Encryption and Secure Sessions

- 12E At-Rest Encryption
- 13G Encryption
- 13H Public Key Encryption
- 13I Secure Sessions

Bypassing Mediation

- OS can enforce authorization policy
  - control the operations processes can perform
- OS enforcement has exceptions and limits
  - privileged users can override file protection
  - passwords can be observed/stolen/guessed
  - bugs may enable malware to gain privileges
  - backups can be accessed w/o the OS
  - file systems can be accessed w/o OS
  - data stored elsewhere is beyond our protection

At-Rest Encryption

- added data protection, beyond file protection
- Disk (or file system) level
  - password must be given at boot or mount time
  - driver or file system does encrypt/decrypt
  - protects computer against unauthorized access
- File level
  - password must be given when file is opened
  - application (or library) does encrypt/decrypt
  - protects file against unauthorized access

Plaintext and Ciphertext

- *Plaintext* is the original form of the message (often referred to as *P*)

Transfer $100 to my savings account

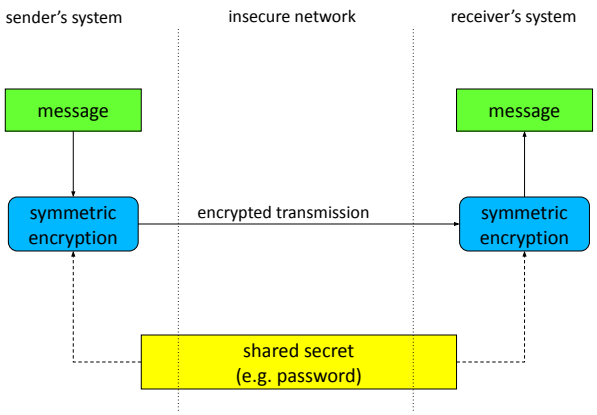
- *Ciphertext* is the encrypted form of the message (often referred to as *C*)

Sqzmredq #099 sn lx rzuhmfr zbbntms

Symmetric Cryptosystems

- *C = E(K,P)*
  - cipher text is encrypted using key and plain text
- *P = D(K,C)*
  - plain text is decrypted using key and cipher text
- *P = D(K, E(K,P))*
  - decryption is the inverse of encryption
  - *E()* and *D()* may be different functions
- Privacy: difficult to infer P from C without K
- Authenticity: difficult to forge P' without K

Simple Symmetric Encryption



Some Popular Symmetric Ciphers

- The Data Encryption Standard (DES)
  - the old US encryption standard (56-bit keys)
  - still fairly widely used, due to legacy
  - weak by modern standards
- The Advanced Encryption Standard (AES)
  - the current US encryption standard (128-256 bit keys)
  - probably the most widely used cipher
- Blowfish
  - popular, general purpose, public domain
  - relatively strong (32-448 bit keys)
- there are many others

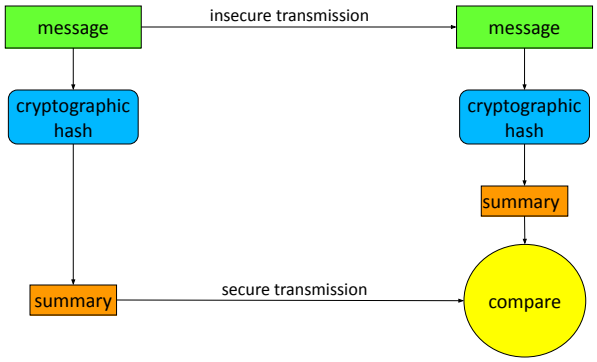
Symmetric Encryption

- Advantages
  - privacy and authentication in one operation
  - relatively efficient/inexpensive algorithms
  - no central authentication services required
- Disadvantages
  - scalability … establishing keys w/many partners
  - authentication … doesn't work w/new partners
  - privacy … shared secret known by one-too-many
  - weakness … short keys are subject to brute force

Tamper Detection: Cryptographic Hashes

- check-sums often used to detect data corruption
  - add up all bytes in a block, send sum along with data
  - recipient adds up all the received bytes
  - if check-sums agree, the data is probably OK
  - check-sum (parity, CRC, ECC) algorithms are weak
- cryptographic hashes are very strong check-sums
  - unique –two messages won't produce same hash
  - one way – cannot infer original input from output
  - well distributed – any change to input changes output
- much less expensive than encryption

