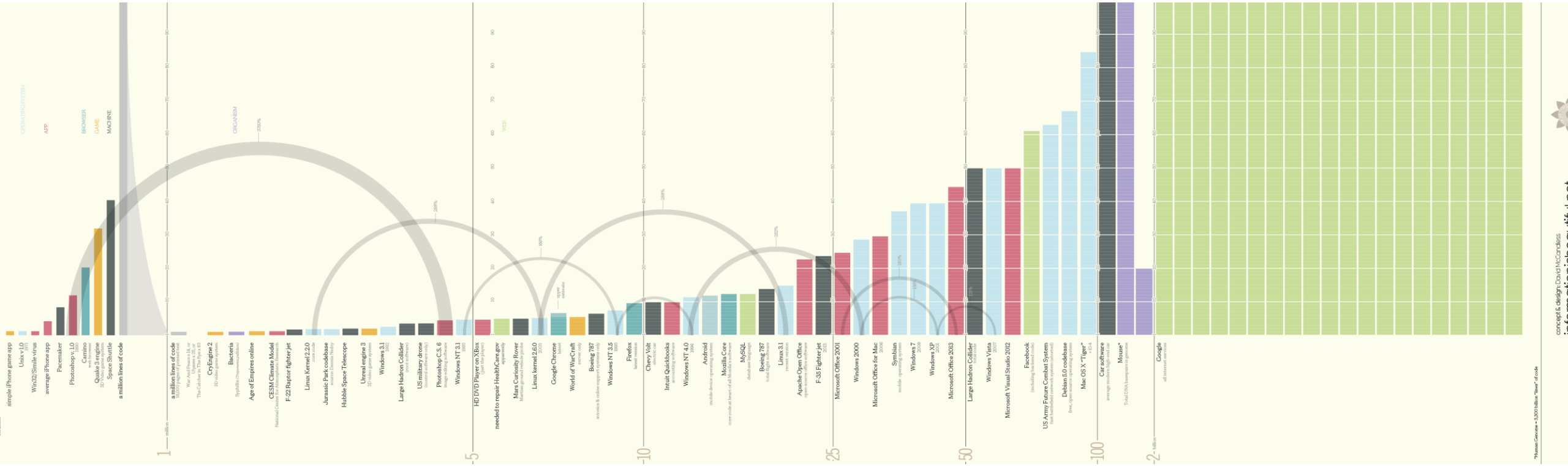
CSSE6400: Software Architecture

Quality attribute: Security

Guangdong Bai

Increasing complexity of modern software



Heartbleed (2014)

Nobody can keep online records safe (2017)

Previous Close	96.66	Market Cap	11.192B
Open	94.40	Beta	1.06
Bid	0.00 x 0	PE Ratio (TTM)	19.69
Ask	0.00 x 0	EPS (TTM)	4.72
Day's Range	90.72 - 95.69	Earnings Date	Oct 24, 2017 - Oct 30, 2017
52 Week Range	89.59 - 147.02	Dividend & Yield	1.56 (1.58%)
Volume	16,707,681	Ex-Dividend Date	2017-08-23
Avg. Volume	1,787,251	1y Target Est	136.92

Why do these happen?

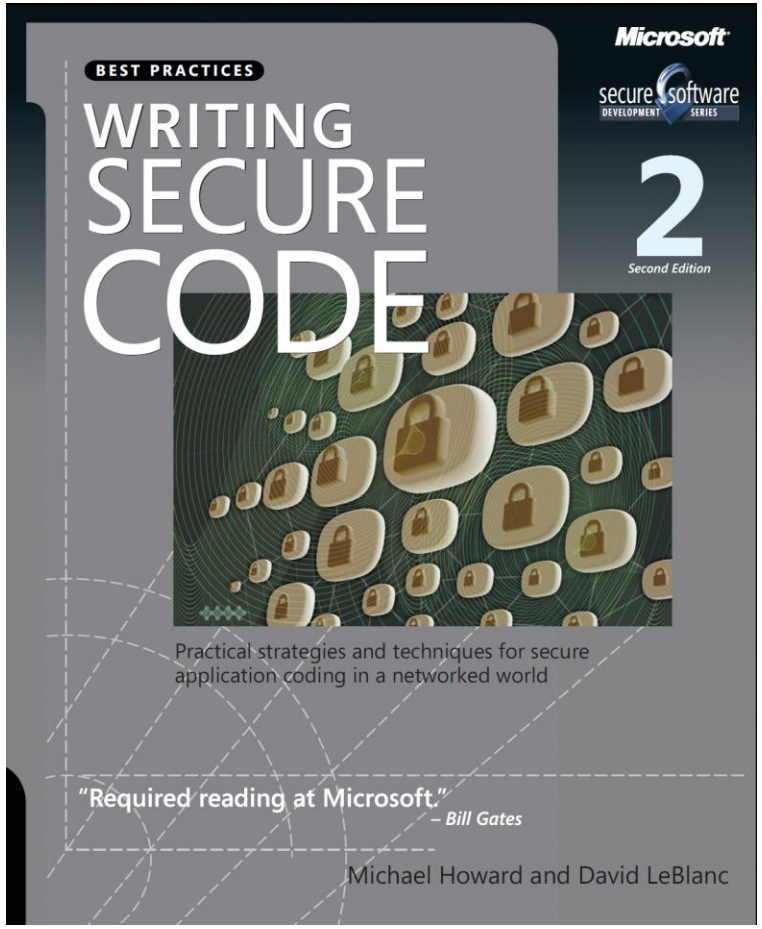
Almost all security failures are caused by software vulnerabilities

