Distributed Systems: Security

CS 111

Summer 2025

Operating System Principles

Peter Reiher

Outline

- Special security concerns for distributed systems
- Ensuring data integrity
- Distributed systems authentication
- Filtering technologies

Security for Distributed Systems

- Security is hard in single machines
- It’s even harder in distributed systems
- Why?

Why Is Distributed Security Harder?

- Your OS cannot guarantee privacy and integrity
  - Some activities happen outside of the machine and its OS
  - Should you trust where they happen?
- Authentication is harder
  - You may not even know who you might deal with
- The wire connecting the user to the system is insecure
  - Eavesdropping, replays, man-in-the-middle attacks
- Even with honest partners, hard to coordinate distributed security
- The Internet is an open network for all
  - Many sites on the Internet try to serve all comers
  - The core Internet makes no judgments on what’s acceptable
  - Even supposedly private systems may be on the Internet

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”
- Availability
  - The network and other nodes must be reachable when they need to be

Elements of Network Security

- Cryptography
  - Symmetric cryptography for protecting bulk transport of data
  - Public key cryptography primarily for authentication
  - *Cryptographic hashes* to detect message alterations
- Digital signatures and public key certificates
  - Powerful tools to authenticate a message's sender
- Filtering technologies
  - Firewalls and the like
  - To keep bad stuff from reaching our machines

Ensuring Data Integrity

- In distributed systems, we get lots of messages from remote sites
- How can we be sure they are the messages our partners actually sent?
  - As opposed to harmful messages sent by those who do not wish us well
- Remember, messages are just a bunch of bits
- Anyone can create any bunch of bits they want

The Problem

- Alice sent Bob a message containing a set of bits X
- Bob got a message apparently from Alice containing a set of bits Y
- Y may or may not be the same as X
- How can Bob tell with high confidence that they are (or are not) the same?

Checksums

- An old technology for detecting data corruption
  - Typically intended for cases of accidental corruption, not malice
- Add up all the bits in the data
- Save the integer result of the addition
- Recalculate the sum to test for corruption
- If bits are randomly flipped, then probably the original sum and the new sum differ

Cryptographic Hashes

- Check-sum (parity, CRC, ECC) algorithms work well for random accidental alterations
- They are weak against malice
  - Too easy for an attacker to make changes that don't change the checksum
- Cryptographic hashes are very strong check-sums
  - Unique –two messages vanishingly unlikely to produce same hash
    - Particularly hard to find two messages with the same hash
  - One way – cannot infer original input from output
  - Well distributed – any change to input changes output

Using Cryptographic Hashes

- Start with a message you want to protect
- Compute a cryptographic hash for that message
  - E.g., using the Secure Hash Algorithm 3 (SHA-3)
- Transmit the hash securely
- Recipient does same computation on received text
  - If both hash results agree, the message is intact
  - If not, the message has been corrupted/compromised

Secure Hash Transport

- Why must the hash be transmitted securely?
  - Cryptographic hashes aren't keyed
  - So anyone can produce them
    - Including a bad guy
- How to transmit hash securely?
  - Encrypt it
  - Unless secrecy required, cheaper than encrypting entire message

Sounds Like a Job For PK

- We’re not really concerned with the secrecy of our encrypted hash
- We only care about its authenticity
- So the sender could encrypt the hash with its private key
- And the receiver could ensure no tampering with the sender’s public key
- If only we could be sure we had the right public key . . .

An Important Public Key Issue

- If I have someone’s public key
  - I can authenticate messages I receive from them
  - I know they were sent by the owner of the private key
- But how can I be sure who owns the private key?
  - How do I know that this is really my bank's public key?
  - Could some swindler have sent me his public key instead?
- I can get Microsoft’s public key when I first buy their OS
  - So I can verify their load modules and updates
  - But how to handle the more general case?

Possible Approaches

1. Set up on-line trusted authorities that I can ask for public keys
   – E.g., "tell me what Amazon's public key is"
2. Create digital documents containing people's public keys
   – With built-in authentication
   – Called *certificates*

- Certificates are the popular option

What Is a Certificate?

