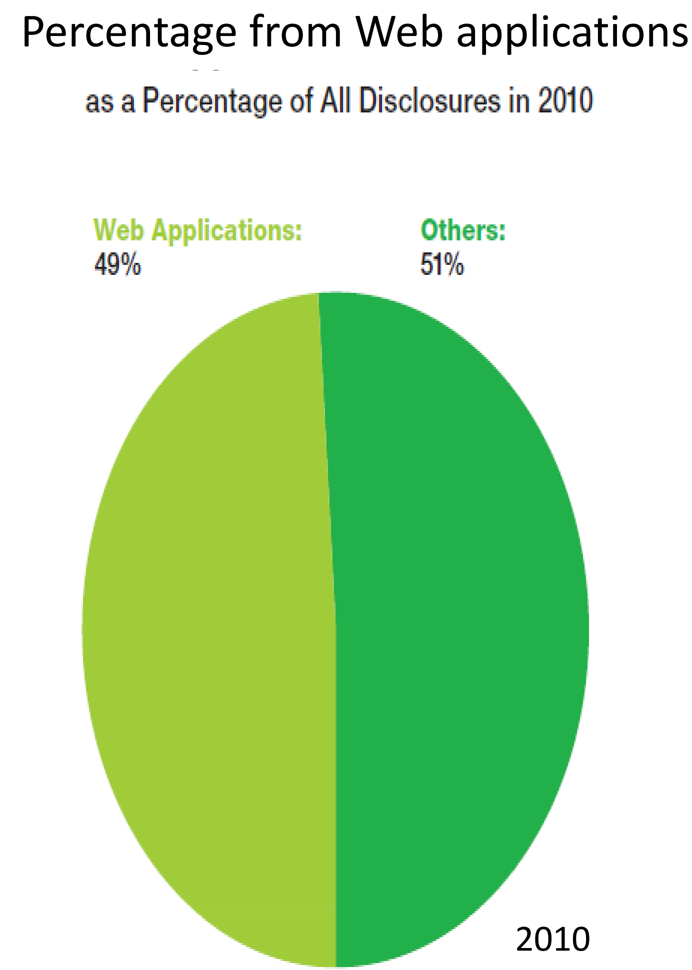
- **Lots of buggy software** (and gullible users)
- **Money can be made from finding and exploiting vulnerabilities**