Are they inevitable? Or can we:

- measure risk associated with software?
- design or verify to prevent them?
- program better to avoid vulnerabilities?
- find vulnerabilities (before attacks)?
- …

Consideration in CSSE6400: security-aware software design

Recommend Reading Materials



Writing Secure Code, by Michael Howard and David LeBlanc, Microsoft Press, 2002.

Building Security In provides information on security principles

OWASP: security issues for web applications (more recently also mobile applications)

CERT Secure Coding: secure coding guidelines for C, C++, Java, Android apps, etc.

Vulnerability tracking

- BugTraq
- CVE (Common Vulnerabilities and Exposures)
- Individual Companies, e.g., Samsung, Android Security Bulletins

Software and Security

Software provides functionality and services

- Primary goal of software: achieving desired behavior
- E.g., on-line exam invigilation

Security is about regulating access to assets

- E.g., information or functionality

Functionality comes with certain risks

- E.g., what are risks of on-line exam invigilation?
  - Privacy, installing backdoor, …

Software security is about managing these risks

- Preventing undesired behavior

Undesired behavior

Stealing information

- Corporate secrets
- Personal information

Confidentiality

Modifying information or functionality

- Installing unwanted software (spyware, bot)
- Destroying records (accounts, logs)

Integrity

Denying access

- Unable to access banking information
- Unable to use the website

Availability

Designing Software with Core Security Concepts

Confidentiality Design

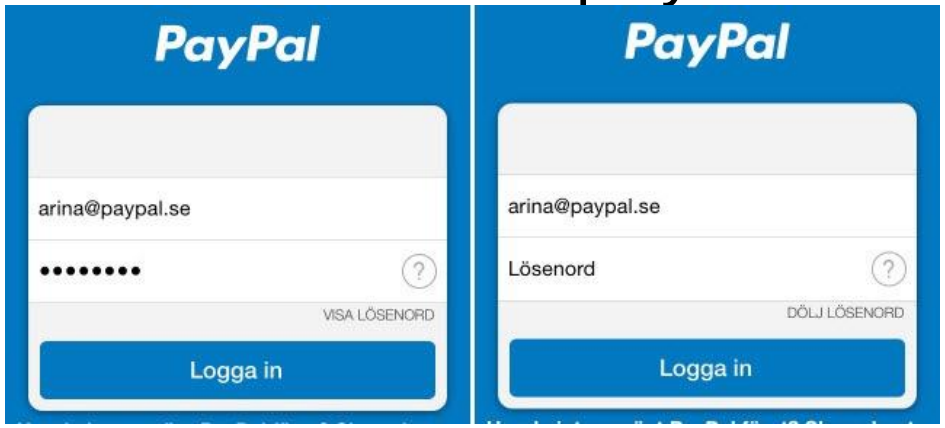
Integrity Design

Availability Design

Confidentiality Design

Disclosure protection can be achieved in several ways using cryptographic and masking techniques.

- Masking is useful for disclosure protection when data is displayed on the screen or on printed forms

