CS162
Operating Systems and Systems Programming
Lecture 21

Security (I)

April 11, 2012
Anthony D. Joseph and Ion Stoica
http://inst.eecs.berkeley.edu/~cs162

Goals for Today

- Conceptual understanding of how to make systems secure
- Key security properties
  - Authentication
  - Data integrity
  - Confidentiality
  - Non-repudiation
- Cryptographic Mechanisms

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne, and lecture notes by Kubiatowicz

What is Computer Security Today?

- Computing in the presence of an adversary!
  - An *adversary* is the security field's defining characteristic
- Reliability, robustness, and fault tolerance
  - Dealing with Mother Nature (random failures)
- Security
  - Dealing with actions of a knowledgeable attacker dedicated to causing harm
  - Surviving malice, and not just mischance
- Wherever there is an adversary, there is a computer security problem!

Protection vs Security

- Protection: one or more mechanisms for controlling the access of programs, processes, or users to resources
  - Page table mechanism
  - Round-robin schedule
  - Data encryption
- Security: use of protection mechanisms to prevent misuse of resources
  - Misuse defined with respect to policy
    - » E.g.: prevent exposure of certain sensitive information
    - » E.g.: prevent unauthorized modification/deletion of data
  - Requires consideration of the external environment within which the system operates
    - » Most well-constructed system cannot protect information if user accidentally reveals password – social engineering challenge

Preventing Misuse

- Types of Misuse:
  - Accidental:
    - If I delete shell, can't log in to fix it!
    - Could make it more difficult by asking: "do you really want to delete the shell?"
  - Intentional:
    - Some high school brat that transfers $3 billion from B to A
    - Criminal organization steals logon credentials via phishing attack
    - Doesn't help to ask if they want to do it (of course!)
- Three Pieces to Security
  - Authentication: who the user actually is
  - Authorization: who is allowed to do what
  - Enforcement: make sure people do only what they are supposed to do
- Loopholes in any carefully constructed system:
  - Log in as superuser and you've circumvented authentication
  - Log in as self and can do anything with your resources; for instance: run program that erases all of your files
  - Can you trust software to correctly enforce Authentication and Authorization?

Analyze to Learn!

- We're going spend study attackers and think about how to break into systems
  - Why spread knowledge that will help bad guys be more effective?
- To protect a system, you have to learn how it can be attacked
  - Civil engineers learn what makes bridges fall down so they can build bridges that last
  - Software engineering is similar
- Security is the same and different!
  - Why?

Challenges in Securing Systems

- Similar:
  - Analyze previous successful attacks
- But, deploy a new defense, they respond, you build a better defense, they respond, you…
  - Need to find ways to anticipate kinds of attacks
- Different:
  - Attackers are intelligent (or some of them are)
  - Attacks will change and get better with time
  - Have to anticipate future attacks
- Security is like a game of chess
  - Except the attackers often get the last move!

Reality: Static Systems

- A deployed system is very hard to change
  - Serious consequences if attackers find a security hole in a widely deployed system
- Goal: Predict *in advance* what attackers might do and eliminate all security holes
- Reality: Have to think like an attacker
- Thinking like an attacker is not always easy
  - Can be fun to try to outwit the system
  - Or can be disconcerting to think about what could go wrong and who could get hurt
- What if you don't anticipate attacks?
  - Analog cellular phones in the 80's and 90's

Real-World Example: Analog Cellular

- 1970's: analog cellular had no security
  - Phones transmit ID/billing info in the clear
  - Assumption: attackers wouldn't bother to assemble equipment to intercept info…
- Attackers built "black boxes" to intercept and clone phones for fraudulent calling
  - Where's the best place to intercept?
  - Cellular operators completely unprepared
- Early 90's, US carriers losing >$1B/yr
  - 70% of LD cellular calls placed from downtown Oakland on Fri nights fraudulent
- Problems: huge capital investment/debt, 5–10 yrs & huge replacement cost

