

More on Cryptography  
CS 136  
Computer Security  
Peter Reiher  
January 19, 2021

# Outline

- Properties of keys
- Certificates
- Key management

# Introduction

- It doesn't matter how strong your encryption algorithm 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
- Ciphers don't get cracked often, but keys get leaked all the time

# Properties of Keys

- Length
- Randomness
- Lifetime
- Secrecy
- Generation

# 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 that number

5924

1a401eca764d2588e5e02067518a1460e333f91d27089903f3658cde0e0bee79

- If the cryptographic hash is a good one, the new number looks pretty random
- Extract what you need from it

1a401eca764d2588e5e02067518a1460e333f91d27089903f3658cde0e0bee79

- Produce new keys by hashing old ones

6028fe87c4df480b74976ed9207142d4d20c4a271c851812d5771d38d54b2488

# Is This Method Secure?

- It depends (in part) on the strength of hash algorithm
  - Does it produces “random” numbers?
  - Is the part you extract itself random?
- Another problem: It falls apart if any key is ever broken or divulged
  - Doesn’t have *perfect forward secrecy*

# Perfect Forward Secrecy

- A highly desirable property in a cryptosystem
- It means that the compromise of a 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

SHA256 hash of 5924 is always  
**1a401eca764d2588e5e02067518a1460**  
**e333f91d27089903f3658cde0e0bee79**

# 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  
of random

But they can screw up, too: recent case of SecureRandom() bug causing skipping of entropy gathering.

- Users do this

Result: some cryptocurrencies used insecure keys.

- They usually try to make it as simple as possible

- And not really random

- Better to have crypto package get its own entropy

# Don't Go Crazy on Randomness

- Make sure you actually get a new key every time
  - A surprisingly common flaw
- 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 ever change it?
- How long should one keep using a given key?

# Why Change Keys?

- Long-lived keys are 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 VoIP call
- Keys used for key distribution can't be changed too often
- Some keys must be stored permanently or at least for a long time

# Key Storage

- Symmetric session keys
  - Avoid storing them permanently
  - Get rid of them when session ends
- Long term symmetric keys
  - E.g., for disk encryption
  - Safe storage is critical
- Private asymmetric keys
  - Usually require long-term storage
  - Safe storage is critical

# Storing a User's Keys

- Where are a user's keys kept?
  - Given they must be used often
- Permanently on the user's machine?
  - What happens if the machine is cracked?
- People can't remember random(ish) keys
  - Hash keys from passwords/passphrases?
- Keep keys on smart cards or hardware security enclaves?
- Get them from key servers?
  - Not a popular solution

# 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 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 Generation

- How do you get a key in the first place?
- For PK ciphers, predefined key generation algorithms
- For local use symmetric keys, random/pseudorandom methods
- How about for shared session keys?

# The Shared Session Key Problem

- Two parties wish to communicate securely across a network
- They don't have a symmetric key
- What do they do to get one?
  - Not just to generate one
  - But to make sure both have it
  - And no one else does

# Key Exchange Protocols

- Network protocols that solve this problem
- Some are based on trusted servers
- Others use PK methods
- There is a way to do it without either trusted servers or PK . . .

# Diffie/Hellman Key Exchange

- Securely exchange a key
  - Without previously sharing any secrets
    - No PK available
    - No other symmetric key either
- Using an insecure channel
  - I.e., the bad guys can hear everything the good guys tell each other

# Exchanging a Key in Diffie/Hellman

- How can Alice and Bob possibly get a shared secret session key in this case?
- First, they set things up
- Alice and Bob agree on a large prime  $n$  and a number  $g$ 
  - $g$  should be primitive root mod  $n$
- $n$  and  $g$  don't need to be secrets
  - Typically predefined in their software

# Exchanging the Key, Con't

- Alice secretly chooses a large random integer  $x$  and sends Bob  $X = g^x \bmod n$
- Bob secretly chooses a random large integer  $y$  and sends Alice  $Y = g^y \bmod n$  Alice knows  $x$
- Alice computes  $k = Y^x \bmod n$  Bob knows  $y$
- Bob computes  $k' = X^y \bmod n$
- $k$  and  $k'$  are both equal to  $g^{xy} \bmod n$
- But nobody else can compute  $k$  or  $k'$

# Why Can't Others Get the Secret?

- What do they know?
  - $n$ ,  $g$ ,  $X$ , and  $Y$
  - Not  $x$  or  $y$
- Knowing  $X$  and  $y$  gets you  $k$
- Knowing  $Y$  and  $x$  gets you  $k'$ 
  - $k = k'$
- Knowing  $X$  and  $Y$  gets you nothing
  - Unless you compute the discrete logarithm to obtain  $x$  or  $y$ 
    - Which is believed to be hard

# A Snake in Diffie/Hellman's Grass

- Diffie/Hellman “guarantees” that two parties share a secret
- But it doesn't guarantee who those two parties are
- How does Alice know whether the  $Y$  she heard actually was sent by Bob?
  - What if it was sent by an attacker?