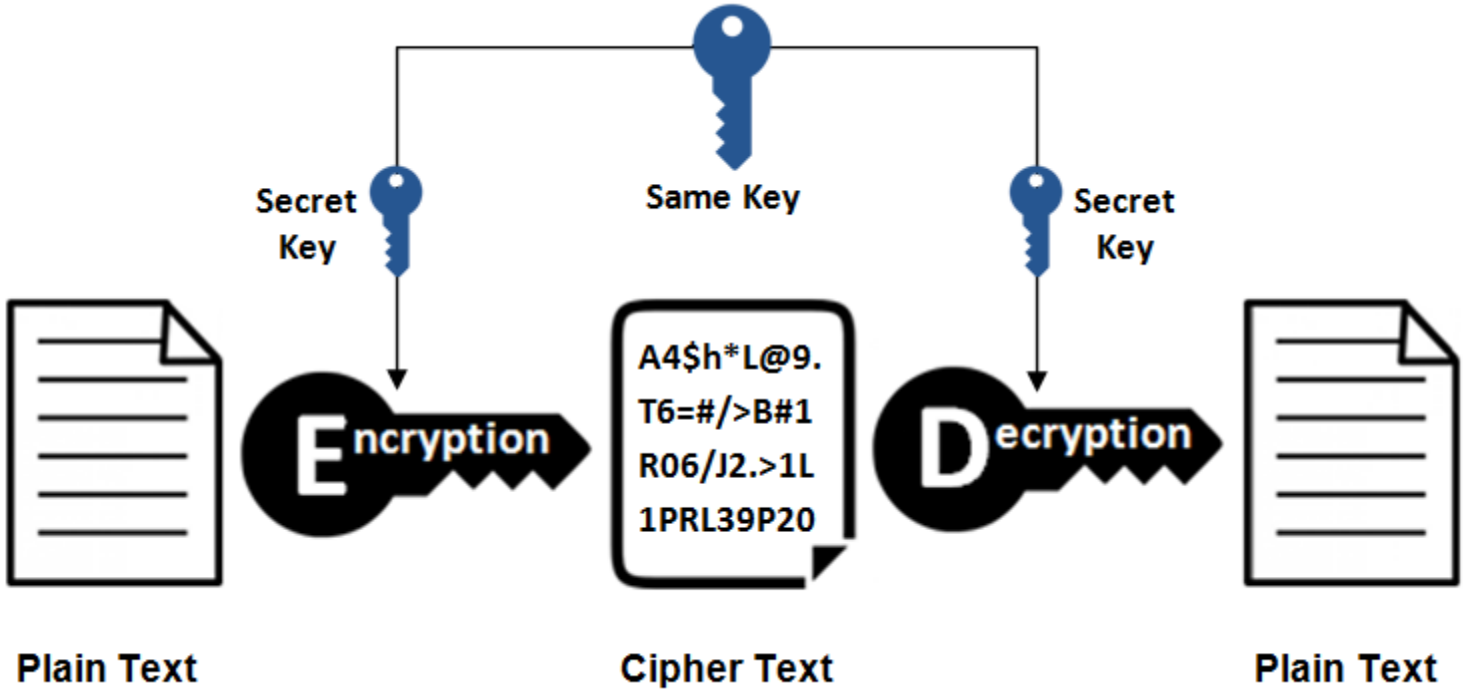


- Cryptographic techniques are useful for assurance of confidentiality when the data is transmitted or stored in transactional data stores or offline archives

Symmetric Algorithms

Symmetric algorithms are characterized by using a single key for encryption and decryption operations that is shared between the sender and the receiver

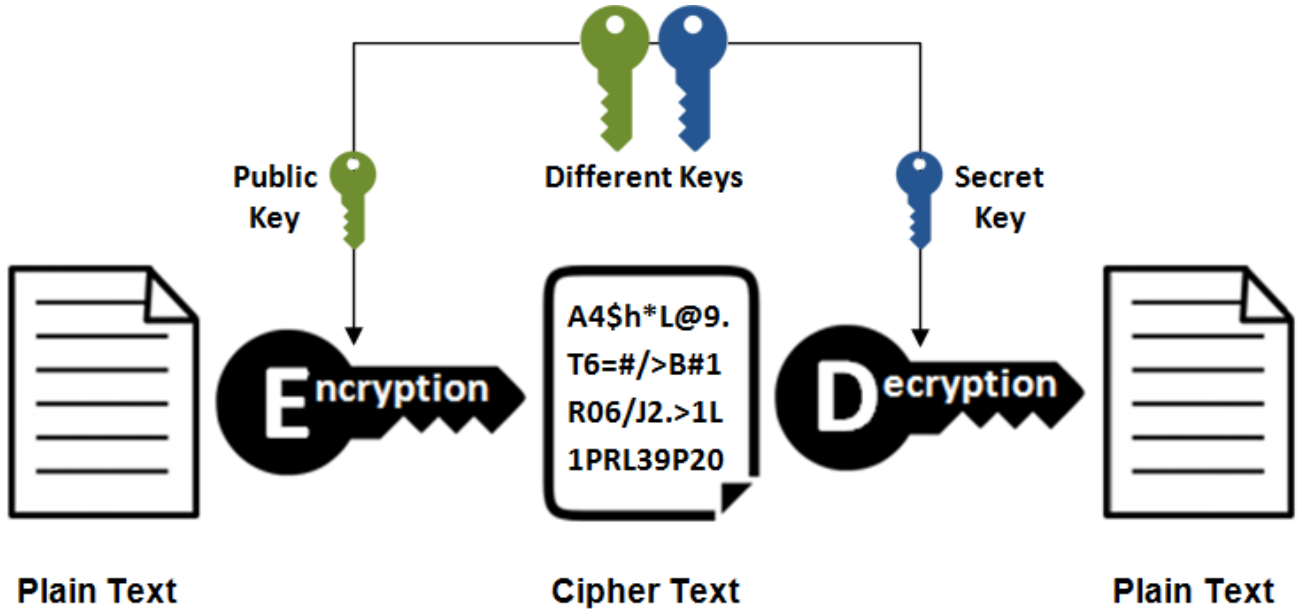
- E.g., DES, 3DES, Blowfish, AES



Asymmetric Algorithms

Two keys that are mathematically related to each other are used

- Private key: the key to be held secret
- Public key: disclosed to anyone with whom secure communications and transactions need to occur.
- It should be computationally infeasible to derive the private key from the public key.