Lesson Learned

- Failing to anticipate types of attacks, or underestimating the threat, can be costly
- Security design requires studying attacks
  - Security experts spend a lot of time trying to come up with new attacks
  - Sounds counter-productive (why help the attackers?), but it is better to learn about vulnerabilities before the system is deployed than after
- If you know about the possible attacks in advance, you can design a system to resist those attacks
  - But, anything else is a toss of the dice…

A Process for Security Evaluation

- How do we think about the ways that an adversary might use to penetrate system security or otherwise cause mischief?
- We need a framework to help you think through these issues
- Start with *security requirements* or in other words:
  - What properties do we want the system to have, even when it is under attack?
  - What are we trying to protect from the attacker?
  - Or, to look at it the other way around, what are we trying to prevent?

Security Requirements

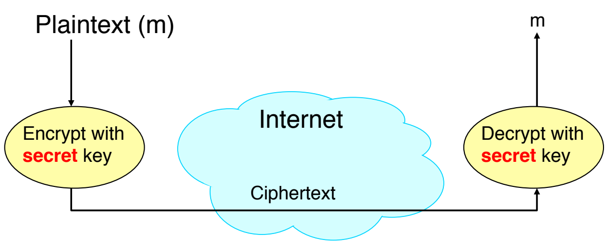
- Authentication
  - Ensures that a user is who is claiming to be
- Data integrity
  - Ensure that data is not changed from source to destination or after being written on a storage device
- Confidentiality
  - Ensures that data is read only by authorized users
- Non-repudiation
  - Sender/client can't later claim didn't send/write data
  - Receiver/server can't claim didn't receive/write data

Securing Communication: Cryptography

- Cryptography: *communication in the presence of adversaries*
- Studied for thousands of years
  - See the Simon Singh's *The Code Book* for an excellent, highly readable history
- Central goal: confidentiality
  - How to encode information so that an adversary can't extract it, but a friend can
- General premise: there is a key, possession of which allows decoding, but without which decoding is infeasible
  - Thus, key must be kept secret and not guessable

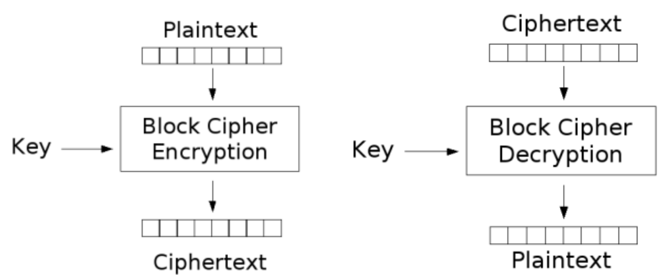
Using Symmetric Keys

- Same key for encryption and decryption



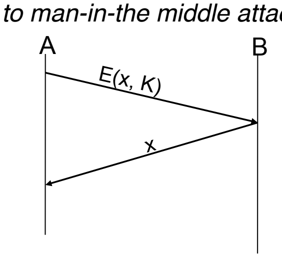
Symmetric Keys

- Can just XOR plaintext with the key
  - Easy to implement, but easy to break using frequency analysis
  - Unbreakable alternative: XOR with one-time pad
- More sophisticated (e.g., block cipher) algorithms
  - Works with a *block size* (e.g., 64 bits)
    - To encrypt a stream, can encrypt blocks separately, or link them



Authentication via Secret Key

- Main idea: entity proves identity by decrypting a secret encrypted with its own key
  - K – secret key shared only by A and B
- A can asks B to authenticate itself by decrypting a nonce, i.e., random value, x
  - Avoid replay attacks (attacker impersonating client or server)
- *Vulnerable to man-in-the middle attack*



