Operating Systems Principles

Distributed Systems

Mark Kampe
(markk@cs.ucla.edu)

Distributed Systems

Goals of Distributed Systems

- scalability and performance
  - apps require more resources than one computer has
  - grow system capacity /bandwidth to meet demand
- improved reliability and availability
  - 24x7 service despite disk/computer/software failures
- ease of use, with reduced operating expenses
  - centralized management of all services and systems
  - buy (better) services rather than computer equipment
- enable new collaboration and business models
  - collaborations that span system (or national) boundaries
  - a global free market for a wide range of new services

Peter Deutsch's "Seven Falacies of Network Computing"

1. network is reliable
2. no latency (instant response time)
3. available bandwidth is infinite
4. network is secure
5. network topology & membership are stable
6. network admin is complete & consistent
7. cost of transporting additional data is zero

Bottom Line: true transparency is not achievable

Heterogenous Interoperability

- heterogenous clients
  - different instruction set architectures
  - different operating systems and versions
- heterogenous servers
  - different implementations
  - offered by competing service providers
- heterogenous networks
  - public and private
  - managed by different orgs in different countries

Fundmental Building Blocks Change

- the old model
  - programs run in processes
  - programs use APIs to access system resources
  - API services implemented by OS and libraries
- the new model
  - clients and servers run on nodes
  - clients use APIs to access services
  - API services are exchanged via protocols
- local is a (very important) special case

Changing Paradigms

- network connectivity becomes "a given"
  - new applications assume/exploit connectivity
  - new distributed programming paradigms emerge
  - new functionality depends on network services
- applications demand new kinds of services:
  - location independent operations
  - rendezvous between cooperating processes
  - WAN scale communication, synchronization

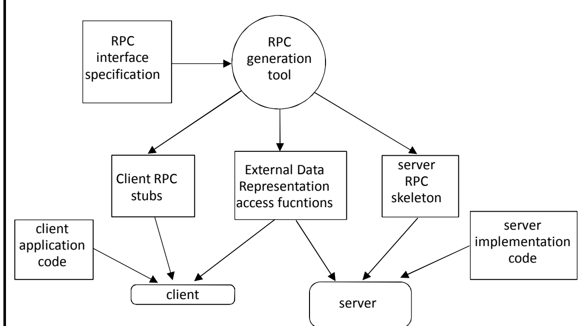
General Paradigm – RPC

- procedure calls – a fundamental paradigm
  - primary unit of computation in most languages
  - unit of information hiding in most methodologies
  - primary level of interface specification
- a natural boundary between client and server
  - turn procedure calls into message send/receives
- a few limitations
  - no implicit parameters/returns (e.g. global variables)
  - no call-by-reference parameters
  - much slower than procedure calls (TANSTAAFL)

Remote Procedure Call Concepts

- Interface Specification
  - methods, parameter types, return types
- eXternal Data Representation
  - machine independent data-type representations
  - may have optimizations for like client/server
- client stub
  - client-side proxy for a method in the API
- server stub (or skeleton)
  - server-side recipient for API invocations

Remote Procedure Calls – Tool Chain



(RPC – Key Features)

- client application links against local procedures
  - calls local procedures, gets results
- all rpc implementation is inside those procedures
- client application does not know about RPC
  - does not know about formats of messages
  - does not worry about sends, timeouts, resents
  - does not know about external data representation
- all of this is generated automatically by RPC tools
- the key to the tools is the interface specification

RPC is not a complete solution

- client/server binding model
  - expects to be given a live connection
- threading model implementaiton
  - a single thread service requests one-at-a-time
  - numerous one-per-request worker threads
- failure handling
  - client must arrange for timeout and recovery
- higher level abstractions
  - e.g. Microsoft DCOM, Java RMI, DRb, Pyro

Evolving Interaction Paradigms

- HTTP is becoming the preferred transport
  - well supported, tunnels through firewalls
- Simple Object Access Protocol (SOAP)
  - HTTP transport of XML encoded RPC requests
  - options for other transports and encodings
  - supports non-RPC interactions (e.g. transactions)
- REpresentational State Transfer (REST)
  - stateless, scalable, cacheable, layerable
  - operations limited to Create/Read/Update/Delete

Sample SOAP Request

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock/Surya">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```


Sample REST (json) Request

```
{
  "username" : "my_username",
  "password" : "my_password",
  "validation-factors" : {
      "validationFactors" : [
        {
          "name" : "remote_address",
          "value" : "127.0.0.1"
        }
      ]
  }
}
```


How does the OS ensure security?

