COMP0141 Notes

Exam tips

- When thinking about an approach, think about their pros and cons and discuss accordingly

Week 1: Intro / What is security?

How to define a secure system that meets a security policy
How to define a security policy via threat modelling to create a **threat model**

1.1 What is security?

Security in CS uses properties of programs/systems:

- **Correctness**
  - For a given input, program must provide correct output
- **Safety**
  - Well-formed programs cannot have bad (wrong/dangerous) outputs, no matter the input
- **Robustness**
  - Programs must be able to cope with errors in execution

In security, these properties must hold even in presence of a resourceful and strategic adversary (threats)

Security mindset: think about how things can be made to fail (think like an attacker/adversary) so you

More Security Properties

- **Confidentiality**
  - keep data private
- **Integrity**
  - system and data have not been improperly altered
- **Availability**
  - ability to use the system as anticipated
- **Privacy** (***different to confidentiality*** - see below)
  - controlling information revealed by data

- **Authenticity**
  - this document has been written by (someone)
- **Anonymity**
  - I don't want the website to know I'm visiting their website
- **Non-repudiation**
  - Being unable to deny you sent/recieved a message.
- **Plausible deniability**
  - being able to deny you sent a message
- **Forward secrecy**
  - if an adversary can decrypt my mesages, they can't decrypt messages sent/recieved prior to compromise

Security Approaches:

Binary

- system is either **secure** or **insecure**
- used in **cryptography**
  - define capabilities/limitations of attacker, objectives/policy of system
  - secure if policy can't be violated without needing capability the attacker doesn't have, insecure otherwise.
- **pros**:
  - longevity
- **cons**:
  - brittle, hard to define what "right" is
  - expensive

Risk Management

- system is **secure to an acceptable level** - as long as you manage the risk
- largely about cost (for both defender/attacker)
- **pros**:
  - adaptive (adjust based on situation)
- **cons**:
  - arms race
  - hard to evaluate risks, effectiveness of policy, mitigations

1.2 Threat modelling

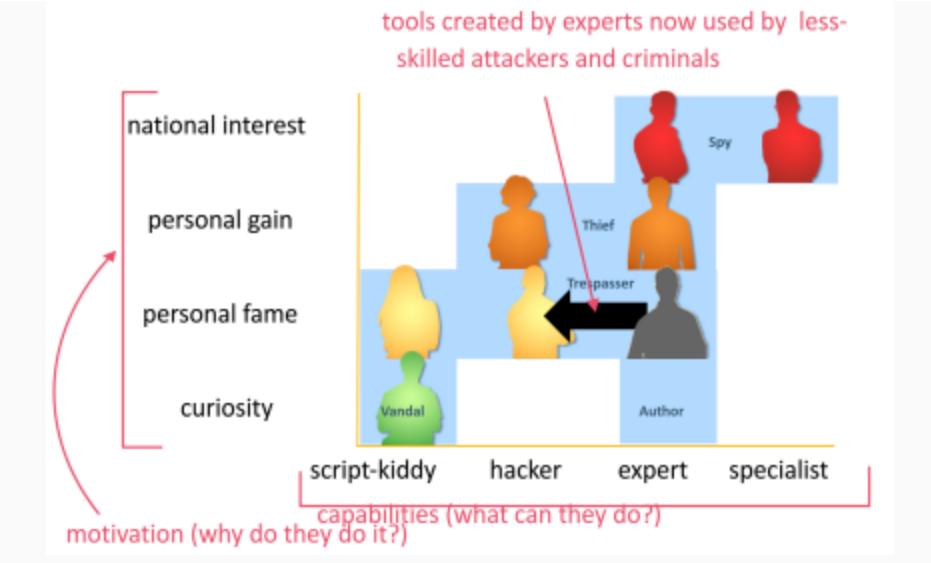
What should security policy address? - used to **create a threat model** to address that policy

- **Threats**
  - who is the adversary (capabilities/their motivation)
  - what resources are available to adversary (that can be leveraged optimally to launch an attack)
- **Vulnerabilities**
  - where can the system break (cracks in the system)
- **Likelihood**
  - how likely is the threat
- **Impact**
  - what if bad things happen
- **Protection**
  - what does it cost? how?
- **Goals** (covered in human-centred security)
  - is the system usable

You put all of them together with protection mechanisms to define a **specific threat model** for a system to **create a security policy** to meet security objectives

Threats

Identifying who is the adversary (threats) and what resources are available for them to attack my system



x-axis - what can they do?
y-axis - why do they do it?

When thinking about adversary, think about the resources - **resourceful strategic adversary**

Resourceful Strategic Adversary

What **resources** are available to the adversary? E.g. what can they

- **observe**? (e.g. network connection)
- **corrupt**? (e.g. router)
- **influence**? (e.g. network configuration)
- **modify**? (e.g. packet data)
- **control**? (e.g. company employee)

attack - a series of actions that allow the adversary to violate a given security policy
strategic adversary - adversary that chooses the **optimal** way to leverage their resources (strategy) to mount an attack

STRIDE

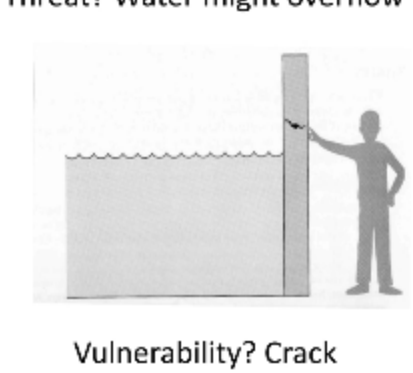
A model of thinking about threats to a system

Actions	Definition	Desired Property
Spoofing	Pretending to be something or someone other than yourself	Authenticity
Tampering	Modifying something on disk, network, memory, or elsewhere	Integrity

Actions	Definition	Desired Property
Repudiation	Claiming that you didn’t do something or were not responsible; can be honest or false	Non-repudiatility
Information disclosure (e.g. data leak)	Someone obtaining information they are not authorized to access	Confidentiality
Denial of service	Exhausting resources needed to provide service	Availability
Elevation of privilege	Allowing someone to do something they are not authorized to do (do things you're not supposed to do)	Authorization

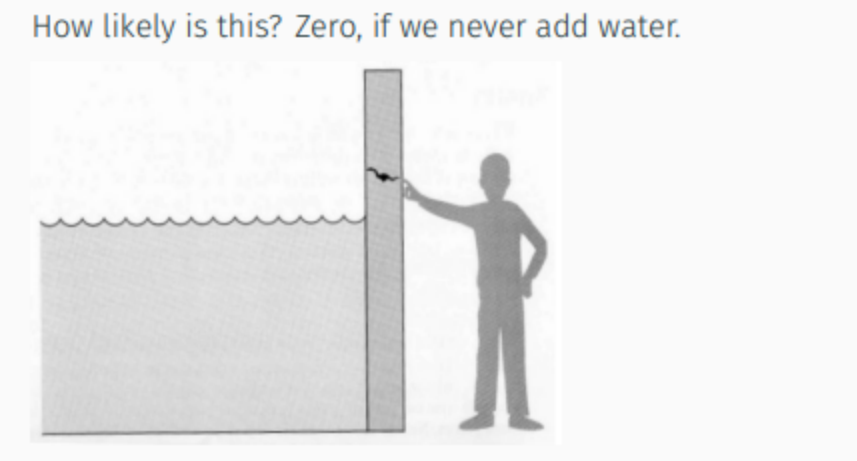
Vulnerabilties

- Finding cracks in the systems (what makes it possible) that allow threats to a system



Likelihood

- How likely will the threat occur



Impact

- what if bad things happen? how much does it matter/cost

Protection

- thinking about protection mechanisms available when thinking about security objectives to achieve them
  - what does it cost?
  - how well does it work?

Threat Model - Defining a secure system

Secure system: one that meets a specific security policy

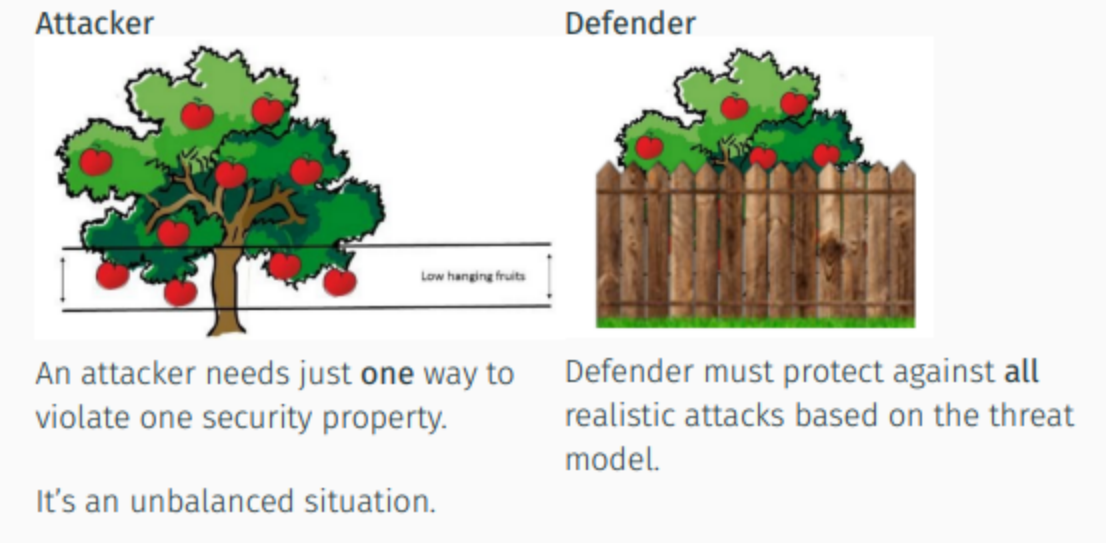
We can use threats, vulnerabiltiies, likelihood, impact and cost (proection mechanisms) used to create a **threat model** that can be used to create a security policy

Think about "Is the system secure under this threat model" - can be modelled as **attacker vs defender**

A system is **secure** if

- there is a **security argument** that an **adversary constrained by a specific threat model cannot violate the security policy**

Attacker vs Defender



Security argument

- rigorous argument that the security mechanisms are maintaining the security policy (subject to assumptions of the threat model)

Security Mechanisms

- Technical mechanism used to ensure that security policy is not violated by an adversary operating within the threat model
- Could be software, hardware, cryptography, people and procedures
- Example
  - Policy: a log cannot be changed by a single employee.
  - Mechanism: Keep a copy of the log on multiple computers such that no one employee has access to all of them

1.3 Threat modelling examples

Think about threats, vulnerabilities, likelihood, impact and protection mechanisms when coming up with **threat models** (as mentioned in previous subsection)

Example 1: Electronic voting

What is the **threat model** for **electronic voting?**

Threats

Identifying capabilities and their motivations

Capabilities?	Motivation?
Voter(s)	Vote as someone else (S)
Election official	Rig the election (T)
Manufacturer of EVM	Learn someone's vote (I)
Software engineer	Prevent others from voting (D)
Cleaner	

Vulnerabilities

Capabilities?	Vulnerabilities
Voter(s)	Weak cryptography/design
Election official	Software/hardware defects
Manufacturer of EVM	Hardware defects
Software engineer	Software defects
Cleaner	Machine access before/after election

Likelihood (might this happen?)

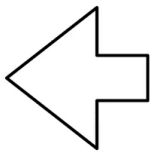
Motivation	Capabilities	Vulnerabilities
Rig the election	Manufacturer of EVM	Hardware defects
yes!	Janitor	Access to machines
Vote as someone	Voter	Weak cryptography
yes!	Software engineer	
Rig the election	Election official	Hardware defects
no!		

Impact (what if bad things happen)

Scale	Motivation?
Small group	Vote as someone else (S)
Huge!	Rig the election (T) Learn
Small to large group	someone's vote (I)
Small to large group	Prevent others from voting (D)

Protection

Cost	Vulnerabilities
Expensive (binary)	Weak cryptography/design
Cheap - expensive	Software/hardware defects
Expensive	Hardware defects
Cheap - expensive	Software defects
Cheap (risk management)	Machine access before/after election



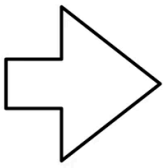
Example 2: Driving a car

Threats

Capabilities?	Motivation?
Device manufacturer	Track your location
Software engineer	Steal your money
Network attacker	See which websites you visit
Someone in close proximity	Steal your credentials
Intimate partner	Stalk you

Vulnerabilities

Capabilities?	Vulnerabilities
Device manufacturer	Owning a computing device
Software engineer	App installation
Network attacker	Internet connection
Someone in close proximity	Human interaction
Intimate partner	Human interaction



Likelihood

Impact

Protection

1.4 Human-centred security

Security is a secondary concern, **not** the primary goal - we need to find a balance between doing things we want and doing them securely

When thinking about policies, we also need to address **goals** (why are people using the system)

Usability

- a measure of how well a **specifc user in specific context** can use a product/design to achieve a **defined goal** effectively, efficiently and satisfactorily

  Is the system usable?
- instead ask: Is the system usable for **this user** with **this goal**?
- not all users are the same
- even the same user can act differently
- not all goals are the same (depends on the user and their goals)

Usability and security

- Systems can be secure but NOT usable, thus insecure
- policies can
  - be too complex
  - provide temptation to cut corners
  - fail to make it clear what the real threats/risks are
  - interfere with what users actually want to do

Unintentional Failures

Security issues can occur due to **unintentional failures** e.g.

- human error
- non-compliance

why do failures happen?

- users lack intuition about complex computing devices
- users are in charge of their own (complex) devices
- hard to estimate risks (or have the wrong mental model - unrealistic threat model)

- security measures feel like they get in the way

Hidden cost: time

Need to factor in time - its a false assumptions that users:

- can devote time/effort to security-related tasks
- are motivated by security
- need more knowledge/training to comply if they are not doing so already

Evaluating Usability

- Conduct a cognitive walkthrough by usability experts
- User studies:
  - Laboratory experiments
  - Diary studiets
  - Interviews
  - Observation of real usage
- Data to collect:
  - Demographics
  - Performance (time, success rate, errors)
  - Opinions and attitudes
  - Actions and decisions

How to improve usability?

Looking back at failures:

- users lack intuition about complex computing devices
  - provide security education / training
- users are in charge of their own (complex) devices
  - make security invisible
- hard to estimate risks
  - help users build more accurate mental models
- security measures feel like they get in the way
  - make security path of least resistance

Week 2: Principles and background

2.1 Design principles

Security describes techniques (security mechanisms) that control who may use or modify the computer or information contained in it.
These design principles guide the design and contribute to the implementation of a system without security flaws

The 8 design principles - ELLF COPS
(from: The Protection of Information in Computer Systems [1975])

Principles	Description	Justification
Economy of Mechanisms	Keep the design as simple and small as possible (KISS principle)	Easier to test, verify, have fewer potential vulnerabiltiies
Least Privilege	Minimum privileges needed to complete the job	Helps to limit potential damage from accidents/errors and reduce unwanted interactions among privileged programs
Least Common Mechanism	Share as little as possible among users	Minimise amount / use of shared mechanisms
Fail-safe Default	Deny access by default and grant access only when explicit permission exists	Minimises unauthorized access by requiring explicit permissions rather than implicit access (deny by default) so access is given deliberately only.
Complete Mediation	Check every access to every object everytime	Tightly control access to objects without any bypassing (subverting)
Open Design	Design should not be secret but rely on key management (security by obscurity)	Permits mechanisms to be examined by many reviewers without concern that review may itself compromise the safeguards
Psychological Acceptability	Should be easy to use	Users would try to circumvent security mechanisms that are too complex
Separation of Privilege (Responsibilities)	Should grant access based on more than one piece of information	Creates a situation where no single accident, deception, breach is sufficient to compromise protected information e.g. two-key nuclear launch

Other principles:

Principles	Description	Justification
Defense in depth	having more than one security measures	Allows system can resist attack even if a single security vulnerability is discovered/security feature is bypassed
Design for updating	plan for the safe/reliable installation of security updates	no system is likely to remain free from security vulnerabiltiies forever
Prudent Paranoia	Never underestimate effort adversary will go to	Underestimating adversaries' capabilities, resources, and determination can lead to inadequate security measures - anticipate worst case scenarios when designing systems
Privacy Promotion	Don't collect more data than strictly necessary	Doesn't allow users to be exposed to more privacy violations, reducing liability, building user trust and minimising risk exposure

Trusted Computing Base (TCB)

- refers to every component of system upon which the security policy relies
    - could be hardware, software, etc
- in other words, if something goes wrong then security policy may be violated
    - needs to be kept small
- example of economy of mechanisms

2.2 Networks Primer

Knowing about how the internet works
How does a web request work, in steps:

1. **Find content host**
    - get IP address
2. **Request Content**
    - send GET request to IP address (routing via internet backbone)
3. **Receive Content**
    1. wait for response from IP address and render webpage (in .html format) and ennjoy

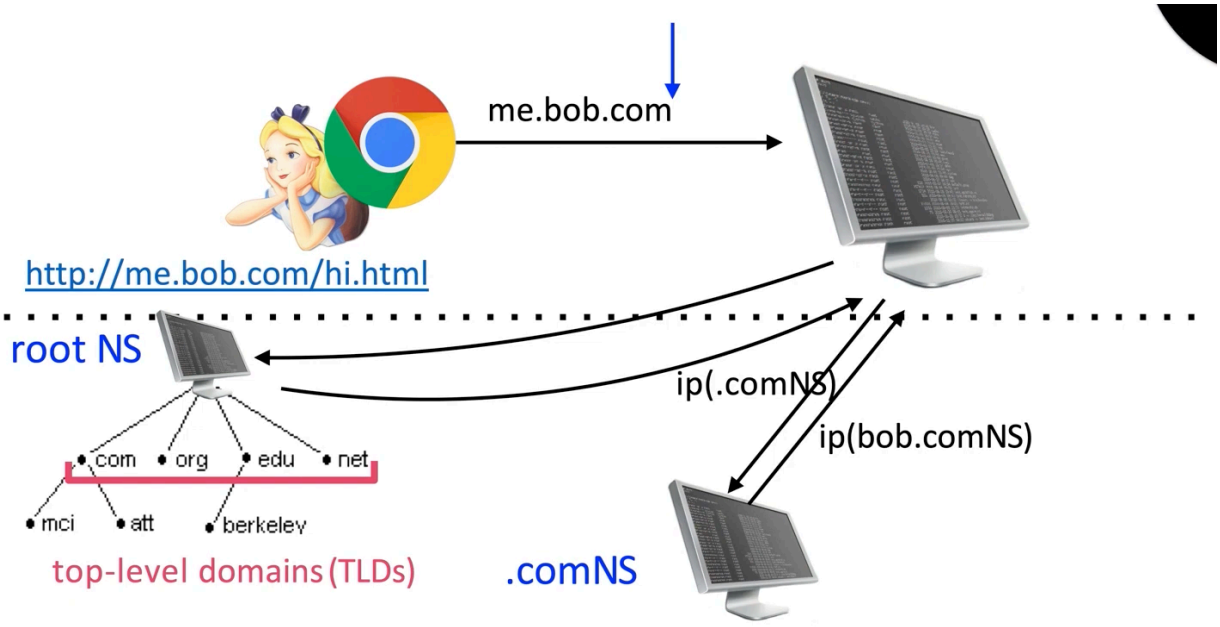
Step 1: Find Content Host

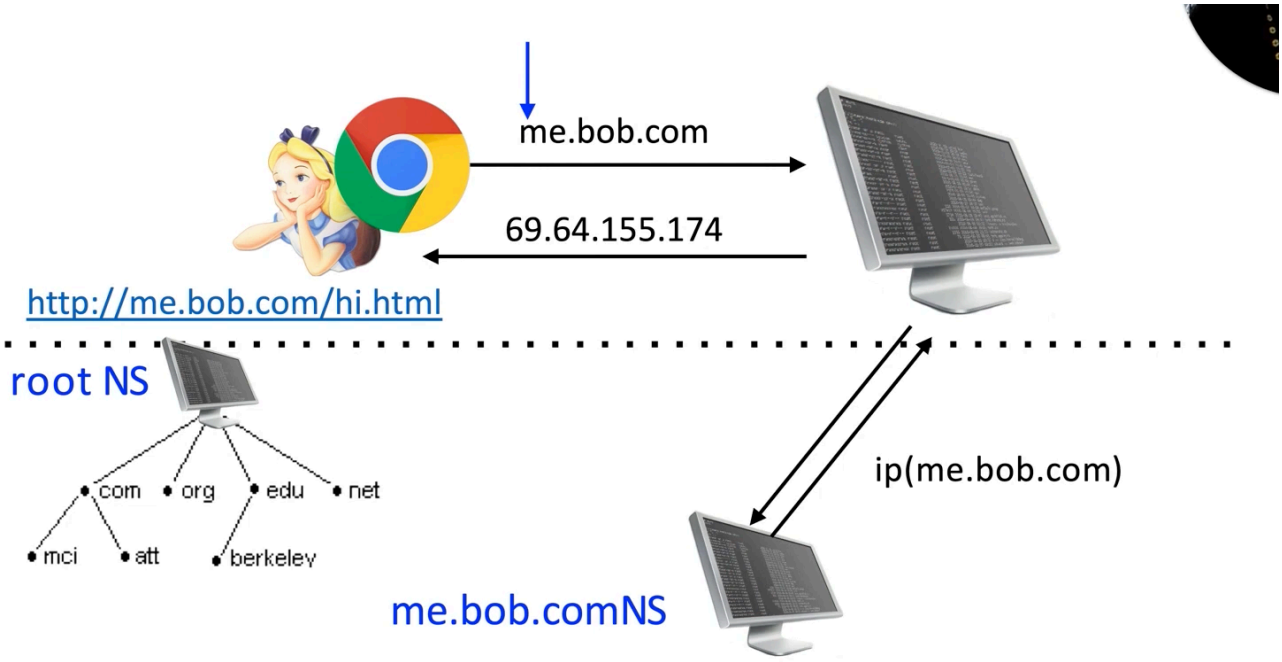
- **via URL**, which consists of:
    - protocol specifier (e.g. http/https)