Notation: E(m,k) – encrypt message m with key k

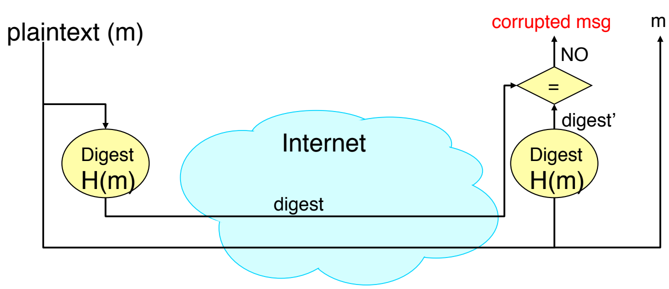
Symmetric Key Ciphers - DES & AES

- Data Encryption Standard (DES)
  - – Developed by IBM in 1970s, standardized by NBS/NIST
  - – 56-bit key (decreased from 64 bits at NSA's request)
  - – Still fairly strong other than brute-forcing the key space
    - » But custom hardware can crack a key in < 24 hours
  - – Today many financial institutions use Triple DES
    - = DES applied 3 times, with 3 keys totaling 168 bits
- Advanced Encryption Standard (AES)
  - Replacement for DES standardized in 2002
  - Key size: 128, 192 or 256 bits
- How fundamentally strong are they?
  - No one knows (no proofs exist)

Integrity: Cryptographic Hashes

- Basic building block for integrity: *hashing*
  - – Associate hash with byte-stream, receiver verifies match
    - » Assures data hasn't been modified, either accidentally – or maliciously
- Approach:
  - Sender computes a *digest* of message m, i.e., H(m)
    - » H() is a publicly known *hash function*
  - Send digest (d = H(m)) to receiver in a secure way, e.g.,
    - » Using another physical channel
    - » Using encryption
  - Upon receiving m and d, receiver re-computes H(m) to see whether result agrees with d

Using Hashing for Integrity



Standard Cryptographic Hash Functions

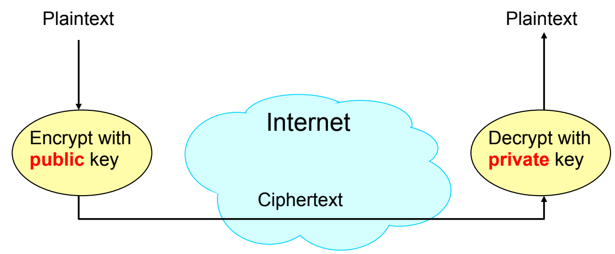
- MD5 (Message Digest version 5)
  - – Developed in 1991 (Rivest)
  - – Produces 128 bit hashes
  - – Widely used (RFC 1321)
  - – Broken (1996-2008): Attacks that find collisions
- SHA-1 (Secure Hash Algorithm)
  - – Developed by NSA in 1995 as successor to MD5
  - – Produces 160 bit hashes
  - – Widely used (SSL/TLS, SSH, PGP, IPSEC)
  - – Broken in 2005, government use discontinued in 2010
- SHA-2 (2001)
  - – Family of SHA-224, SHA-256, SHA-384, SHA-512

Asymmetric Encryption *(Public Key)*

- Idea: use two *different* keys, one to encrypt (*e*) and one to decrypt (*d*)
  - A key pair
- Crucial property: knowing *e* does not give away *d*
- Therefore *e* can be public: everyone knows it!
- If Alice wants to send to Bob, she fetches Bob's public key (say from Bob's home page) and encrypts with it
  - Alice can't decrypt what she's sending to Bob …
  - … but then, neither can anyone else (except Bob)

Public Key / Asymmetric Encryption

- Sender uses receiver's public key
  - Advertised to everyone
- Receiver uses complementary private key
  - Must be kept secret



Public Key Cryptography

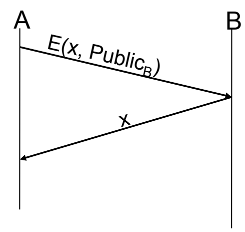
- Invented in the 1970s
  - *Revolutionized* cryptography
  - (Was actually invented earlier by British intelligence)
- How can we construct an encryption/decryption algorithm using a key pair with the public/private properties?
  - Answer: Number Theory
- Most fully developed approach: RSA
  - Rivest / Shamir / Adleman, 1977; RFC 3447
  - Based on modular multiplication of very large integers
  - Very widely used (e.g., ssh, SSL/TLS for `https`)