1. Marketplace for vulnerabilities
2. Marketplace for owned machines
3. Methods to profit from owned client machines

current state of computer security

MITRE tracks vulnerability disclosures

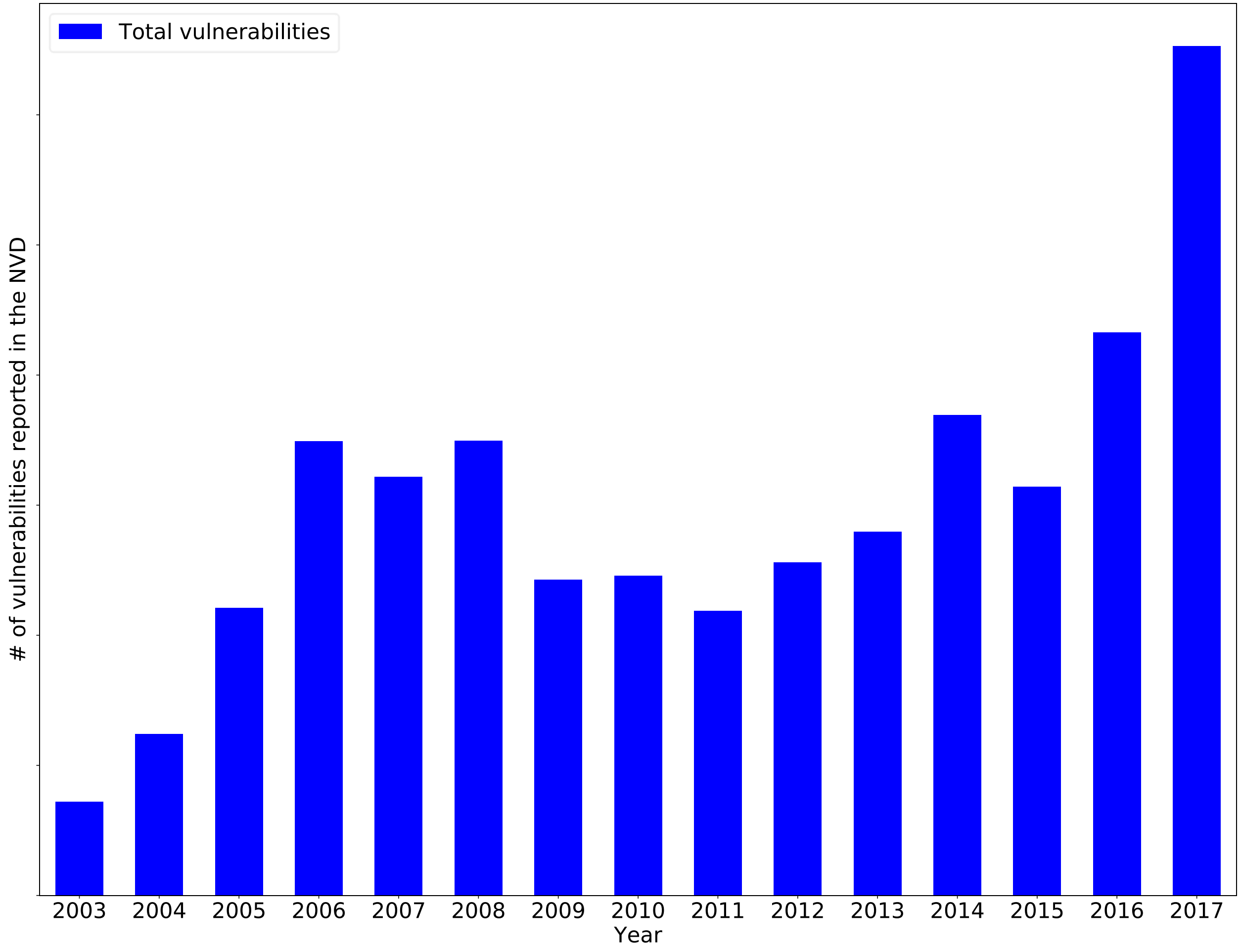




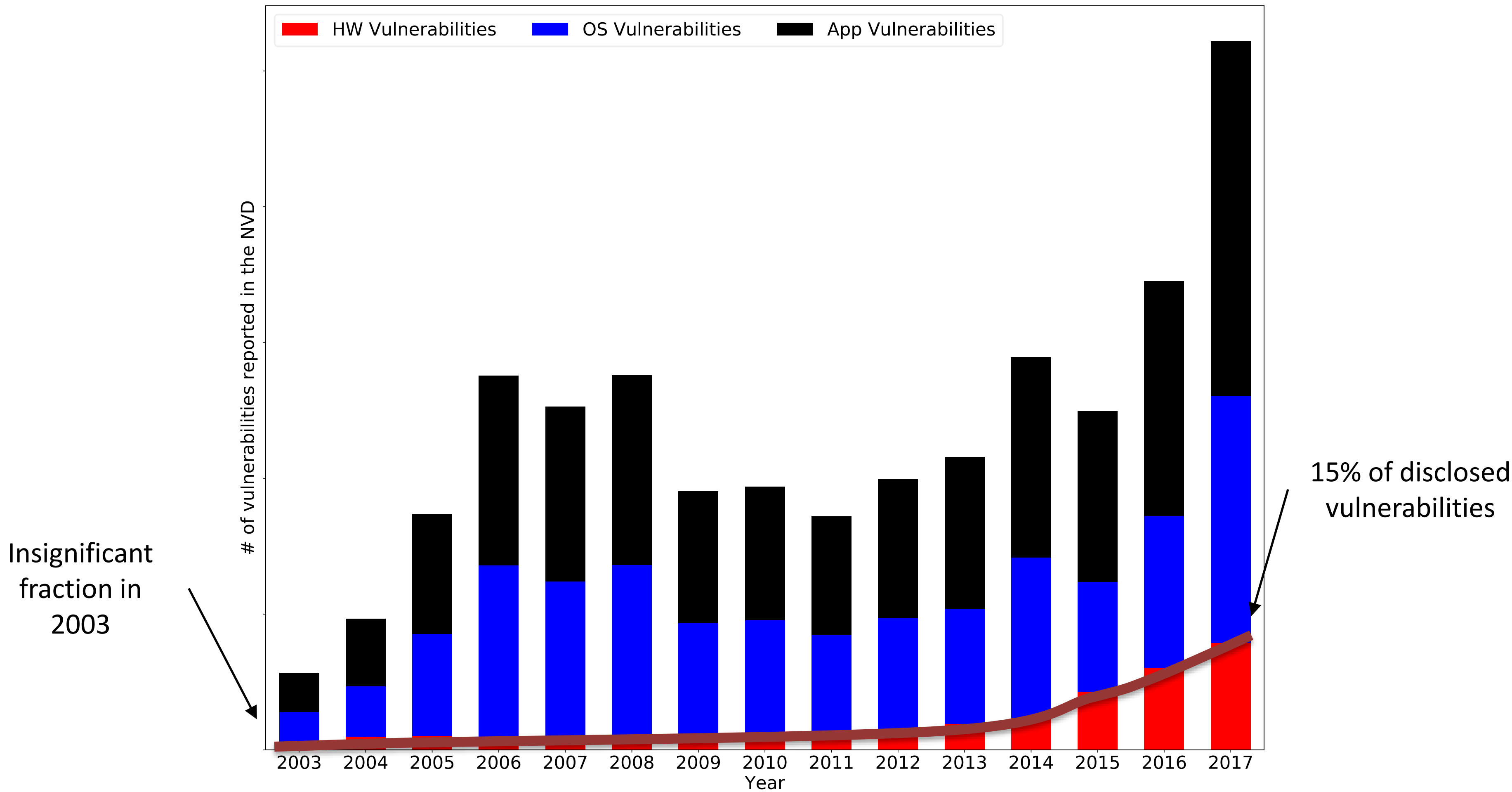
Source: IBM X-Force, Mar 2011

Data: http://cve.mitre.org/

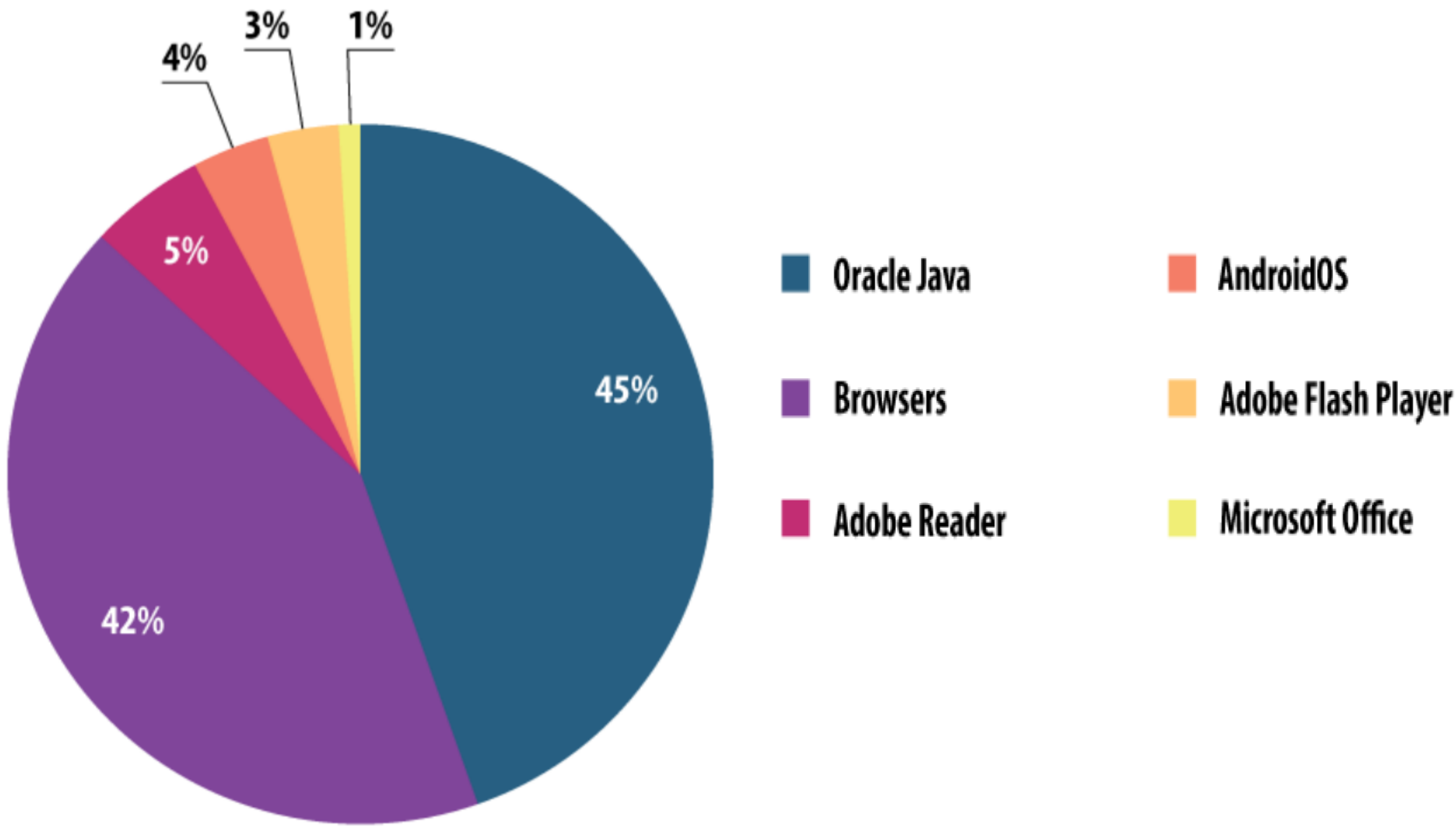
CVEs in the NVD



Data shows HW vulns growing

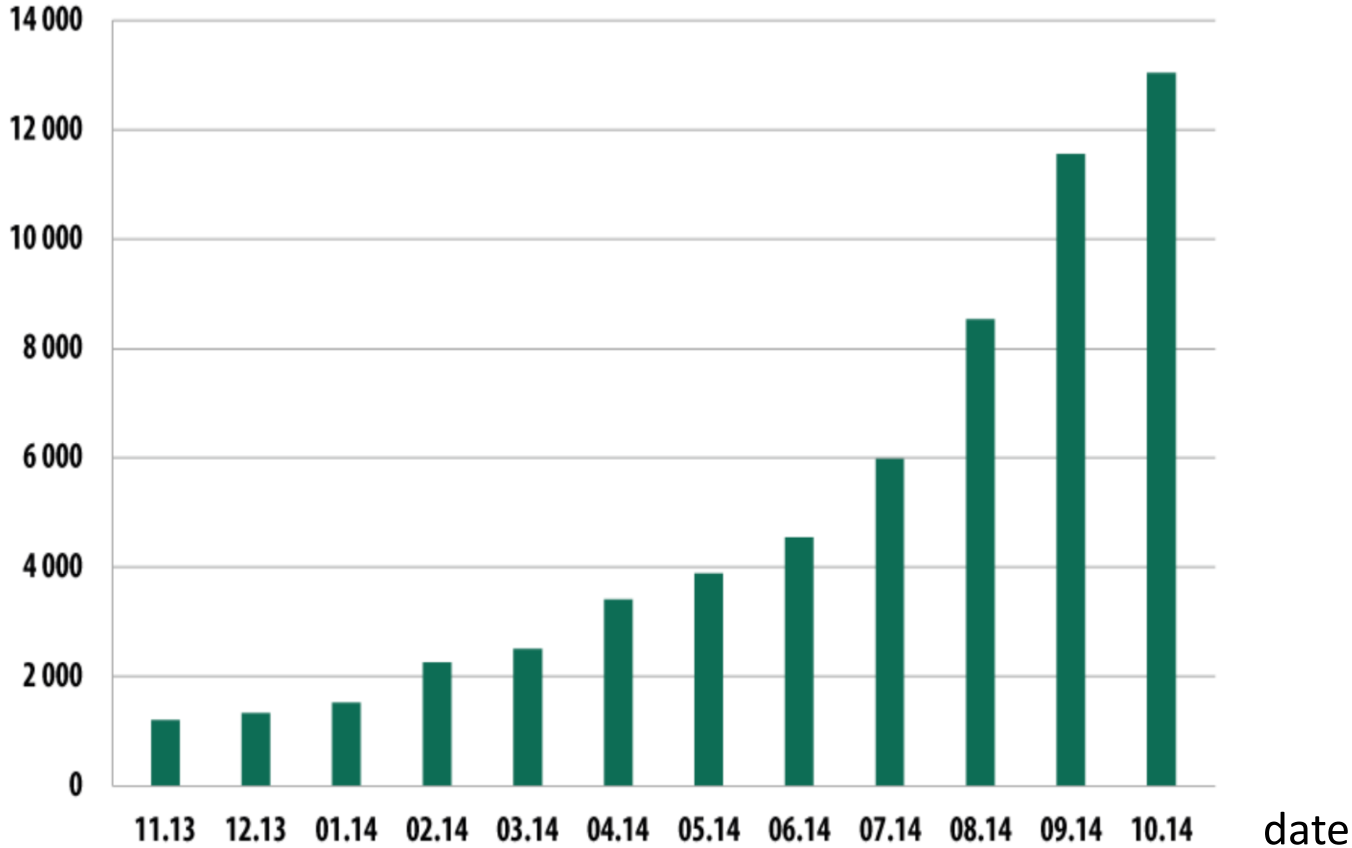


Vulnerable applications being exploited



Source: Kaspersky Security Bulletin 2014

Mobile malware (Nov. 2013 – Oct. 2014)



The rise of mobile banking Trojans (Kaspersky Security Bulletin 2014)

Introduction

Sample attacks

The computer security problem

Two factors:

- **Lots of buggy software** (and gullible users)
- **Money can be made from finding and exploiting vulnerabilities**

1. Marketplace for vulnerabilities
2. Marketplace for owned machines (PPI)
3. Methods to profit from owned client machines

Where is all this buggy software?

- Easy answer: everywhere
- More detailed answer:
  - Vulnerabilities on the client side
  - Vulnerabilities on the server side
  - Vulnerabilities in the network!
  - Inside your organization
  - In other words, everywhere!

Client Side Attacks

1. Compromise user machine (via vuln SW)
2. Install malware on owned machines
3. ???
4. PROFIT

Why own machines: (1) IP address and bandwidth stealing

Attacker's goal: look like a random Internet user

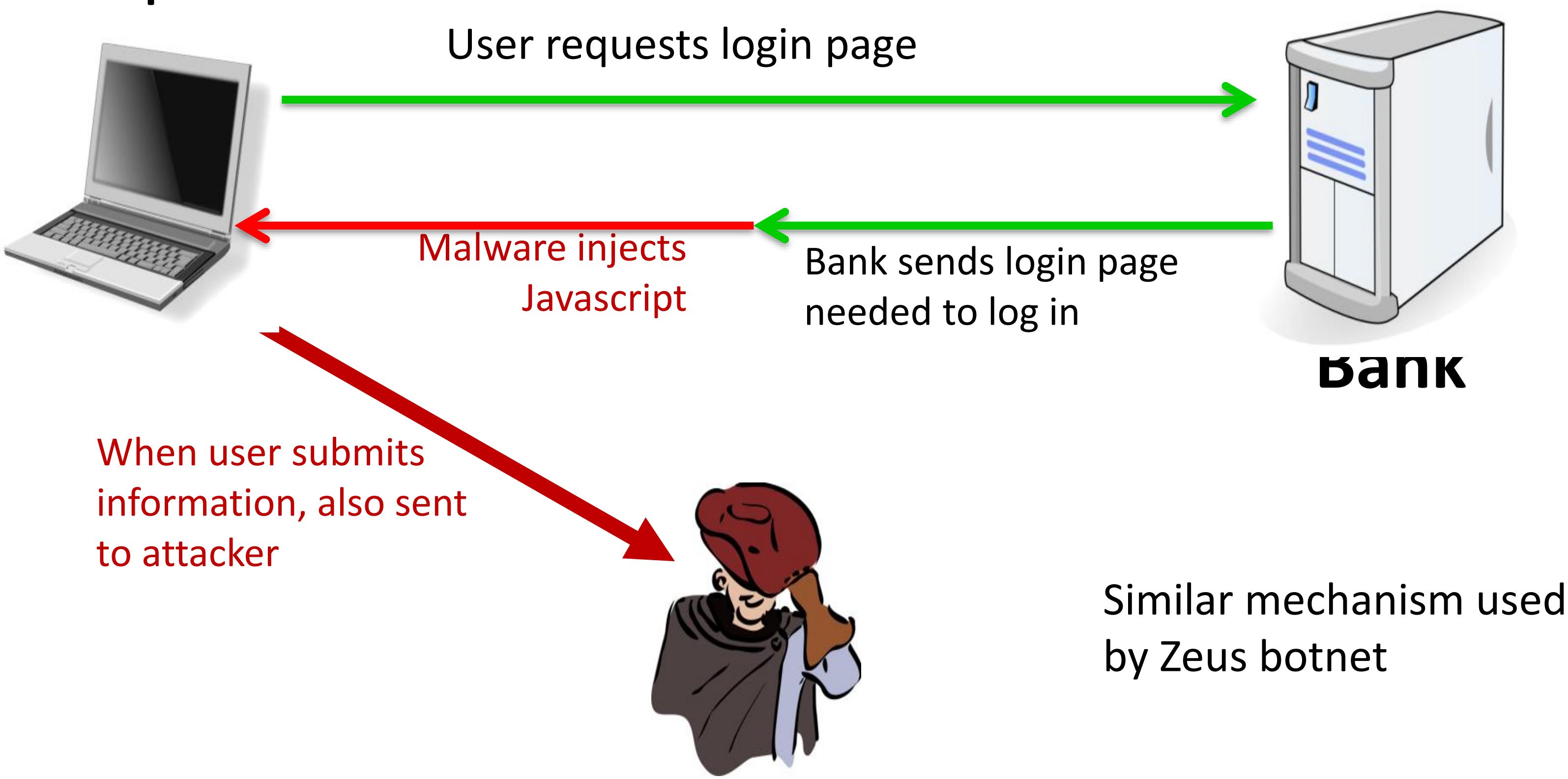
Use IP address of infected machine or phone for:

- **Spam** (e.g. the storm botnet)
  - 1:12M pharma spams leads to purchase
  - 1:260K greeting card spams leads to infection
- **Denial of Service:**
  - Services: 1 hour ($20), 24 hours (100$)
- **Click fraud** (e.g. Clickbot.a)

Why own machines: (2) steal user credentials and/or inject ads

Keyloggers that steal banking passwords, web passwords, gaming passwords

Example: SilentBanker (and many like it)



Server-side attacks

1. Compromise server software
2. Install malware on {web,mail,DNS,…}-server
3. ???
4. PROFIT

Server-side attacks

- Financial data theft: often credit card numbers
  - Example: Target attack (2013) ≈ 140M CC numbers stolen
  - Many similar attacks (e.g., Equifax)
- Political motivation:
  - Aurora, Tunisia Facebook (Feb. 2011), GitHub (Mar. 2015)
- Infect visiting users

Example: Mpack

- PHP tools installed on compromised web sites
  - Embedded as an iframe on infected page
  - Infects browsers that visit site
- Features
  - Mgmt console provides stats on infection rates
  - Sold for several hundred dollars
  - Customer care can be purchased, one-year contract
- Impact: 500,000 infected sites (via SQL injection)
  - Several defenses: e.g. Google safe browsing

Network Attacks

1. Compromise some part of the Internet
2. Misroute/inspect/modify traffic via this part
3. ???
4. Profit

MyEtherWallet Hack (Apr 2018)

- MyEtherWallet – online “wallet” for Ethereum
- Can send/receive money via Eth blockchain
- What’s the impact of stealing pwd to MEW?
  - Thief can steal all your money on Eth blockchain
- So how did the hack work?