    - domain name (e.g. bob.com)
    - page location (e.g. /hi.html)
- **goal**: get IP address for domain name via DNS (domain name server - mapping of IP address to domain names)

Domain Name System

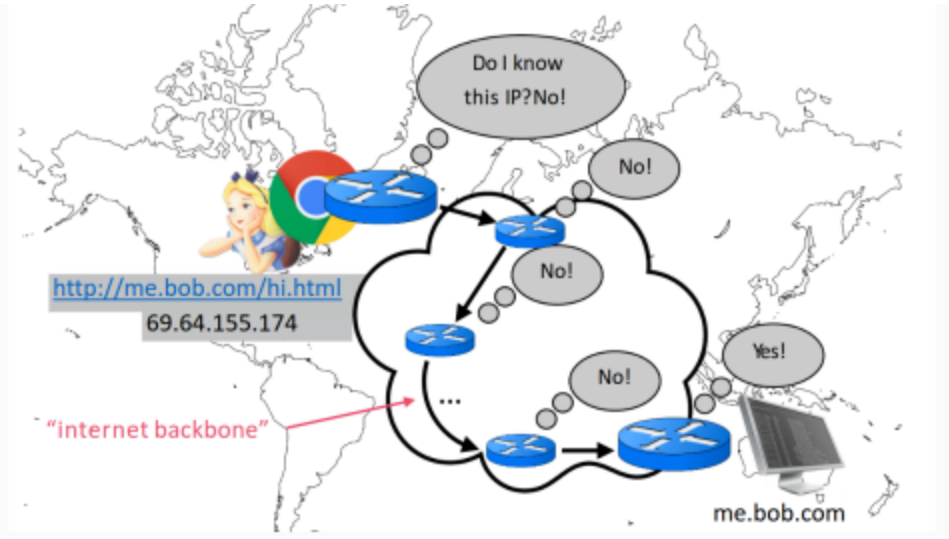
- Find the IP address for the server we want to contact (domain name)
- Starts with TLD (top level domain)
- Then query the next level and then finally the last level
- DNS results are cached by web browser
- Root servers are hard coded
- You use routing to find the name servers





Step 2: Request Content

- Done via **routing** (via internet backbone)
- Direct connection via leased lines
  - pros:
    - incredibly fast
    - reliable
    - secure
  - cons:
    - incredibly expensive
- many networks are connected to other networks (even via undersea cables too :o)
- data gets transmitted from 1 network to another, until it reaches the destination

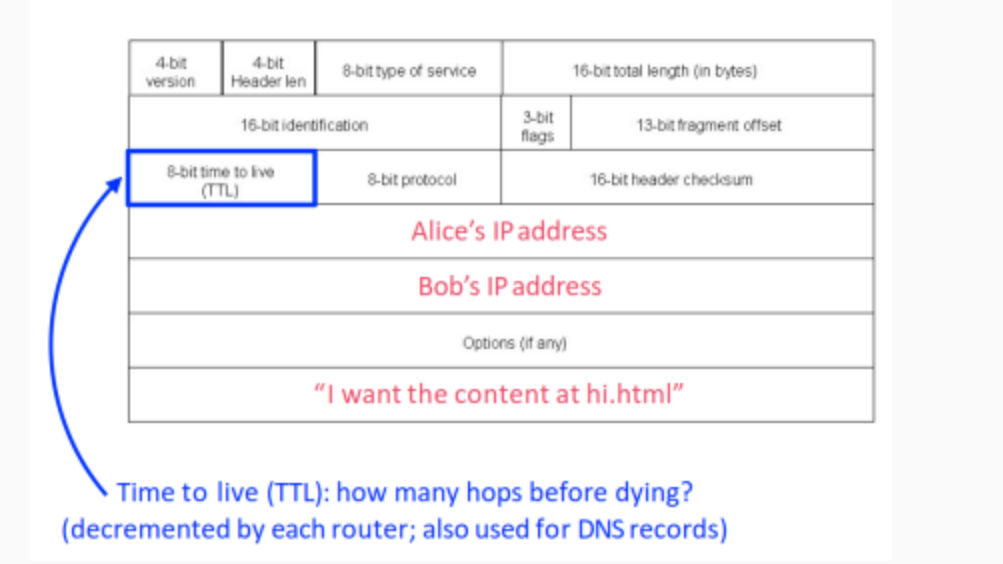


Routing

- Information that routers sent are in **packets**

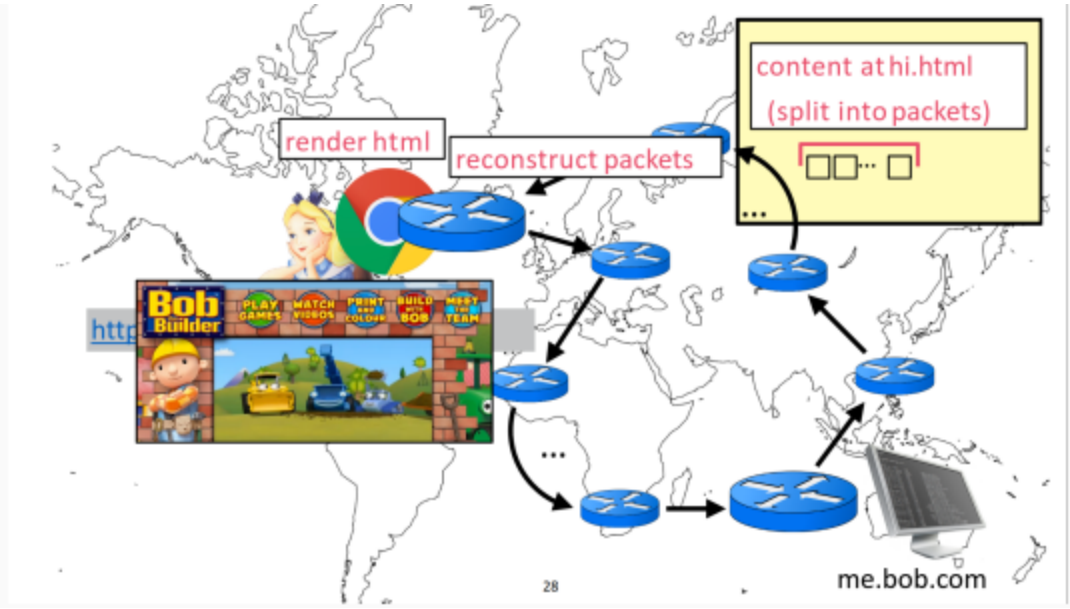
Packets

- Packets contain a TTL (time to live) - thus they can't go aroun din a circle



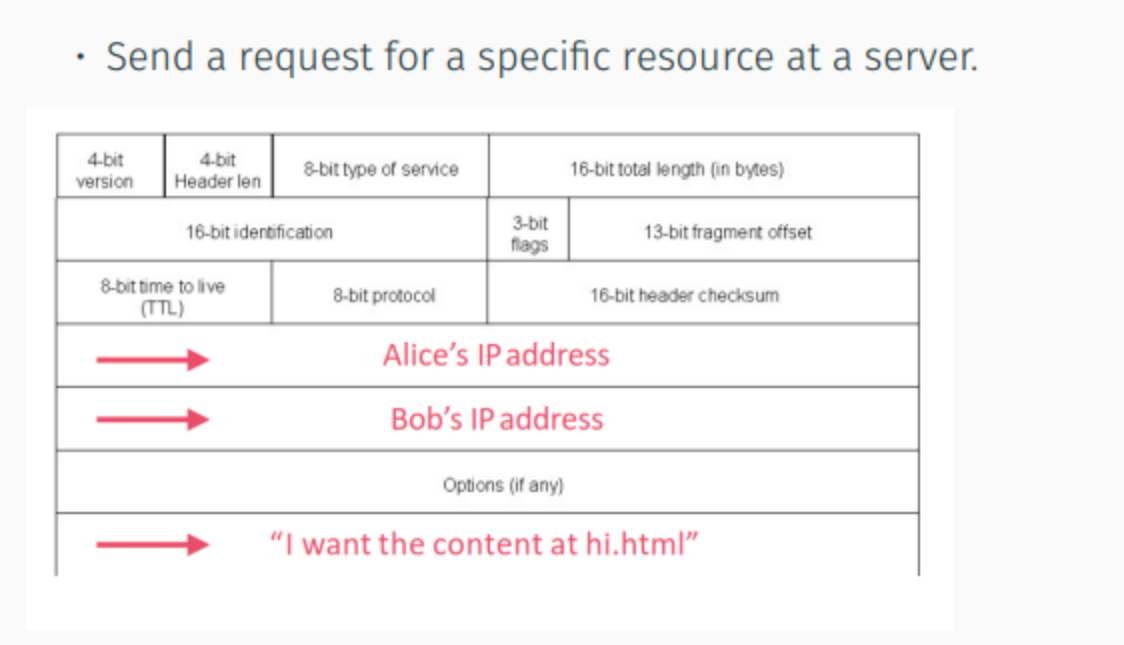
Step 3: Receive Content (Response)

- Split response content from server into packets
- Send **packets** from server to client (via routing)
- Reconstruct packets
- Render webpage (in html)
- Note that HTTPS protect the content / HTTP is not private

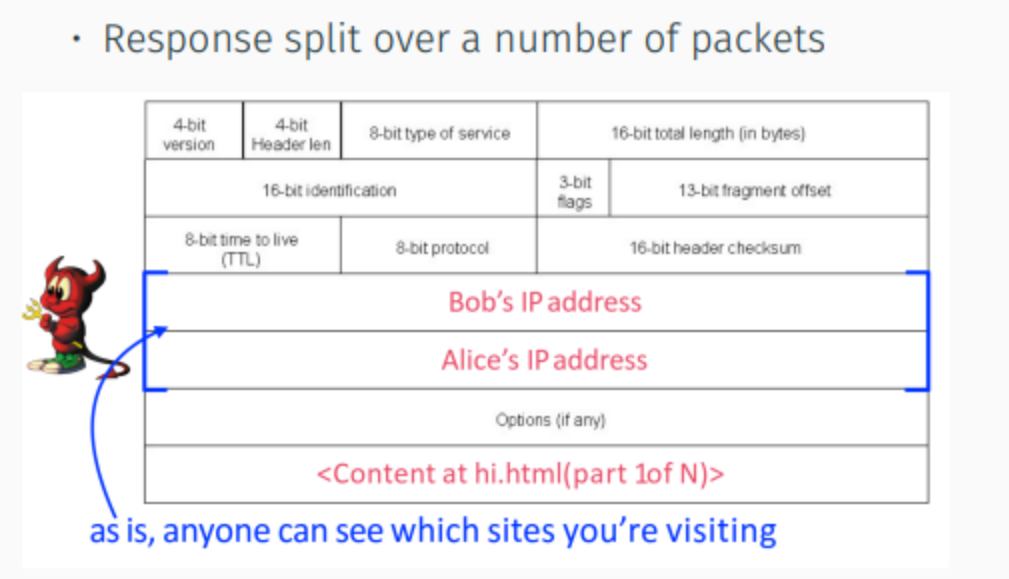


Example:

Request:



Response:



2.3 Math Background

Notation

Sets: $A = \{1, 2\}, B = \{a, b, c\}$
Cardinality (number of items in the set): $|A| = 2, |B| = 3$
Set inclusion: $(x \in A)$ means x is a member of A ($1 \in A, 3 \notin A$)
Division: $x|y$ = x (evenly) divides y ($y = ax$ for some integer a) = $x \mod y = 0$
Set conditions: $\{x : 2|x\}$ = given set contains all x that are even

Modular Arithmetic

Given integers $x > 0, y, z$, we can write $z = y \mod x$ when $x|(z-y)$

e.g. $5 = 1 \mod 2, 18 = 3 \mod 5$

we can find a unique a and $y \in \{0, 1, \ldots, x-1\}$ s.t.
$z = ax + y$

- quotient: a
- remainder: y

e.g.

- $5 = 2 * 2 + 1$
- $18 = 3 * 5 + 3$

if $z = ax + y$ then $z = y \mod x$ (because $z - y = ax$ and $x|ax$ - x evenly divides ax)

Greatest Common Divisor (GCD)

GCD of a and b = largest number that evenly divides both a and b
$gcd(a, b) = max(\{d : d|a \text{ and } d|b\})$

For all a, b, there are r, s s.t. $gcd(a, b) = ra + sb$

Euclidean Algorithm

- Used to compute the $gcd(a, b)$ for $a > b$
- start by finding r_0 s.t. $a = q_0 b + r_0$ ($r_0 = a - q_0 b$)
- then r_1 st $b = \$q_1 r_0 + r_1$ (so $r_1 = b \mod r_0$)
- eventually you will get $r_{n-2} = q_n r_{n-1} + 0$
- the final non-remainder r_{n-1} will get you the gcd of a and b

Why does it work:

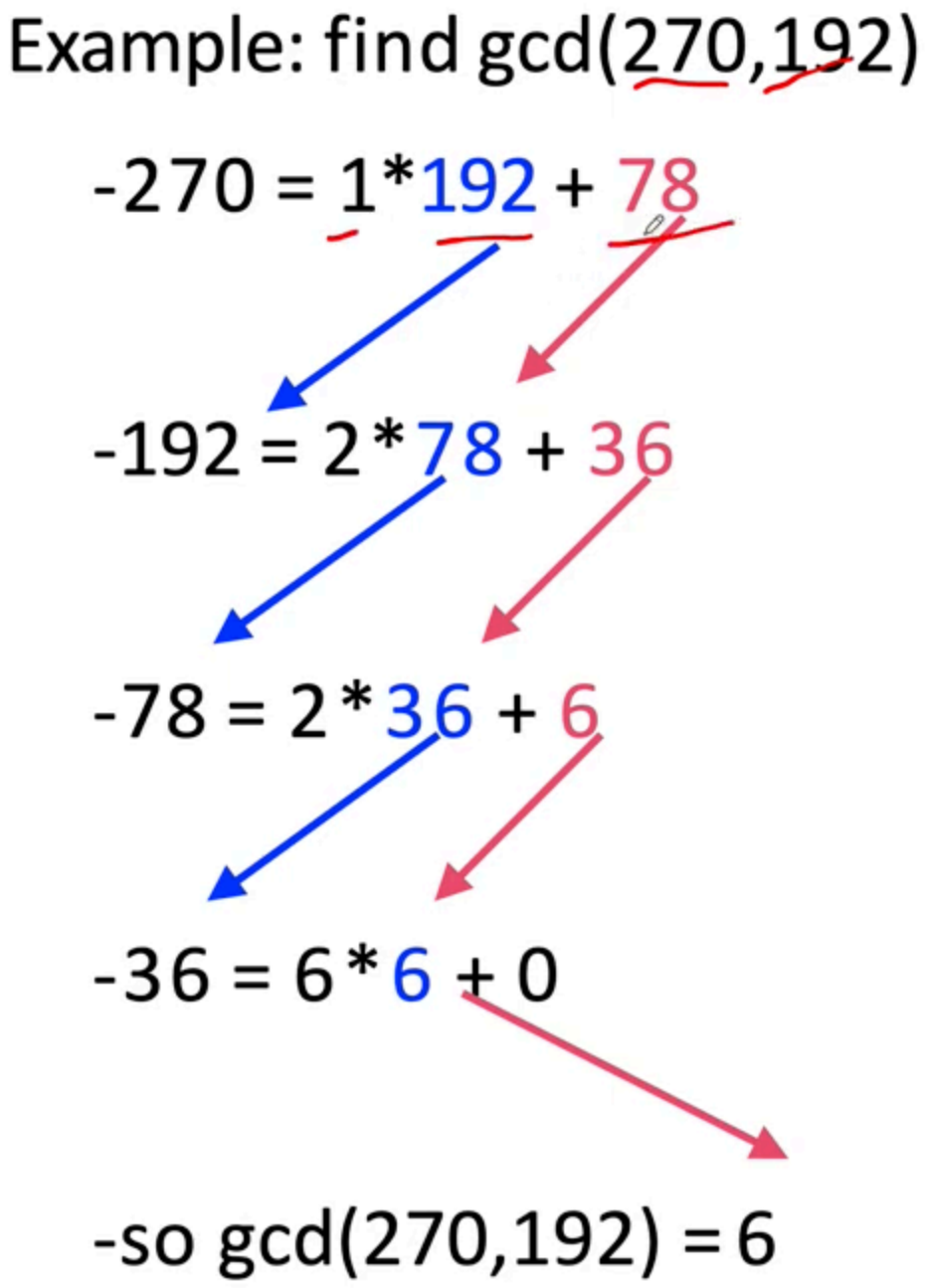
- there are finitely many numbers between a and 0 so it terminates
- bottom up approach (r_{n-1} divides a and b):

  $r_{n-2} = q_n r_{n-1} + 0 \rightarrow r_{n-1} = 0 \bmod r_{n-2}$

  $r_{n-3} = q_{n-1} r_{n-2} + r_{n-1} \rightarrow r_{n-1} = 0 \bmod r_{n-3}$

  $\ldots \rightarrow r_{n-1} = 0 \bmod a \text{ and } 0 \bmod b$ (so r_{n-1} is a common divisor)
-

Example:



Extended Euclidean Algorithm

- Used to calculate r, s such that $gcd(a, b) = ra + sb$
- add in two extra variables with same quotient

  In the extended algorithm, add in two extra variables with the same quotient: $s_i = s_{i-2} - q_{i-1}s_{i-1}$ and $t_i = t_{i-2} - q_{i-1}t_{i-1}$, start with $s_0 = 1, s_1 = 0, t_0 = 0$, and $t_1 = 1$(because a = 1*a + 0*b and b = 0*a + 1*b)

- example

  Example: find s and t such that gcd(18,13) = s*18 + t*13 18 mod 13

  - (r) 5 = 18 - 1*13, (s) 1- 0 = 1, (t) 0 - 1= -1

  13 mod 5

  - (r) 3 = 13 - 2*5, (s) 0 - 2*1 = -2, (t) 1- 2*(-1) = 3

  5 mod 3

  - (r) 2 = 5 - 1*3, (s) 1- 1*(-2) = 3, (t) -1 - 1*(3) = -4

  3 mod 2

  - (r) 1= 3 - 1*2, (s) -2 - 1*(3) = -5, (t) 3 - 1*(-4) = 7

  2 mod 1

  - (r) 0 = 2 - 2*1

  So 1= -5*18 +7*13

  gcd is a linear combination between two numbers with quotiennts

Modular Arithmetic

Associativity (both for + and *):

$(a + b \bmod N) + c \bmod N$

$= a + (b + c \bmod N) \bmod N$

$= a + b + c \bmod N$

$(ab \bmod N)c \bmod N$

$= a(bc \bmod N) \bmod N$

$= abc \bmod N$

Modular Exponentiations

We often use modular exponentiations $g^x \bmod p$ ($g*g*\ldots*g \bmod p$)

x times

Again, usual exponentiation rulesapply

- $g^x g^y = g^{x+y} \bmod p$
- $(g^x)^y = g^{xy} \bmod p$

Commutative Ring (Z/NZ)

Associative: $(a + b) + c = a + (b + c) \bmod N$

$(ab)c = a(bc) \bmod N$

Distributive: $a(b+c) = ab + ac \bmod N$

$(a+b)c = ac + bc \bmod N$

Commutative: $a + b = b + a \bmod N$

$ab = ba \bmod N$

Additive identity: $a + 0 = 0 + a = a \bmod N$

Multiplicative identity: $1a = a1 = a \bmod N$

Additive inverse: $a + (-a) + (-a) + a = 0 \bmod N$

Multiplicative Inverse:

In general, some numbers have inverses but not all

Example: 3 mod 10

-3 * 7 = 21 = 1+ 20 = 1+ 2*10 = 1mod 10

-so 3 = 7^{-1} mod 10

Example: 2 mod 10

-there is no number b such that 2b = 1mod 10

We define (Z/NZ)* to be all invertible elements in Z/NZ

When does a number have aninverse?

element a has a muiltiplicative inverse $\mod N$ if and only if $gcd(a, N) = 1$

if $gcd(a, N) = 1$, we can write $ra + sN = 1$ so $ra = 1 \mod N$ where $r = a^{-1}$
if a has inverse r, then $ra = 1 \mod N$, which means $ra + sN = 1$ for some s which means $gcd(a, N) = 1$

inverse a^{-1} is unique modulo N and can be efficiently computed using extended euclidean algorithm

Prime Numbers

natural number N is prime if its only divisors is 1 and N
if p is prime and p | ab then p | a or p | b
any natural number N has unique factorisation N = $p_1^{r_1} p_2^{r_2} \dots p_s^{r_s}$

gcd(a,p)$\in \{1, p\}$ when p is prime

Corollary of Bézout's identity: if p is prime then for any a∈{1,…,p-1} there is an inverse r such that 1= ra mod p

Finite Fields

What is $(Z/pZ)^*$ when p is a prime?