- all key resources are kept inside of the OS
  - protected by hardware (mode, memory management)
  - processes cannot access them directly
- all users are authenticated to the OS
  - by a trusted agent that is (essentially) part of the OS
- all access control decisions are made by the OS
  - the only way to access resources is through the OS
  - we trust the OS to ensure privacy and proper sharing
- what if key resources could not be kept in OS?

Network Security – things get worse

- the OS cannot guarantee privacy and integrity
  - network transactions happen outside of the OS
- authentication
  - all possible agents may not be in local password file
- "man-in-the-middle" attacks
  - wire connecting the user to the system is insecure
- systems are open to vandalism and espionage
  - many systems are purposely open to the public
  - even supposedly private systems may be on internet

Man-in-the-Middle Attacks

- assume someone watching all network traffic
  - your traffic is being routed through many machines
  - most internet traffic is not encrypted
  - snooping utilities are widely available
  - passwords may be sent in clear text
- assume someone can forge messages from you
  - your traffic is being routed through many machines
  - some of them may be owned by bad people
  - they can hijack connection after you log in
  - they can replay previous messages, forge new ones

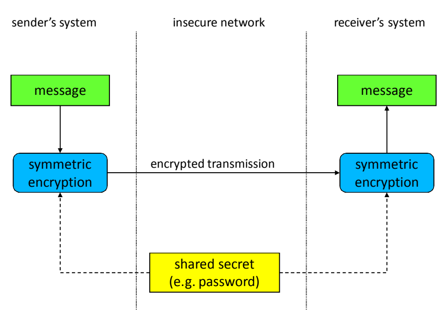
Goals of Network Security

- secure conversations
  - privacy: only you and your partner know what is said
  - integrity: nobody can tamper with your messages
- positive identification of both parties
  - authentication of the identity of message sender
  - assurance that a message is not a replay or forgery
  - non-repudiation: he cannot claim "I didn't say that"
- they must be assured in an insecure environment
  - messages are exchanged over public networks
  - messages are filtered through private computers

Elements of Network Security

- simple symmetric encryption
  - can be used to ensure both privacy and integrity
- cryptographic hashes
  - powerful tamper detection
- public key encryption
  - basis for modern digital privacy and authentication
- digital signatures and public key certificates
  - powerful tools to authenticate a message's sender
- delegated authority
  - enabling us to trust a stranger's credentials

Simple Symmetric Encryption



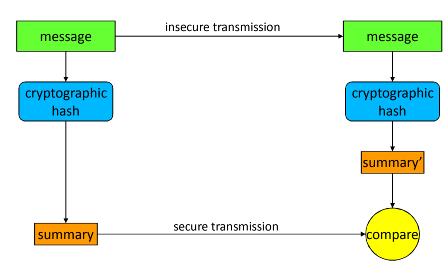
Symmetric Encryption

- simple fast algorithms
  - encryption and decryption use the same key
  - requires sender and receiver to both know the key
- symmetric encryption provides privacy
  - in order to decrypt the data, you must know the key
- symmetric encryption provides integrity
  - in order to generate false messages, you must know the key
- symmetric encryption algorithms are weak
  - if someone watches long enough they will figure out the key
  - a secret among two people is known by one too many

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

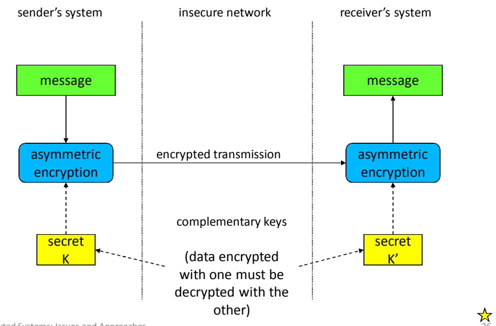
Cryptographic Hash Authentication



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 does same computation on 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
  - or else bad guy could just forge the validation hash

Asymmetric Encryption (public key)



Public Key Encryption

- an asymmetric (two key) encryption technique
  - one key is private – (not shared) only the 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 with sender's public key, it must be from sender
- these can be combined for authentication + privacy

How does the OS ensure security?