- Essentially a data structure
- Containing an identity and a matching public key
  - And usually other information
  - Like an expiration date
- Also containing a *digital signature* of those items
  - Encrypted evidence that something is true
- Signature usually signed by someone I trust
  - And whose public key I already have

Using Public Key Certificates

- If I know public key of the authority who signed it
  - I can validate that the signature is correct
  - And that 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
  - Which really means I know and trust their public key

I can even do it off-line

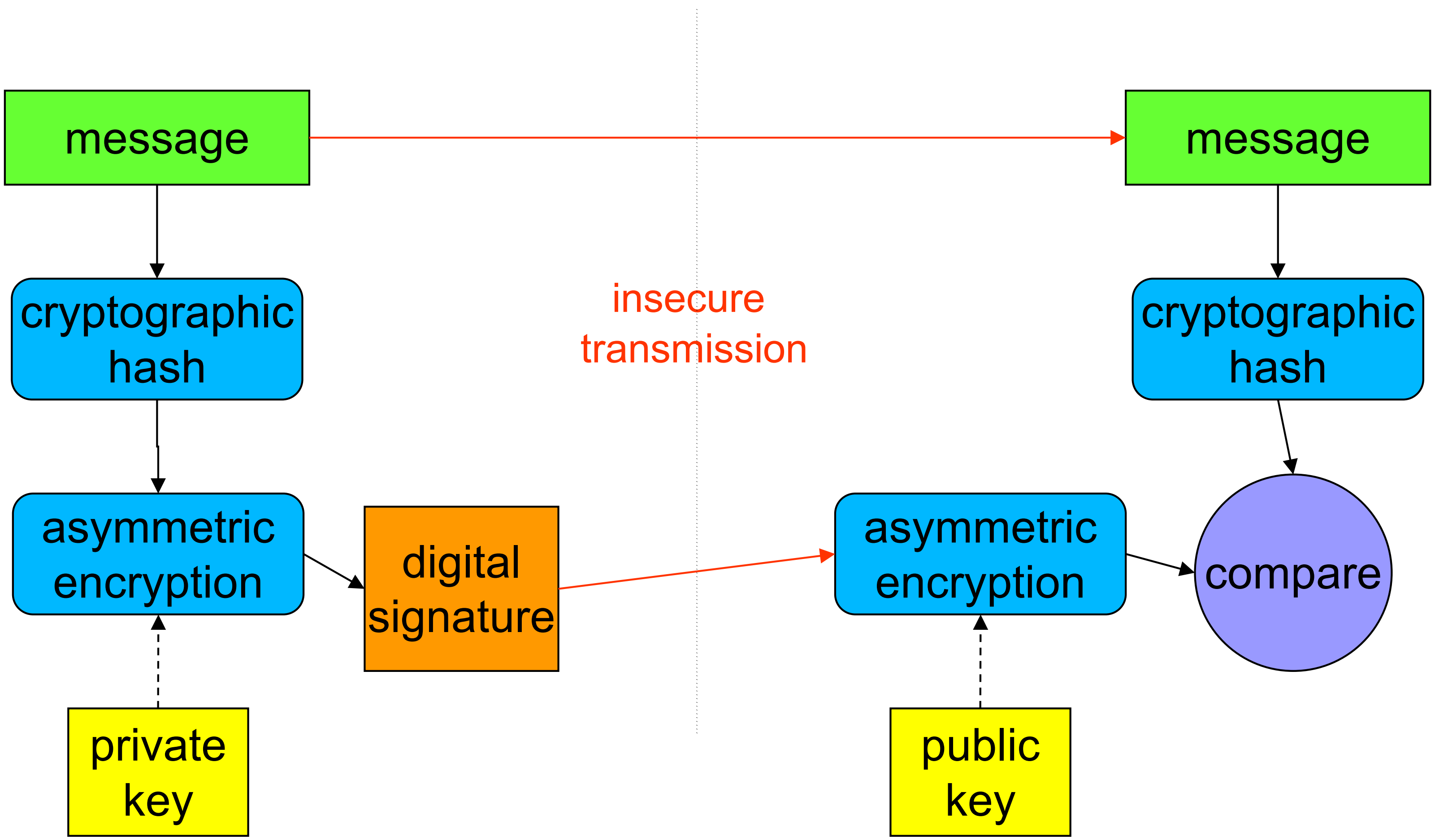
A Chicken and Egg Problem

- I can learn the public key of a new partner using his certificate
- But to use his certificate, I need the public key of whoever signed it
- So how do I get <u>that</u> public key?
- Ultimately, *out of band*
  - Which means through some other means
- Commonly by having the key in a trusted program, like a web browser
- Or hand delivered