The elements that have gcd(x,p) = 1, but for a prime this is all x not divisible by p, so $F_p^* = \{1,2,3,\ldots,p-1\}$

A field is a commutative ring where all non-zero elements are (multiplicatively) invertible

F_p is a field with p elements (or order p) so is called a finite field

Prime Order Finite Fields

Associative: $(a + b) + c = a + (b + c) \bmod N$

$(ab)c = a(bc) \bmod N$

Distributive: $a(b+c) = ab + ac \bmod N$

$(a+b)c = ac + bc \bmod N$

Commutative: $a + b = b + a \bmod N$

$ab = ba \bmod N$

Additive identity: $a + 0 = 0 + a = a \bmod N$

Multiplicative identity: $1a = a1 = a \bmod N$

Additive inverse: $a + (-a) + (-a) + a = 0 \bmod N$

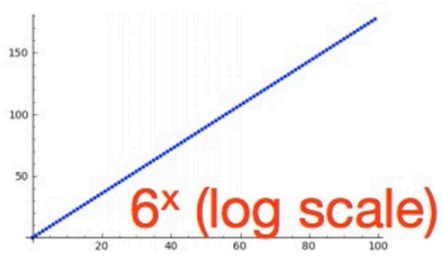
Multiplicative inverse: $a * a^{-1} = 1 \bmod N$

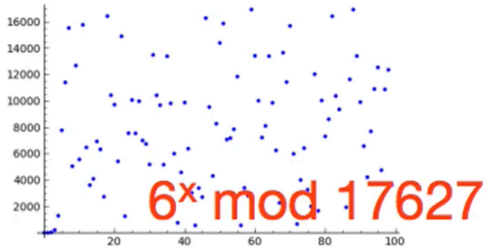
2.4 Math and Cryptography

Discrete Logarithm

Discrete logarithm problem: for a fixed prime p, given g and y, find x such that $g^x = y \bmod p$

Example: $6^x = 10000 \bmod 17627$





This problem seems to be very difficult to solve

(like for modern computers and large enough p, until the heat death of the sun)

Ring (Z/NZ)*

Euler totient function φ is $\varphi(N) = |\{x \text{ in } \{0,\ldots,N-1\} \mid \gcd(x,N) = 1\}|$

Euler's theorem: $x^{\varphi(N)} = 1 \bmod N$ for $x \in (Z/NZ)^*$ Now

let N = pq for p and q two different odd primes $\varphi(N) =$

$$\varphi(pq) = pq - |x : \gcd(x,pq) \neq 1\}|$$
$$= pq - |\{x : p \mid x\}| - |\{x : q \mid x\}| + |\{0\}|$$
$$= pq - q - p + 1 = (p-1)(q-1)$$

This means that $(Z/NZ)^*$ has $\varphi(N) = (p-1)(q-1)$ elements

RSA

RSA problem: given an integer N = pq, find p and q

Example: the RSA-1024 challenge is to find p and q for N =

135066410865995223349603216278805969938881475605667027524485143851526510604859533833940287150571909441798207282164471551373680419703964191743046496589274256239341020864383202110372958725762358509643110564073501508187510676594629205563685529475213500852879413773285339061097505443349998111500569772368909275 63

One-Way Functions

More generally, discrete log and RSA are examples of something called a one-way function

This is a function f( ) such that
- (1) it is easy to compute f(x) for all x, but
- (2) it is assumed to be very difficult to compute x given f(x), or in fact to compute any y such that f(y) = f(x)

Discrete log: $f(x) = g^x \bmod p$

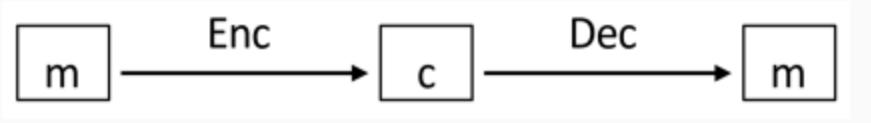
RSA: f(p,q) = pq

Week 3 Confidentiality

- About how to **keep data private** where think about how:
  - to communicate secretly
  - to establish a basis for secure communication

Cryptographer: person woh makes cryptography
Crytanalyst: person who breaks cryptography

Code: semantic translation (A means B)
Ciphertext: encryption of underlying plaintext



Note that cryptography uses binary view of security: secure or insecure

- if you get it right - secure for decades
- if wrong - no security at all!

Encryption

Threat Model

Motivation:

- recover key: learn all future plaintexts
- recover plaintext: learn this specific plaintext
- distinguish plaintext: learn a single bit about plaintext

Capabilities:

- known ciphertext: know ciphertext
- known algorithm: know scheme used to encrypt
- known plaintext: (Partial) information about plaintext
- chosen plaintext: adversary picked plaintext
- chosen ciphertext: adversary picked ciphertext

strongest security statement: adversary with strongest capabilities can't achieve even the weakest goal

Historical ciphers

Monoalphabetic substitution

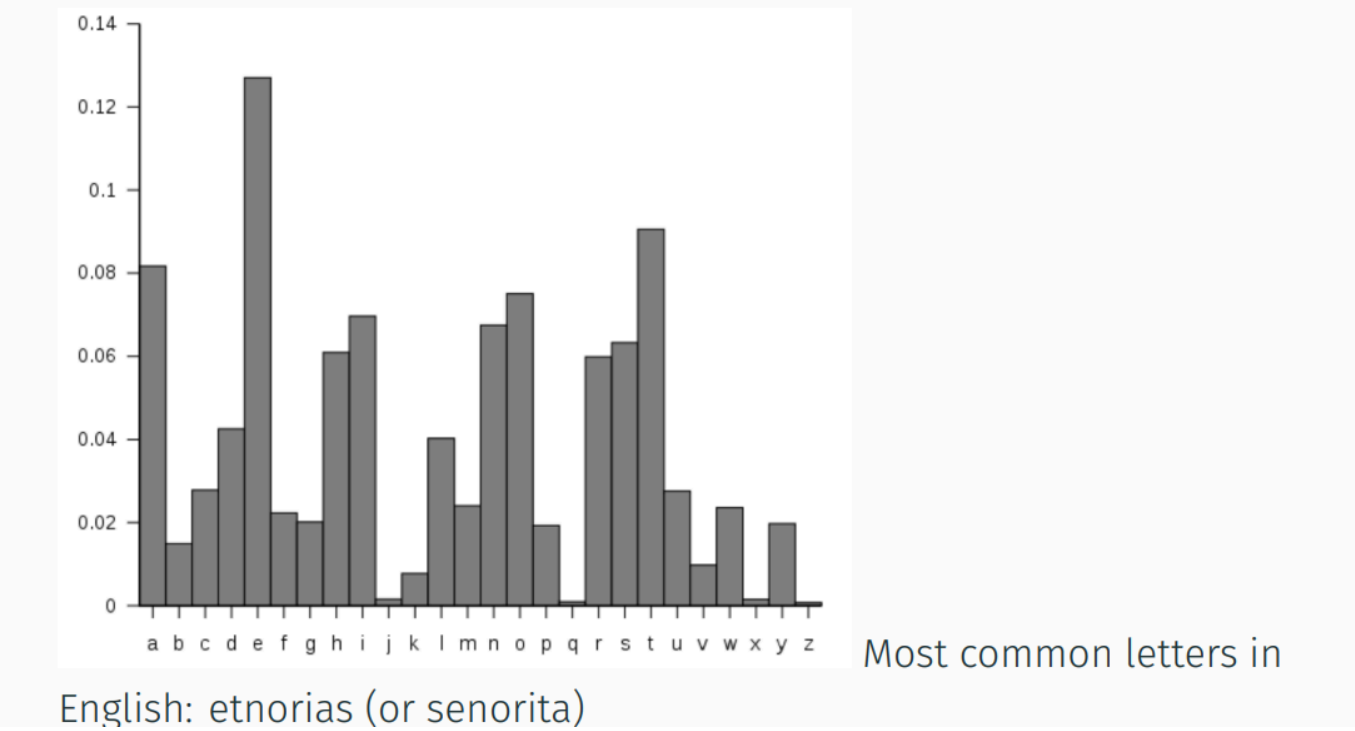
Monoalphabetic substitution cipher applies permutation $\pi : \Sigma \to \Sigma'$

in caesar cipher, π is rotation: $\beta : \beta +$ key $\mod 26$
More generally, could have $\pi(a) = o, \pi(b) = m$, etc, or Σ' might not be the same language as Σ

Example: Caesar shift cipher

Shifting where 'A' becomes 'D'

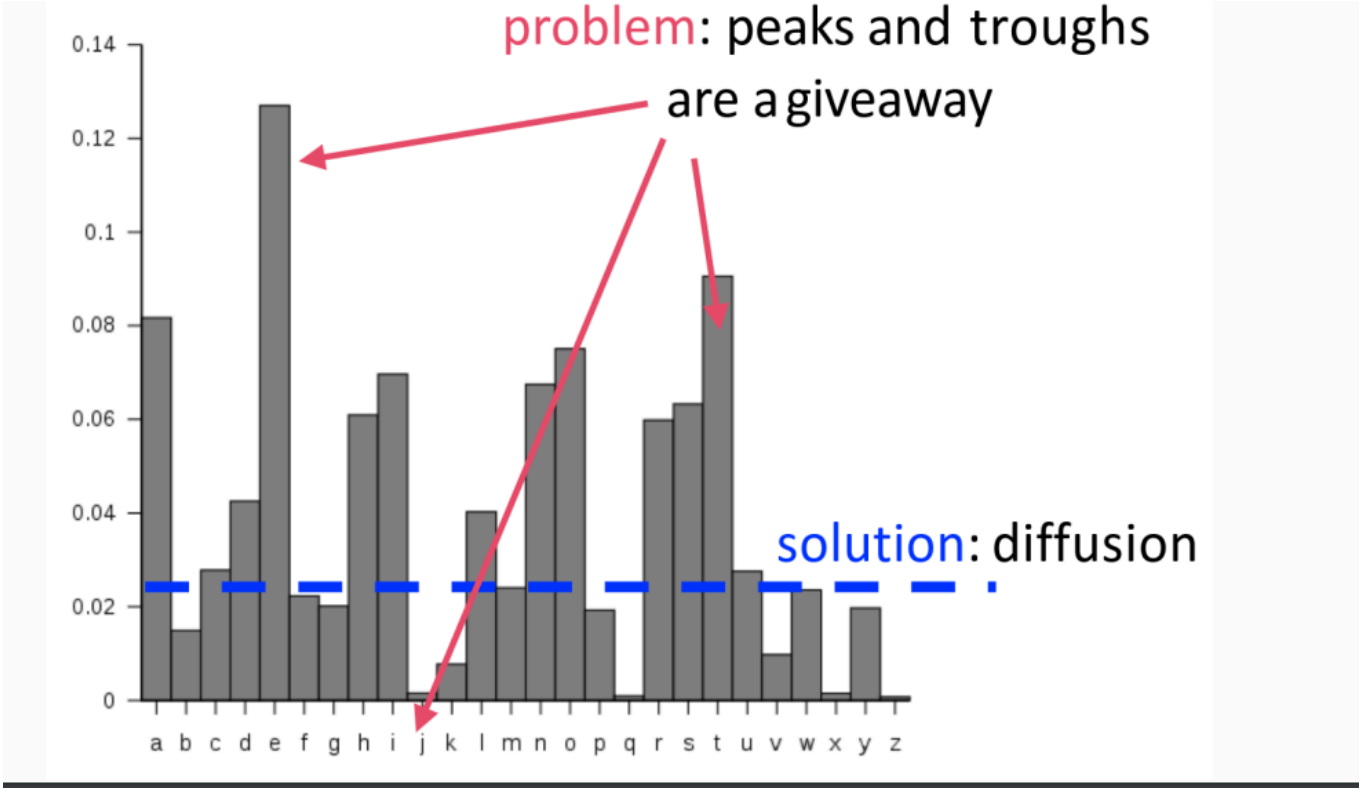
Frequency Analysis



knowing the distribution of letters in plaintext by how common are these letters are in the language

you find the most common letter and make a guess on the caesar shift (e.g. h -> e shift means key is x(23))

problem of frequency analysis and solution

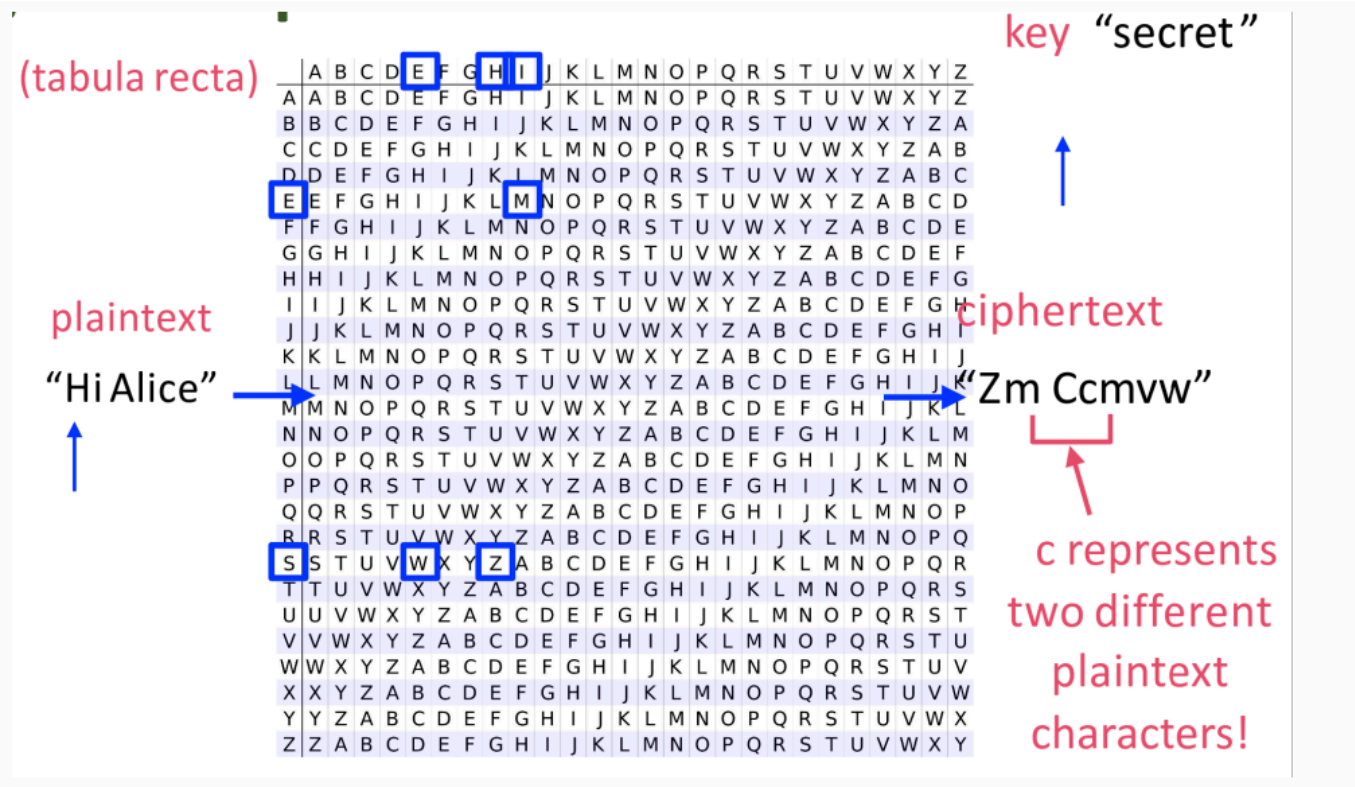


Polyalphabetic Substitution

- Polyalphabetic substitutino cipher rotates through permuitations $\pi : \Sigma \rightarrow \Sigma'$
- e.g. rotor machines like Engima

Example: Vigenere Cipher

Extension of the Caesar cipher which uses a table (columns to encrypt, rows to decrypt)
Uses multiple caesar ciphers where a different key is used for each cipher



e.g. "Hi Alice"

Shift H to what is "s" -> "Z"
"i" to what is "e" (4) -> "m"

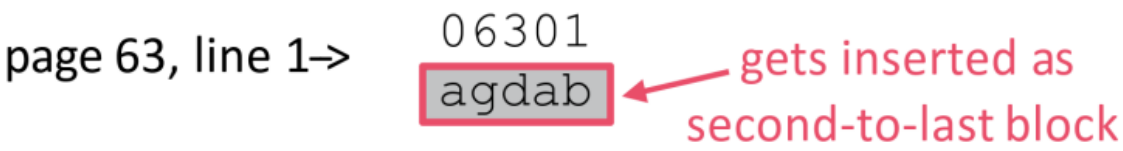
result: "Zm Ccmvw"

Vigenere Cryptanalysis

- same key letters encrypt same plaintext letters
- repeated n-grams reveal length of key (because distance between = multiple of key length, so key length = lcd(distances))
- **problem**: key length reduces to monoalphabetic
- **solution**: use a really long key

Running Key Cipher

- Split ciphertext into blocks of five charcters
- use indicator block to say where in keyt to begin



- **problem**: repetition in key yield patterns

- **solution**: use a long random key

One-Time Pad

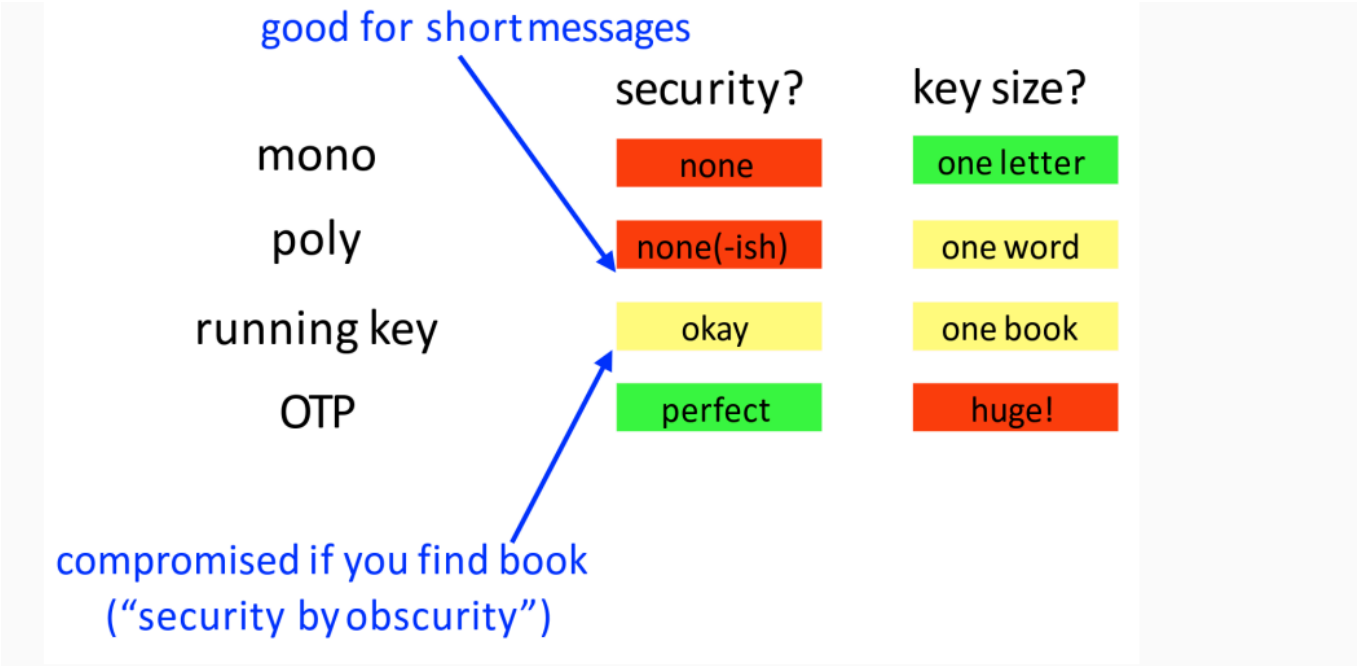
```
hialice|hibob|howsitgoing|okayyou|howsitgoing
ujakjywibavnscknkveoldxhinrovngdytlwkhyinncrhih
    ↓
bravraa|iiwbt|rbgnmhrrfuo|fyvlers|skgzgbtbken
```