MyEtherWallet Hack

1. Create a fake website that looks like original
2. Ask users to type in email/password
3. Use their data to steal all their money
4. …
5. PROFIT??

Q: What’s the flaw in this plan?

A: Nobody will visit this fake website!

Actual MyEtherWallet Hack

1. Hack Amazon DNS (Route 53) to give response pointing to bogus webserver
   - How do to do this?
   - BGP hijacking to route queries meant for Route 53 to malicious DNS server hosted by attackers
2. Now give IP address of bogus site
3. Users will now visit a bogus site even though they typed myetherwallet.com into browser

Insider attacks: example

Hidden trap door in Linux (nov 2003)

- Allows attacker to take over a computer
- Practically undetectable change (uncovered via CVS logs)

Inserted line in wait4()

```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
            retval = -EINVAL;
```

Looks like a standard error check, but …

See: http://lwn.net/Articles/57135/

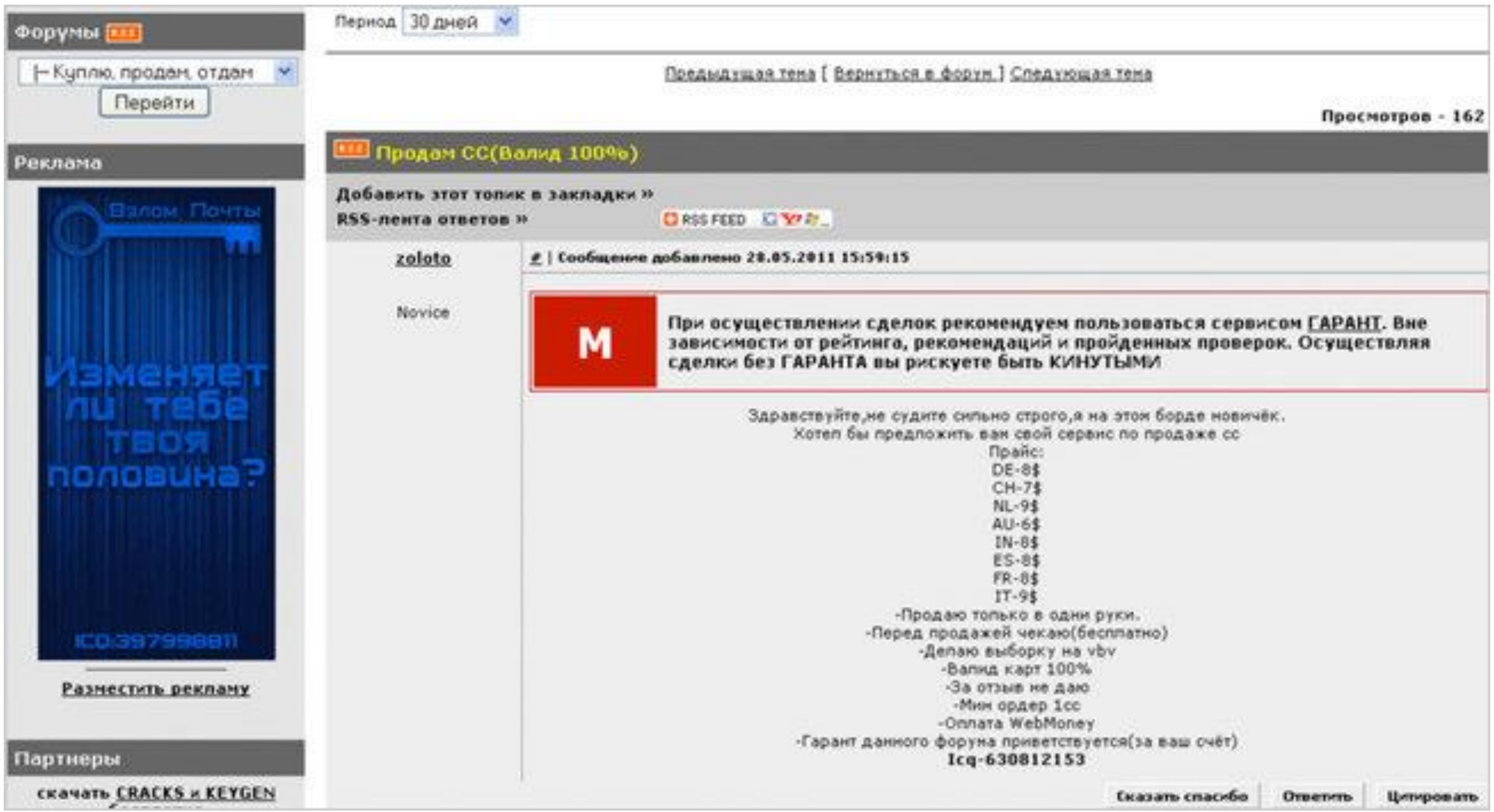
Many more examples

- Access to SIPRnet and CD-RW: 260,000 cables $\Rightarrow$ Wikileaks
- SysAdmin for city of SF government. Changed passwords, locking out city from router access
  - https://www.cio.com.au/article/255165/sorting_facts_terry_childs_case/?pp=4&fp=&pf=1&fpid=
- Inside logic bomb took down 2000 UBS servers
  - https://www.theregister.co.uk/2006/12/13/ubs_logic_bomber_sentenced/

Introduction

The Marketplace for Vulnerabilities

Hacker zoloto offered credit cards for sale on the Web site HackZone .ru.



Marketplace for Vulnerabilities

Option 1: bug bounty programs (many)

- Google Vulnerability Reward Program: up to $100K
- Microsoft Bounty Program: up to $100K
- Mozilla Bug Bounty program: $500 - $3000
- Pwn2Own competition: $15K

Option 2:

- ZDI : $2K – $25K

Marketplace for Vulnerabilities

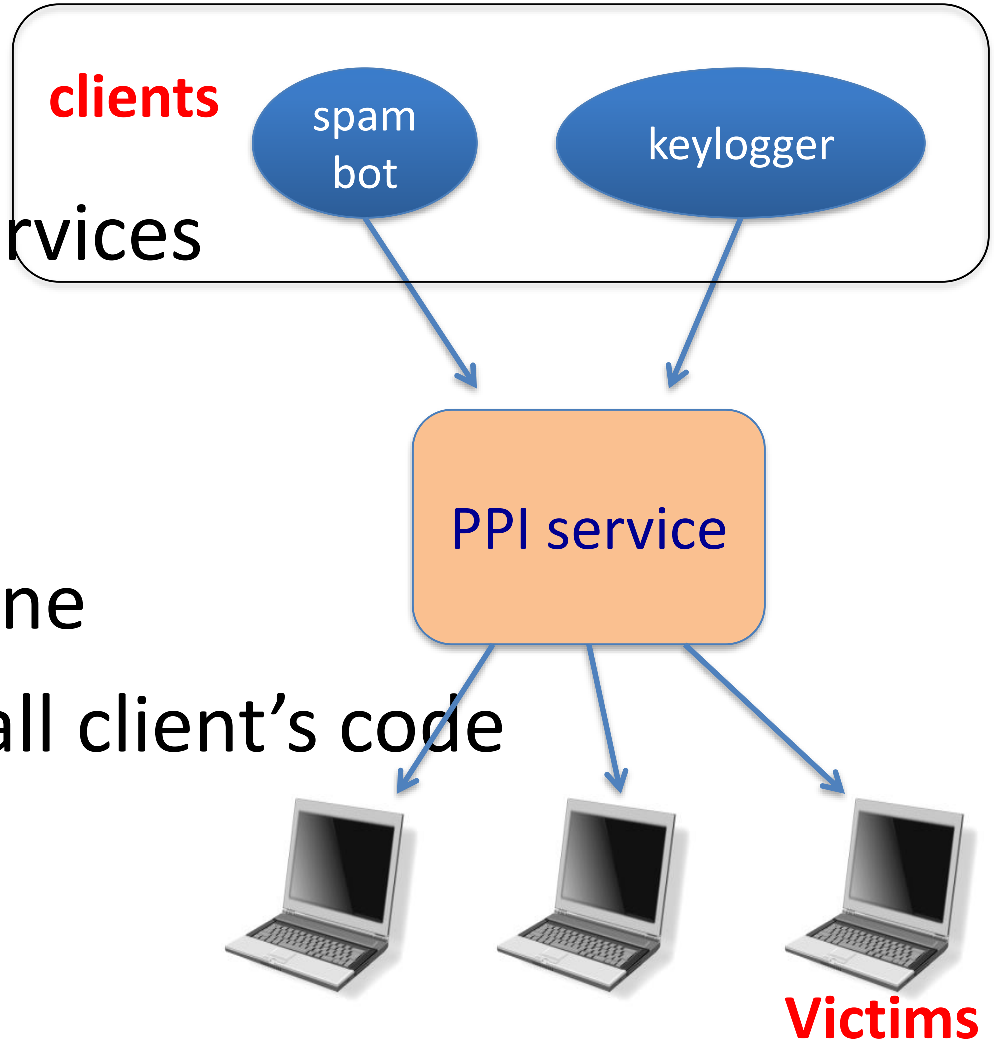
Option 3: Black Market

ADOBE READER	$5,000-$30,000
MAC OSX	$20,000-$50,000
ANDROID	$30,000-$60,000
FLASH OR JAVA BROWSER PLUG-INS	$40,000-$100,000
MICROSOFT WORD	$50,000-$100,000
WINDOWS	$60,000-$120,000
FIREFOX OR SAFARI	$60,000-$150,000
CHROME OR INTERNET EXPLORER	$80,000-$200,000
IOS	$100,000-$250,000

Source: Andy Greenberg (Forbes, 3/23/2012)