Properties of RSA

- Requires generating large, random prime numbers
  - Algorithms exist for quickly finding these (probabilistic!)
- Requires exponentiating very large numbers
  - Again, fairly fast algorithms exist
- Overall, much slower than symmetric key crypto
  - One general strategy: use public key crypto to exchange a (short) symmetric session key
    - » Use that key then with AES or such
- How difficult is recovering d, the private key?
  - Equivalent to finding prime factors of a large number
    - » Many have tried - believed to be very hard (= brute force only)
    - » (Though *quantum computers* can do so in polynomial time!)

Simple Public Key Authentication

- Each side need only to know the other side's public key
  - No secret key need be shared
- A encrypts a nonce (random number) x
  - Avoid replay attacks, e.g., attacker impersonating client or server
- B proves it can recover x
- A can authenticate itself to B in the same way



Notation: E(m,k) – encrypt message m with key k

Administrivia

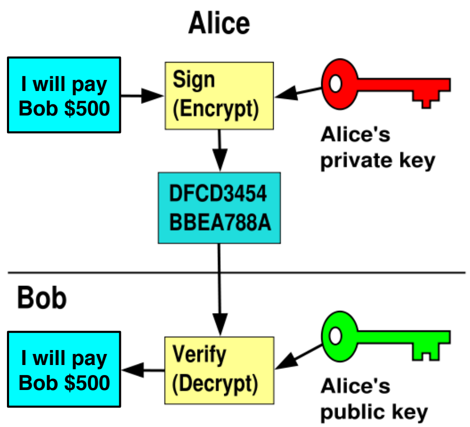
- Project 4 will be posted by Friday
  - Extends project 3 with replication and encryption using multiple servers and coordinated updates using 2PC
  - Design document due Mon 4/23 at 11:59PM
  - Code due Thu 5/3 at 11:59PM

5min Break

Non-Repudiation: RSA Crypto & Signatures

- Suppose Alice has published public key K_E
- If she wishes to prove who she is, she can send a message x encrypted with her private key K_D (i.e., she sends $E(x, K_D)$)
  - Anyone knowing Alice's public key K_E can recover x, verify that Alice must have sent the message
    - » It provides a signature
  - Alice can't deny it ⇒ non-repudiation

RSA Crypto & Signatures (cont'd)



Digital Certificates

- How do you know K_E is Alice's public key?
- Trusted authority (e.g., Verisign) signs binding between Alice and K_E with its private key $KV_{private}$
  - C = E({Alice, K_E}, $KV_{private}$)
  - C: digital certificate
- Alice: distribute her digital certificate, C
- Anyone: use trusted authority's KV_{public}, to extract Alice's public key from C
  - {Alice, K_E} = D(C, KV_{public})

Summary of Our Crypto Toolkit

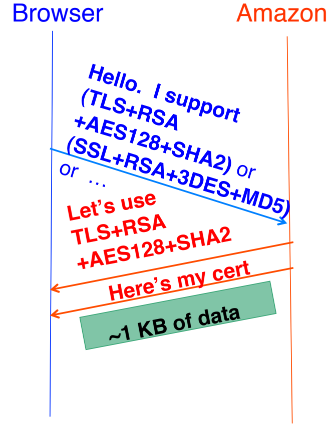
- If we can securely distribute a key, then
  - Symmetric ciphers (e.g., AES) offer fast, presumably strong confidentiality
- Public key cryptography does away with (potentially major) problem of secure key distribution
  - But: not as computationally efficient
    - » Often addressed by using public key crypto to exchange a session key
- Digital signature binds the public key to an entity