- Also known as perfect substitution cipher
- Any ciphertext could decrypt to any plaintext (if you use the key just once; otherwise reduces to running key)
- no way to break this unless you have the key
- encrypt one letter at a time where its encrypted independently (hard to obtain the key)
- hard for adversary to try and intercept/decrypt the generated ciphertext

Key Material

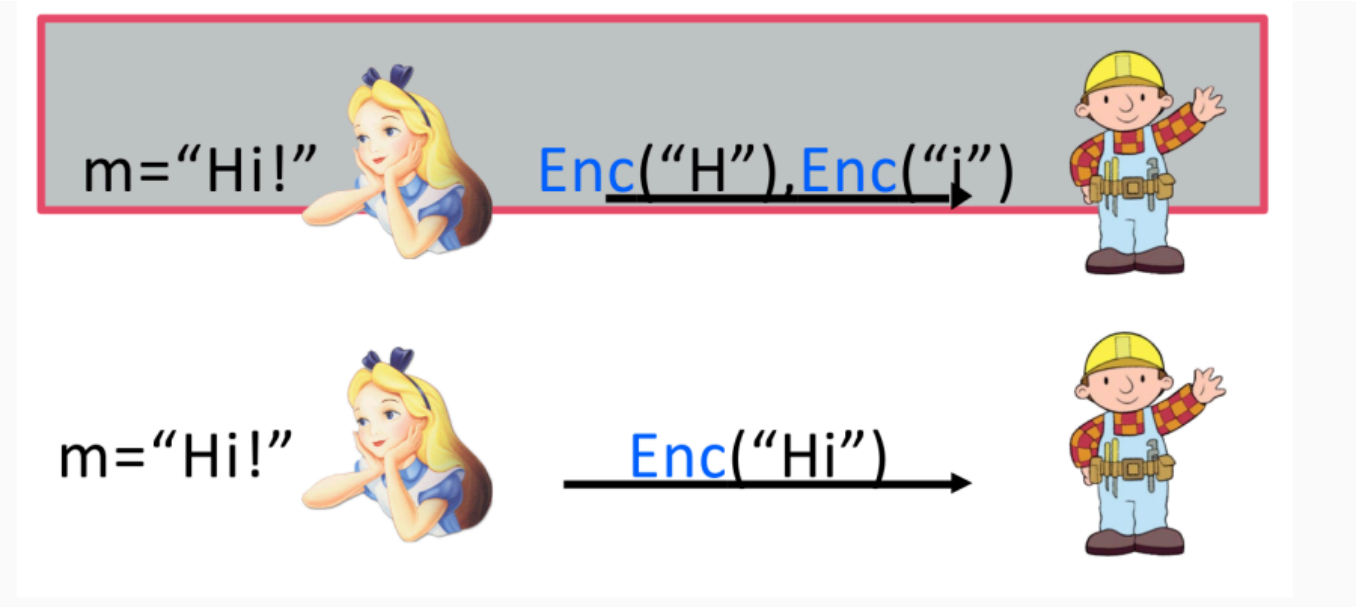
- key should be as long as the plaintext
- maybe use pages of keys which is to be destroyed after use

Trade-Offs of historical ciphers

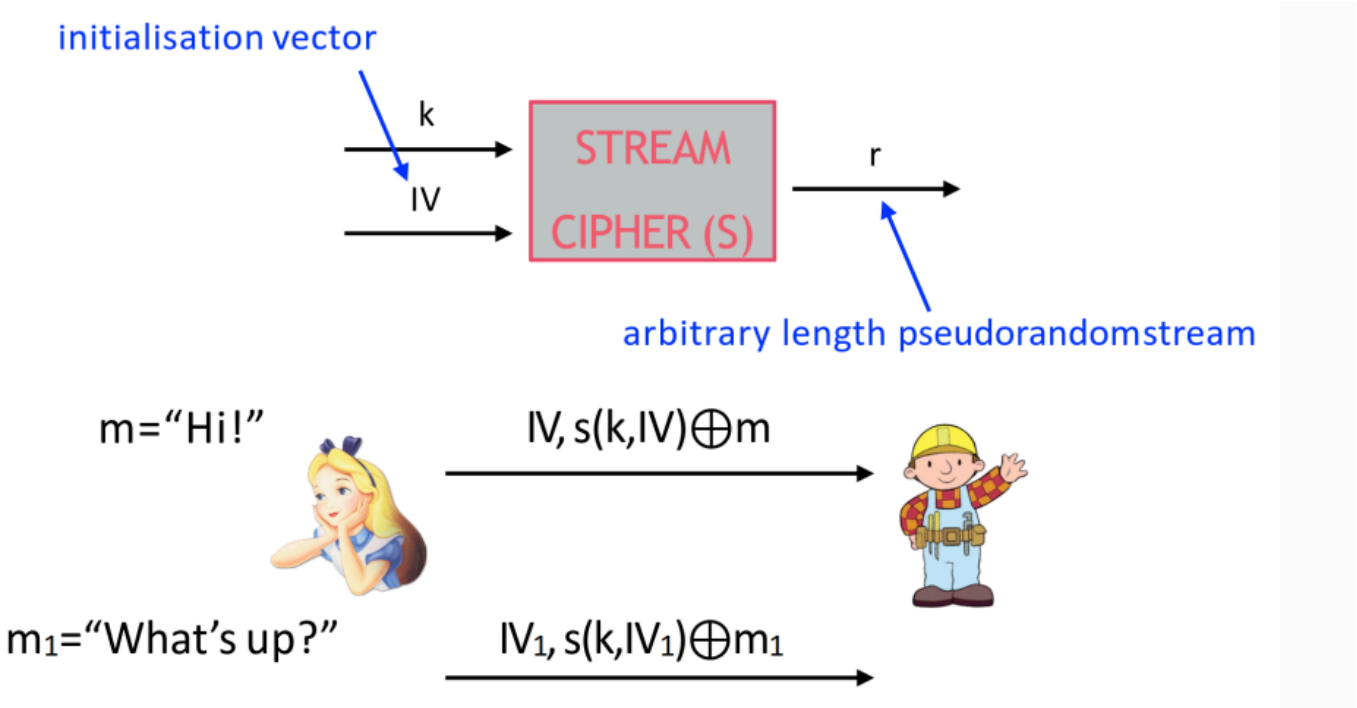


Symmetric Key Cryptography

Two types: **stream ciphers** (encrypted/decrypted one bit at a time) and **block ciphers** (encrypted/decrypted in blocks)



Stream Ciphers - bit by bit

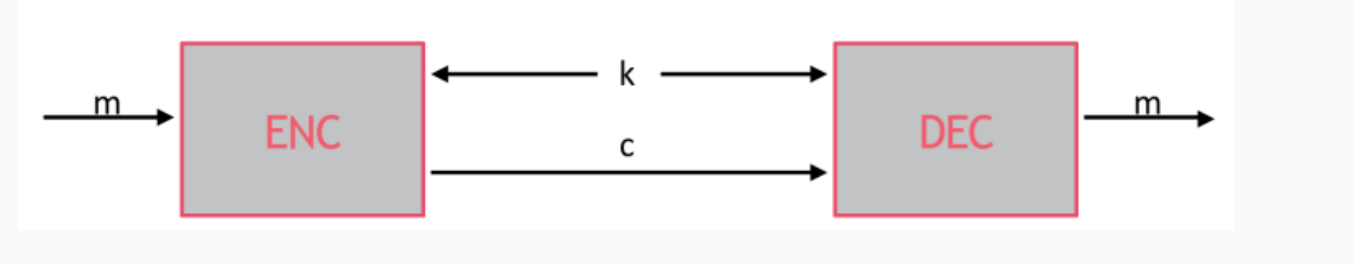


Randomness s(k,IV) is designed to mimic randomness in one-time pad: easier to generate but less secure (heuristics)

like with otp, if Alice re-uses same IV then there is NO security

Block Ciphers - done in blocks

Encrypt blocks of plaintext (block length same length as key)



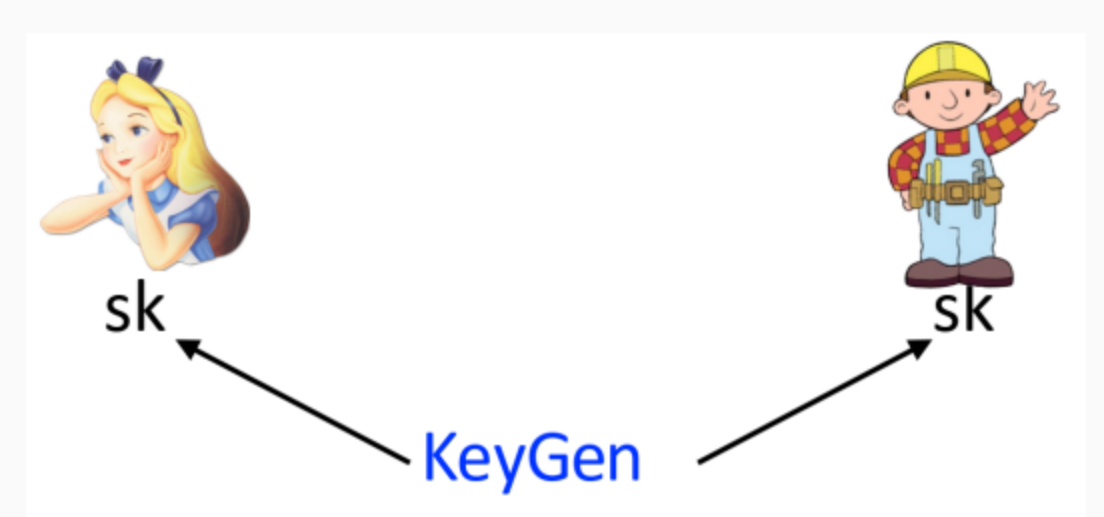
Key is short random string (128, 192, or 256 bits)

Plaintext/Ciphertexts are short blocks of same length (if plaintext is shorter than key it must be padded to match)

Correctness: Dec(k, Enc(k,m)) = m
Security, without k, Enc acts as a random permutation

Symmetric Encryption

Sender and receiver has same key
Key establishment (generation) is used when two parties agree on a key

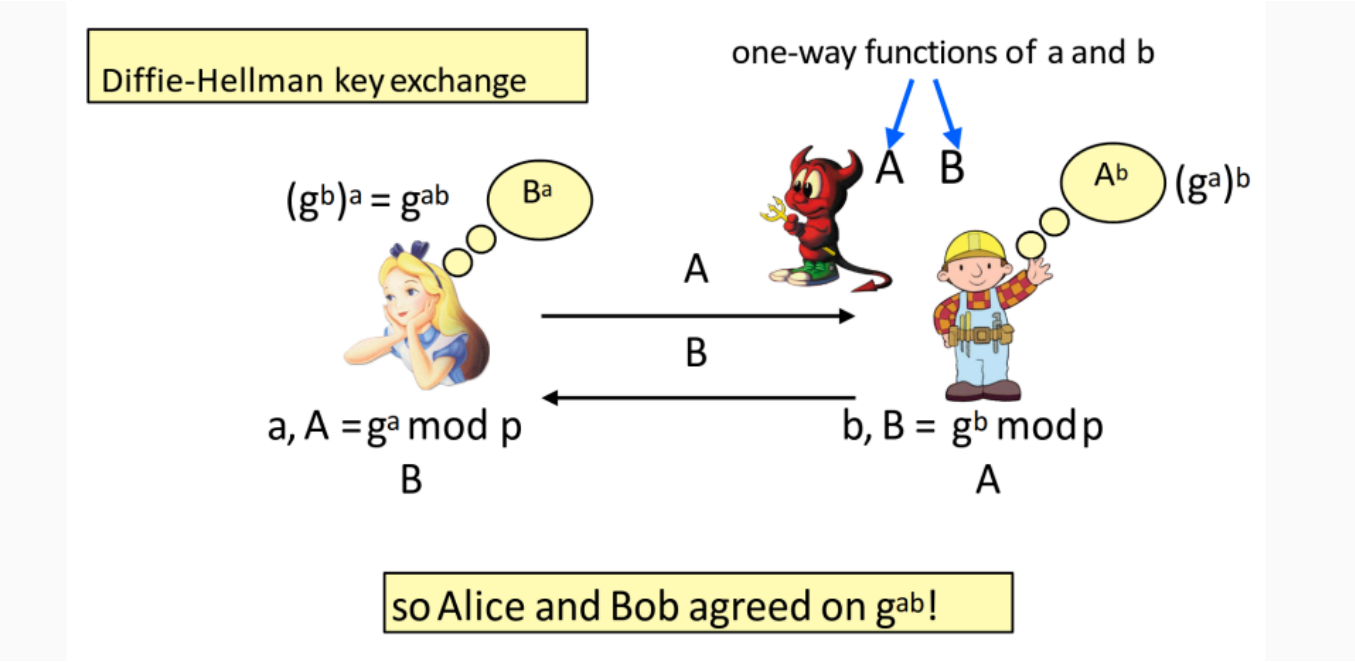


Issues with sharing key:

- need a secure channel to prevent eavesdropping

Diffie-Hellman key exchange

Allow keys to be shared over insecure channels (uses number theory/modular arithmetic properties to have a shared key)



The agreed key is in form g^{ab}
alice generates random number (aka her private key) a and perform the calculation $g^a \mod p$ which obtains A
She will send A (aka $g^a \mod p$) to Bob (she also can send g and p as they can be public (agreed by both Alice and Bob), but NOT a)
Note that p must be a prime number and g is typically a primitive root modulo p
g^a cannot convert to a unless performing discrete logarithms (which is computationally costly to perform - takes a lot of reosurces)

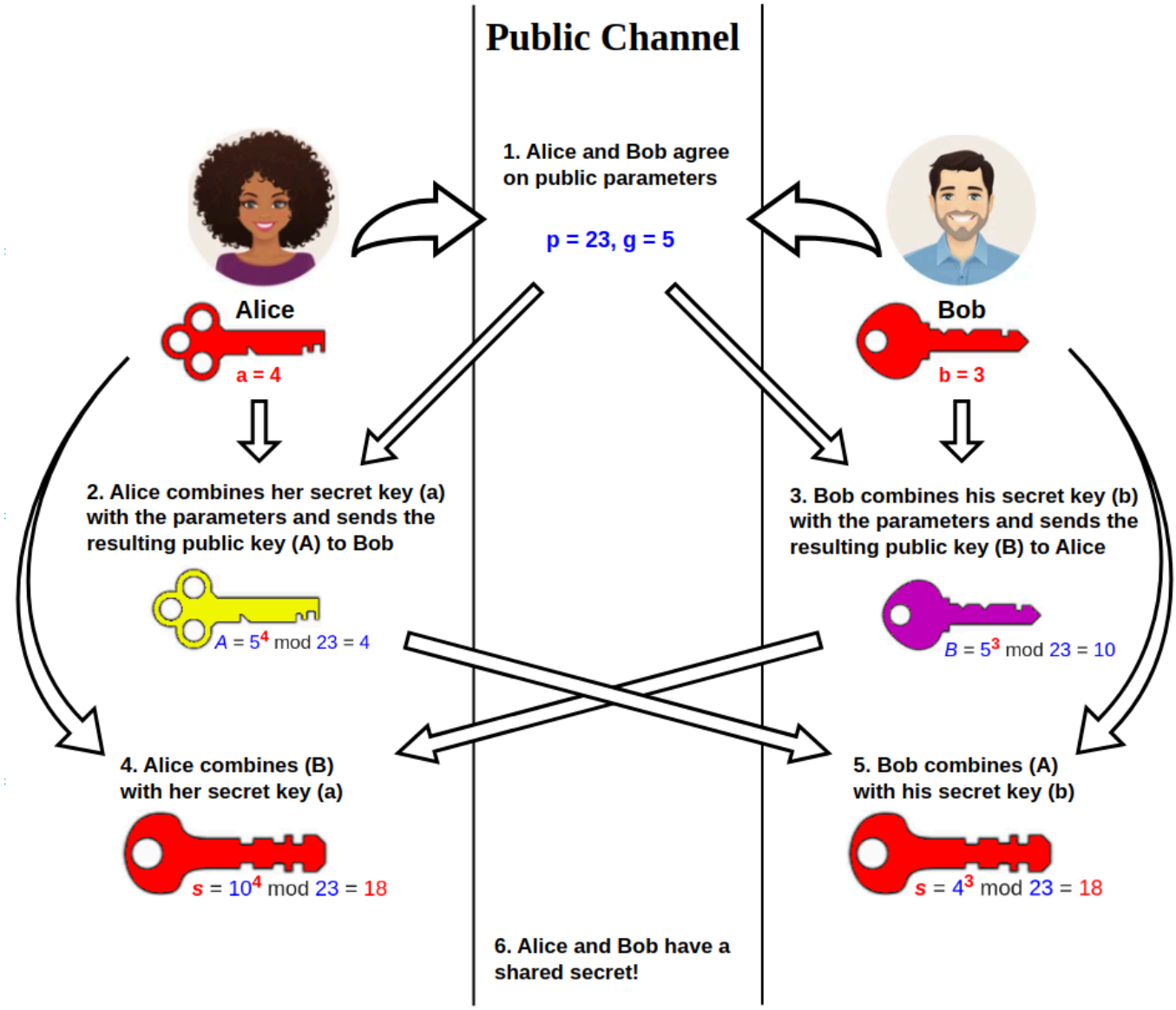
Bob generates secret number (his private key) b and performs $g^b \mod p$ which is expressed as B, which is sent to ALice over the channel

shared key: g^{ab} (equiavlent to g^{ba})

all Bob has to do with A is to raise it to the power of b which gets $(g^a)^b$ and vice versa (which is the agreed symmetric key)

summary:

- Alice computes: K = B^a mod p = (g^b)^a mod p = g^(ab) mod p
- Bob computes: K = A^b mod p = (g^a)^b mod p = g^(ab) mod p
- They will both reach the same value despite starting with different information