Cryptographic Hash Authentication



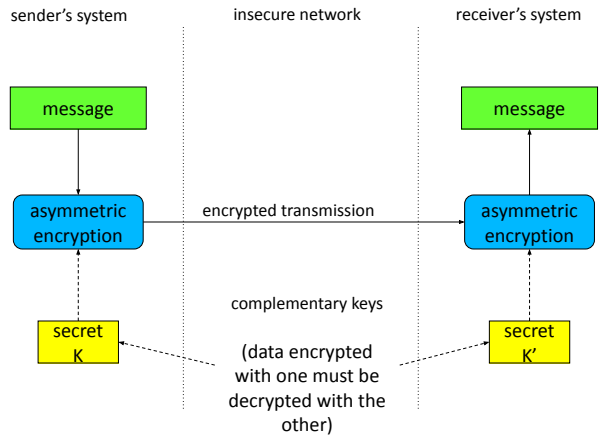
(Using Cryptographic Hashes)

- start with a message you want to protect
- compute a cryptographic hash for that message
  - e.g. using the Message Digest 5 (MD5) algorithm
- transmit the hash over a separate channel
- recipient computes hash of received text
  - if both hash results agree, the message is intact
  - else message has been corrupted/compromised
- hash must be delivered over a secure channel
  - encrypted, or otherwise separate and trusted
  - or else bad guy could just forge the validation hash

Asymmetric Cryptosystems

- Encryption and decryption use different keys
  - $C = E(K_E, P)$
  - $P = D(K_D, C)$
  - $P = D(K_D, E(K_E, P))$
- Often works the other way, too
  - $C = E(K_D, P)$
  - $P = D(K_E, C)$
  - $P = D(K_D, E(K_E, P))$
- Public Key (PK) encryption is such a system
  - K_E is called the *public key*, K_D is called the *private key*
  - it is very difficult to infer K_D from *D, E, C, P* and K_E

Asymmetric Encryption (public key)



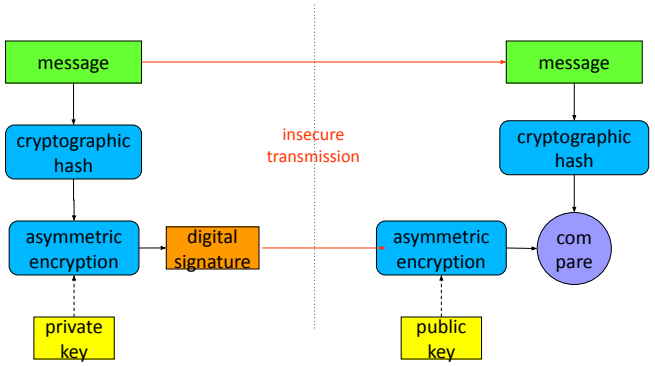
(Public Key Encryption)

- an asymmetric (two key) encryption technique
  - one key is private – (not shared) only key owner knows it
  - one key is public – it is advertised to the entire world
- it can be used to implement "your eyes only" privacy
  - encrypt a message with the recipient's public key
  - the message can only be decrypted with his private key
- it can be used to implement guaranteed signatures
  - sender encrypts message with his own private key
  - if it decrypts w/sender's public key, it must be from sender
- these can be combined for authentication + privacy

Example Public Key Ciphers

- RSA
  - the most popular public key algorithm
  - used on pretty much everyone's computer
- Elliptic curve cryptography
  - an alternative to RSA
  - tends to have better performance
  - not as widely used or studied

Digital Signatures



(Signing a message)

- encrypting a message with private key signs it
  - only you could have encrypted it, it must be from you
  - it has not been tampered with since you wrote it
- encrypting everything w/private key is a bad idea
  - if use a key too much, someone will eventually crack it
  - asymmetric encryption is extremely slow
- no need to encrypt whole message w/private key
  - compute a cryptographic hash of your message
  - encrypt the cryptographic hash with your private key
  - faster and safer than encrypting whole message

Using Digital Signatures

- much better than ink signatures or fingerprints
  - uniquely identify the document signer
  - uniquely identify the document that was signed
  - signature cannot be copied onto another document
- we know document has not been tampered with
  - we can recompute the cryptographic hash at any time
  - confirm it matches message the sender signed
  - sender cannot later claim not to have signed message
- digitally signed contracts can be legally binding
  - several states have passed such legislation

Can we trust public keys?