Back To Our Original Problem

- We wanted to prove to a remote site that we created a piece of data
  - And that it hasn't been altered in transit
- Cryptographic hashes will allow receivers to verify that the data hasn't been changed
  - Provided the hash itself hasn't been changed
- Encrypting the hash with one's private key will prevent alteration of the hash
  - And show that I created the hash and data
- ***Digital signatures***

Digital Signatures



For Example,

- How do we know we can trust a software update?
  - Is it really the new update to Windows, or actually dangerous, evil code?
  - Digital signatures can answer this question
- Designate a certification authority
  - Perhaps the OS manufacturer (Microsoft, Apple, ...)
- They verify the reliability of the software
  - By code review, by testing, etc.
  - They sign a certified module with their private key
- We can verify the signature with their public key
  - Which we got from a certificate
  - And checked using the public key of the signing authority
- That proves the module was certified by them
- And that the module hasn’t been tampered with

Why Bother With the Hash?

- Given we know someone's public key, why create a hash and encrypt that?
- Why not just encrypt the entire message with the private key?
- The receiver could check authenticity and integrity by decrypting with the public key
- But asymmetric cryptography is very, very computationally expensive
  - Hashes are small
  - Full messages could be <u>very</u> large

Distributed Systems Authentication

- We can use similar methods for authentication as in single machine systems
  - Passwords, biometrics, etc.
- But there are new concerns when we are authenticating across the network
  1. Has authentication evidence been tampered with in transit?
  2. Do we fully trust the remote machine providing that evidence?

Solving the Problems

- Most (not all) security problems in networking are solved with cryptography
- Problem 1 can be solved by encrypting the transmission
  - If tampering occurs, proper use of crypto detects it
- We would still need to solve the key distribution issues, of course
  - Maybe certificates can help with that?

What About Problem 2?

- In a general distributed system, we are cooperating with other computers
  – But that doesn't mean we trust them fully
  – Not as much as we trust our own operating system
- The remote computers might be working with many different users
  – And we might have varying trust for those users
- If a partner computer asks us to do something, it may depend on which remote user is asking

Authenticating Remote Users

- Distributed system authentication is typically done in one of two ways

1. The remote site authenticates the party and sends encrypted confirmation
   – The local site itself need not verify the identity
2. The remote site sends an encrypted form some kind of authentication evidence
   – Which the remote site might not have checked
   – Which is then analyzed by the local site

The First Approach

- The remote site authenticates its local user
- It sends an authentication message to our site
  - Maybe with an indication of the user's identity
  - Maybe with an indication of what the user is allowed to do (perhaps without identity)
- These indications must be cryptographically signed by the remote site
- This approach implies we totally trust the remote site

The Second Approach

- The local site demands authentication information from the remote user
  – Via message, of course
- The remote user provides that information
  – Almost always through the remote computer
  – Also via message
- The information should be cryptographically protected in transit
- Sounds like we don’t need to trust the remote computer, in this case

But . . .

- The remote user must create authentication information
  – Generally using resources on the remote computer
  – Which are under control of that computer's OS
- Which means the remote computer learns the authentication information
  – And thus might be able to fake it in the future
  – Pretending to be the remote user itself
- Perhaps we can solve this with the right form of challenge/response authentication

Ongoing Distributed System Authentication

- Many distributed system operations span multiple messages
  - And perhaps hours or days of time
- Must we ask for passwords or challenge responses or biometrics every time?
  - That would be expensive, though safe
- Why not ask once?
  - And then determine that subsequent communications are with the same party?

This should sound familiar

How To Do It?

- Authenticate using something like we've just discussed
- Once the remote party is authenticated, set up a symmetric key between you and them
  - Known to no one else
- Encrypt all future communications with that key
- Only your authenticated partner could have created messages with that key
  - So any properly encrypted message is authentic

There Are Complexities

- Can we prevent replays?
  - Where an attacker copies an authentic message when it is sent
  - And sends it again
- Can we handle lost or dropped messages?
  - Which attackers may cause to happen
- How long should we accept the authentication?
  - For a few messages?
  - For an entire session?
  - Forever?