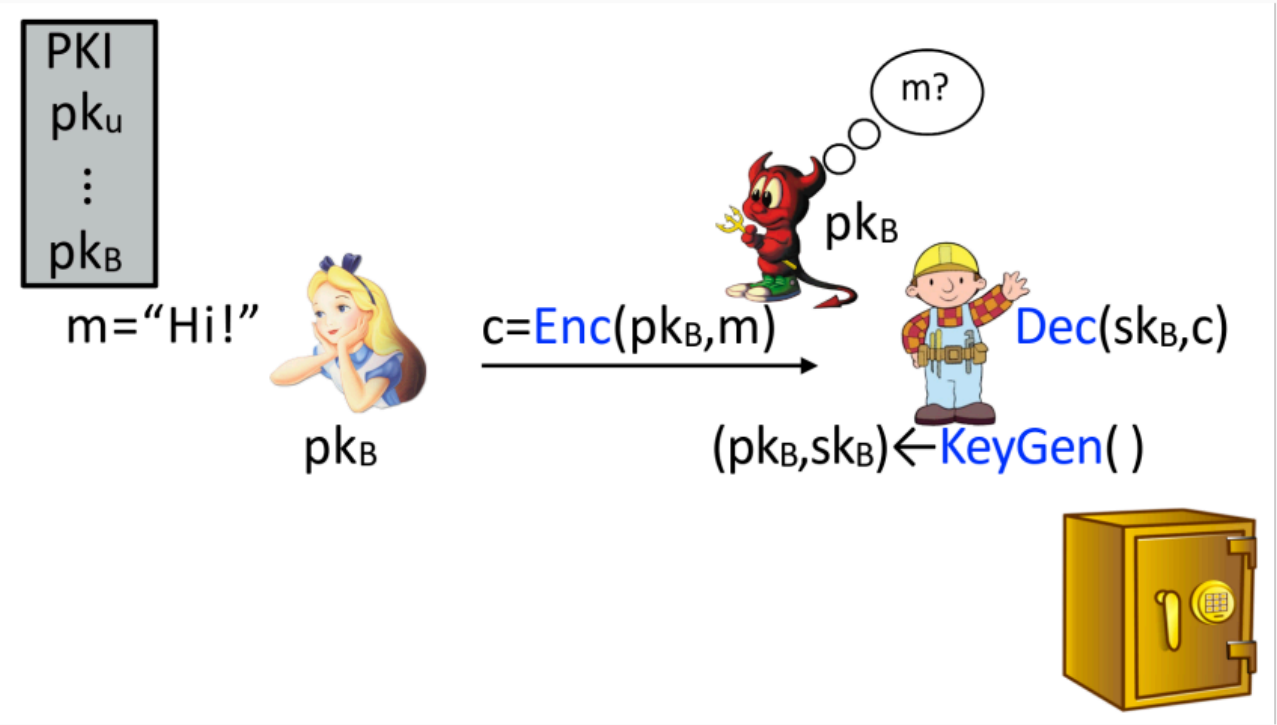


Public key cryptography (Asymmetric encryption)

Uses generated **two keys**:

- **public key** -> encrypt messages
  - will be published to a PKI
- **secret (private) key** -> decrypt received messages
  - kept to the user

PKI - public key infrastructure (directory to store public keys)
pk_u - public key of a user (e.g. pk_B = Bob's public key)



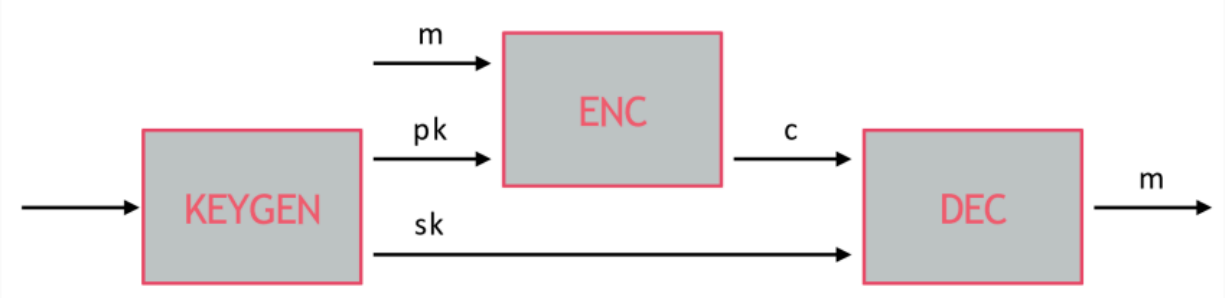
Steps to generate a key:

- A key generator generates a public/secret key pair and the public key is then published to a PKI.

Steps to send a message:

- **Alice** will retrieve **Bob's public key**, from a PKI, and **encrypt** her message using Bob's public key to obtain a **ciphertext**
- The encrypted message (**ciphertext**) will be sent to Bob where it'll be **decrypted** using his **secret (private) key** to obtain the original **plaintext**

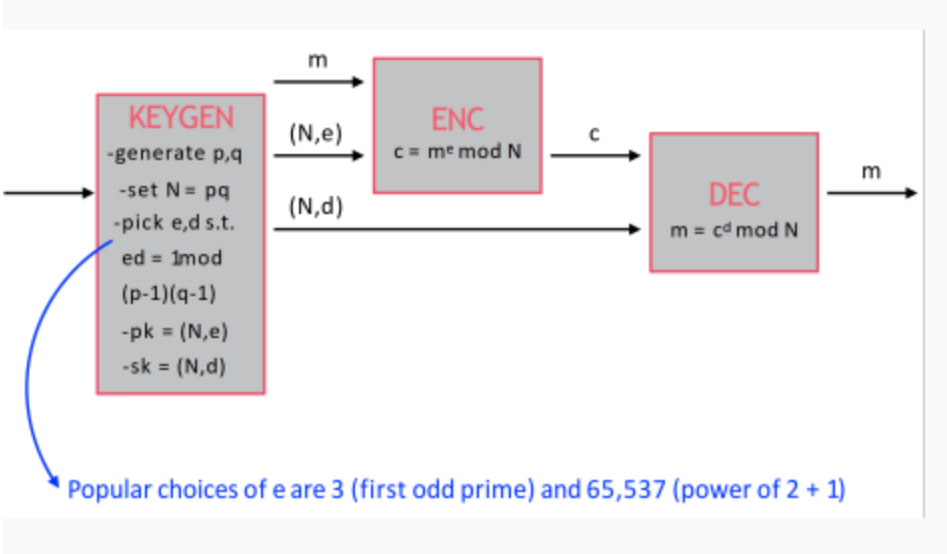
Threat model:
motivation: distinguish plaintext - learn a single bit about plaintext
capabilities: chosen plaintext- adversary picked plaintext



Correctness: For all (pk, sk) produced by KeyGen and messages m, $Dec(sk, Enc(pk, m)) = m$

Security: an adversary who can see decryptions of chosen ciphertexts and can piuck two arbitrary plaintexts should not be able to distinguish the encryption of one of them from the encryption of the other (IND-CCA security)
encrypting the same message twice won't get you the same result

RSA Encryption



- Involves 3 steps:
  - key generation
  - encryption
  - decrpytion

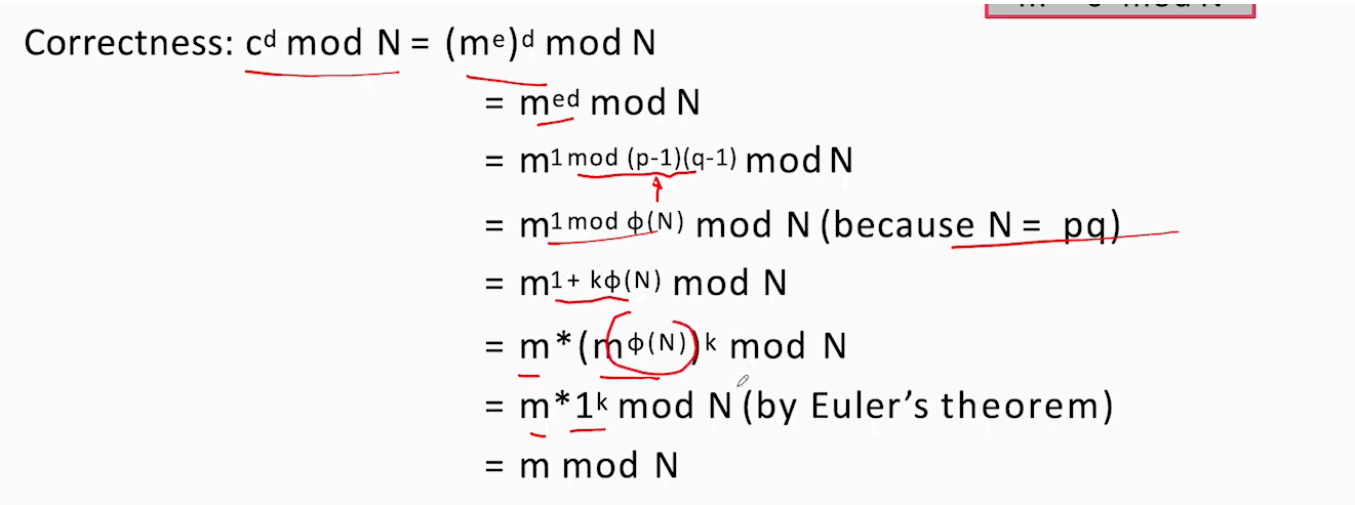
Steps:

- Choose 2 large primes p and q (can be 1024,2048 bits long)
- Let $N = p \times q$ (multiply thjem goether)
- choose a secret key e s.t. $gcd(e, \varphi(N)) - 1$
  - (where $\varphi(N) = (p-1) \times (q-1)$)
- find a public key d such that $e \times d = 1 \mod \varphi(N)$
  - where e and d needs to be **co-prime**
- d is modular inverse of e : $d = e^{-1} \mod \varphi(N)$

public key will be (N, d), private key will be (N, e)

to encrypt message m, compute $c = m^e \mod N$
to decrypt ciphertext c, compute $m = c^d \mod \varphi(N)$

correctness:



Security of RSA

What if you could factor N?

- compute $\phi(N) = (p-1)(q-1)$
- compute $d = e^{-1} \mod (p-1)(q-1)$
- use the d to decrypt

shows you can break RSA even if you only know $\phi(N)$

BUT itll take a very long time (computationally) to factor N (maybe quantum computers can do it - but they do not exist yet)

- hence we use RSA-2048 (as RSA-1024 is dangerous - can take maybe 10 years to crack)

Why textbook RSA:

- no IND-CCA - message recovery attack
  - no IND-CPA - adversary who can pick M_0 and m_1 can compute $(m_b)^e \mod N$ for $b \in \{0,1\}$ so can clearly distinguish
  - Enc is completely deterministic (no randomness to hide value of m)
  - hence we use $\mathrm{RSA\text{-}OAEP}$

Message Recovery Attack

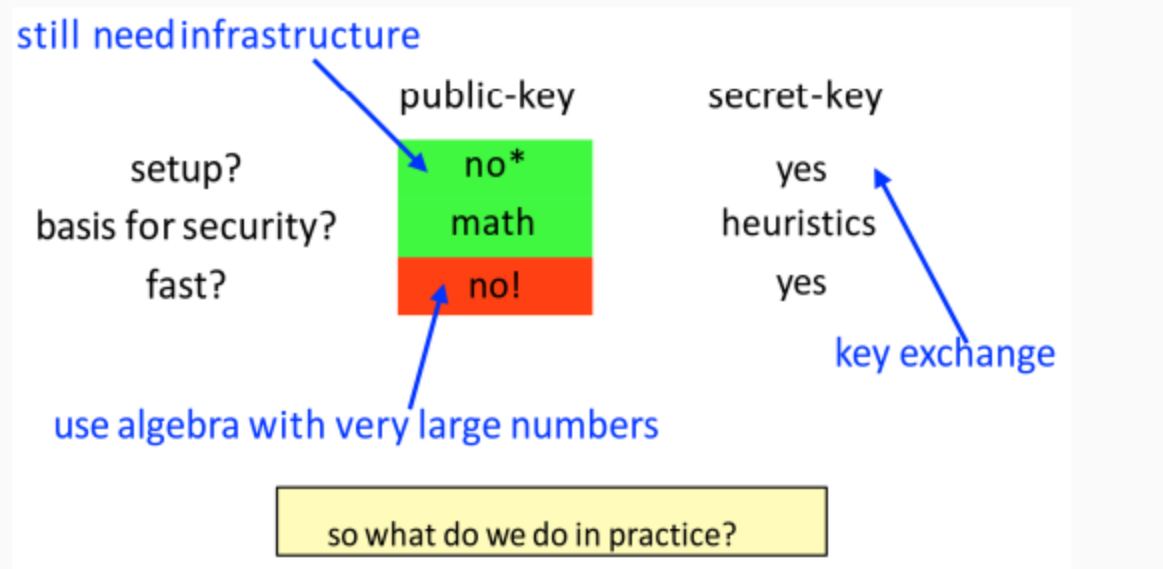
Given priavte key (N, e) and ciphertext c:
compute $C_r = c \times r^e \mod N$

get decrpytion m_r of c_r (chosen ciphertext)

Compute:

$$
\begin{aligned}
m_r \cdot r^{-1} \mod N &= (c \cdot r^e)^d \cdot r^{-1} \mod N \\
&= c^d \cdot r^{ed} \cdot r^{-1} \mod N \\
&= (m^e)^d \cdot r^{ed} \cdot r^{-1} \mod N \\
&= m^{ed} \cdot r \cdot r^{-1} \mod N \\
&= m \mod N
\end{aligned}
$$

Sumamry of Public Key Encryption

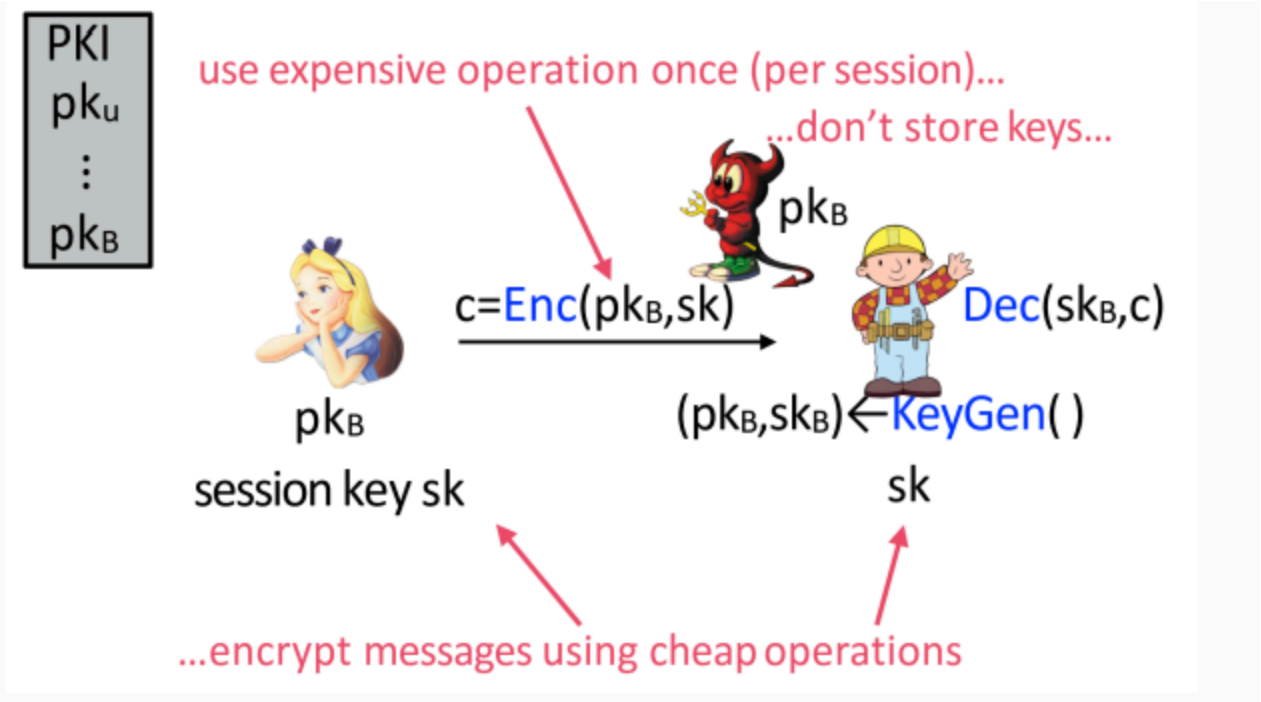


Encrypting web traffic

When receiving content where they're split in packets, these content needs to be encrypted/decrypted

Anyone can read web traffic (i.e. routers)

Session Keys



Uses both symmetric/public key cryptography
Alice will generate a sesion key s_k which is encrypted with Bob's public key pk_B
Bob will decrypt the session key with his secret key sk_B
Session key is used as symmetric key and bothw ill use the symmetric key for content exchange

Packet Format:

4-bit version	4-bit Header len	8-bit type of service	16-bit total length (in bytes)
16-bit identification		3-bit flags	13-bit fragment offset
8-bit time to live (TTL)	8-bit protocol	16-bit header checksum	
Bob's IP address			
Alice's IP address			
Options (if any)			
Enc(sk,<Content at `hi.html` (part 1of N)>)			

An example of **Hybrid Encryption**

To encrypt a long message m:

- pick a random (symmetric) sesion key K

- encrypt K with $c_1 = PKE.Enc(pk, K)$
- encrypt M with c_2 = $SKE.Enc(K, m)$
- ciphertext is $c = (c_1, c_2)$

to decrypt and recover m:

- compute $K = PKE.Dec(sk, c_1)$
- compute $m = SKE.Dec(K, c_2)$
- ciphertext is $c = (c1, c2)$

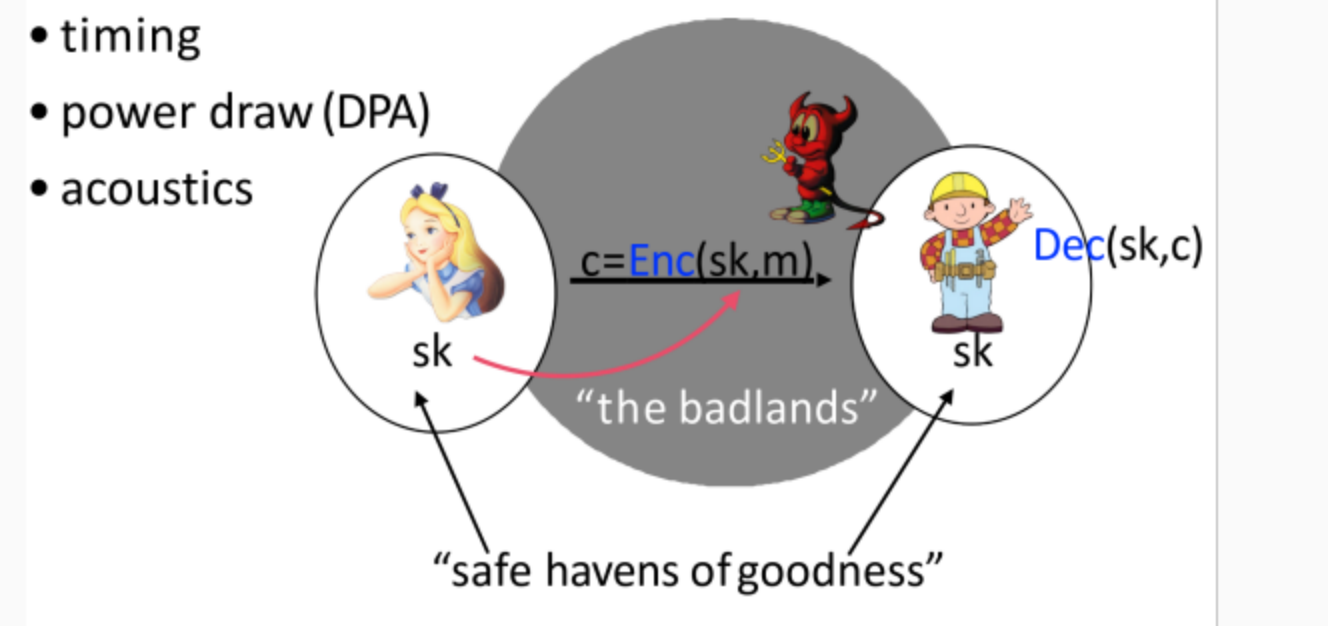
Does encrypted web traffic still reveal IP addresses?

- **yes**, but to avoid this, you use proxies or onion routing (e.g. Tor)

Is communication channel the only attack surface

- no! side channels exploit weakness on either side

Side Channels



Adversary can be in side channels where they can intercept

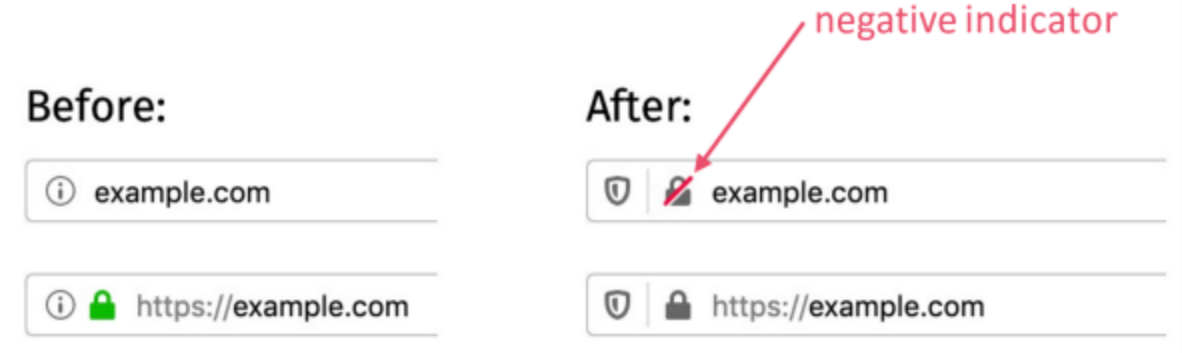
Using Encryption on the Web

HTTPS: green padlock to show traffic was encrypted (you are using secure HTTP aka HTTPS)

HTTP indicators in 2018

Status	Chrome	Firefox	Edge	Safari	Opera	Notes
HTTP	Not secure	www				No indicator shown.
Mixed HTTPS (active content)	https://	https://				Some browsers warn against mixed active content.
Mixed HTTPS (passive content)	Secure	https://				All browsers show a positive indicator. Most commonly a lock icon.
HTTPS (DV)	Secure	https://				All browsers show a positive indicator.
HTTPS (OV)	Secure	https://				All browsers show a positive indicator.
HTTPS (EV)	SSL Corp [US]	SSL Corp [US]	SSL Corp [US]	SSL Corp	SSL Corp [US]	All browsers show a positive indicator, but most browsers show the verified name of the organization in a green color.

