

Key Management
CS 136
Computer Security
Peter Reiher
February 7, 2008

Outline

- Properties of keys
- Key management
- Key servers
 - Kerberos
- 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

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?

- Clearly, more bits is more secure
- Why not a whole lot of key bits, then?
- Much encryption done in hardware
 - More bits in hardware costs more
- Software encryption slows down as you add more bits, too
 - Public key cryptography costs are highly dependent on key length

Key Randomness

- Brute force attacks assume you chose your key at random
- If the attacker can get any knowledge about your mechanism of choosing a key, he can substantially reduce brute force costs
- How good is your random number generator?

Generating Random Keys

- Well, don't use `rand()`
- 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

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*

Random Noise

- Observe an event that is likely to be random
- Assign bit values to possible outcomes
- Record or generate them as needed
- Sources:
 - Physical processes (cosmic rays, etc.)
 - Real world processes (variations in disk drive delay, keystroke delays, etc.)

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.

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 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 Measures 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
- Use a key server that handles revocation and security problems well

Kerberos

- Probably the most widely used and well-known key server
- Originally developed at MIT
 - As part of Project Athena
- Uses trusted third parties
 - And symmetric cryptography
- Provides authentication in key service

The Kerberos Model

- Clients and servers sit on the network
- Clients want to interact securely with servers
 - Using a fresh key for each session
- Kerberos' job is to distribute keys to ensure that security
- Scalability is a concern
- Meant for single admin domain

Basic Kerberos Approach

- Servers provide real services
 - You need to authenticate to them
 - Authentication info is called a *ticket*
- Special servers provide authentication information
 - *Ticket-granting servers*
 - They need authentication information, too
- Kerberos server for primal authentication

Using Kerberos

- Client logs into system
- Contacts Kerberos server and authenticates himself
- Kerberos gives him a special ticket
- That ticket authenticates him to ticket-granting servers
 - Who give him more tickets

Using Kerberos, Con't

- Servers require tickets from clients to provide services
 - Tickets from particular ticket-granting servers
- Everything based on symmetric cryptography
- Timestamps used to invalidate tickets

Potential Weaknesses in Kerberos

- Timestamp-based attacks
- Password-guessing attacks
- Replacement of Kerberos software
 - The server is probably well protected
 - But are the clients?
 - Not unique to Kerberos

Certificates

- An increasingly popular 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 ~600 million 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
- The single authority at the top produces certificates for the next layer down
- And so on, recursively

Using Certificates From Hierarchies

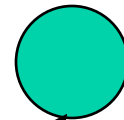
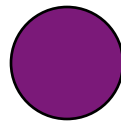
- I get a new certificate
- I don't know the signing authority
- But the certificate also contains that authority's certificate
- Perhaps I know the authority who signed this authority's certificate

Extracting the Authentication

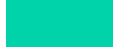
- Using the public key of the higher level authority,
 - Extract the public key of the signing authority from the certificate
- Now I know his public key, and it's authenticated
- I can now extract the user's key and authenticate it



A Example

Alice gets a message with a certificate

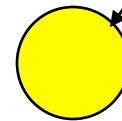




Then she uses 
to check 

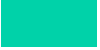
Should Alice believe that he's really  ?

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?

Certificates and Trust

- Ultimately, the point of a certificate is to determine if something is trusted
 - Do I trust the request 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?

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

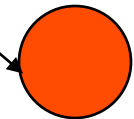
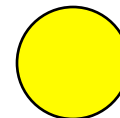
Alice likes how  verifies identity

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
- 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

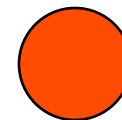
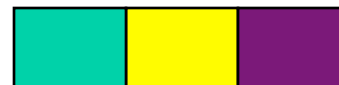
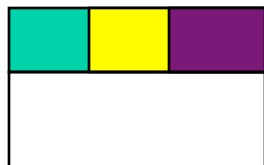
- A general problem for keys, certificates, access control lists, etc.
- How does the system revoke something related to trust?
- In a network environment
- Safely, efficiently, etc.

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 . . .

The Web of Trust Model

- Public keys are still passed around signed by others
- But your trust in others is based on your personal trust of them
 - Not on a formal certification hierarchy
 - “I work in the office next to Bob, so I trust Bob’s certifications”

Certificates in the Web of Trust

- Any user can sign any other user's public key
- When a new user presents me his public key, he gives me one or more certificates signed by others
- If I trust any of those others, I trust the new user's public key

Limitations on the Web of Trust

- The web tends to grow
 - “I trust Alice, who trusts Bob, who trusts Carol, who trusts Dave, . . . , who trusts Lisa, who trusts Mallory”
 - Just because Lisa trusts Mallory doesn’t mean I should
- Example of transitive trust problems
- Working system needs concept of degrees of trust

Advantages and Disadvantages of Web of Trust Model

- + Scales very well
- + No central authority
- + Very flexible
- May be hard to assign degrees of trust
- Revocation may be difficult
- May be hard to tell who you will and won't trust