Putting It All Together - `HTTPS`

- What happens when you click on `https://www.amazon.com`?
- `https` = "Use HTTP over SSL/TLS"
  - SSL = Secure Socket Layer
  - TSL = Transport Layer Security
    - » Successor to SSL
  - Provides security layer (authentication, encryption) on top of TCP
    - » Fairly transparent to applications

HTTPS Connection (SSL/TLS) (cont'd)

- Browser (client) connects via TCP to Amazon's `HTTPS` server
- Client sends over list of crypto protocols it supports
- Server picks protocols to use for this session
- Server sends over its certificate
- (all of this is in the clear)



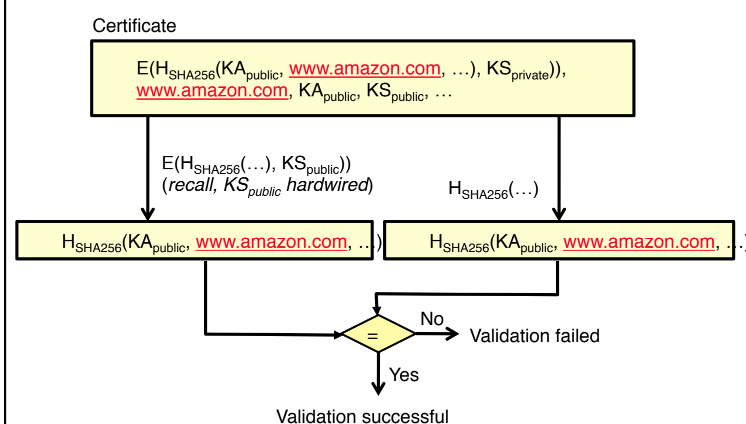
Inside the Server's Certificate

- Name associated with cert (e.g., Amazon)
- Amazon's RSA public key
- A bunch of auxiliary info (physical address, type of cert, expiration time)
- Name of certificate's signatory (who signed it)
- A public-key signature of a hash (SHA-256) of all this
  - Constructed using the signatory's private RSA key, i.e.,
  - Cert = $E(H_{SHA256}(KA_{public}$, www.amazon.com, …), $KS_{private})$)
    - » KA_{public}: Amazon's public key
    - » $KS_{private}$: signatory (certificate authority) public key
- …

Validating Amazon's Identity

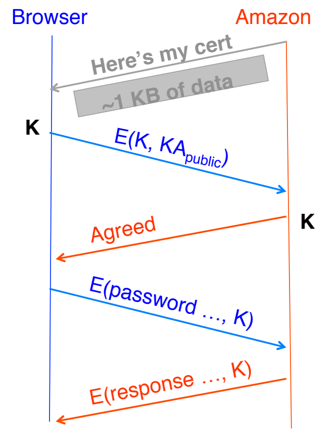
- How does the browser authenticate certificate signatory?
  - Certificates of several certificate authorities (e.g., Verisign) are hardwired into the browser (or OS)
- If it can't find the cert, then warns the user that site has not been verified
  - And may ask whether to continue
  - Note, can still proceed, just without authentication
- Browser uses public key in signatory's cert to decrypt signature
  - Compares with its own SHA-256 hash of Amazon's cert
- Assuming signature matches, now have high confidence it's indeed Amazon …
  - … <u>assuming signatory is trustworthy</u>
  - *DigiNotar CA breach (July-Sept 2011): Google, Yahoo!, Mozilla, Tor project, Wordpress, … (531 total certificates)*

Certificate Validation



HTTPS Connection (SSL/TLS) cont'd

- Browser constructs a random *session key* K used for data communication
  - Private key for bulk crypto
- Browser encrypts K using Amazon's public key
- Browser sends $E(K, KA_{public})$ to server
- Browser displays 
- All subsequent comm. encrypted w/ symmetric cipher (e.g., AES128) using key K
  - E.g., client can authenticate using a password



Authentication: Passwords

- Shared secret between two parties
- Since only user knows password, someone types correct password ⇒ must be user typing it
- Very common technique
- System must keep copy of secret to check against passwords
  - What if malicious user gains access to list of passwords?
    - » Need to obscure information somehow
  - Mechanism: utilize a transformation that is difficult to reverse without the right key (e.g. encryption)

Passwords: Secrecy

- Example: UNIX /etc/passwd file
  - passwd→one way transform(hash)→encrypted passwd
  - System stores only encrypted version, so OK even if someone reads the file!
  - When you type in your password, system compares encrypted version

Passwords: How easy to guess?