Integrity Design

Integrity of software and data can be accomplished using any one of the following techniques or a combination of the techniques

- Hashing (or hash functions)
- Resource locking

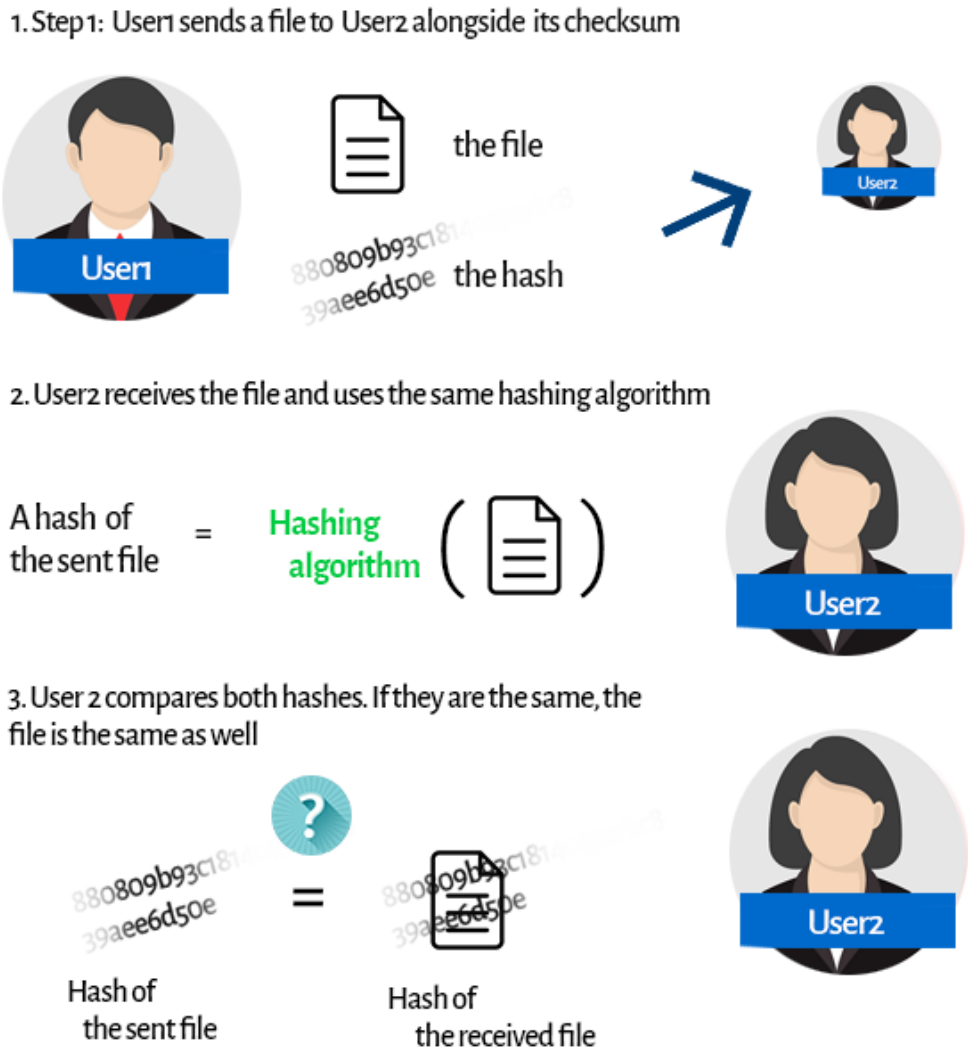
Hashing (Hash Functions)

Hash functions are used to condense variable length inputs into an irreversible, collision free, fixed-sized output known as a message digest or hash value.