- all key resources are kept inside of the OS
  - protected by hardware (mode, memory management)
  - processes cannot access them directly
- all users are authenticated to the OS
  - by a trusted agent that is (essentially) part of the OS
- all access control decisions are made by the OS
  - the only way to access resources is through the OS
  - we trust the OS to ensure privacy and proper sharing
- what if key resources could not be kept in OS?

Network Security – things get worse

- the OS cannot guarantee privacy and integrity
  - network transactions happen outside of the OS
- authentication
  - all possible agents may not be in local password file
- "man-in-the-middle" attacks
  - wire connecting the user to the system is insecure
- systems are open to vandalism and espionage
  - many systems are purposely open to the public
  - even supposedly private systems may be on internet

Man-in-the-Middle Attacks

- assume someone watching all network traffic
  - your traffic is being routed through many machines
  - most internet traffic is not encrypted
  - snooping utilities are widely available
  - passwords may be sent in clear text
- assume someone can forge messages from you
  - your traffic is being routed through many machines
  - some of them may be owned by bad people
  - they can hijack connection after you log in
  - they can replay previous messages, forge new ones

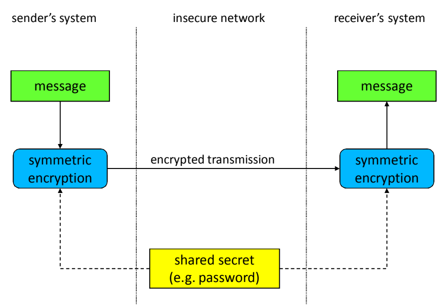
Goals of Network Security

- secure conversations
  - privacy: only you and your partner know what is said
  - integrity: nobody can tamper with your messages
- positive identification of both parties
  - authentication of the identity of message sender
  - assurance that a message is not a replay or forgery
  - non-repudiation: he cannot claim "I didn't say that"
- they must be assured in an insecure environment
  - messages are exchanged over public networks
  - messages are filtered through private computers

Elements of Network Security

- simple symmetric encryption
  - can be used to ensure both privacy and integrity
- cryptographic hashes
  - powerful tamper detection
- public key encryption
  - basis for modern digital privacy and authentication
- digital signatures and public key certificates
  - powerful tools to authenticate a message's sender
- delegated authority
  - enabling us to trust a stranger's credentials

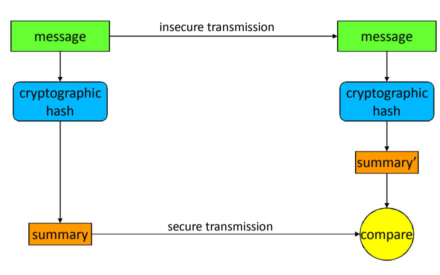
Simple Symmetric Encryption



(Symmetric Encryption)

- simple fast algorithms
  - encryption and decryption use the same key
  - requires sender and receiver to both know the key
- symmetric encryption provides privacy
  - in order to decrypt the data, you must know the key
- symmetric encryption provides integrity
  - in order to generate false messages, you must know key
- symmetric encryption algorithms are weak
  - if someone watches long enough they can determine key
  - a secret by shared two people is known by one too many

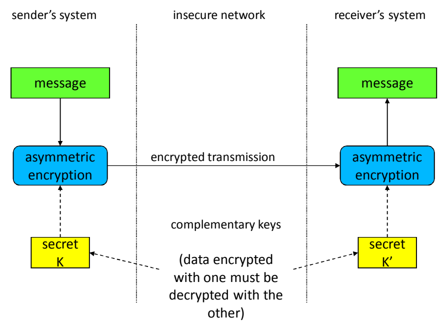
Cryptographic Hash Authentication



(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 does same computation on 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
  - or else bad guy could just forge the validation hash

Asymmetric Encryption (public key)



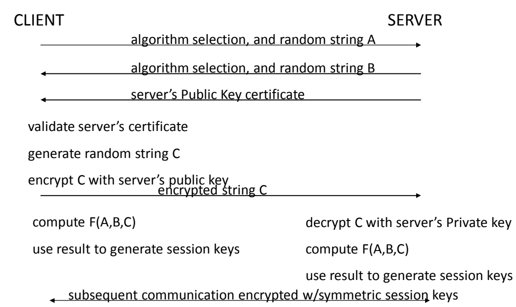
(Public Key Encryption)