Marketplace for owned machines

Pay-per-install (PPI) services

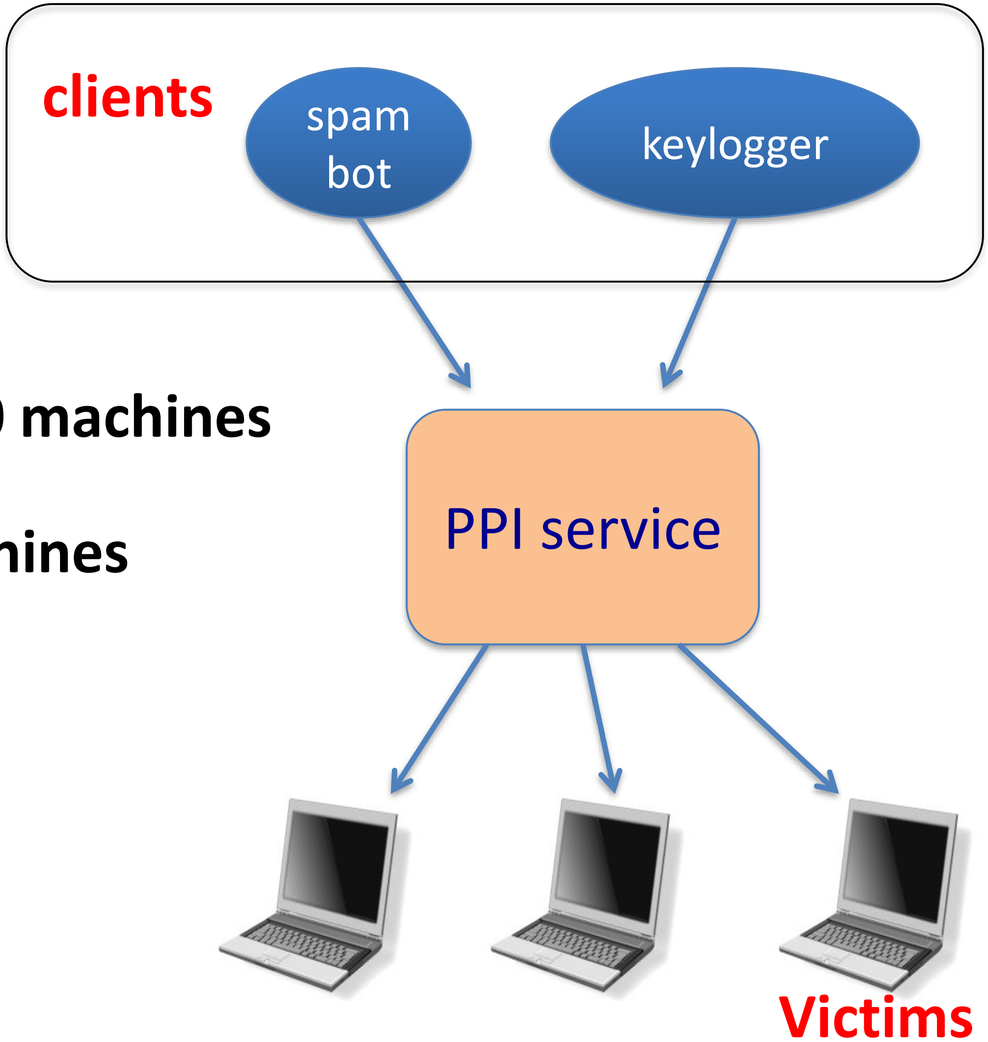


PPI operation:

1. Own victim's machine
2. Download and install client's code
3. Charge client

Source: Cabalerro et al. (www.icir.org/vern/papers/ppi-usesec11.pdf)

Marketplace for owned machines



Cost: **US - $100-180 / 1000 machines**

Asia - $7-8 / 1000 machines

Source: Cabalerro et al. (www.icir.org/vern/papers/ppi-usesec11.pdf)

Ken Thompson’s clever Trojan

Ken Thompson, co-author of UNIX, recounted a story of how he created a version of the C compiler that, when presented with the source code for the "login" program, would automatically compile in a backdoor to allow him entry to the system.

This is only half the story, though. In order to hide this trojan horse, Ken also added to this version of "cc" the ability to recognize if it was recompiling itself to make sure that the newly compiled C compiler contained both the "login" backdoor, and the code to insert both trojans into a newly compiled C compiler. In this way, the source code for the C compiler would never show that these trojans existed.

What is Security?

- Achieving something in the presence of adversaries
  - Internet is full of adversaries
  - Insider adversaries for air-gapped systems
  - So, system designers need to worry about security

What is Security?

- A High Level Plan for Security Centric System Design
  - Policy: “Only X can access file F”
  - Common goals: Confidentiality, Integrity, Availability
  - Threat Models: “Can Y physically grab the file server?”
  - Mechanisms: The knobs that can be controlled to uphold your security policy, but also be flexible to uphold a different policy
  - Resulting Goal: “No way the adversary in the threat model to violate policy”

Why is security hard?

- Need to guarantee policy, assuming threat models
- Often policy and specification are ill-defined
- Code can be difficult to reason about
  - Gap between programmer intent and code behavior
- Difficult to think of all ways an attacker might break in
- Threat models often open-ended (negative models)
  - Easier to check positive goal: “X can access file F”
- Weakest link matters
  - Cannot secure code that you don’t control but rely on

Perfect Security is not achievable

- Best effort
- Need to manage security risk vs. benefit tradeoff
- Risk based security model
- Each system will have breaking point – need to analyze and understand – e.g., pentesting
- Manual auditing often can help
- Make the cost of attack high – deterrence
  - Either by law
  - Technologically

Why policy matters in security

- Example: Sarah Palin’s email account hacked
  - Yahoo accounts have username/password and security questions
  - User can login with username/password
  - If user forgets password – can reset by answering security question
  - Security questions are sometimes easier to guess
  - Some one guessed Palin’s highschool, birthday etc
  - Policy amounts to: can log in with either password or security questions

Policy Matters: iCloud Leaks

- August 2014, 500+ private pictures of celebrities were posted on 4chan
- Initially believed to have been brute-force guessing exploiting the fact that iCloud didn’t rate limit password checks
- Later turned out to be spear-fishing. Attacker sent emails saying account has been compromised and made it look like they’re from Apple/Google

Policy Matters

- All three of these examples are not “bugs”
- Code did exactly what it was supposed to do
- Problem was in the policy aka specification
  - Should not allow password recovery using only security questions
  - Should not allow bruteforcing of passwords
  - Should educate users about phishing attacks

What to do?

- Think hard about implications of policy statements
- Some policy checking tools can help – but you need to specify ‘what is bad’
- Difficult in distributed systems: don’t know what everyone is doing

What might go wrong in threat models/assumptions?

- Human factors not accounted for: ex. Phishing attack
- Computational assumptions change over time:
  - MIT's kerberos system used 56-bit DES keys since mid 1980s
  - Now it costs about $100 to get it cracked
- All SSL certificate CAs are fully trusted
  - To connect to an SSL-enabled website, your browser verifies the cetificate
    - Certificate is a combination of server's host name, and cryptographic key, signed by a trusted CA
  - 100s of CAs are trusted by most browsers
  - In 2011, two CAs were compromised – issued fake certificates for many domains (google, yahoo, tor, …)
  - http://en.wikipedia.org/wiki/Comodo_Group
  - http://en.wikipedia.org/wiki/DigiNotar

Limitations in Assumptions

- Assuming your hardware is trustworthy
  - If NSA is your adversary – it is not necessarily true
    - https://www.schneier.com/blog/archives/2013/12/more_about_the.html
- Assuming good randomness in cryptography
  - Often source of randomness may not be good, and keys may be compromised
  - https://factorable.net/weakkeys12.extended.pdf
- Assuming OS to be secure
  - Bugs? Backdoors? Trojans?
- Machine is disconnected from the Network
  - Did not stop stuxnet worm

What to do to avoid limitations in threat models?

- More explicit and formalized threat models to understand possible weaknesses
- Simpler and more general threat models
- Better design may lessen reliance on certain assumptions
  - E.g., alternative trust models that does not rely on full trust in CAs
  - E.g., authentication mechanisms that aren't susceptible to phishing

Problems with mechanisms

- Bugs in security mechanism (e.g. OS kernel) lead to vulnerabilities (e.g. CVE-2010-0003)
- Might get pwned by code you didn't know existed (e.g., Intel SMM and SMI)
- If application is enforcing security, application bugs can lead to vulnerabilities
  - Example: Missing access control checks in Citigroup's credit card website http://www.nytimes.com/2011/06/14/technology/14security.html?_r=0
  - Example: Android's Java SecureRandom weakness leads to bitcoin theft

Some implementation bugs

- Buffer overflow, Use-after-free, Double-free
- Decrementing stack pointer past the end of stack – into some other memory location
  - http://www.invisiblethingslab.com/resources/misc-2010/xorg-large-memory-attacks.pdf
- Not checking sanity of inputs
  - SQL injection (e.g., see XKCD on next slide)
  - Command injection (e.g., ShellShock)

What's the takeaway?

- Security is hard, but also hugely important – both from tech and societal perspective
- Practicing engineers should be aware of security concerns in computing systems
- This course gives an overview of these topics

This course

Goals:

- Be aware of exploit techniques
- Learn to defend and avoid common exploits
- Learn to architect secure systems

This Course

Part 0: Introduction and Review

- Revisit a few crypto primitives

Part 1: Basics (architecting for security)

- Securing apps, OS, and legacy code
  Isolation, authentication, and access control

Part 2: Web security (defending against a web attacker)

- Securing websites, browser security model

Part 3: Network security (defending against a network attacker)

- Monitoring and architecting secure networks.

Part 4: Hardware Security (SGX, Hardware Trojans)

Looking Forward

- Next: revisiting crypto primitives
  - Assignment #1 (Programming Assgn) goes out
- Then buffer overflows and related attacks
  - Assignment #2 (CTF)
- After that, secure system architecture
- …

Explaining Clickfraud

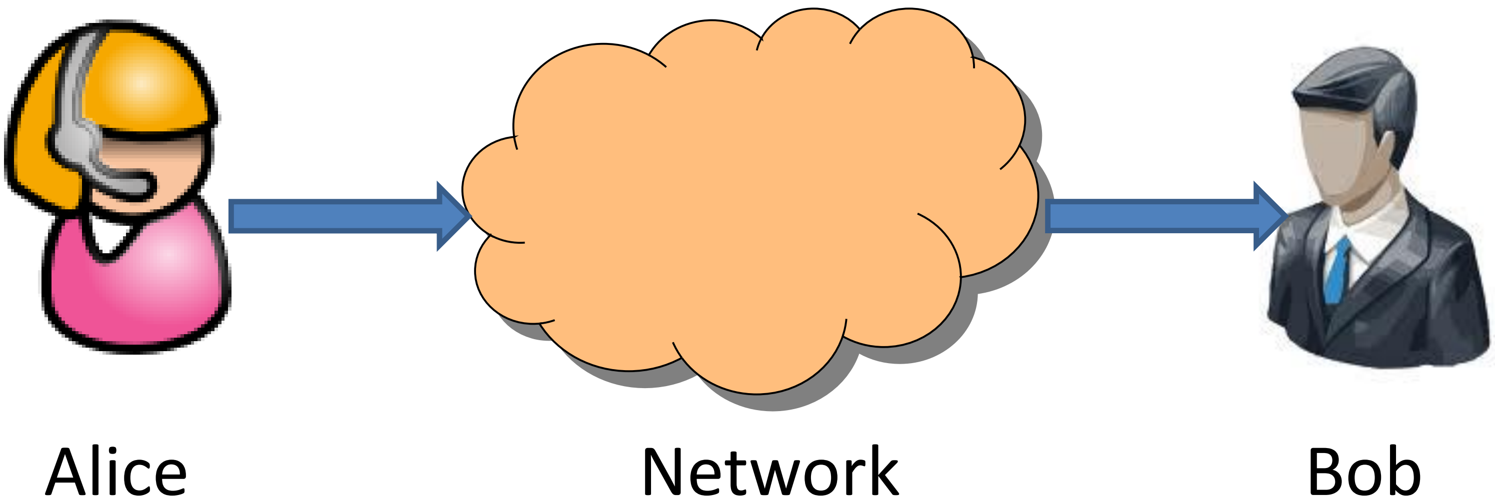
- Google ad cost is PPC (pay per click)
- Google shares some of this revenue with website which generated (also PPC)
- If you are a website operator, what do you do?
- Get fake clicks using botnets
- Fabio Gasperini: Clickfraud prosecution in US

