

Cryptographic Keys  
CS 136  
Computer Security  
Peter Reiher  
October 16, 2014

# Outline

- Properties of keys
- Key management
- Key servers
- Certificates

# Introduction

- It doesn't matter how strong your encryption algorithm is
- Or how secure your protocol is
- If the opponents can get hold of your keys, your security is gone
- Proper use of keys is crucial to security in computing systems

# Properties of Keys

- Length
- Randomness
- Lifetime
- Secrecy

# Key Length

- If your cryptographic algorithm is otherwise perfect, its strength depends on key length
- Since the only attack is a brute force attempt to discover the key
- The longer the key, the more brute force required

# Are There Real Costs for Key Length?

- Generally, more bits is more secure
- Why not a whole lot of key bits, then?
- Some encryption done in hardware
  - More bits in hardware costs more
- Some software encryption slows down as you add more bits, too
  - Public key cryptography especially
- Some algorithms have defined key lengths only
- If the attack isn't brute force, key length might not help

# Key Randomness

- Brute force attacks assume you chose your key at random
- If attacker learns how you chose your key
  - He can reduce brute force costs
- How good is your random number generator?

# Generating Random Keys

- Well, don't use `rand()`<sup>1</sup>
- The closer the method chosen approaches true randomness, the better
- But, generally, don't want to rely on exotic hardware
- True randomness is not essential
  - Need same statistical properties
  - And non-reproducibility

<sup>1</sup>See <http://eprint.iacr.org/2013/338.pdf> for details



# Cryptographic Methods

- Start with a random number
- Use a cryptographic hash on it
- If the cryptographic hash is a good one, the new number looks pretty random
- Produce new keys by hashing old ones
- Depends on strength of hash algorithm
- Falls apart if any key is ever broken
  - Doesn't have *perfect forward secrecy*

# Perfect Forward Secrecy

- A highly desirable property in a cryptosystem
- It means that the compromise of any one session key will not compromise any other
  - E.g., don't derive one key from another using a repeatable algorithm
- Keys do get divulged, so minimize the resulting damage

# Random Noise

- Observe an event that is likely to be random
  - Physical processes (cosmic rays, etc.)
  - Real world processes (variations in disk drive delay, keystroke delays, etc.)
- Assign bit values to possible outcomes
- Record or generate them as needed
- More formally described as *gathering entropy*
- Keys derived with proper use of randomness have good perfect forward secrecy

# On Users and Randomness

- Some crypto packages require users to provide entropy
  - To bootstrap key generation or other uses of randomness
- Users do this badly (often very badly)
- They usually try to do something simple
  - And not really random
- Better to have crypto package get its own entropy

# Don't Go Crazy on Randomness

- Make sure it's non-reproducible
  - So attackers can't play it back
- Make sure there aren't obvious patterns
- Attacking truly unknown patterns in fairly random numbers is extremely challenging
  - They'll probably mug you, instead

# Key Lifetime

- If a good key's so hard to find,
  - Why every change it?
- How long should one keep using a given key?

# Why Change Keys?

- Long-lived keys more likely to be compromised
- The longer a key lives, the more data is exposed if it's compromised
- The longer a key lives, the more resources opponents can (and will) devote to breaking it
- The more a key is used, the easier the cryptanalysis on it
- **A secret that cannot be readily changed should be regarded as a vulnerability**

# Practicalities of Key Lifetimes

- In some cases, changing keys is inconvenient
  - E.g., encryption of data files
- Keys used for specific communications sessions should be changed often
  - E.g., new key for each phone call
- Keys used for key distribution can't be changed too often



# Destroying Old Keys

- Never keep a key around longer than necessary
  - Gives opponents more opportunities
- Destroy keys securely
  - For computers, remember that information may be in multiple places
    - Caches, virtual memory pages, freed file blocks, stack frames, etc.
  - Real modern attacks based on finding old keys in unlikely places

# Key Storage

- The flip side of destroying keys –
  - You'd better be sure you don't lose a key while you still need it
- Without the key, you can't read the encrypted data
  - Kind of a bummer, if you wanted to
- Key storage is one approach