- if I have a public key
  - I can authenticate received messages
  - I know they were sent by the owner of the private key
- but how do I know who that person is?
  - can I be sure who a public key belongs to?
  - how do I know that this is really my bank's public key?
  - could some swindler have sent me his key instead?
- I would like a certificate of authenticity
  - a digital Notary stamp
  - certifying who the real owner of a public key is

Public Key Certificates

```
Certificate:
  Data:
    Version: v3; Serial Number: 3;
    Issuer: OU=Ace Certificate Authority, O=Ace Industry, C=US
    Validity: Not  After: Sun Oct 17 18:36:25 1999
    Subject: CN=Jane Doe, OU=Finance, O=Ace Industry, C=US
    Subject Public Key Info: Algorithm: PKCS #1 RSA Encryption
      Public Key: Modulus:
          00:ca:fa:79:98:8f:19:f8:d7:de:e4:49:80:48:e6:2a:2a:86:
          ...
  Signature:
    Algorithm: PKCS #1 MD5 With RSA Encryption
    Signature:
      6d:23:af:f3:d3:b6:7a:df:90:df:cd:7e:18:6c:01:69:8e:54:65:fc:06:
      ...
```


(What Is a PK Certificate?)

- Essentially a data structure
  - name and description of an actor
  - public key belonging to that actor
  - validity/expiration information
- Signed by someone I trust
  - whose public key I already have
  - a digital Notary Public
- Testifying that the actor owns the public key
  - and (by implication) the matching private key

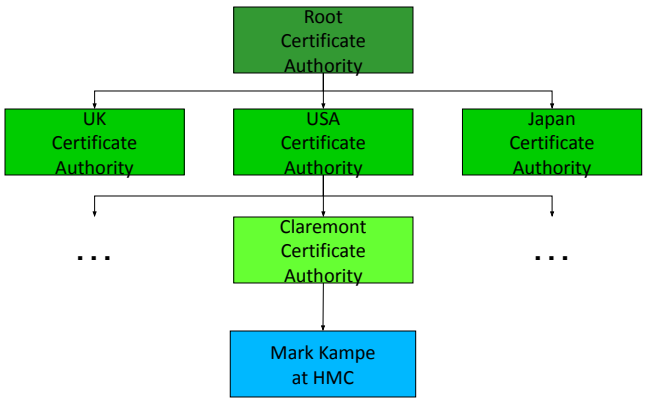
Using Public Key Certificates

- if I know public key of the authority who signed it
  - I can (locally) validate the signature is correct
  - I can tell the certificate has not been tampered with
- if I trust the authority who signed the certificate
  - I can trust they authenticated the certificate owner
  - e.g. we trust drivers licenses and passports
- but first I must know and trust signing authority
  - everybody knows and trusts RSA as an authority
  - does that mean that only RSA can sign certificates?

Delegated Authority

- I can accept certificates from a known authority
  - not practical for one authority to issue all certificates
  - how to validate certificates from unknown authority
- what if he has a certificate
  - that is signed by an authority I know and trust
  - that authorizes him to issue certificates
- if I trust RSA, I should also trust their "delegates"
  - perhaps I can also trust people they delegate
  - but I would need to see the entire chain of certificates

Certificate Authority Hierarchy



A Chicken and Egg Problem

- certificate is a formal introduction to a new partner
  - I can trust he is who he claims to be
  - if I can validate the certificate
  - by following the chain of delegated trust
- How do I trust the authority at the end of the chain?
- Ultimately through some other mechanism
  - OS or browser comes with an initial set of certificates
  - hand delivered (as in our IOT security project)
  - down-loaded, over a secure channel, from trusted site
  - you decide to accept a new certificate

A Principle of Key Use

- Both symmetric and PK crypto require secret keys
  - if key gets out, we lose both privacy and authentication
- The more you use a key, the less secure it becomes
  - the key stays around in various places longer
  - there are more opportunities for an attacker to get it
  - there is more incentive for attacker to get it
  - given enough time, any key can be brute forced
- Therefore:
  - ensure your keys are both long and high-entropy
  - use a given key as little as possible , change them often
    - the longer you keep it, the less you should use it

Practical Public Key Encryption

- Public Key Encryption algorithms are expensive
  - 10x to 100x as expensive as symmetric ones
  - key distribution is also complex and expensive
- We should use PKE as little as possible
  - for initial authentication/validation
  - to negotiate/exchange symmetric session keys
- Communication should use symmetric encryption
  - use short-lived, disposable, session keys
  - much less expensive to encrypt/decrypt