- 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

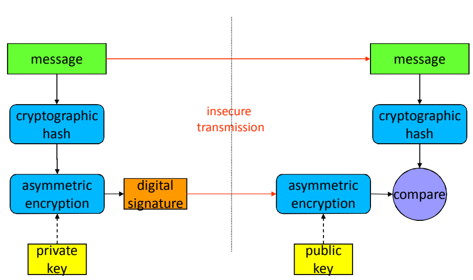
example: Secure Socket Layer

- 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 based authentication of client
  - if server requires authentication and non-repudiation
- uses symmetric encryption with session keys
  - safety of public key, efficiency of symmetric

SSL session establishment



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

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

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
  - guaranteeing 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:
      ...
```

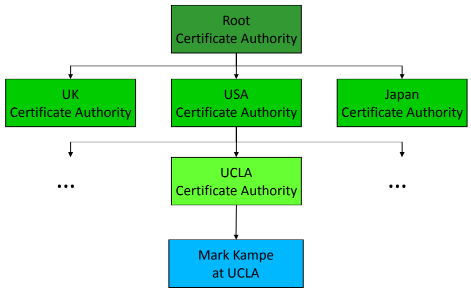

Using Public Key Certificates

- if I know public key of the authority who signed it
  - I can 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



Distributed Synchronization

- spatial separation
  - different processes run on different systems
  - no shared memory for (atomic instruction) locks
  - they are controlled by different operating systems
- temporal separation
  - can't "totally order" spatially separated events
  - before/simultaneous/after lose their meaning
- independent modes of failure
  - one partner can die, while others continue

Leases – more robust locks

- obtained from resource manager
  - gives client exclusive right to update the file
  - lease "cookie" must be passed to server w/update
  - lease can be released at end of critical section
- only valid for a limited period of time
  - after which the lease cookie expires
    - updates with stale cookies are not permitted
  - after which new leases can be granted
- handles a wide range of failures
  - process, client node, server node, network

Lock Breaking and Recovery

- revoking an expired lease is fairly easy
  - lease cookie includes a "good until" time
  - any operation involving a "stale cookie" fails
- this makes it safe to issue a new lease
  - old lease-holder can no longer access object
  - was object left in a "reasonable" state?
- object must be restored to last "good" state
  - roll back to state prior to the aborted lease
  - implement all-or-none transactions

Distributed Consensus

- achieving simultaneous, unanimous agreement
  - even in the presence of node & network failures
  - required: agreement, termination, validity, integrity
  - desired: bounded time
- consensus algorithms tend to be complex
  - and may take a long time to converge
- they tend to be used sparingly
  - e.g. use consensus to elect a leader
  - who makes all subsequent decisions by fiat

Typical Consensus Algorithm

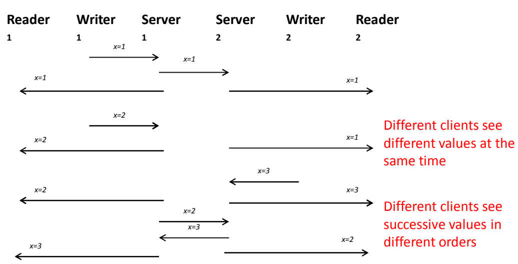
1. Each interested member broadcasts his nomination.
2. All parties evaluate the received proposals according to a fixed and well known rule.
3. After allowing a reasonable time for proposals, each voter acknowledges the best proposal it has seen.
4. If a proposal has a majority of the votes, the proposing member broadcasts a claim that the question has been resolved.
5. Each party that agrees with the winner's claim acknowledges the announced resolution.
6. Election is over when a quorum acknowledges the result.

Assignments

- for the next lecture:
  - Arpaci ch 48 … NFS (Network File System)
  - Arpaci ch 49 … AFS (Andrew File System)
  - Kerberos … authentication/authorization
  - Wikipedia: ACID Semantics

Supplementary Slides

Distributed Temporal Separation



1. The system does not have a scalar state. State is a vector.
2. There is no total ordering; There are only partial orderings.

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