Negative indicators - shows that web traffic is not encrypted



Why negative indicators?

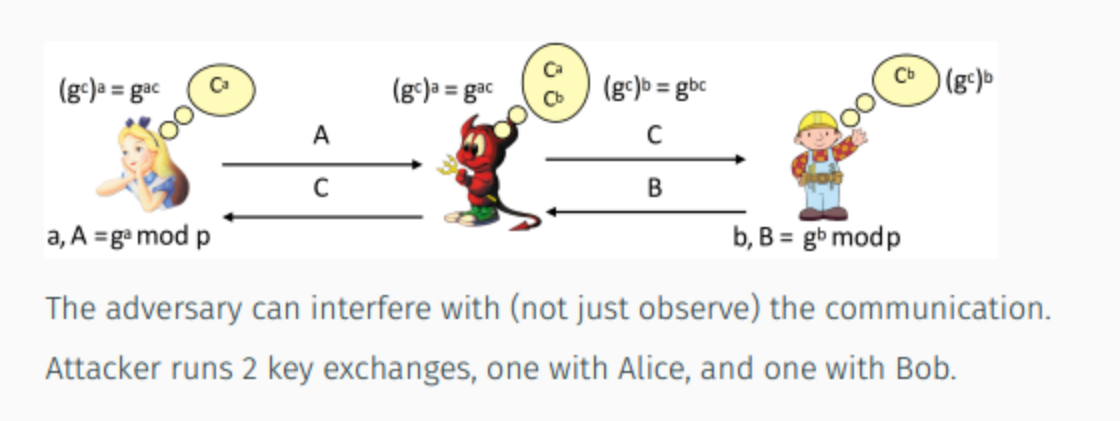
- positive indicators were not usable and did not protect users - hence browsers are moving towards negative indicators instead

Week 4: Integrity

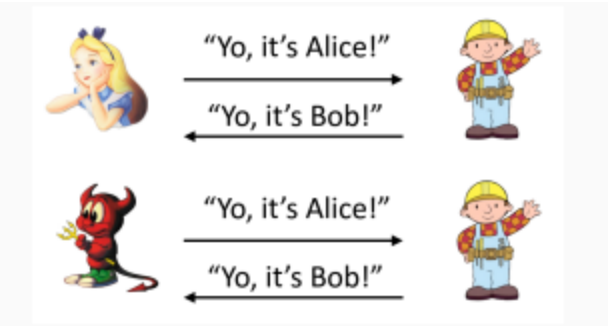
What is **integrity**?

- **system and data have not been improperly altered**
- how to :
  - know who you're talking to
  - know what data you're getting

Both public key encryption / diffie-hellman key exchange can be subject to **man-in-the-middle (MitM) attack** by an adversary

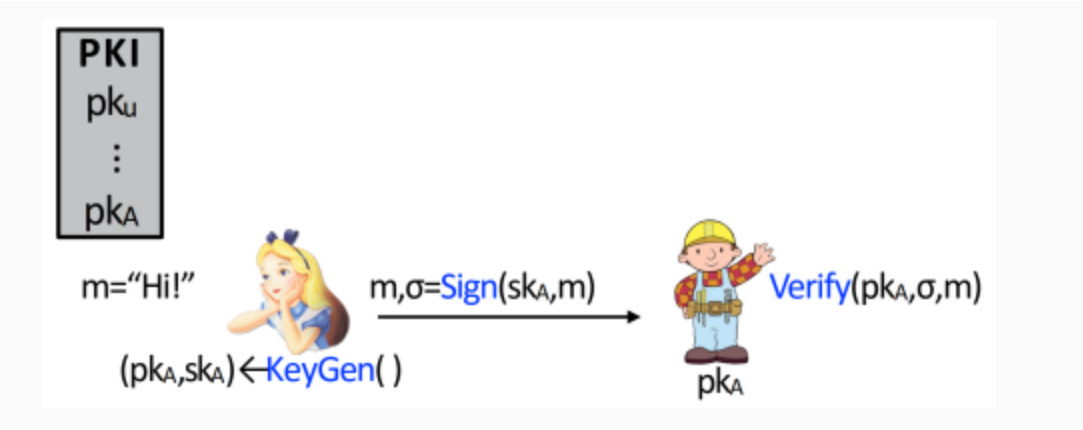


Hence the need for both parties to **identify each other** (integrity) before exchanging keys/messages - but needs to be "unforgeable"
e.g.



4.1 Digital Signatures / MACs

Digital Signatures



Similar to public key encryption but uses the keys the other way around (the inverse of PKE)

Example (Alice/Bob):

- Alice's **public/secret keys** pairs are **generated** (via a key generation algorithm) at the start - public key gets published to PKI, secret key is only for herself
- **Digital signature** σ for a message m can be **generated** using **Alice's secret key** sk_a which is then **signed** $(m, \sigma = Sign(sk_A, m))$
- Alice's public key is obtained from PKI, where Bob can **verify** the digital signature of the message using **Alice's public key** if the signature is valid $(Verify(pk_A, \sigma, m))$
- The result of the verifcation is returned either 1 or 0

Correctness: valid signatures using valid keys will verify properly (for all k, m and $(pk, sk) \in [KeyGen(1^k)], Verify(pk, m, Sign(sk, m)) = 1)$

Unforgeabiity (EUF-CMA): for a given public key, an adversary can't produce new signatures that verify $((pk, sk) \rightarrow KeyGen(1^k)$, A gets pk and access to oracle $SIgn(m)$, can't output (σ, m) for m not queried to $Sign)$

Threat Model for Digital Signatures
Motivation:

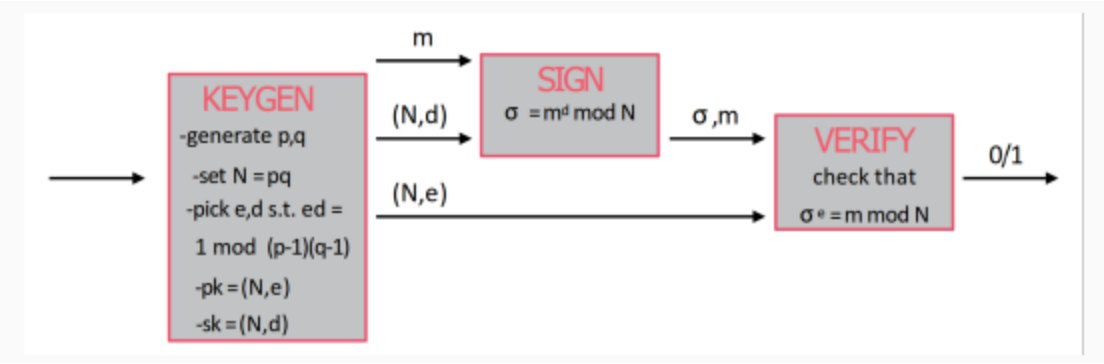
- recover key: sign all future messages
- forge signature: pretend to be someone else

Capabilities:

- known algorithm: know scheme used to sign

- Known signature: (partial) ifnroamtion about signature
- Chosen m,essage: adversary picked messages

Digital Signature - RSA



Key generation is same - use secret key to generate signature, public key to verify

Correctness of RSA

Key Gen: $N = pq$ for large primes p, q. Pick e, d such that $ed = 1 \mod (p-1)(q-1)$. Set $pk = (N, e)$ and $sk = (N, d)$

SIgn: $\sigma = m^d \mod N$. Verify $\sigma^e = m \mod N$?

Correctness:

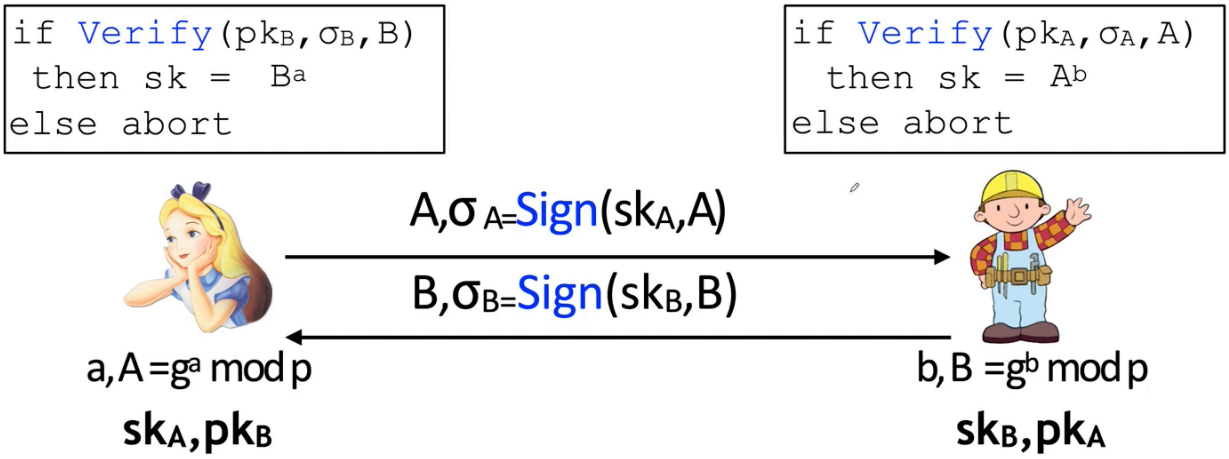
$$
\begin{aligned}
\sigma^e \mod N &= (m^d)^e \mod N \\
&= m^{ed} \mod N \\
&= m^{1 \bmod (p-1)(q-1)} \mod N \\
&= m^{1 \bmod \varphi(N)} \mod N \text{ (because N = pq)} \\
&= m^{1+h\varphi(N)} \mod N \\
&= m \cdot (m^{\varphi(N)})^h \mod N \\
&= m \cdot 1^h \mod N \text{ (by Euler's theorem)} \\
&= m \mod N
\end{aligned}
$$

Security of RSA

Why "textbook" RSA?

- No EUF-CMA: if adversary gets signatures
- σ_1 on m_1 and σ_2 on m_2 then it can create a valid signature $\sigma = \sigma_1.\sigma_2$ on $m_1.m_2$
- this works as the function $f(m) = md$ is homomorphic, so $f(m_1).f(m_2) = f(m_1.m_2)$

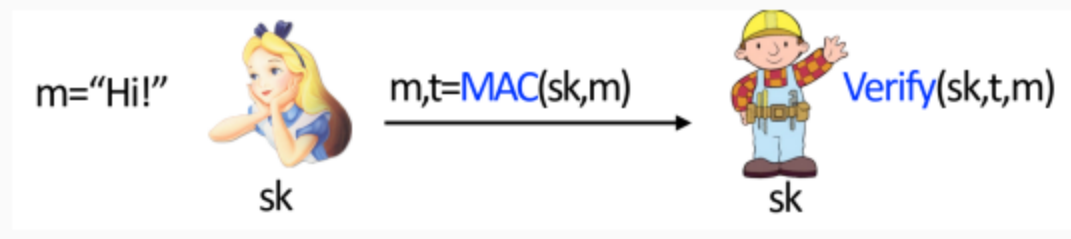
Digital Signatures in Diffie-Hellman Key Exchange



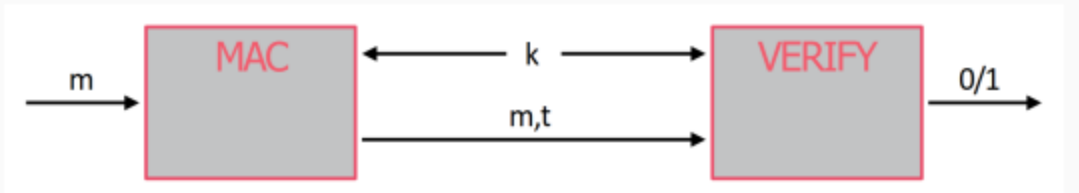
Ensure Alice is actually talking to Bob (not an adversary impersonating Bob), and vice versa

Message Authentication Codes (MAC)

Digital signatures but uses symmetric encryption

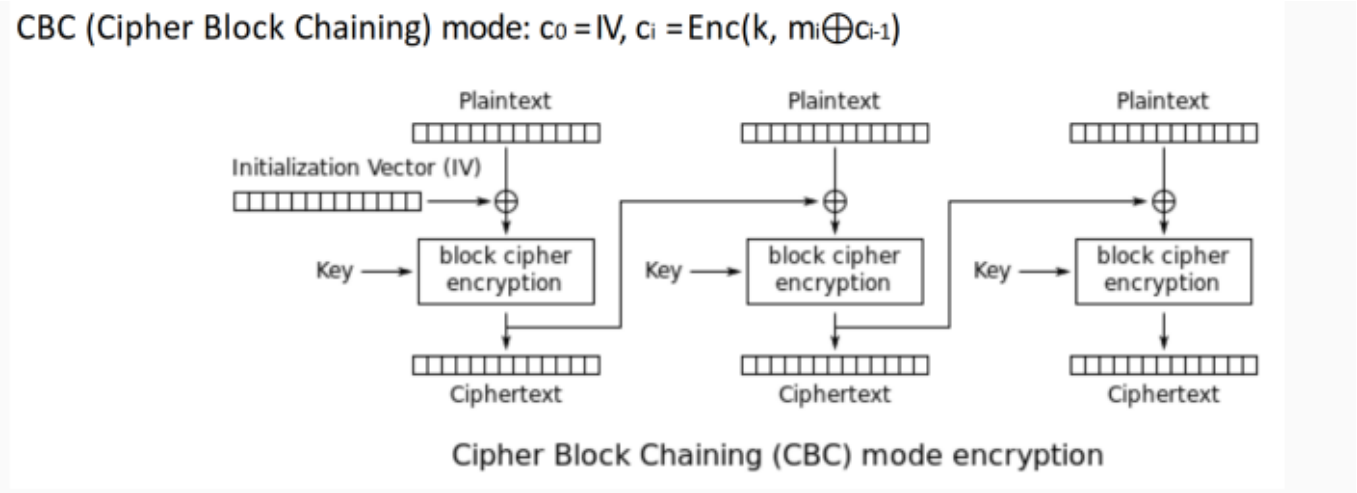


Generate a "tag" t fon message m using secret key. Verify with the same key
Not everyone can do verification - only those with secret key (that is shared) can do that



Correctness: $Verify(k, m, MAC(k, m)) = 1$
Unforgeability: hard to generate $(m, MAC(k, m))$ without knowing k

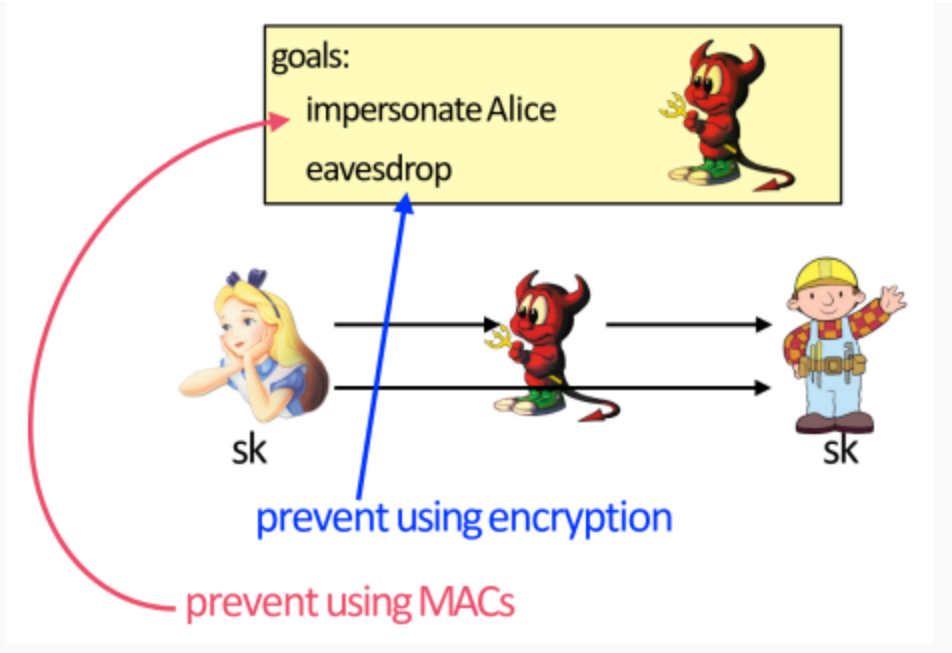
MACs from AES-CBC
IV - initialisation vector



Can use last block of this as MAC: $MAC(k, (m_1, \ldots, m_n)) = c_n$ using fixed IV for c_0, $Verify(k, m, t)$ recomputes MAc and checks equality with t

Authenticated Encryption (AEAD)

Combining confidentiality (encryption) and integrity (MAC)



Threat Model for AEAD

Motivation:

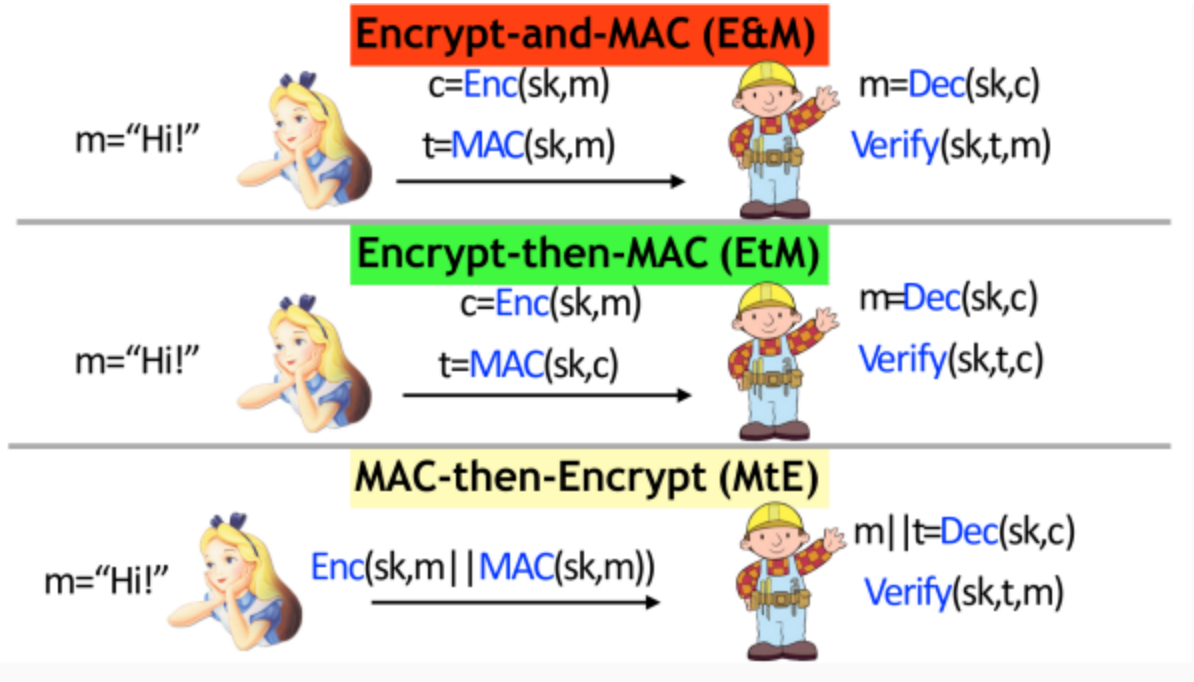
- recover key: learn all future pl;aintext
- recover plaintext: learn this specific plaintext
- distinguish plaintext: learn a single bit about plaintext
- forge plaintext: ciphertext decrypts to plaintext never encrypted by the sender (INT-PTXT)

Capabilities:

- known algorithm: know schemes used to encrypt / MAC

- known ciphertext: (Partial) information about ciphertext
- chosen message: adversary picked messages
- chosen ciphertext: adversary picked ciphertext

Constructing AEAD



Encrypt-and-MAC (E&M):

- you send encrypted message / MAc for other party to decrypt/verify

Encrypt-then-MAC (EtM)

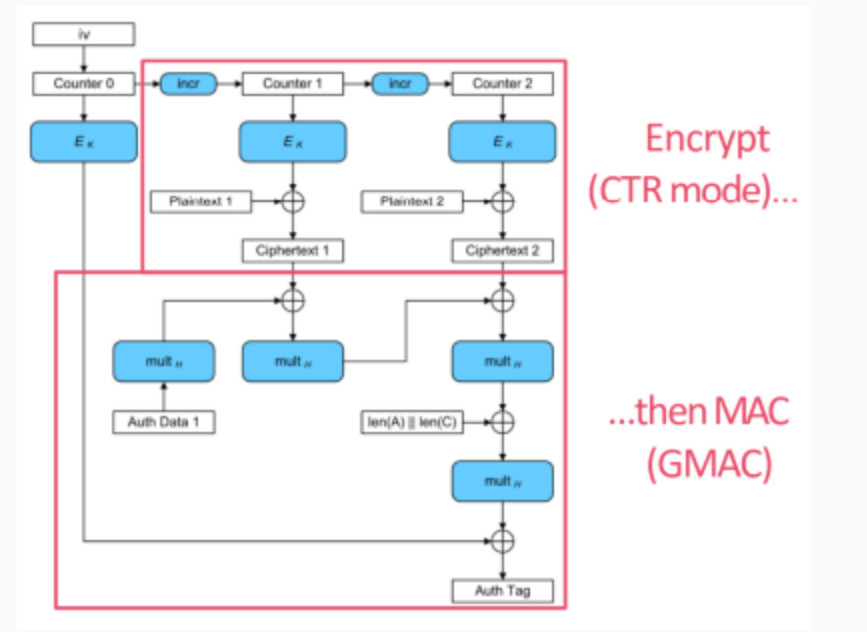
- you encrypt a message, then perform MAC on that ciphertext and send both
- other party will decrypt message / verify the MAC

MAC-then-Encrypt (MtE)

- just one message is sent - encryption using secret key the message concatenated to the MAC (MAC the message and then encrypt)
- Once reach the other party, you decrypt the message and then verify the MAC

Galois Counter Mode (GCM)

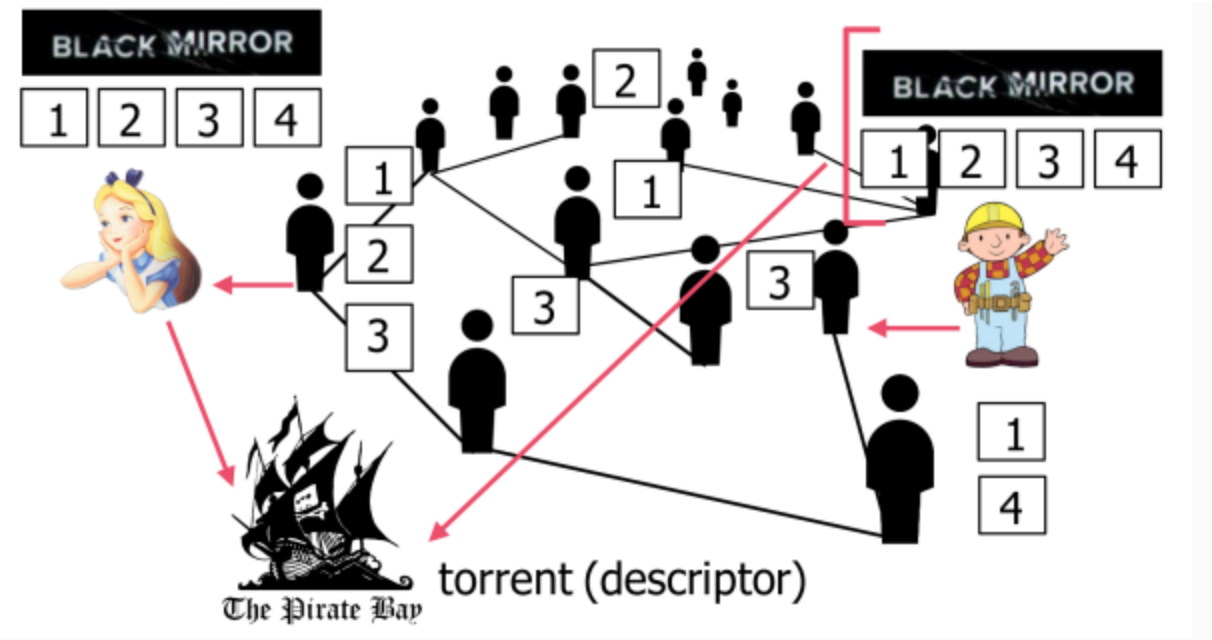
- MAC on encrypted block ciphers that are converted from messages broken into blocks



4.2 Hash Functions

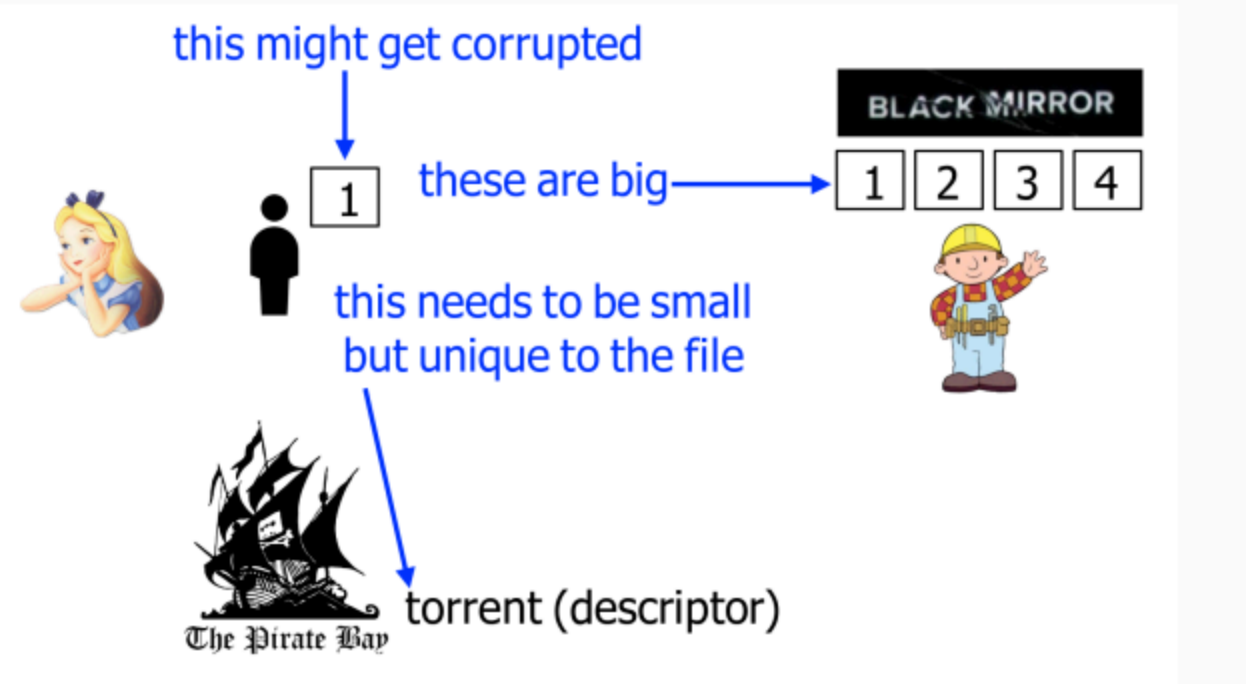
Focusing on knowing what data you're getting by hash functions

BitTorrent



You get a torrent (descriptor) which is split into 4 segments and sent to people (peer-to-peer)

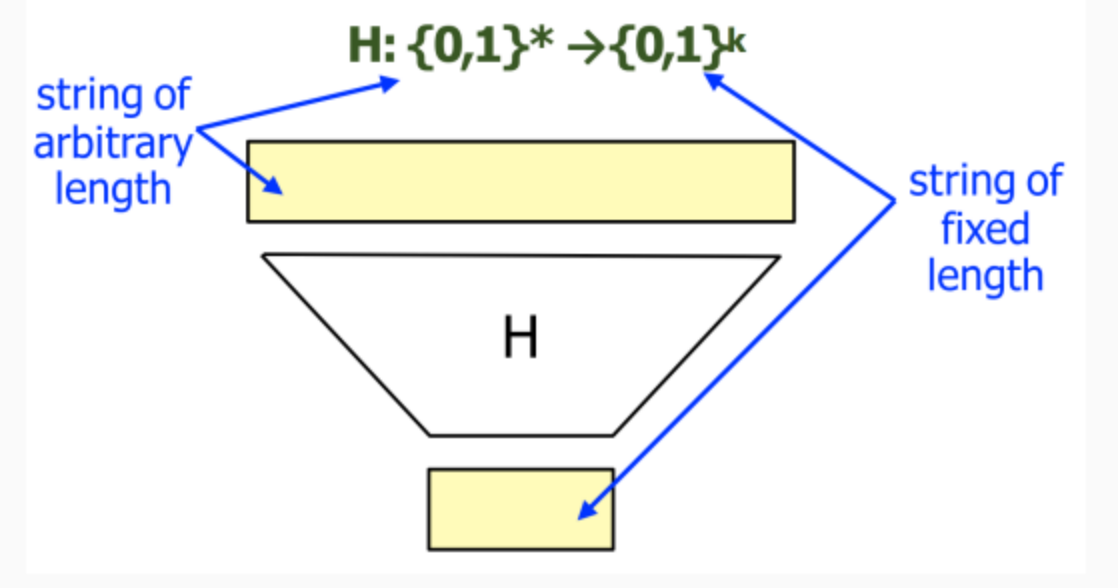
issue:



Find a way to find a unique representation of the file

Hash Function

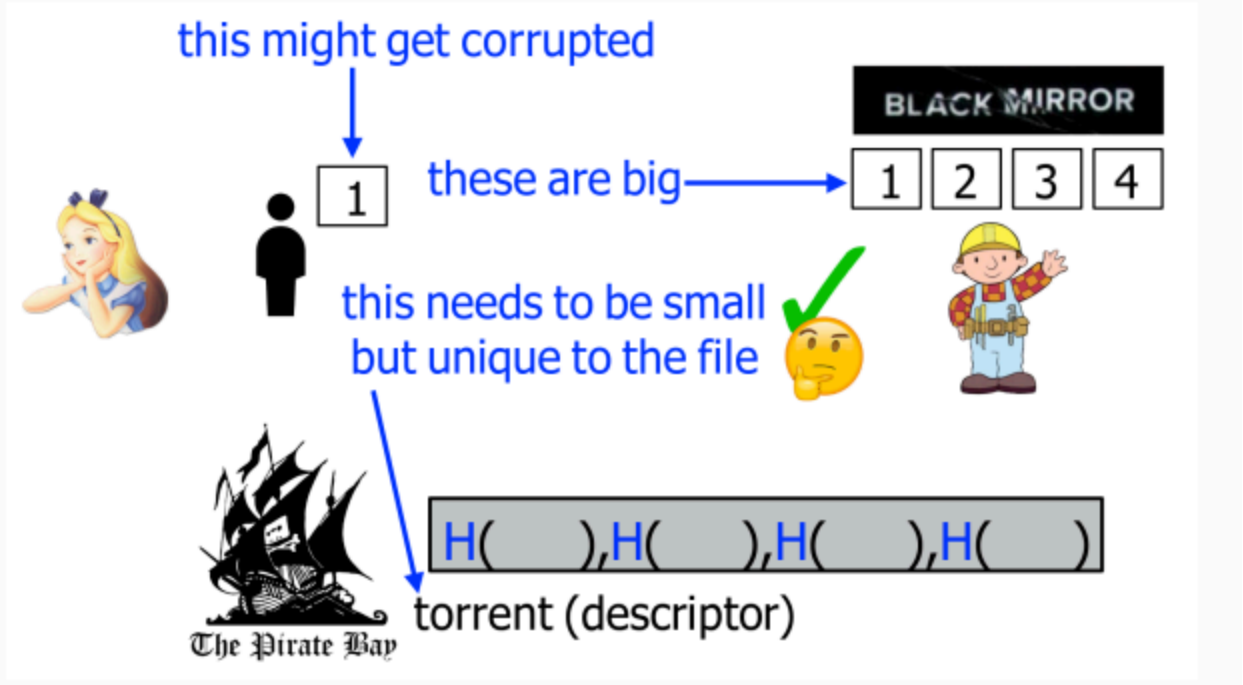
- Similar to check sum



  Takes an arbitrary length (* represents any number of bits) input and gives a fixed-length output

Going from gigabytes to bits (256 bit hashes)

Back to the BitTorrent example:



There can be inputs that correspond to the same outputs (do not do that as they must be unique)

Properties of Hash Function
Uniformity: even small changes in input can yhield big changes in output
Uniqueness: given $h = H(m)$, should be a very low chance of collision $(m_2 \text{ s.t. } H(m_2) = h)$, where given a hash functions, there is must not be the same hash output for two different messages

Example of cryptographic hashes

SHA256 hashes of:

tristan
769aee4006843e728012d6090511f730bb05957a81baf9a182d1b0d411d99b9b

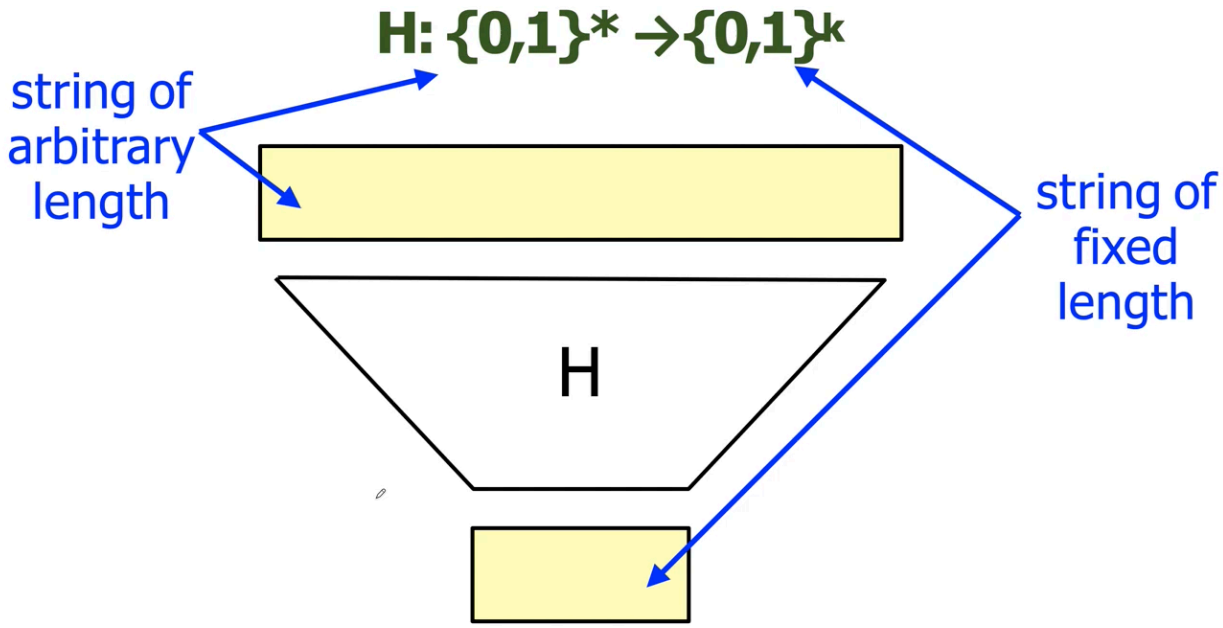
tristan1
af087d0d27e90cca55b92eb08d76161f45d650f92ec0a774d4f4edd4652e8f52

Tristan
6f8034cbc6beb650c8103c431cf5453eeb3ddd7d4df136ec27794ed100b934ba

Tristan1
23f531399e56929a66293b2df704eb557f8eb69d80598ef7cf1749964abc3b95

Hash values are expressed in hexadecimal (0-F)
Slightly different inputs obtain very different hash values (fixed length - 256 bits)
they must be unique

Cryptographic Hash Function

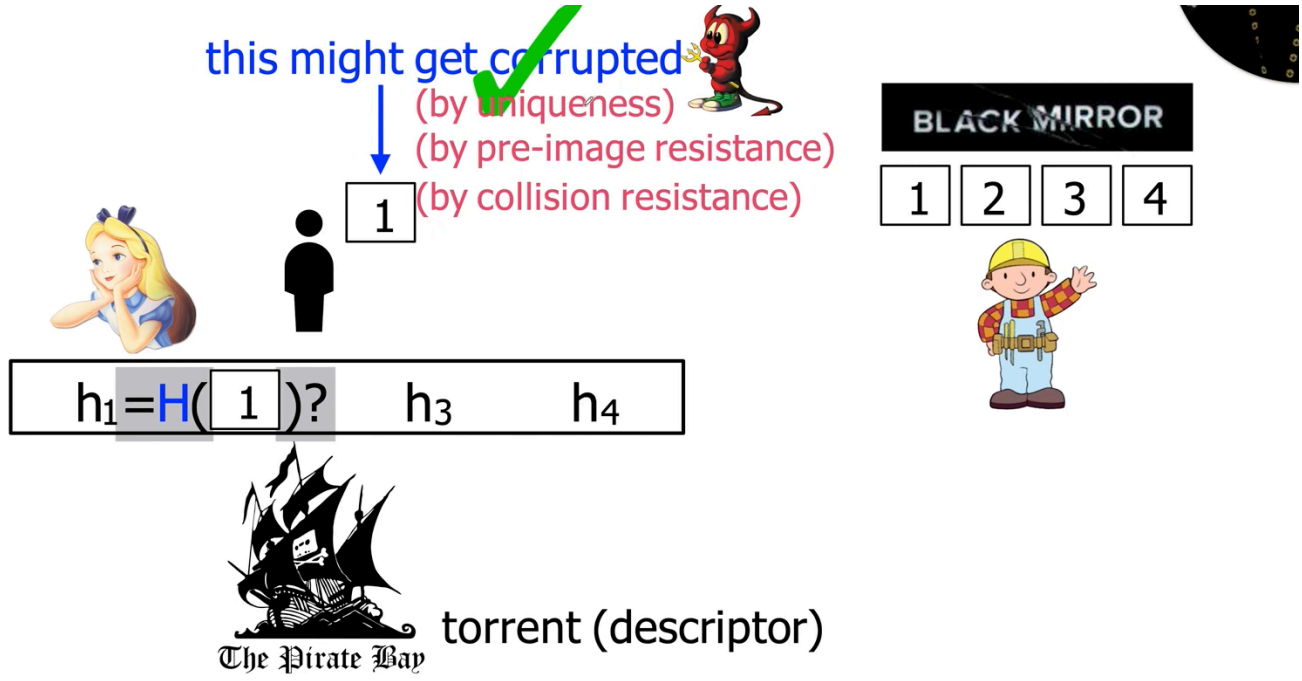


more of preventing bugs than preventing security attacks

Main properties of Cryptographic Hash Function

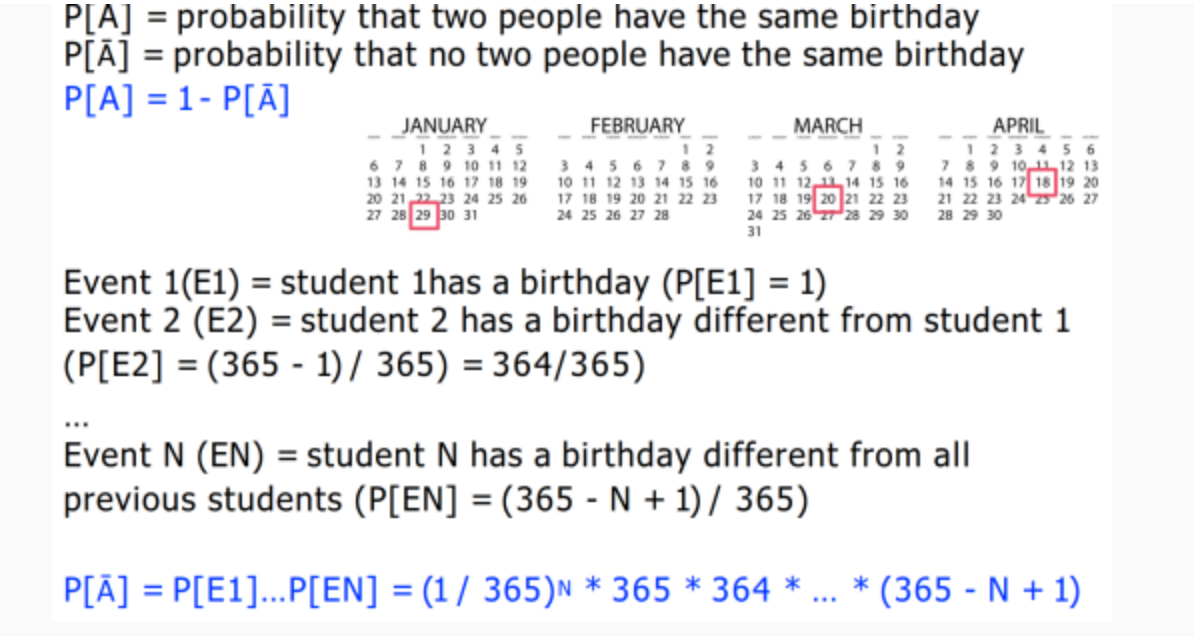
- **pre-image resistance**: given h, hard to find m such that $H(m) = h$
- **collision resistance**: hard to find x and y such that $x \neq y$ but $H(x) = H(y)$