- MD5, SHA-1, SHA256

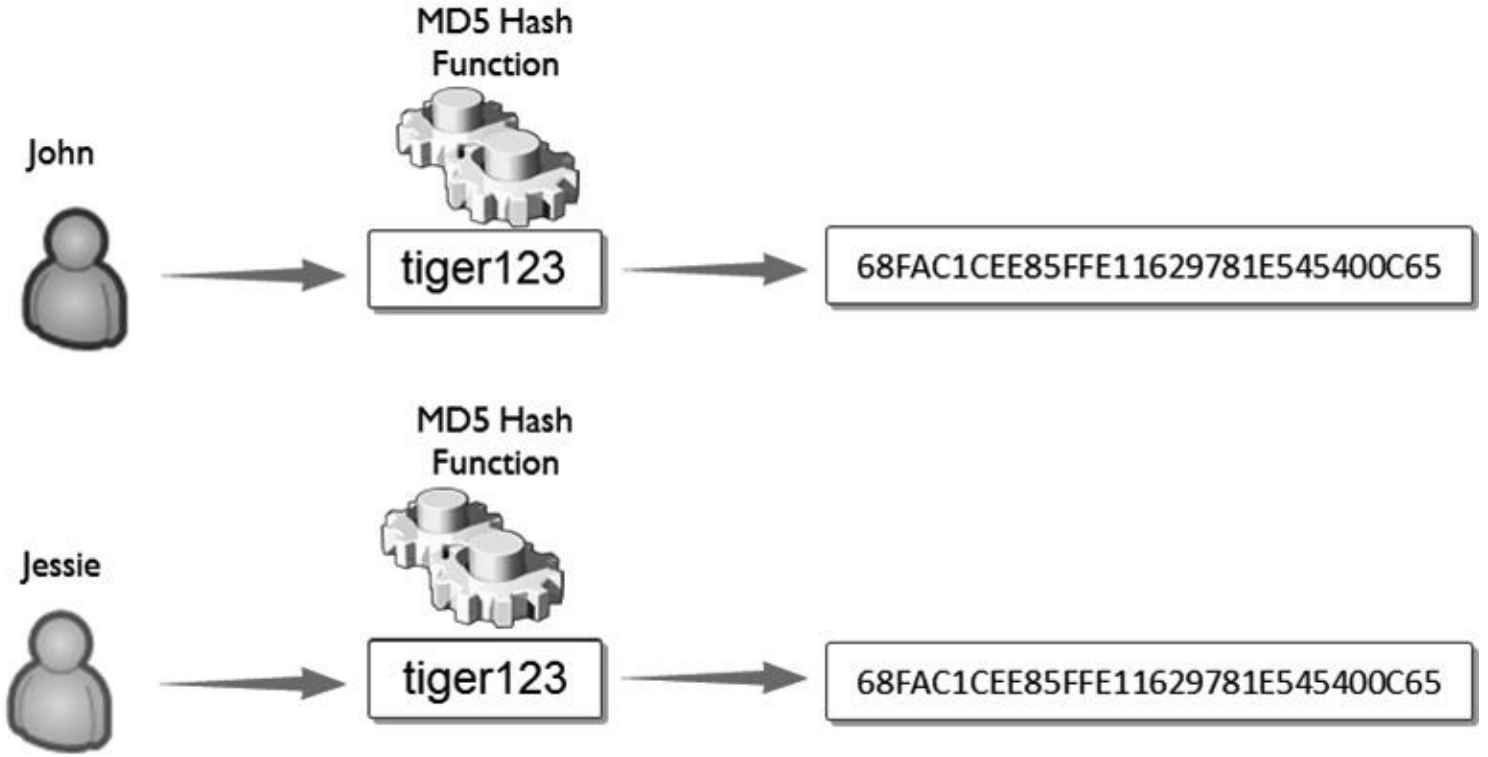
Hashing can also be used for confidentiality design