# What Is Key Storage?

- Saving a copy of a cryptographic key “somewhere else”
- Securely store a key in some safe place
- If you lose it accidentally, get it back from storage location
- Prevents encrypted data from becoming unreadable

# Where Should You Store Keys?

- Must not be accessible to an attacker
  - Don't want him to get hold of all your keys
  - Don't want them readily available if your machine is hacked
- But relatively accessible when needed
- Usually on a separate machine

# How Do You Get Keys There?

- And back
- Each new key must be transported to the key server
- Not much saved if transport mechanism is compromised
- Need carefully designed/implemented mechanism for moving keys

# Key Secrecy

- Seems obvious
- Of course you keep your keys secret
- However, not always handled well in the real world
- Particularly with public key cryptography

# Some Problems With Key Sharing

- Private keys are often shared
  - Same private key used on multiple machines
  - For multiple users
  - Stored in “convenient” places
  - Perhaps backed up on tapes in plaintext form

# Why Do People Do This?

- For convenience
- To share expensive certificates
- Because they aren't thinking clearly
- Because they don't know any better
- A recent example:
  - RuggedCom's Rugged Operating System for power plant control systems
  - Private key embedded in executable



## To Make It Clear,

- **PRIVATE KEYS ARE PRIVATE!**
- They are for use by a single user
- They should never be shared or given away
- They must never be left lying around in insecure places
  - Widely distributed executables are insecure
  - Just because it's tedious to decipher executables doesn't mean can't be done
- The entire security of PK systems depends on the secrecy of the private key!

# Key Management

- Choosing long, random keys doesn't do you any good if your clerk is selling them for \$10 a pop at the back door
- Or if you keep a plaintext list of them on a computer on the net whose root password is "root"
- Proper key management is crucial

# Desirable Properties in a Key Management System

- Secure
- Fast
- Low overhead for users
- Scalable
- Adaptable
  - Encryption algorithms
  - Applications
  - Key lengths

# Users and Keys

- Where are a user's keys kept?
- Permanently on the user's machine?
  - What happens if the machine is cracked?
- But people can't remember random(ish) keys
  - Hash keys from passwords/passphrases?
- Keep keys on smart cards?
- Get them from key servers?

## Key Servers

- Special machines whose task is to generate, store and manage keys
- Generally for many parties
- Possibly Internet-wide
- Obviously, key servers are highly trusted

# Security of Key Servers

- The key server is the cracker's holy grail
  - If they break the key server, everything else goes with it
- What can you do to protect it?

# Security for Key Servers

- Don't run anything else on the machine
- Use extraordinary care in setting it up and administering it
- Watch it carefully
- Use a key server that stores as few keys permanently as possible
  - At odds with need for key storage
- Use a key server that handles revocation and security problems well

# Single Machine Key Servers

- Typically integrated into the web browser
  - Often called *key chains* or *password vaults*
- Stores single user's keys or passwords for various web sites
- Usually protected with an overall access key
- Obvious, encrypted versions stored on local disk



# Security Issues for Single Machine Key Servers

- Don't consider one that doesn't store keys encrypted
- Issues of single sign-on
  - If computer left unattended
  - In case of remote hacking
    - Anything done by your web browser is “you”

# Local Key Servers

- Can run your own key server machine
  - Stores copies of all keys you use
- Possibly creates keys when needed
- Uses careful methods to communicate with machines using it
- E.g., Oracle Key Manager 3
  - Primarily intended for tapes

# Key Storage Services

- Third party stores your keys for you
  - In encrypted form they can't read
- ANSI standard (X9.24) describes how third party services should work
- Not generally popular
- HyperSafe Remote Key System is one example
- Variants may become important for cloud computing

# Certificates

- A ubiquitous form of authentication
- Generally used with public key cryptography
- A signed electronic document proving you are who you claim to be
- Often used to help distribute other keys

# Public Key Certificates

- The most common kind of certificate
- Addresses the biggest challenge in widespread use of public keys
  - How do I know whose key it is?
- Essentially, a copy of your public key signed by a trusted authority
- Presentation of the certificate alone serves as authentication of your public key