Transport Layer Security (TLS)

- A general solution for securing network communication
- Built on top of existing socket IPC
- Establishes a secure link between two parties
  - Privacy – nobody can snoop on conversation
  - Integrity – nobody can generate fake messages
- With some degree of authentication
  - How much varies by need and situation
- PK used to distribute a symmetric session key
  - New key for each new socket
- Rest of data transport switches to symmetric crypto
  - Giving safety of public key and efficiency of symmetric

TLS Authentication

- TLS can use PK to authenticate both parties
  - Sender authenticated to receiver
    - Is this my customer or an evil hacker?
  - Receiver authenticated to sender
    - Am I really talking to my merchant or an evil hacker?
- Typically it's only the latter
- Why?
- Who has a public key of their own?
  - Amazon, Microsoft, Google, etc. do
  - But do you or your friend or your grandmother?

So What Does Amazon Do?

- Sometimes they don't care
- If some random party wants to browse their web pages, so what?
- If they care, they require authentication after TLS has been set up
  - Perhaps by password or some other mechanism
  - Since that later authentication is over TLS, they can trust it and use it for ongoing communications

But I Don't Always Have to Log In

- If you interact with Amazon or Facebook or Google, you don't always provide a password
- I logged in three months ago and Amazon still considers me authentic
  - Even if I shut down my browser since then
  - Implying I closed the secure socket
- How's that work?
- Cookies

Cookies For Authentication

- Let's say Amazon has authenticated you
  – Maybe via password
- Amazon creates an encrypted piece of data
  – Indicating who you are
- Amazon sends it to you in the form of a *cookie*
- Cookies are pieces of data that your web browser sends back to web sites
- So when you interact again with Amazon, you automatically send the cookie they gave you
  – Which authenticates you

Filtering Technologies

- Cryptography doesn't solve all your distributed system security problems
- Sometimes you might not want messages
  - Because they might be damaging
  - Because they might interfere with legitimate work
  - Because there are just too many of them
- The Internet likes to deliver messages, good or bad
  - But we don't always like receiving them

What Needs to Be Filtered

- A computer is commonly part of a local area network (LAN)
  - A set of computers all connected via some networking technology
- Which in turn connects to the Internet as a whole
- Most filtering issues involve bad stuff coming in via the Internet
  - Less often bad stuff coming from the your LAN partners

Filtering Internet Traffic

- The Internet won't do it for you
- But your LAN typically connects to the Internet at only a few points
  - So all messages coming in to you and your LAN partners enter via those points
- So you can filter at those points
- Pass all messages entering your LAN through a special machine
  - Which can drop the stuff you don't want