- E.g., password storage on Web server

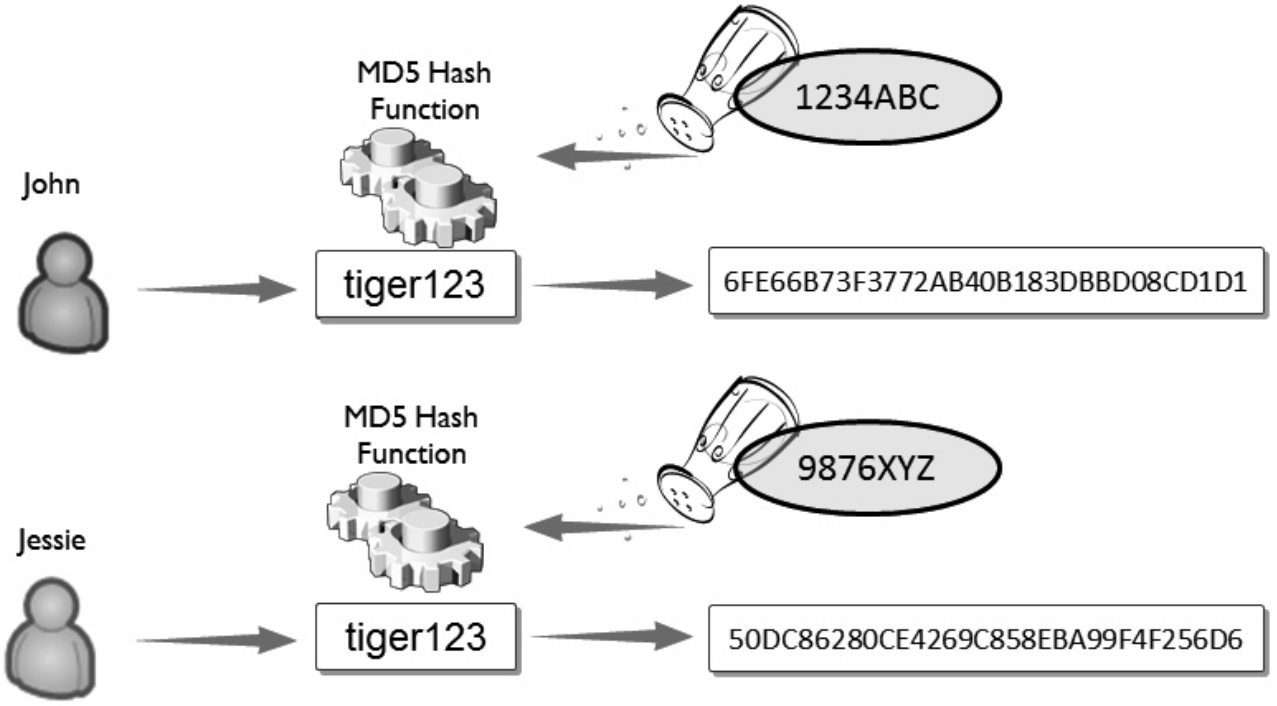


Prebuilt dictionary attacks

A dictionary attack is an attempt to thwart security protection mechanisms by using an exhaustive list (like a list of words from a dictionary)



Salted hash



Design considerations should take into account the security aspects related to the generation of the salt, which should be unique to each user and random

Resource locking

Two concurrent operations are not allowed on the same object

- e.g. say a record in the database

An online banking app with web client and mobile client

Availability Design

Replication, Failover and Scalability techniques can also be used to design the software for availability

Replication

- A single point of failure is characterized by having no redundancy capabilities and this can undesirably affect end-users when a failure occurs
- By replicating data, databases and software across multiple computer systems, a degree of redundancy is made possible
- Replication usually follows a master-slave or primary-secondary backup scheme

Availability Design *cont*.

Failover

- Failover refers to the automatic switching from an active transactional software, server, system, hardware component or network to a standby (or redundant) system.

Scalability Techniques

- Vertical scaling means that additional resources are added to the existing node
  - Memory, storage, etc.
- Horizontal scaling means that newer nodes are added to the existing node

Secure Software Design Principles

Secure Software Design Principles

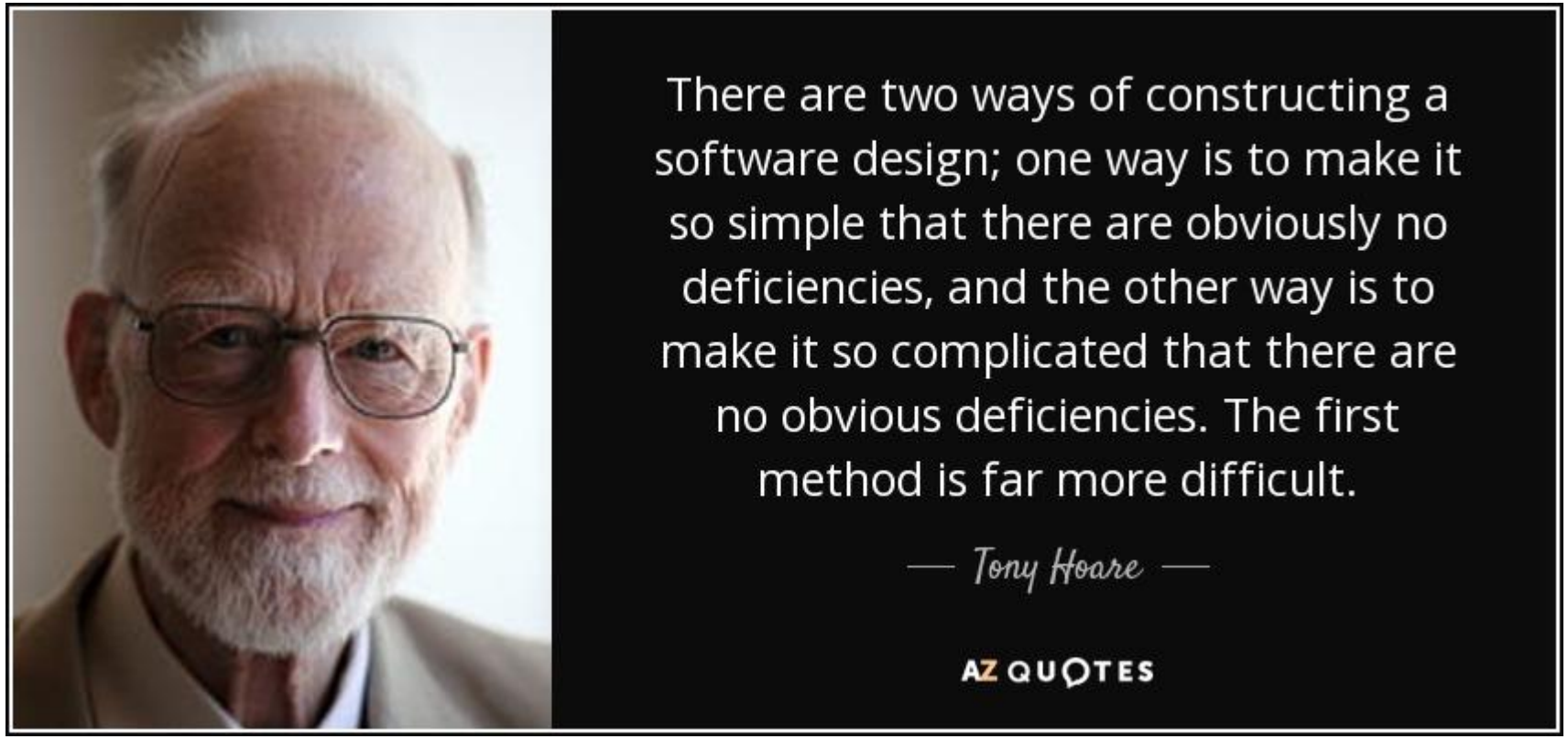
Principle #1: Favor simplicity
- Secure by default
- Do not expect expert users