# Implementation of Public Key Certificates

- Set up a universally trusted authority
- Every user presents his public key to the authority
- The authority returns a certificate
  - Containing the user's public key signed by the authority's private key
- In essence, a special type of key server

# Checking a Certificate

- Every user keeps a copy of the authority's public key
- When a new user wants to talk to you, he gives you his certificate
- Decrypt the certificate using the authority's public key
- You now have an authenticated public key for the new user
- Authority need not be checked on-line

# Scaling Issues of Certificates

- If there are 1-2 billion Internet users needing certificates, can one authority serve them all?
- Probably not
- So you need multiple authorities
- Does that mean everyone needs to store the public keys of all authorities?



# Certification Hierarchies

- Arrange certification authorities hierarchically
- Single authority at the top produces certificates for the next layer down
- And so on, recursively

# Using Certificates From Hierarchies

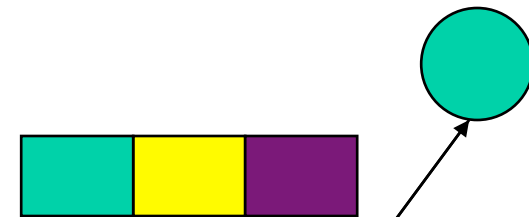
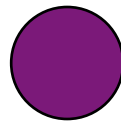
- I get a new certificate
- I don't know the signing authority
- But the certificate also contains that authority's certificate
- Perhaps I know the authority who signed this authority's certificate

# Extracting the Authentication

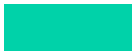
- Using the public key of the higher level authority,
  - Extract the public key of the signing authority from the certificate
- Now I know his public key, and it's authenticated
- I can now extract the user's key and authenticate it



# A Example

Alice gets a message with a certificate

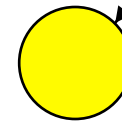




Then she uses  to check 

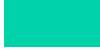
Should Alice believe that he's really  ?

So she uses  to check 

Give me a certificate saying that I'm 



Alice has never heard of  But she has heard of 

How can  prove who he is?



# Certification Hierarchies Reality

- Not really what's used
  - For the most part
- Instead, we rely on large numbers of independent certifying authorities
  - Exception is that each of them may have internal hierarchy
- Essentially, a big list
- Is this really better?

# Certificates and Trust

- Ultimately, the point of a certificate is to determine if something is trusted
  - Do I trust the request enough to perform some financial transaction?
- So, Trustysign.com signed this certificate
- How much confidence should I have in the certificate?

# Potential Problems in the Certification Process

- What measures did Trustysign.com use before issuing the certificate?
- Is the certificate itself still valid?
- Is Trustysign.com's signature/certificate still valid?
- Who is trustworthy enough to be at the top of the hierarchy?

# Trustworthiness of Certificate Authority


- How did Trustysign.com issue the certificate?
- Did it get an in-person sworn affidavit from the certificate's owner?
- Did it phone up the owner to verify it was him?
- Did it just accept the word of the requestor that he was who he claimed to be?
- Has authority been compromised?




# What Does a Certificate Really Tell Me?


- That the certificate authority (CA) tied a public/private key pair to identification information
- Generally doesn't tell me why the CA thought the binding was proper
- I may have different standards than that CA




# Showing a Problem Using the Example

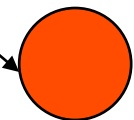
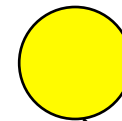
Alice likes how  verifies identity

But is she equally happy with how  verifies identity?



Does she even know how  verifies identity?

What if  uses 's lax policies to pretend to be  ?




# Another Big Problem


- Things change
  - E.g., recent compromise of Adobe private keys
- One result of change is that what used to be safe or trusted isn't any more
- If there is trust-related information out in the network, what will happen when things change?

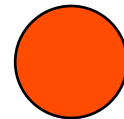
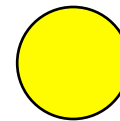
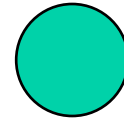
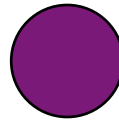
# Revocation

- A general problem for keys, certificates, access control lists, etc.
- How does the system revoke something related to trust?
- In a network environment
- Safely, efficiently, etc.
- Related to revocation problem for capabilities

# Revisiting Our Example

Someone discovers  
that  has obtained  
a false certificate for

How does Alice make sure  
that she's not accepting 's  
false certificate?