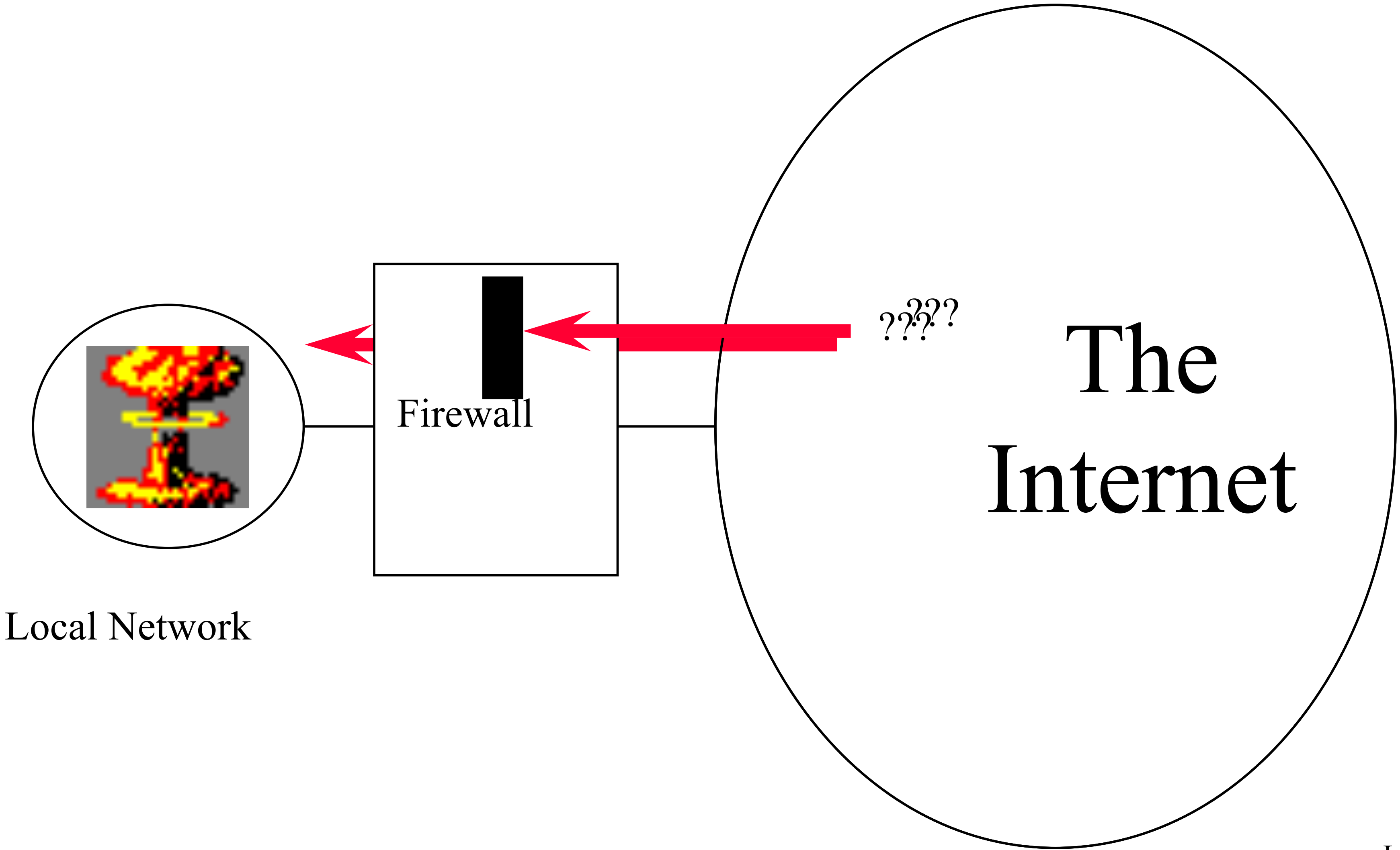
Firewalls

- A machine to protect a network from malicious external attacks
- Typically a machine that sits between a LAN/WAN and the Internet
- Running special software to regulate network traffic

The Brass Tacks of Firewalls

- What do they really do?
- Examine each incoming packet
- Decide to let the packet through or drop it
  - Criteria could be simple or complex
- Perhaps log the decision
- Maybe send rejected packets elsewhere

Typical Use of a Firewall



Firewalls and Perimeter Defense

- Firewalls implement a form of security called *perimeter defense*
- Protect the inside of something by defending the outside strongly
  - The firewall machine is often called a *bastion host*
- Control the entry and exit points
- If nothing bad can get in, I'm safe, right?

Weaknesses of Perimeter Defense Models

- Breaching the perimeter compromises all security
- Generally hard to impossible to create a perfect perimeter
  - If you keep all bad stuff out,
  - You keep much good stuff out
- Perimeter defense is part of the solution, not the entire solution

Using Multiple Firewalls

- Non-trivial networks often divide themselves into several subnetworks
- Often firewalls are placed between all the local subnetworks
  - And, of course, at Internet connection points
- For example, a public facing web site is behind one firewall
- To get to the company's development network, you must also go through a second firewall

Zero Trust Architectures

- A more secure way of protecting distributed systems
- Avoid having any node in your system automatically trust any other node
  - E.g., don’t just do something because a partner node asked
- Perform authentication and authorization on all remote requests before fulfilling them
- Likely to be more expensive than traditional approaches (but definitely more secure)

Conclusion

- Distributed systems face security challenges because you can’t control remote machines
- Trust issues are more complex in such systems
- You need authentication to determine who is communicating with you
  - Often based on cryptography
- You need privacy and integrity mechanisms
  - Usually based on cryptography
- Filtering technologies like firewalls can reduce load and risk