- Three common ways of compromising passwords
- Password Guessing:
  - Often people use obvious information like birthday, favorite color, girlfriend's name, etc…
  - Trivia question 1: what is the most popular password?
  - Trivia question 2: what is the next most popular password?
  - Answer: (from 32 million stolen passwords– Rockyou 2010) http://www.nytimes.com/2010/01/21/technology/21password.html
- Dictionary Attack (against stolen encrypted list):
  - Work way through dictionary and compare encrypted version of dictionary words with entries in `/etc/passwd`
  - http://www.skullsecurity.org/wiki/index.php/Passwords
- Dumpster Diving:
  - Find pieces of paper with passwords written on them
  - (Also used to get social-security numbers, etc.)

Passwords: How easy to guess? (cont'd)

- Paradox:
  - Short passwords are easy to crack
  - Long ones, people write down!
- Technology means we have to use longer passwords
  - UNIX initially required lowercase, 5-letter passwords: total of 26^5=10million passwords
    - » In 1975, 10ms to check a password→1 day to crack
    - » In 2005, .01µs to check a password→0.1 seconds to crack
  - Takes less time to check for all words in the dictionary!

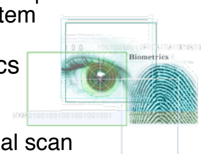
Passwords: Making harder to crack

- Can't make it impossible to crack, but can make it harder
- Technique 1: Extend everyone's password with a unique number ("Salt" – stored in password file)
  - Early UNIX uses 12-bit "salt" ➔dictionary attacks 4096x harder
  - Without salt, could pre-compute all the words in the dictionary hashed with UNIX algorithm (modern salts are 48-128 bits)
- Technique 2: Require more complex passwords
  - Make people use at least 8-character passwords with upper-case, lower-case, and numbers
    - » 70^8=6x10^{14}=6million seconds=69 days@0.01µs/check
  - Unfortunately, people still pick common patterns
    - » e.g. Capitalize first letter of common word, add one digit
- Technique 3: Delay checking of passwords
  - If attacker doesn't have access to `/etc/passwd`, delay every remote login attempt by 1 second
  - Makes it infeasible for rapid-fire dictionary attack

Passwords: Making harder to crack (cont'd)

- Technique 4: Assign very long passwords/passphrases
  - Can have more entropy (randomness→harder to crack)
  - Embed password in a smart card (or ATM card)
    - » Requires physical theft to steal password
    - » Can require PIN from user before authenticates self
  - Better: have smartcard generate pseudorandom number
    - » Client and server share initial seed
    - » Each second/login attempt advances random number

- Technique 5: "Zero-Knowledge Proof"
  - Require a series of challenge-response questions
    - » Distribute secret algorithm to user
    - » Server presents number; user computes something from number; returns answer to server; server never asks same "question" twice
  - Often performed by smartcard plugged into system



- Technique 6: Replace password with Biometrics
  - Use of one or more intrinsic physical or behavioral traits to identify someone
  - Examples: fingerprint reader, palm reader, retinal scan

Conclusion

- Distributed identity: Use cryptography
- Symmetrical (or Private Key) Encryption
  - Single Key used to encode and decode
  - Introduces key-distribution problem
- Public-Key Encryption
  - Two keys: a public key and a private key
  - Slower than private key, but simplifies key-distribution
- Secure Hash Function
  - Used to summarize data
  - Hard to find another block of data with same hash
- Passwords
  - Encrypt them to help hid them
  - Force them to be longer/not amenable to dictionary attack
  - Use zero-knowledge request-response techniques