# Realities of Certificates

- Most OSes come with set of “pre-trusted” certificate authorities
- System automatically processes (i.e., trusts) certificates they sign
- Usually no hierarchy
- If not signed by one of these, present it to the user
  - Who always accepts it . . .

# An Example

- Firefox web browser
- Makes extensive use of certificates to validate entities
  - As do all web browsers
- Comes preconfigured with several certificate authorities
  - Over 200 of them

# Firefox Preconfigured Certificate Authorities

- Some you'd expect:
  - Microsoft, RSA Security, Verisign, etc.
- Some you've probably never heard of:
  - Unizeto Sp. z.o.o., Netlock  
Halozatbiztonsagi Kft., Chungwa  
Telecom Co. Ltd.



# The Upshot

- If Netlock Halozatbiztonsagi Kft. says someone's OK, I trust them
  - I've never heard of Netlock Halozatbiztonsagi Kft.
  - I have no reason to trust Netlock Halozatbiztonsagi Kft.
  - But my system's security depends on them

# The Problem in the Real World

- In 2011, a Dutch authority (DigiNotar) was compromised
- Attackers generated lots of bogus certificates signed by DigiNotar
  - “Properly” signed by that authority
  - For popular web sites
- Until compromise discovered, everyone trusted them

# Effects of DigiNotar Compromise

- Attackers could transparently redirect users to fake sites
  - What looked like Twitter was actually attacker's copycat site
- Allowed attackers to eavesdrop without any hint to users
- Apparently used by authorities in Iran to eavesdrop on dissidents

# How Did the Compromise Occur?

- DigiNotar had crappy security
    - Out-of date antivirus software
    - Poor software patching
    - Weak passwords
    - No auditing of logs
    - Poorly designed local network
  - A company providing security services paid little attention to security
- But how were you supposed to know that?*

# A Firefox Solution

- Certificate key pinning
- Code into the browser the “right” signing authority for particular sites
- So a certificate for Google signed by, say, DigiNotar gets rejected
- Currently only for a couple of big name web sites

# Another Practicality

- Certificates have expiration dates
  - Important for security
  - Otherwise, long-gone entities would still be trusted
- But perfectly good certificates also expire
  - Then what?

# The Reality of Expired Certificates

- When I hear my server's certificate has expired, what do I do?
  - I trust it anyway
  - After all, it's my server
- But pretty much everyone does that
  - For pretty much every certificate
- Not so secure

# The Core Problem With Certificates

- Anyone can create some certificate
- Typical users have no good basis for determining whose certificates to trust
  - They don't even really understand what they mean
- Therefore, they trust almost any certificate



# Should We Worry About Certificate Validity?

- Starting to be a problem
  - Stuxnet is one example
  - Compromise of DigiNotar and Adobe also
  - Increasing incidence of improper issuance, like Verisign handing out Microsoft certificates
- Not the way most attackers break in today
- With all their problems, still not the weakest link
  - But now being exploited, mostly by most sophisticated adversaries

# Heartbleed and Certificates

- OpenSSL relies on certificates and private keys for key distribution
- If Heartbleed compromised those, then serious problems
  - Just like DigiNotar, but for all certificate authorities at once
- Current evidence suggests Heartbleed can expose certificates
  - No evidence it ever did, but . . .
  - So we may need to change all certificates

# Should I Trust Crypto At All?

- Recent revelations suggest that the NSA can read many encrypted messages
- Experts think they mostly have compromised key selection/handling
  - Rather than the crypto itself
- But also possible that particular implementations have been “bugged”

# Some Practical Advice on Crypto

- From Bruce Schneier, who should know
- **Use crypto to protect your data**
- **Trust the math, not the program**
- **Be suspicious of commercial crypto**
  - **Especially from big companies**
- **Use public-domain crypto whenever possible**
  - **Especially standards requiring interoperability**