REVIEW OF CRYPTOGRAPHIC PRIMITIVES

Outline

- Message Authentication Codes: MACs
- Block ciphers
- Public key cryptography
- Diffie-Hellman Key Exchange/PFS

Alice wants to send Bob a message



CIA properties

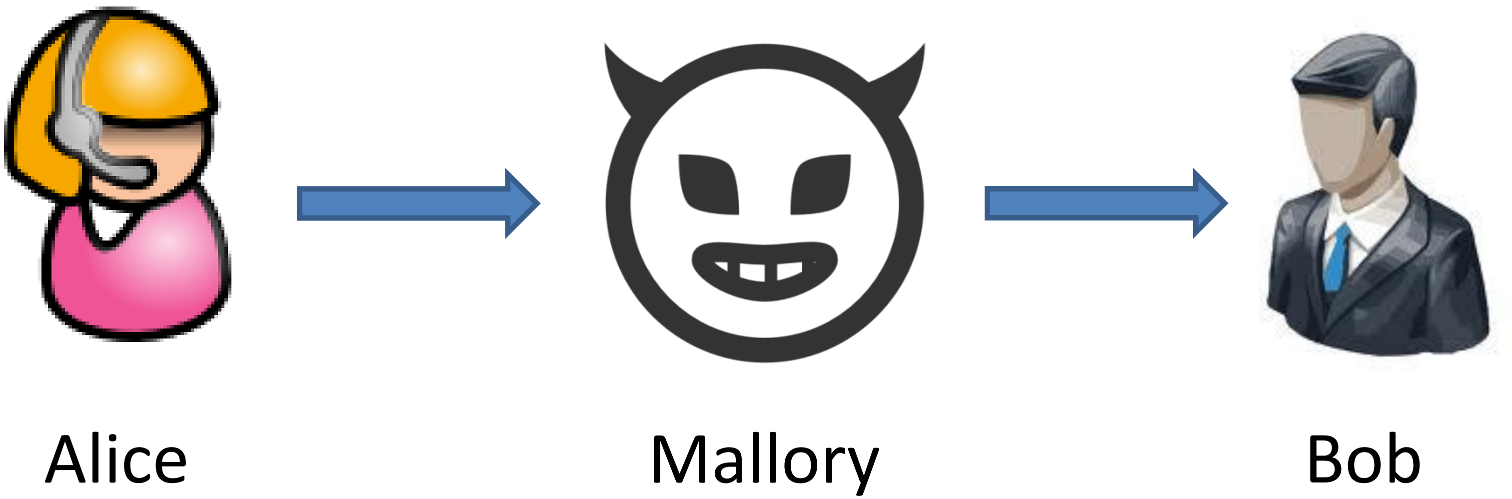
- Confidentiality: keep message secret
- Integrity: prevent message being tampered with
- Availability: make sure message can reach

Alice wants to send Bob a message

Focus on integrity

Assume worst case: Mallory controls network

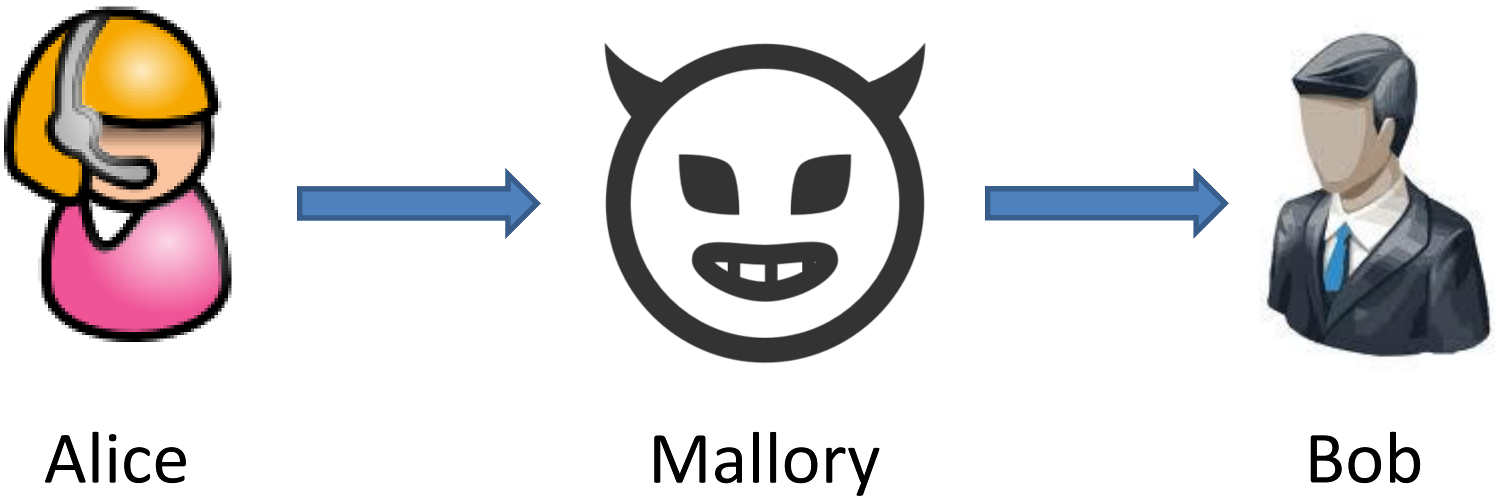
Threat model



Mallory can see, modify, forge messages

Wants to trick Bob into accepting a message that Alice didn't send

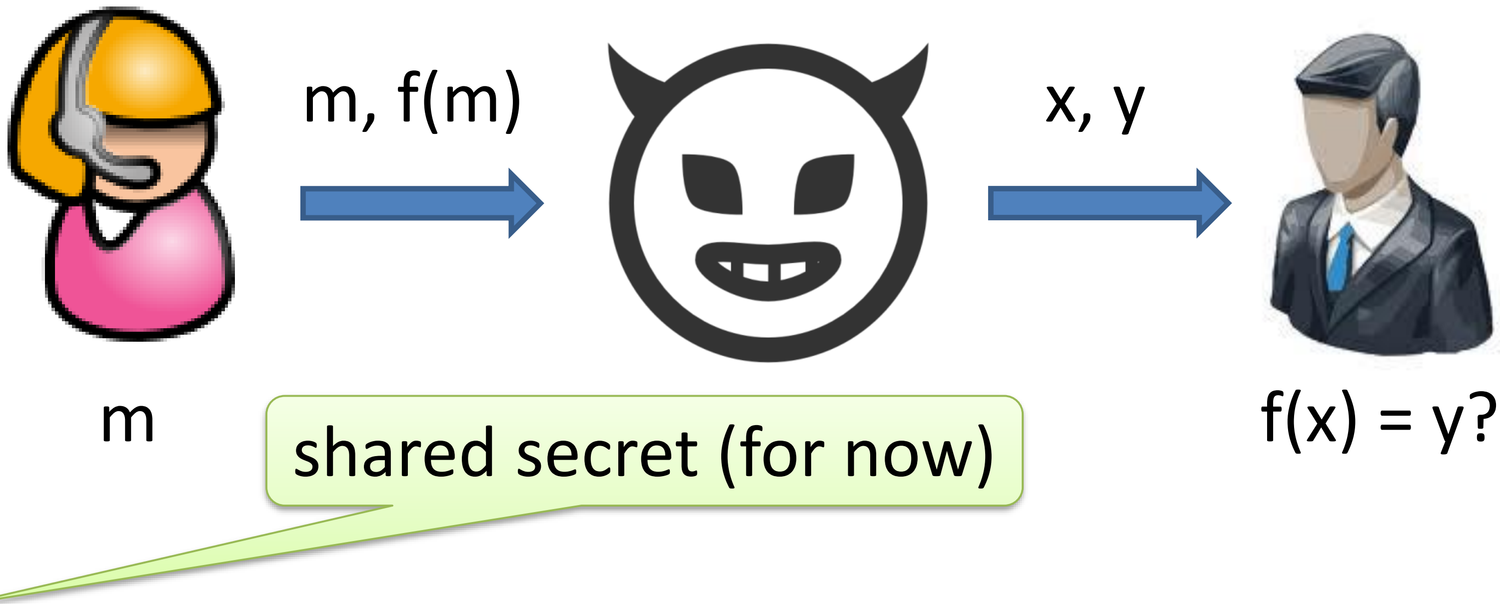
A shared secret



Bob knows something about Alice beforehand.
Something Mallory *does not* know.

Even integrity depends on secrecy.

Message Authentication



f is a **MAC (Message Authentication Code)**

- Deterministic (not randomized)
- Easy for Alice and Bob to compute
- Mallory doesn't know how to compute

A secret function

We're sunk if Mallory can determine *f*

Or even if Mallory can learn $f(x)$ for any $x \;!= m$

Simplest approach: random function

Different random value for each possible input *m*

Limit *m* to *256* bits:

table of size 2^{256} to implement random function

Provable security

We want to prove this is secure

Modern cryptography is all about proofs

Why?

- Design once and forget
  - Tired of build-and-break cycle
- Small set of primitives to focus on
- Doesn't matter how clever the adversary is

Secure MAC game: Us vs. Mallory

Repeat until Mallory says stop

Mallory chooses m_i

We tell Mallory $f(m_i)$

Mallory picks $m' \notin \{m_i\}$ and guesses $f(m')$

f is a secure MAC if

Mallory can't do better than random guessing

Pseudorandom function (PRF)

A random function is too hard to implement

Let’s pick a function that’s almost random but easy to implement

“Pseudorandom function” (PRF)

“Looks random” or “as good as random”

Pseudorandom function

Typical approach

Public “family” of functions f_0, f_1, f_2 …

$f_k(m) = f(k, m)$

Secret key k

Pick and use f_k

Kerckhoffs's principle

Use a <u>public function family</u> and a randomly chosen <u>secret key</u>.

Advantages:

1. can quantify probability that key will be guessed
2. different people can use the same functions with different keys
3. can change key if needed (something goes wrong)

Attempt at PRF

Secret key k (say 256 bits)

$f_k(m) = k \oplus m$ (assume messages also 256 bits)

Discussion: is this a good PRF?

“PRF game”: Us vs. Mallory

We pick with 50/50 probability

either a real random function, or

$g = f_k$ for random k

Repeat until Mallory says stop:

Mallory chooses m_i

we announce $g(m_i)$

Mallory wins if she guesses whether we chose truly random function or PRF

PRF is secure if Mallory’s advantage over random is “negligible”

With no constraints Mallory always wins

Request a few input/output pairs, then go through every possible *k* ("brute force")

Homework [optional]: Prove that this wins for Mallory

Exponentials to the rescue!