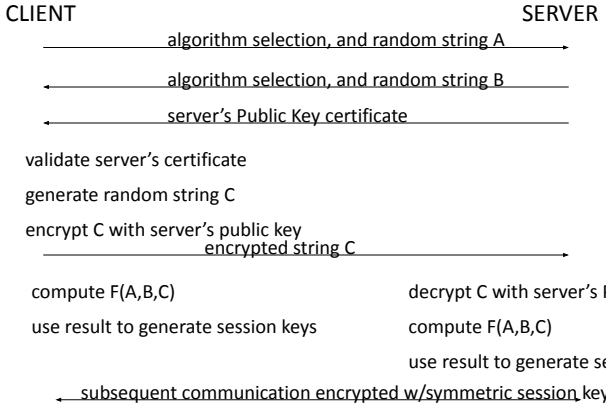
Symmetric *and* Asymmetric Encryption

- Use asymmetric to start the session
  - e.g. RSA or other Public Key mechanism
  - authenticate the parties
  - securely establish initial session key
- Use symmetric encryption for the session
  - e.g. DES or AES
  - very efficient algorithm based on negotiated key
- Periodically move to new session key
  - e.g. sequence based on initial session key
  - e.g. “switch to new key” message

example: Secure Socket Layer

- establishes secure two-way communication
  - privacy – nobody can snoop on conversation
  - integrity – nobody can generate fake messages
- certificate based authentication of server
  - client knows what server he is talking to
- optional certificate client authentication
  - if server requires authentication/non-repudiation
- uses PK to negotiate symmetric session keys
  - safety of public key, efficiency of symmetric

SSL session establishment



Reading and Assignments

Reading (after you return from Thanks Giving)**:**

- Arpaci C48 … Network File System
- Kampe: authentication services

Supplementary Slides

The Nature of PK Algorithms

- Usually based on some problem in mathematics
  - Like factoring extremely large numbers
- Security less dependent on brute force
- More on the complexity of the underlying problem
- Also implies choosing key pairs is complex and expensive

Security of PK Systems

- Based on solving the underlying problem
  - E.g., for RSA, factoring large numbers
- In 2009, a 768 bit RSA key was successfully factored
- Research on integer factorization suggests keys up to 2048 bits may be insecure
  - In 2013, Google went from 1024 to 2048 bit keys
- Size will keep increasing
- The longer the key, the more expensive the encryption and decryption

Diffie-Hellman Key Exchange

- each party has a secret, known only to them
  - call them *x* and *y*
- publicly negotiate two additional numbers
  - *n*: a large prime
  - *g*: a primitive root mod *n*
- each computes a number and sends to other
  - *X* = *g^x* mod *n*
  - *Y* = *g^y* mod *n*
- each can now compute a shared secret: *k*
  - *k* = *Y^x* mod *n*
  - *k* = *X^y* mod *n*

Signed Load Modules

- how do we know we can trust a program?
  - digital signatures can provide this
- designate a certification authority
  - perhaps the OS manufacturer (Microsoft, Sun, ...)
- they verify the reliability of the software
  - by code review, by testing, etc
  - sign certified module with their private key
- we can verify signature with their public key
  - proves the module was certified by them
  - proves the module has not been tampered with

example: Kerberos

- establishes secure two-way session
  - privacy – nobody can snoop on conversation
  - integrity – nobody can generate fake messages
- independent authentication of client & server
  - each side is assured of other side's identity
- based on secret symmetric encryption keys
  - DES key, known only to owner and Kerberos
- Kerberos generates symmetric session keys
  - distributes them securely to client and server

example: KERBEROS

- establishes a secure client/server session
  - each side is assured of partner's identity
  - session is secure against "man in middle" attacks
- digital signatures using symmetric encryption
  - every agent has a secret (symmetric) key
  - that key is known only to agent, and to KERBEROS
- request to KERBEROS encrypted w/client key
  - KERBEROS can decrypt it, authenticating requester
- response from KERBEROS is two-part work ticket
  - part 1: encrypted with client's key
    - symmetric session key, part 2 (to be forwarded to server)
  - part 2: encrypted with server's key
    - client ID, ticket duration, and symmetric session key

KERBEROS Work Tickets