Principle #2: Trust with reluctance
- Employ a small trusted computing base
- Grant the least privilege possible
  - Input validation
  - Flow Restriction
  - Compartmentalize

Principle #3: Defend in depth
- Use community resources
- No security by obscurity

Principle #4: Monitor and trace

Simplicity


There are two ways of constructing a software design; one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.
— Tony Hoare —
AZ QUOTES

Principle #1: Favor Simplicity

Keep it so simple it is obviously correct

- Applies to the external interface, the internal design, and the implementation

> "We've seen **security bugs in almost everything**: operating systems, applications programs, network hardware and software, and security products themselves. **This is a direct result of the complexity of these systems**. The more complex a system is--the more options it has, the more functionality it has, the more interfaces it has, the more interactions it has--the harder it is to analyze [its security]" -- Bruce Schneider

Secure by default

Some **configuration** or **usage choices affect** a system's **security**

- The length of cryptographic keys
- The choice of a password
- Which inputs are deemed valid

The default choice should be a secure one

- **Default key length is secure** (e.g. 2048-bit for RSA)
- **No default password**: cannot run the system without picking a strong password
- **Whitelist** valid objects, rather than blacklist invalid ones
  - E.g., don't render images from unknown sources

Hacker Breached HealthCare.gov Insurance Site

The Hacker Uploaded Malicious Software, But Consumers' Personal Data Didn't Appear to Be Taken

Washington officials said they are concerned an intruder gained access to the HealthCare.gov network through a basic security flaw. The server had low security settings because it was never meant to be connected to the Internet, the HHS official said. When the hacker broke in, it was only guarded by a default password, which often is easy to crack.

"There was a door left open," the official said.

Do not expect expert users

Software designers should consider how the **mindset and abilities** of (the least sophisticated) **users** will affect **security**

Favor simple user interfaces

- **Natural or obvious choice is the secure choice**
  - Or avoid choices at all, if possible
- **Don't have users make frequent security decisions**
  - Want to avoid user fatigue. How often should you remind them to change their password?

Principle #2: Trust with Reluctance

Whole system security depends on the **secure operation of its parts**

Improve security by reducing the need to trust

- Reducing the parts / people needed to be trusted
- Not making unnecessary assumptions
  - If you use 3rd party code, how do you know what it does?
  - If you are not a crypto expert, why do you think you can design/implement your own crypto algorithm?

Trusted Computing Base

Keep the **trusted computing base small** and simple to **reduce overall susceptibility to compromise**

- The trust computing base comprises the system components that must work correctly to ensure security

Eg. OS Kernels

- Kernels enforce security policies, but are often millions of lines of code
  - Compromise in a device driver compromises security overall
- Better: Minimize size of kernel to reduce trusted components
  - Micro-kernel designs move device drivers outside kernel

Least Privilege

Don’t give a part of the system more privileges than it needs to do its job (“need to know basis”)

E.g. Mail program delegates to editor for authoring mails – vi, emacs

- Many editors permit escaping to a command shell to run arbitrary programs: too much privilege!
- Better design: use a restricted editor (pico)

Trust is Transitive

If you trust something, you trust what it trusts

- This trust can be misplaced

Previous email client example

- Mailer delegates to an arbitrary editor
- The editor permits running arbitrary code
- Hence the mailer permits running arbitrary code

Input validation

Input validation is a kind of trust with reluctance

- You are trusting a subsystem only under certain circumstances
  - Validate that those circumstances hold

Examples

- Trust a given function if the range of its parameters is limited (within the length of a buffer)
- Trust a client form field if it contains no <script> tags (and other code-interpretable strings)