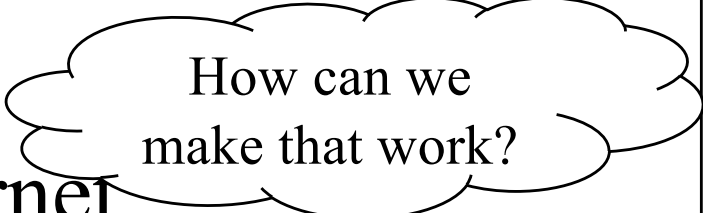


# The Core of the Problem

- Diffie/Hellman does not authenticate the parties
  - It does not provide evidence of who is talking
- To get the real effect you want, you need to authenticate the parties
  - Either during the key exchange
  - Or after you've got the encrypted session

# The Ubiquity of the Problem

- You actually always require authentication in any key distribution
- Otherwise, you don't know who else has the key
- A core problem for the Internet
  - At the base of securing all Internet commerce
- But Diffie-Hellman variants are used everywhere in the Internet<sup>o</sup>



How can we  
make that work?

# Authentication for Key Distribution

- You have a key on one machine
- You need to get it to another machine
- Since it's over a network, you need assurance that it's the right key
  - From the intended party
- How to get that assurance?

# Basic Approaches

- Key servers
- Certificates

# Key Servers

- Machines whose job it is to distribute keys to other machines
- Clients can authenticate themselves to the key server
- Key server can authenticate itself to the clients
- So clients need not authenticate other clients
- Note the transitive trust issue here
- Not the popular solution

# Certificates

- Server-less authentication
  - Used throughout the Internet
- What is a certificate?
- A signed electronic document proving you are who you claim to be
- Most often used to solve key distribution problem

# 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

# Certificates and Diffie-Hellman

- Certificates are used to solve the Diffie-Hellman authentication problem
- Each party includes a digital signature of its number ( $X$  or  $Y$ )
- The other party extracts the sender's PK from its certificate
- And then checks the digital signature
  - Giving assurance of who you're exchanging a key with

# Well, Sort Of

- Not what's usually done in the Internet
- Common case is ordinary user wants to talk to a big server
- Ordinary users don't have certificates
- Big servers do
- So the server signs and the user doesn't

*Consider the implications of that . . .*

# 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

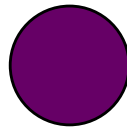
- I get a new certificate
- I don't know the signing authority
- But the certificate also contains that authority's certificate
- Perhaps I know the authority who signed this authority's certificate

# 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

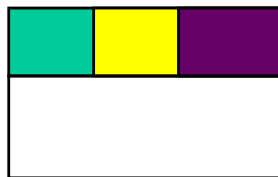
Alice gets a message with a certificate





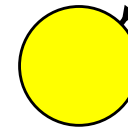
Then she uses   
to check 

Should Alice  
believe that he's  
really  ?

So she uses   
to check 




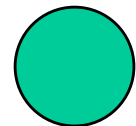
Alice has never  
heard of   
But she has  
heard of 



Give me a  
certificate  
saying that  
I'm 



How can   
prove who  
he is?





# Certification Hierarchies Reality

- We don't use a single certification hierarchy
- Instead, we rely on large numbers of independent certifying authorities
  - Each of which may have its own 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?

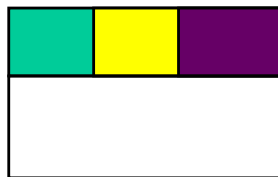
# What Does a Certificate Really Tell Me?


- 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

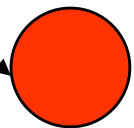
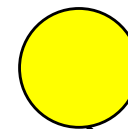
Alice likes how  verifies identity

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., 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, etc.
- How does the system revoke something related to t
- In a network environn
- Safely, efficiently, etc.
- Related to revocation problem for capabilities

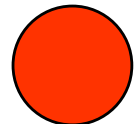
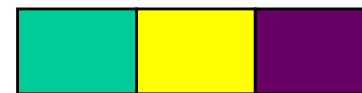
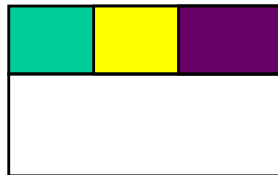
Revocation in a distributed system is always hard



# 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 (or browsers) 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
  - Nearly 100 of them
  - Many are at the top of a hierarchy
  - So there are several hundred certificates

# 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 Kft., Krajowa Izba Rozliczeniowa S.A.

# The Upshot

- If Netlock Kft. says someone's OK, I trust them
  - I've never heard of Netlock Kft.
  - I have no reason to trust Netlock 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?*



# 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
  - Recent report of attackers deceptively obtaining certificates for corporate executives
- 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

# One Approach

- OCSP
- An online system that indicates if certificates have been revoked
- Used in different ways by different OSes and browsers
- Obviously requires network access
  - And ability to reach OCSP site