Require Mallory to be *efficient*

Theorem:
If f is a PRF, then f is a secure MAC

Proof: by contradiction

Assume f is not a secure MAC

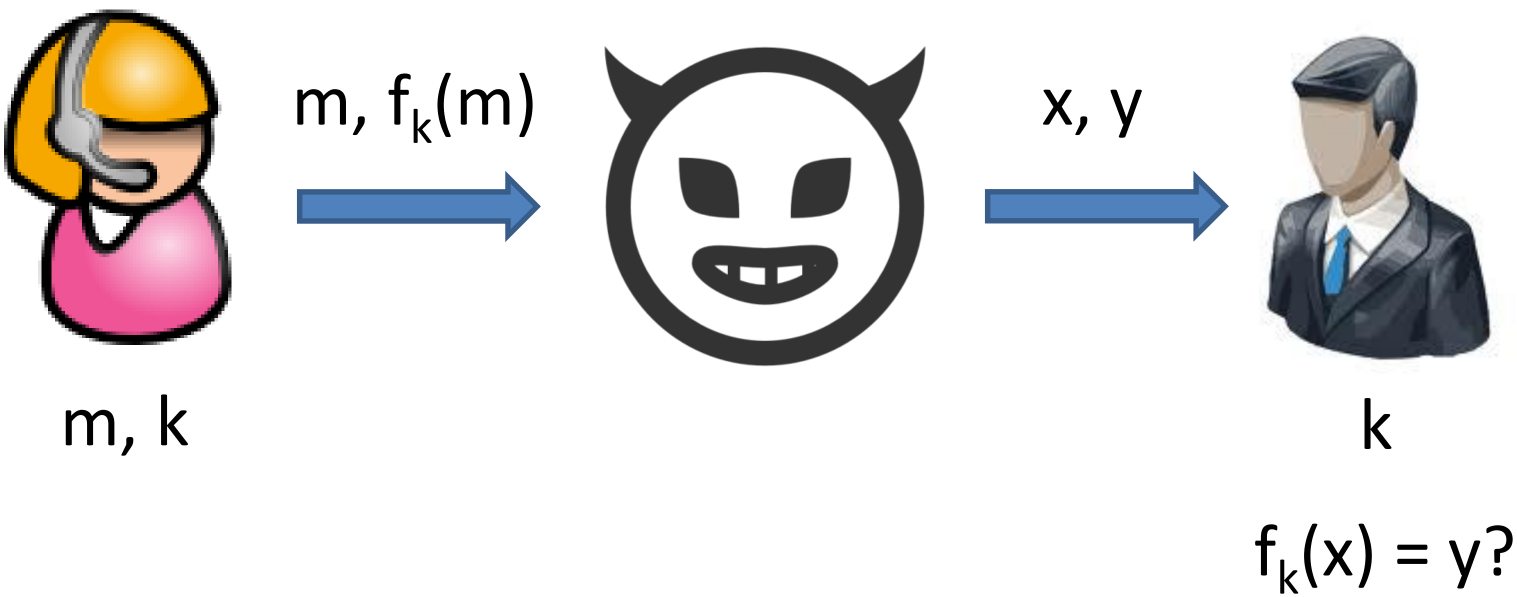
Then there's an adversary (algorithm) that wins secure MAC game

Use this algorithm as a subroutine to win PRF game

So f is not a PRF

Homework: complete this proof

MAC using shared secret key



Annoying question:
Do PRFs actually exist?

Dirty secret of crypto:

everything interesting relies on assumptions

So what do we use in practice?

Based on hash functions

Hash function:
The Swiss army knife of crypto

Popular examples:

- MD5 (has weaknesses, shouldn't be used)
- SHA-1, SHA-256

Typical construction:

"Merkle-Damgård"

Ralph Merkle

Hash function:

takes any string as input

fixed-size output (we'll use 256 bits)

efficiently computable

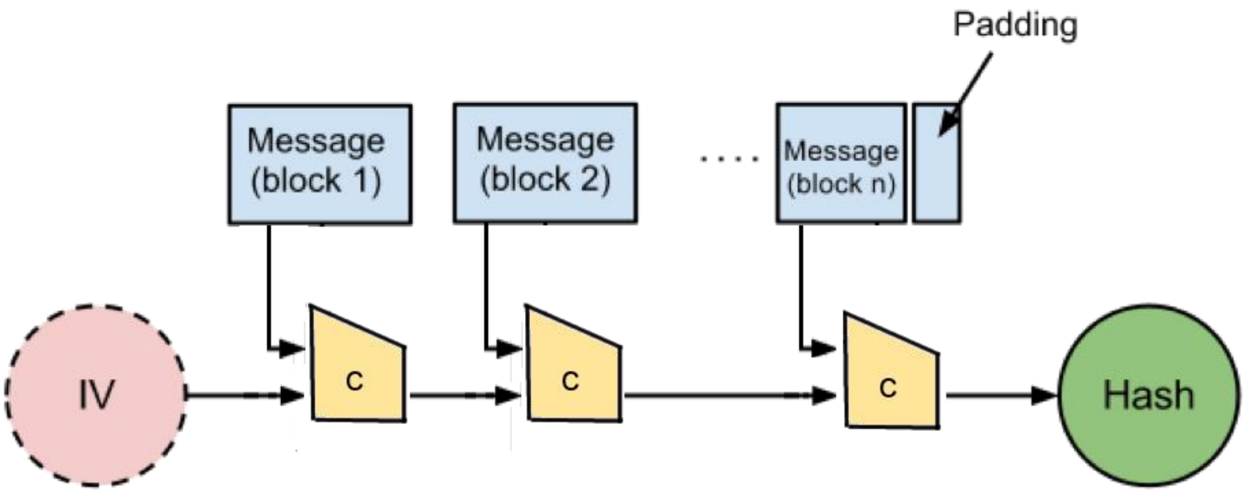
Security properties:

collision-free

hiding (preimage resistance)

puzzle-friendly

Merkle-Damgård construction



- Break input into blocks (say 512 bits)
  Pad the last block
- Apply “compression function” to message block together with output of previous stage
- Compression function designed to look really hairy
- IV = initialization vector

Hash-based MAC

Q. Is a Hash(k || msg) a secure MAC?

A. No! "Length-extension attack"

Knowing $f_k(msg)$ *(i.e., f(k || msg))* lets adversary compute $f_k(msg \,||\, app)$ without knowing the key

Homework: verify this

How to fix: HMAC

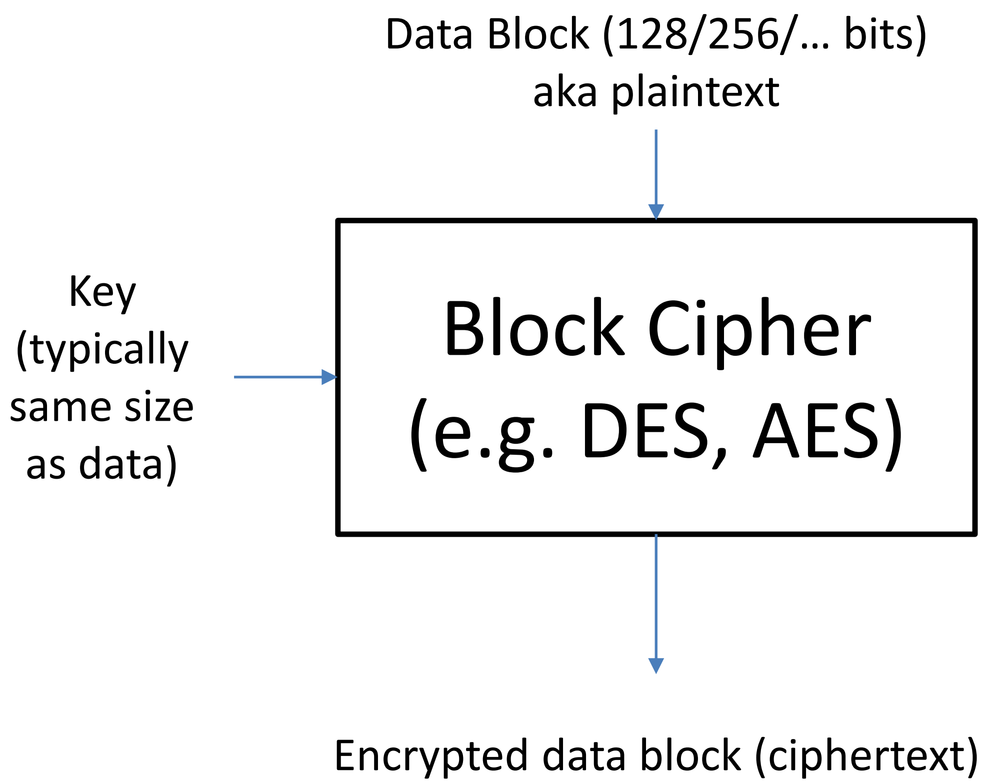
$$HMAC(k,m) = H(\underline{k \oplus z_1} \,||\, \underline{H(k \oplus z_2 \,||\, m)})$$

z_1 and z_2 are constants

If you want to learn only one thing about MAC, then this is it!

BLOCK CIPHERS

Block Cipher

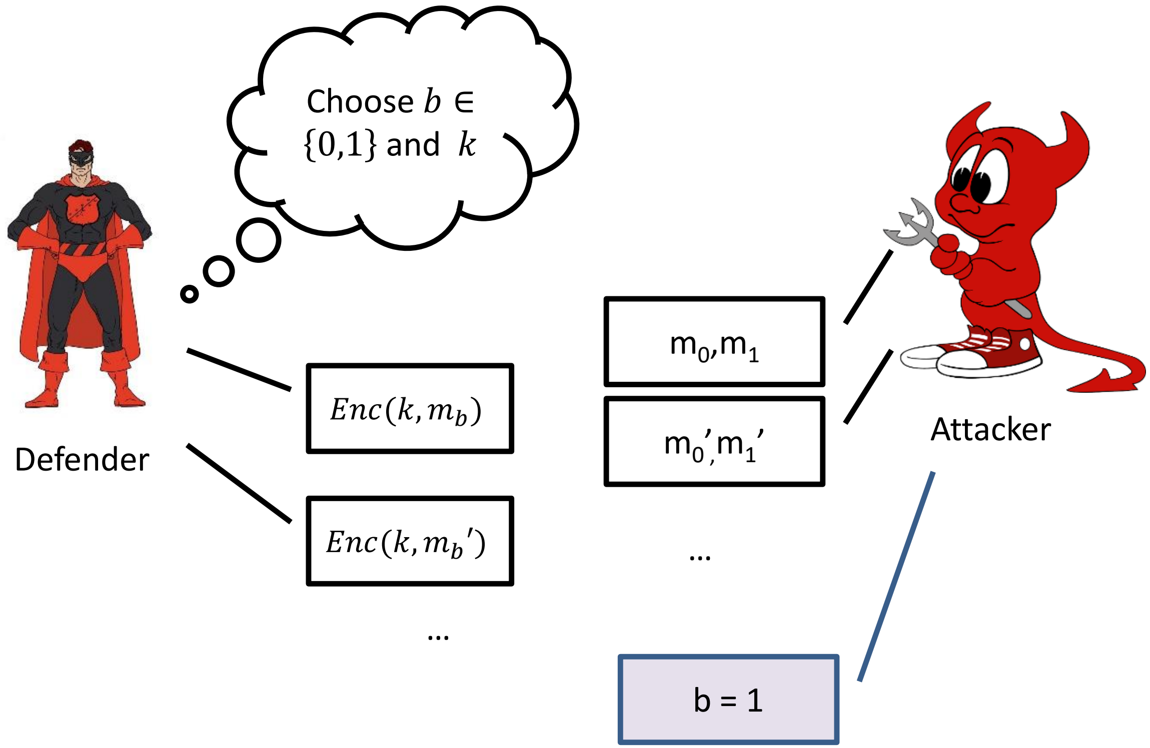


- Q: What if I have message > 256 bits in size?
- Easy answer: split into blocks
- How exactly this is done matters
- Different modes of operation

What are some properties of a good block cipher?

- Should it be deterministic? (Why or why not?)
  - A: No, because small messages can be enumerated
- How about a cipher that is as follows:
  - E(k, x, r) = (r, r XOR x XOR k)
  - We will make sure to use a unique r for each message
- What about an encryption scheme that preserves order?
  - x1 < x2 => E(k, x1) < E(k, x2)
  - When might this be useful?
  - Why might this be a bad idea?