Flow Restriction

A good overall system goal is to restrict flow of sensitive data as much as possible

E.g. Admission system at UQ receives student application as PDF files

- A typical design would allow university administrators to download these files for viewing on their local computers
  - But then compromise of these computers leaks private information
  - Better: PDFs only viewable in browser; no data downloaded to client machine

Compartmentalization

Isolate a system component in a compartment or sandbox, reducing its privilege by making certain interactions impossible

- Browser extensions run in a sandbox on the browser
- Isolate Flash Player

Example: Isolate Flash Player

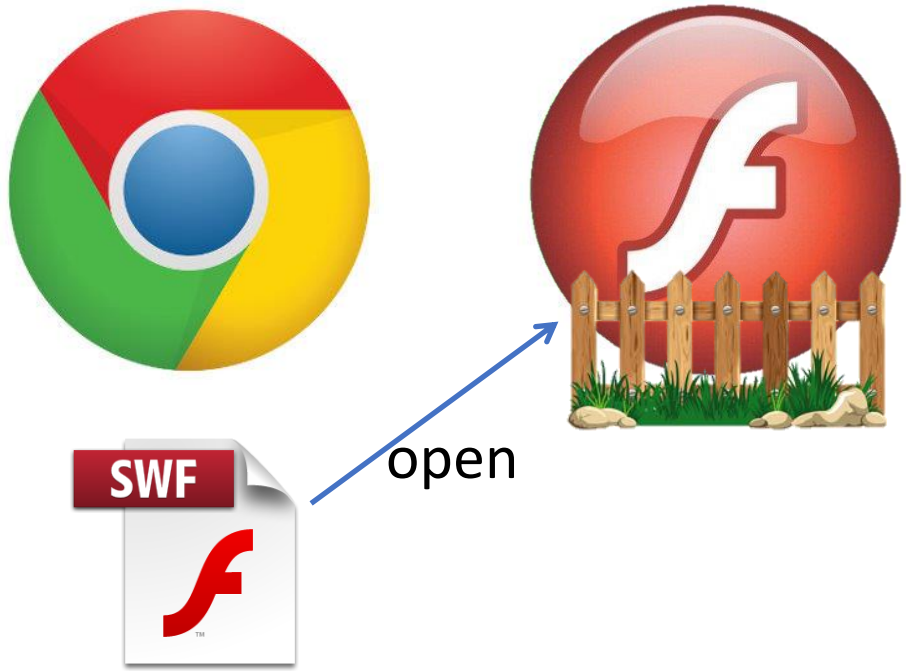
Receive .swf code, save it

Call fork to create a new process

In the new process, open the file

Call exec to run Flash player

Call seccomp-bpf to compartmentalize



Principle #3: Defense in Depth

Security by diversity

- If one layer is broken, there is another of a materially different character that needs to be bypassed

E.g.

- Use a firewall for preventing access via non-web ports
- Encrypt account data at rest
- Multi-factor authentication

Use community resources

User hardened code, perhaps from other projects

- Crypto libraries
- But make sure it meets your needs (test it)

Vet designs publically

- No security by obscurity

Stay up on recent threats and research

NIST for **standards**

OWASP, CERT, Bugtraq for **vulnerability reports**

SANS news bites **for latest top threats**

Academic and industry conferences and journals for longer term trends, technology and risks

Principle #4: Monitoring and Traceability

If you are attacked, how will you know it?

- Once you learn, how will you discern the cause?

Software must be designed to log relevant operational information

- What to log? E.g. events handled, packets processed, requests satisfied
- Log aggregation: Correlate activities of multiple applications when diagnosing a breach

Top 10 Design Flaws: Do not …

1. Assume trust, rather than explicitly give it or award it.
2. Use an authentication mechanism that can be bypassed or tampered with.
3. Authorize without considering sufficient context.
4. Confuse data and control instructions, and process control instructions from untrusted sources.
5. Fail to validate data explicitly and comprehensively.
6. Fail to use cryptography correctly.
7. Fail to identify sensitive data and how to handle it.
8. Ignore the users.
9. Integrate external components without considering their attack surface.
10. Rigidly constrain future changes to objects and actors.

Failure: Bad (or wrong) Crypto

Don’t roll your own crypto

- Both design and implementation are hard to get right

Don’t assume it gives you something it doesn’t, mostly

- Encryption algorithms may protect confidentiality but not integrity
- Hashing protects integrity but not confidentiality

Know how to use it properly

- Use properly generated keys of sufficient size
- Protect the keys from compromise
  - Don’t hard-code them, or embed them in released binary

Security Principals

Design with Core Security Concepts

- CIA
- Authentication, Authorization and Accountability

Security Design Principles

- Principle #1: Favor simplicity
- Principle #2: Trust with reluctance
- Principle #3: Defend in depth
- Principle #4: Monitor and trace