IND-CPA Game



Game Initialization

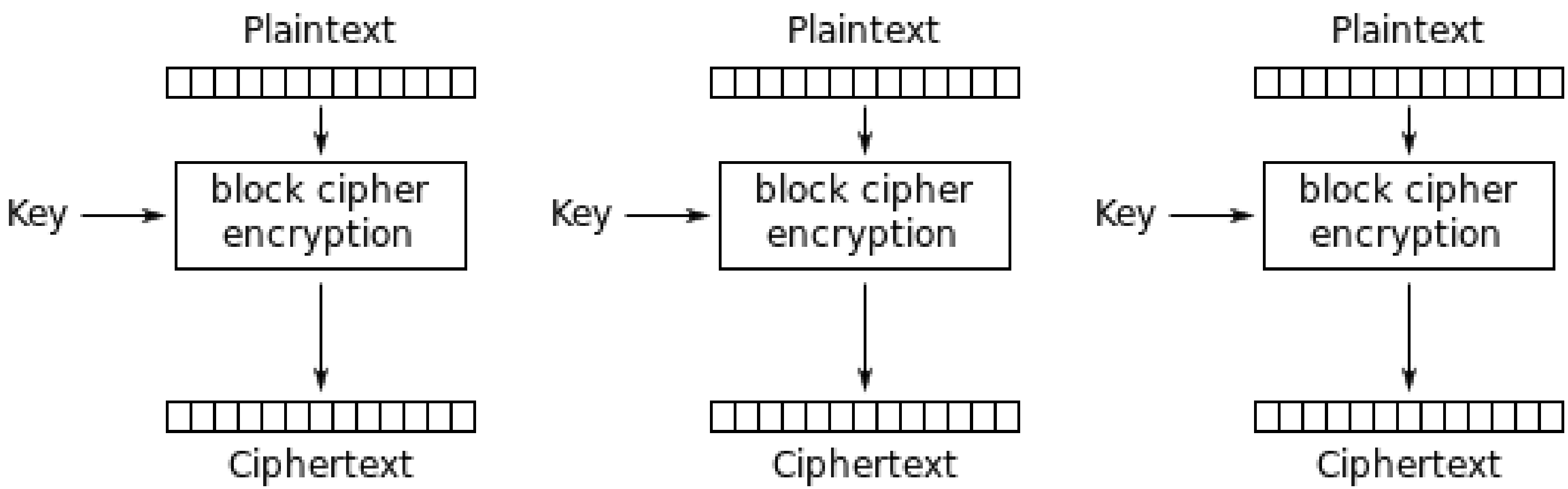
- Defender chooses $b \in \{0,1\}$ and k

Game Execution

- Attacker can ask for encryption for arbitrary pairs of messages (m_0, m_1), …
- Defender returns $Enc(k, m_b)$ for each query

Finalization: Attacker wins if she can determine b with a polynomial number of queries

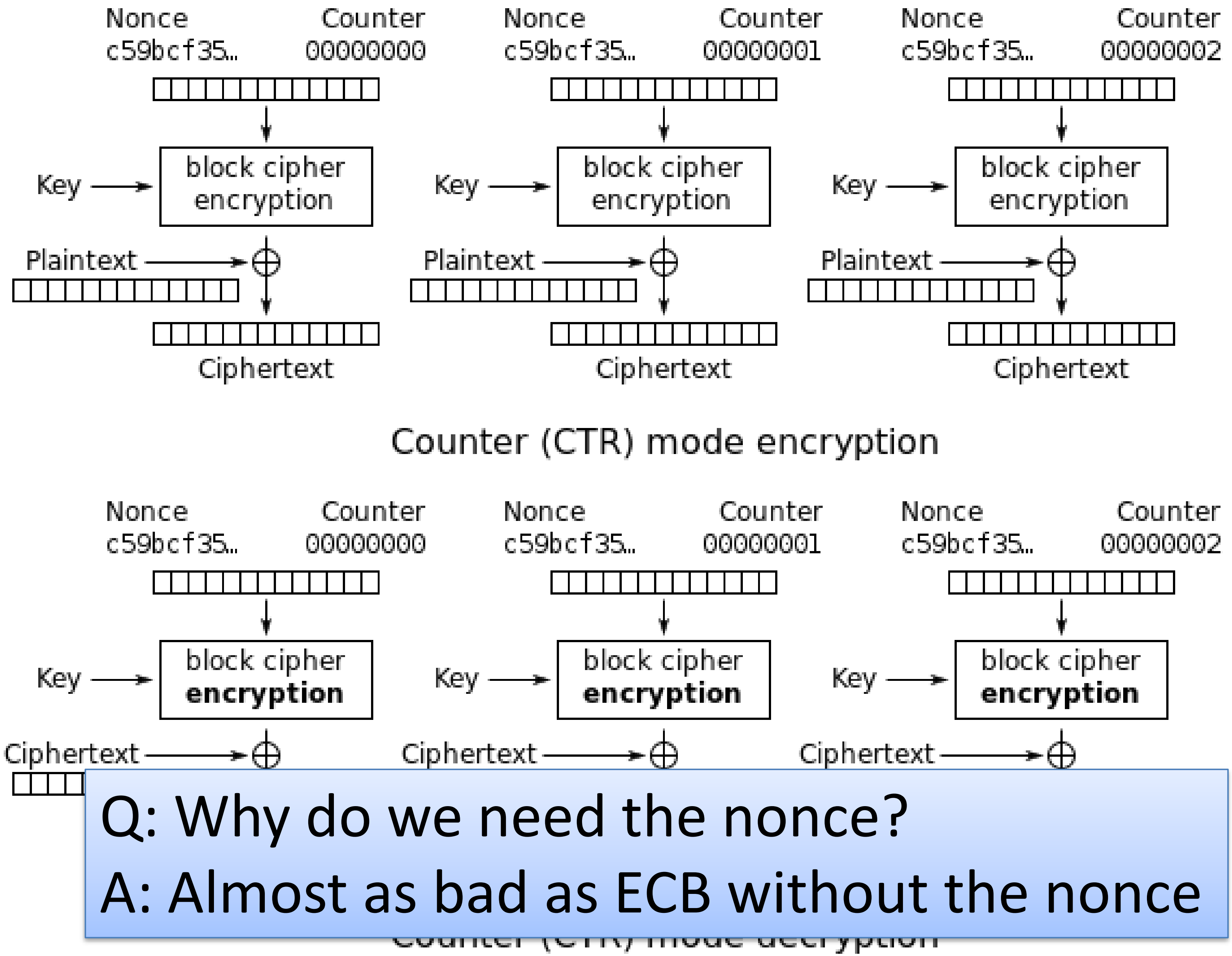
Block ciphers: ECB mode



Electronic Codebook (ECB) mode encryption

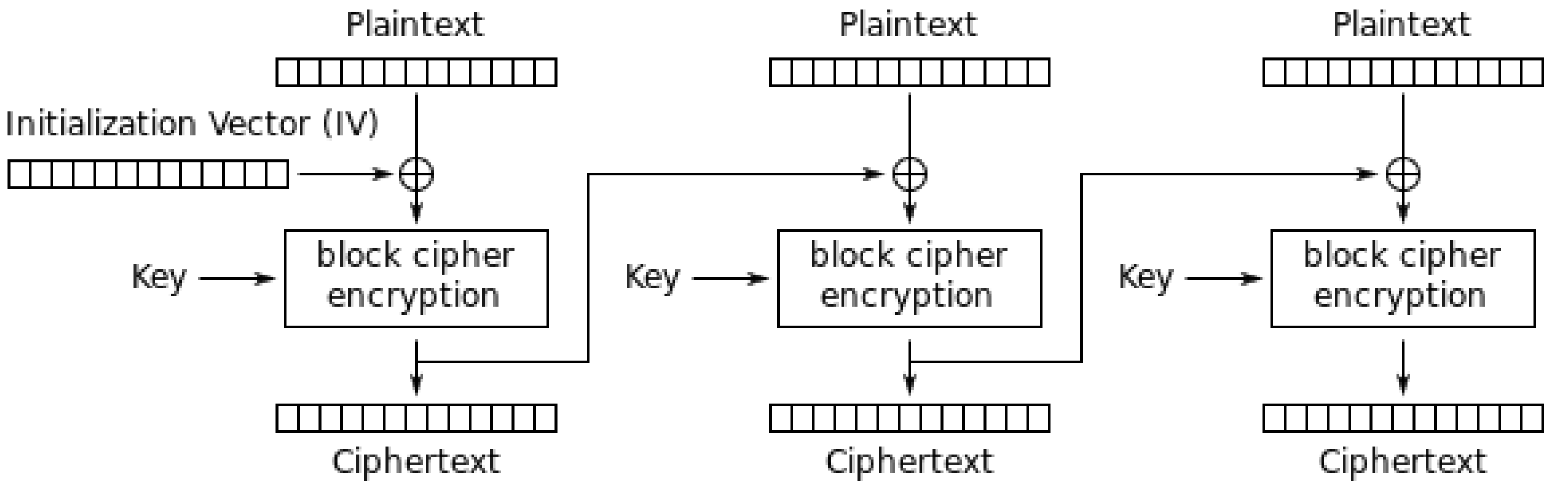
Question: What is the problem with ECB?

Same input block results in the same output block

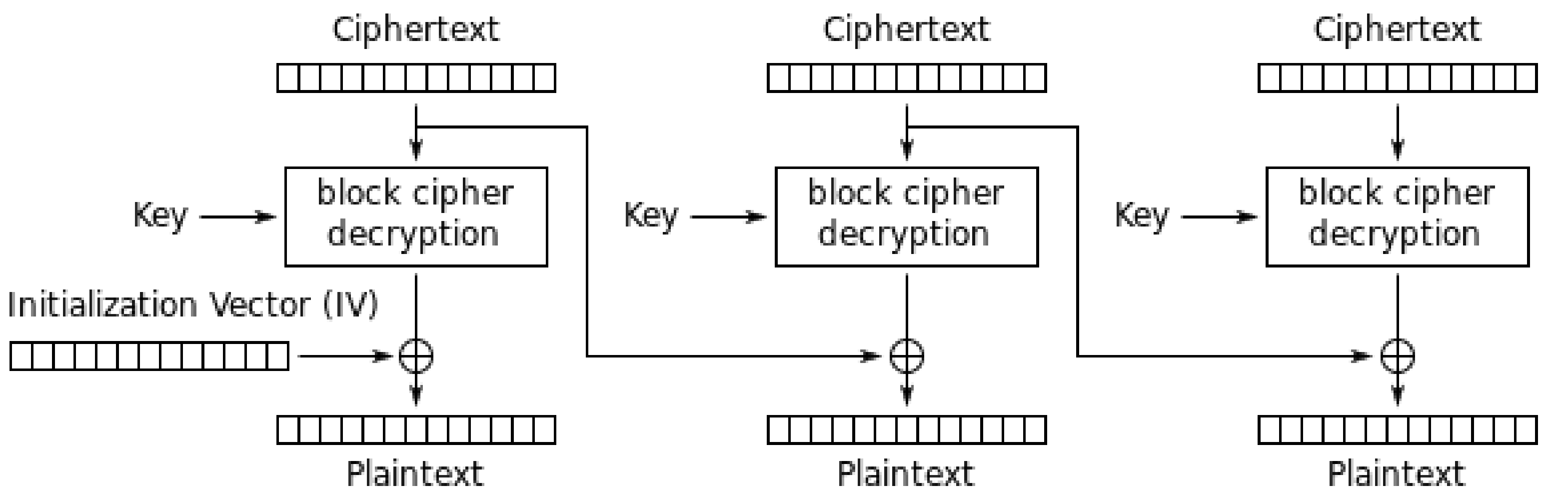


Q: Why do we need the nonce?

A: Almost as bad as ECB without the nonce



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Public key encryption

RSA

Encryption: What happens with N people?

- All N share a key?
  - Read each others' messages
  - Model: crypto is the only protection
  - Always broadcast to the network
  - No such thing as sending a message only to one recipient
  - Another problem: impersonate each other
- Each pair share a key?
  - N keys per participant
  - Setting up keys is a nightmare

Asymmetric cryptography

- Different key for encryption and decryption
(Or creating and verifying auth code)
- One key-pair per person, not pair of people
- One key in the pair is kept secret
The other is given to everyone (published)
- Secret key can't be derived from public key
Doesn't matter if public key can be derived from secret key
- **Discussion: encryption or decryption key public?
Auth creation or verification key public?**

RSA function

Large random primes

- Alice generates N = pq and
  e relatively prime to (p-1)(q-1)
- Euclid's algo to find d s.t.
  ed % (p-1)(q-1) = 1
- Publishes (N, e). Keeps (d, p, q) secret

- RSA(N, e, x) = x^e % N
  RSA(N, d, y) = y^d % N

Inverses

Trapdoor permutation

- Permutation
  Easy to compute

- Hard to invert
  Except if trapdoor is known

RSA Encryption – OAEP encoding

n: RSA modulus length

m: message

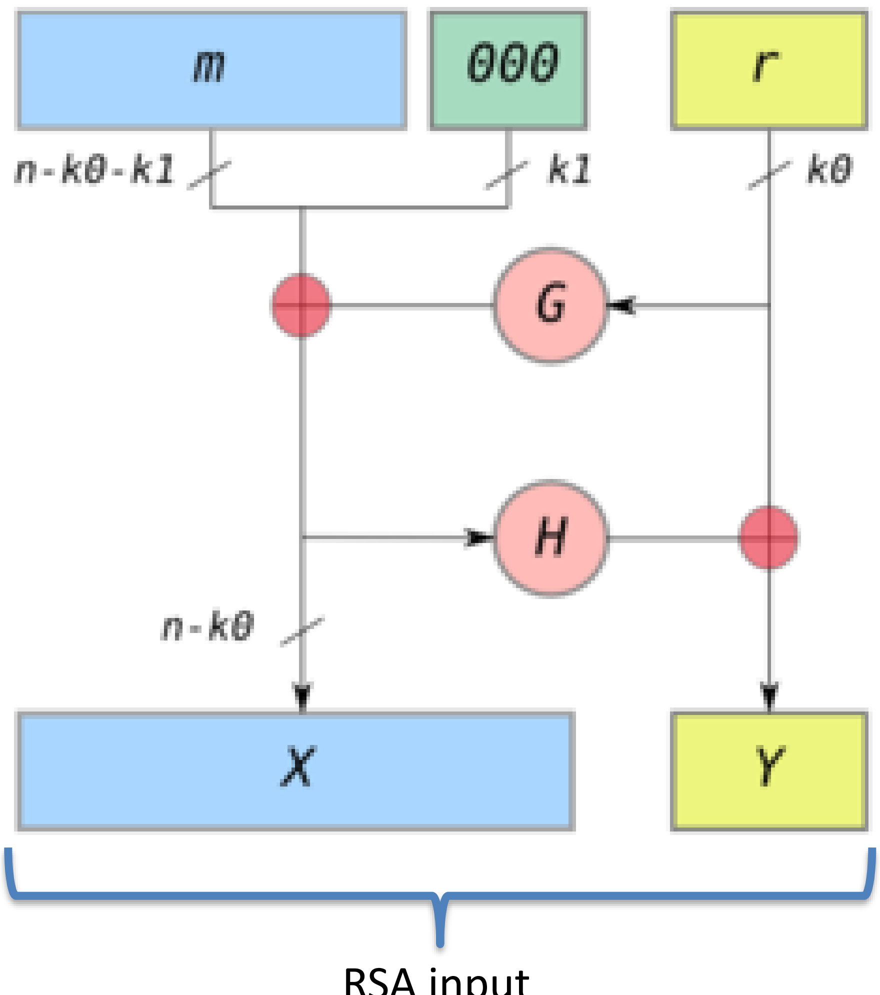
000: padding

r: random nonce

G: PRG

H: hash function

k_0, k_1: 128 bits

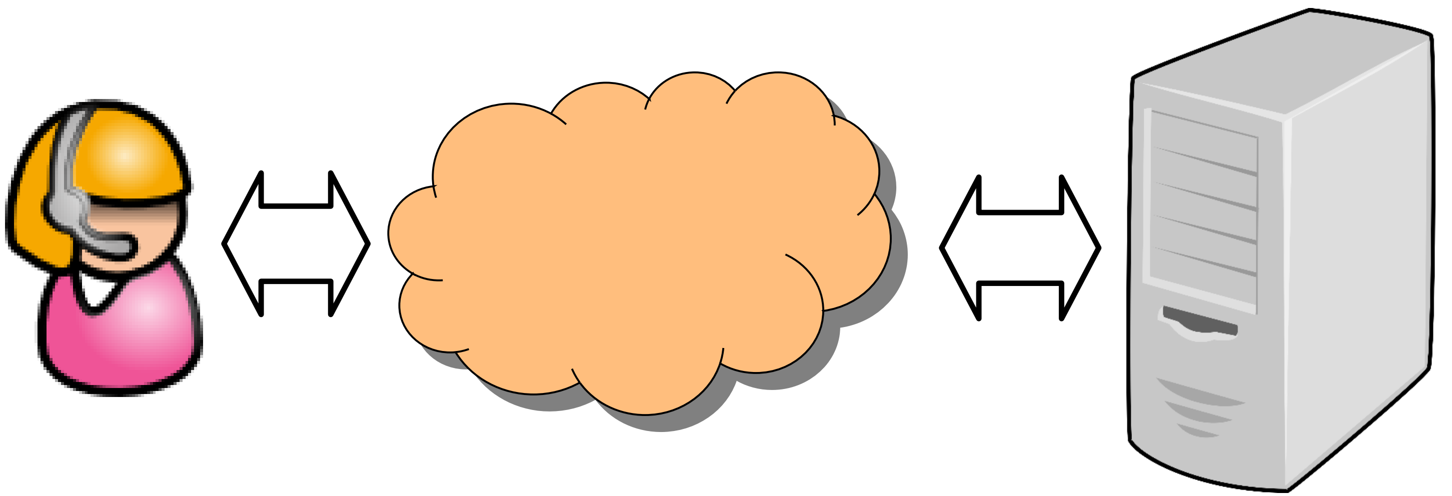


PKCS #1

https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding

D-H EXCHANGE AND PERFECT FORWARD SECRECY

Strawman SSL

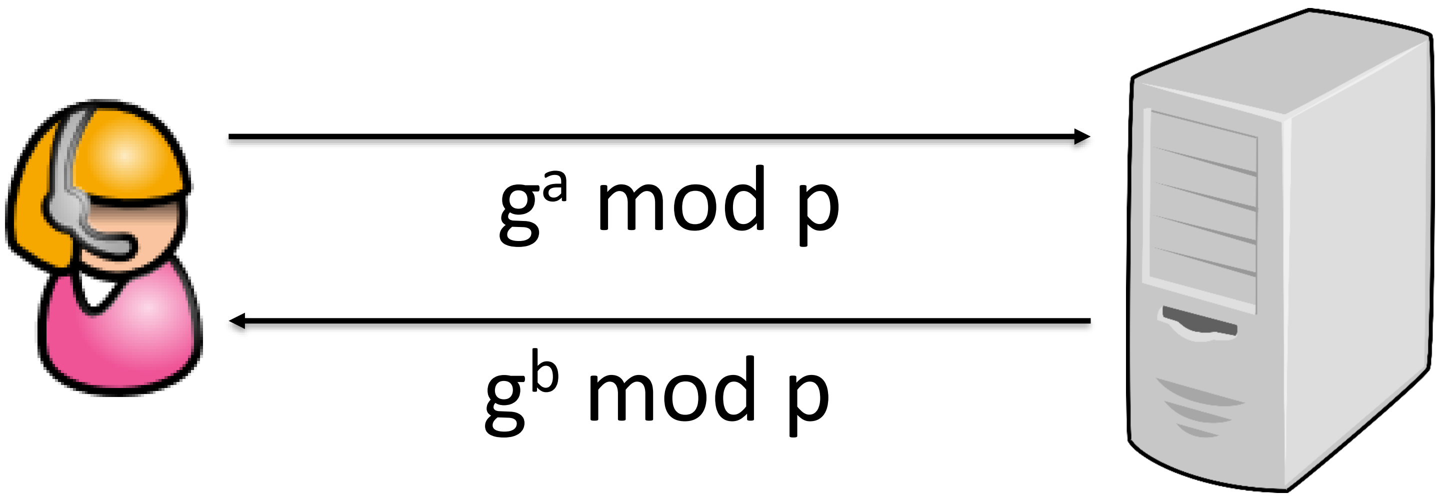


- Alice gets public key of webserver from CA
- Sends session key encrypted using this pubkey
- Server and alice communicate using this key

Problems with this protocol?

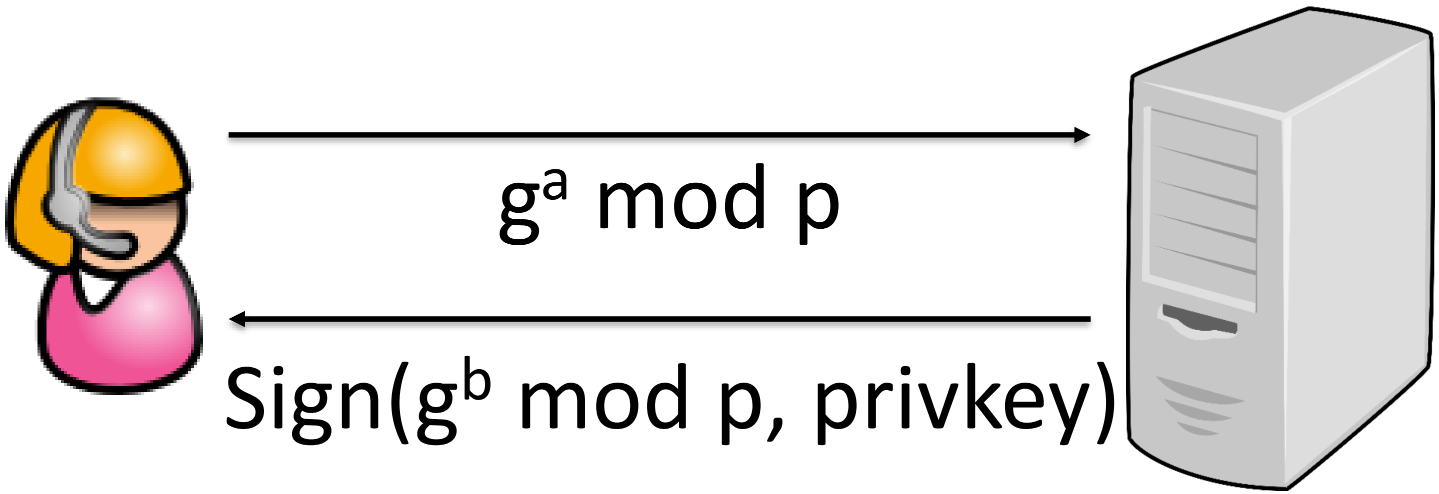
- What if server private key is compromised?

Diffie-Hellman Key Exchange



- p is a prime, g is called a generator
- After exchange, both parties know $g^{ab} \bmod p$
- More importantly, nobody else knows g^{ab}
- This holds even if privkey is compromised **in future**
- Satisfies property of forward secrecy

Diffie-Hellman Key Exchange



- p is a prime, g is called a generator
- After exchange, both parties know g^{ab} mod p
- More importantly, nobody else knows g^{ab}
- This holds even if privkey is compromised **in future**
- Satisfies property of forward secrecy