Example with BitTorrent



Collision Attack

- how quickly can we find a collision $x_1 \neq x_2$ such that $H(x_1) = H(x_2)$
- Using the birthday paradox:
  - cosnidering a class of N students with random birthday (meaning birthdays follow a uniform distribution over the days of the year)
  - how large does N need to be before there is more than 50% chance of having 2 students with the same birthday
  - answer: $\sqrt{365} \approx 23$



going back to the question:

- pick different $x_1, \ldots, x_{\sqrt{N}}$ (where $N = 2^n$ for $H : \{0,1\}^* \to \{0,1\}^n$)
- **compute**: $y_1 = H(x_1), \ldots, y_{\sqrt{N}} = H(x_{\sqrt{N}})$ and look for a collision
- this has almost **40% chance** of finding a collision
- **memory cost**: $3n(2^{\frac{n}{2}})$
- **computational** cost: $2^{\frac{n}{2}}$ hash evaluations

example:

	n	birthday	shortcut
MD4	128	64	2
MD5	160	80	21
RIPEMD	128	64	18
RIPEMD160	160	80	
SHA-0	160	80	34
SHA-1	160	80	(51)
SHA-256	256	128	
SHA-3	256	128	

Applications of Hash Functions:

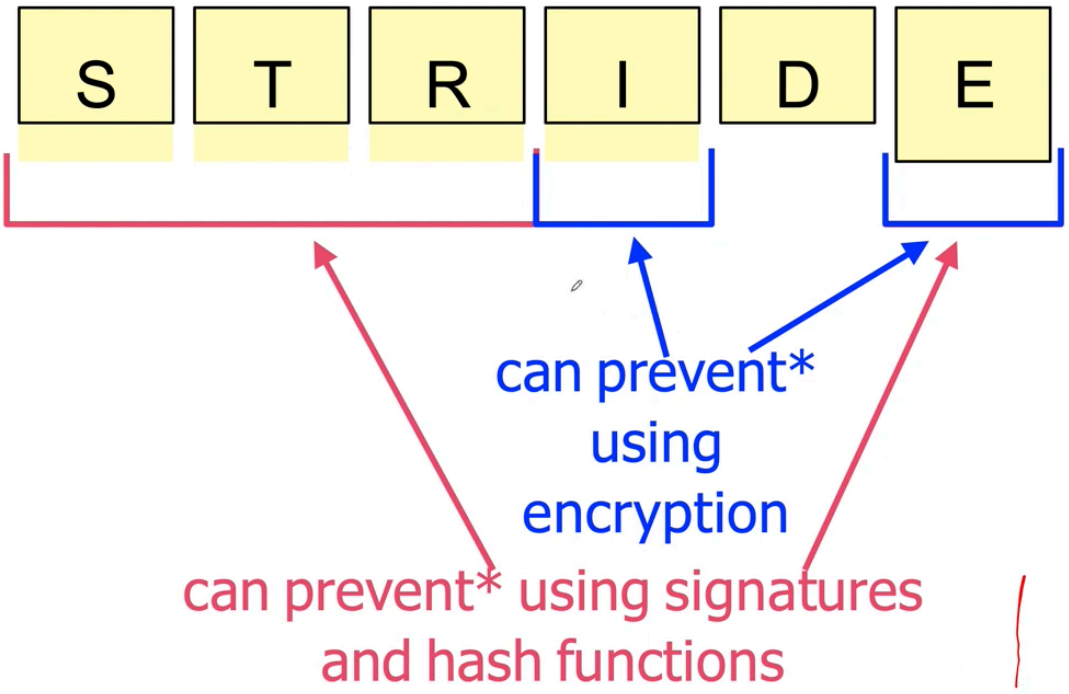
- file checksum
- MACs
- digital signatures
- commiittments
- blockchains
- virus scanning
- password storage
- and many more!

Cryptographic Primitives summarised so far

	setup?	confidentiality/ integrity?	fast?
SE	yes	confidentiality	yes
PKE	no*	confidentiality	no
digital signature	no*	integrity	no
MAC	yes	integrity	yes
OWF	no	confidentiality*	no
hash function	no	integrity	yes
AE	yes	both	yes

Recap of STRIDE:

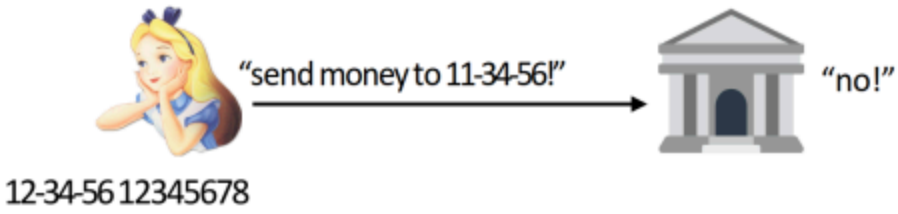
- Spoofing
  - integrity
- Tampering
  - integrity
- Repudiation
  - integrity
- Information Disclosure
  - confidentiality
- Denial of Service
  - availability
- Elevation of Privilege
  - both confidentiality / integrity



4.3 Uses of Hash Functions

Checksum

Used to detect errors introduced by humans (replaced digits, transposition, phonetic, etc.)



Also useful for errors due to corruption (inevitable for big files)

HMAC

Hash-based MAC

uses hash function to obtain a MAC

$$HMAC(K, m) = H((K \oplus opad)||H((K \oplus ipad)||m))$$

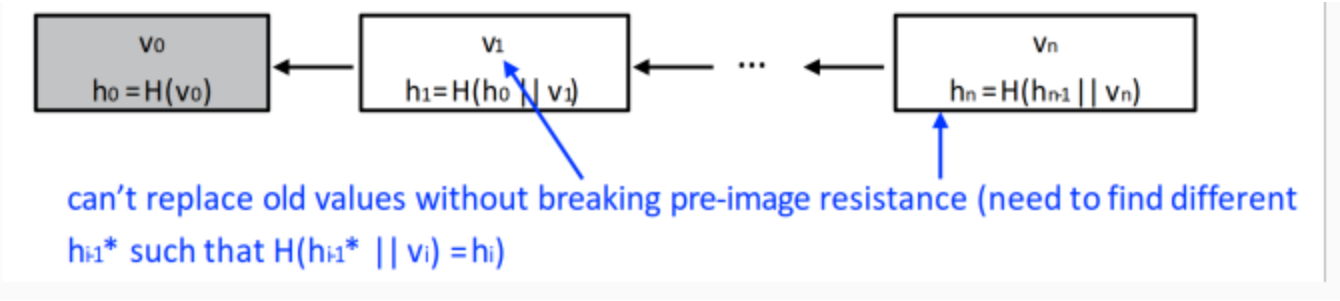
- First compute $h_{inner} = H((K \oplus ipad)||m)$
- Then compute $H((K \oplus opad)||h_{inner})$

where $opad$ and $ipad$ are fixed strings

if we do something simpler - this results in a **length extension attack** (attack on MAC)

Blockchains

Hashes can commit to entire series of values to form **tamper-evident data structure**

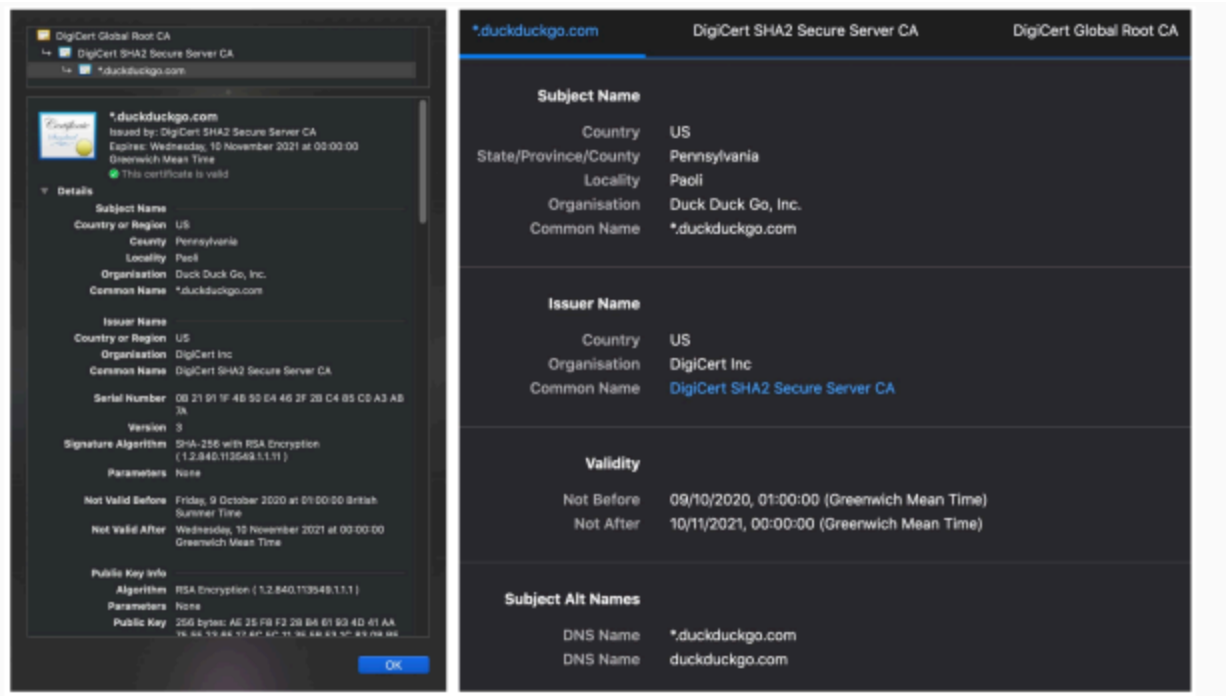


In blockchain, value represents blocks (collections of transactions + "proof-of-work"), so if we wanted to erase an old transaction we would have to rewrite the entire history and redo all the "work" since then

4.4 Digital Certificates

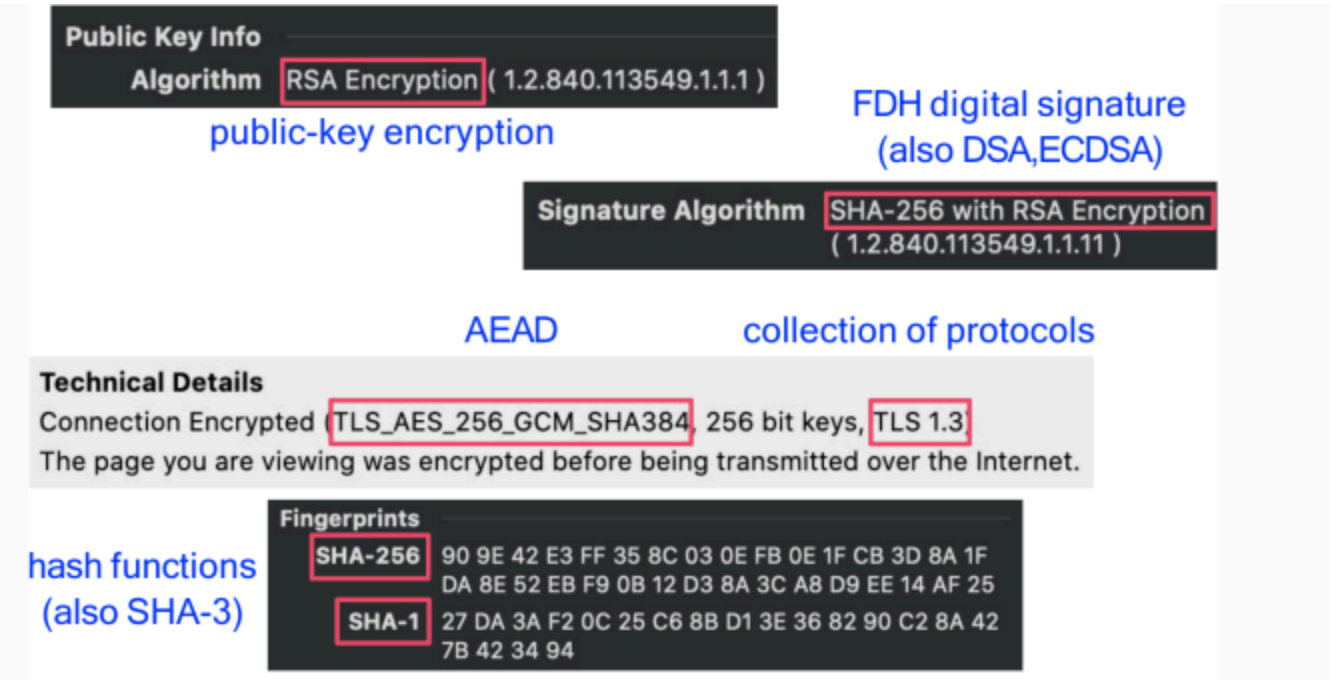
You can receive the digital certificate (public key of the website) of the website when clicking on the padlock on your browser when browsing websites

example:



digital certificates need to ensure secret communication, the right public key and right certificate

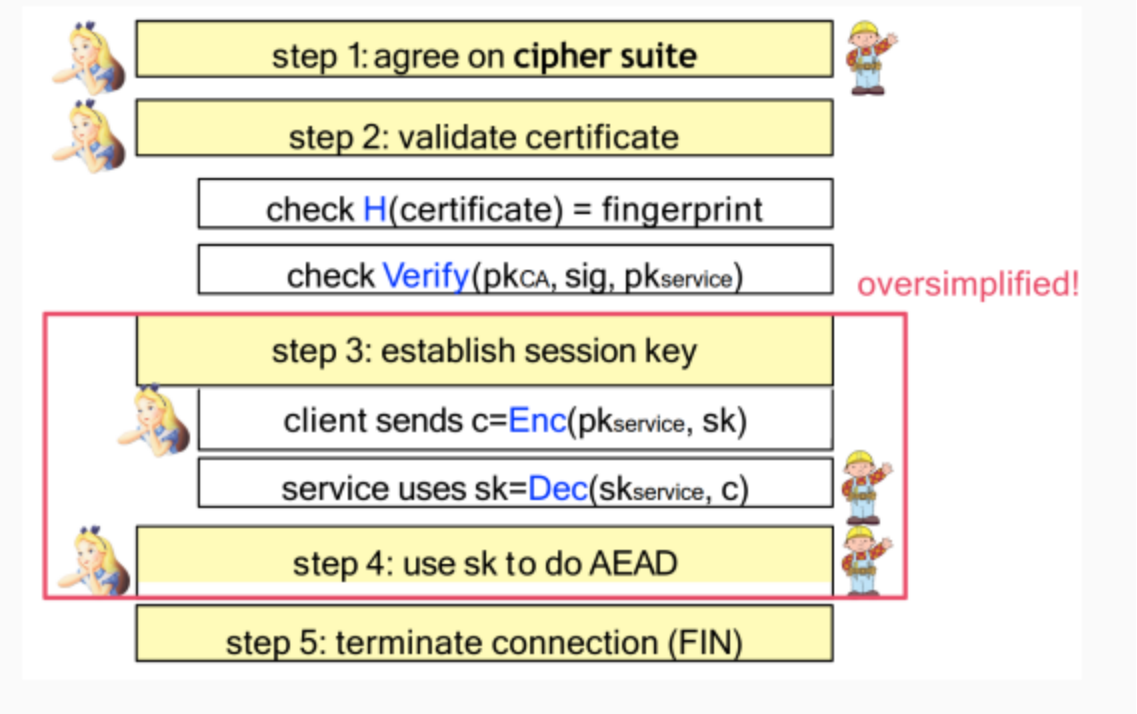
exmaple standards:



Tells you what encryption algorithms can be used e.g. RSA,

SSL/TLS Handshake

An SSL/TLS handshake happens when you visit a website



Browser generates a session key and uses the website's public key to encrypt that session key
Send that key to the website which is then decrypted, hence obtaining a shared session key

TLS/SSL

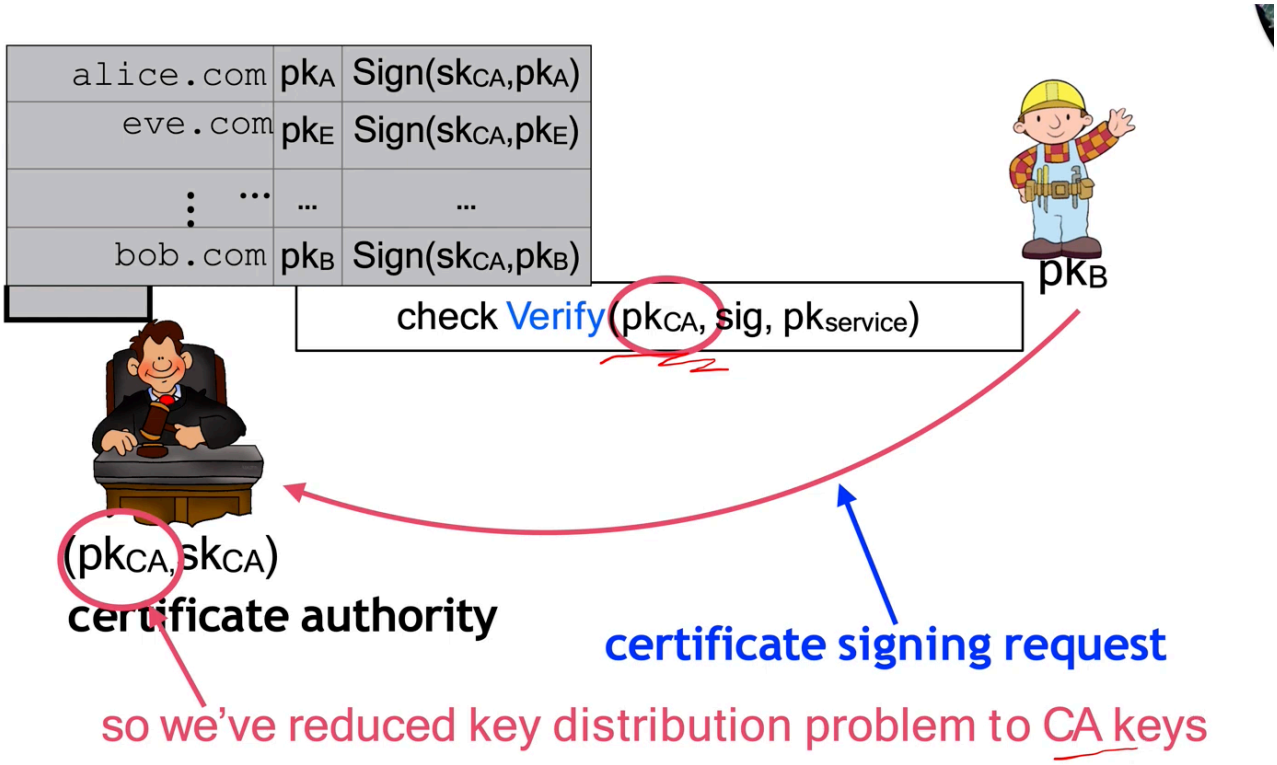
Transport Layer SEcurity - standard for secure communication on Internet today (SSL was the predecessor)

HTTPS -> you're running HTTP over TLS

Note that the digital certificates need to be signed by a Certificate Authority (CA) as a form of validation (prevents phishing)

Browsers have preloaded public keys of CA (there is a limited amount of CAs that exist) that they use to verify signatures

Certificate Authority

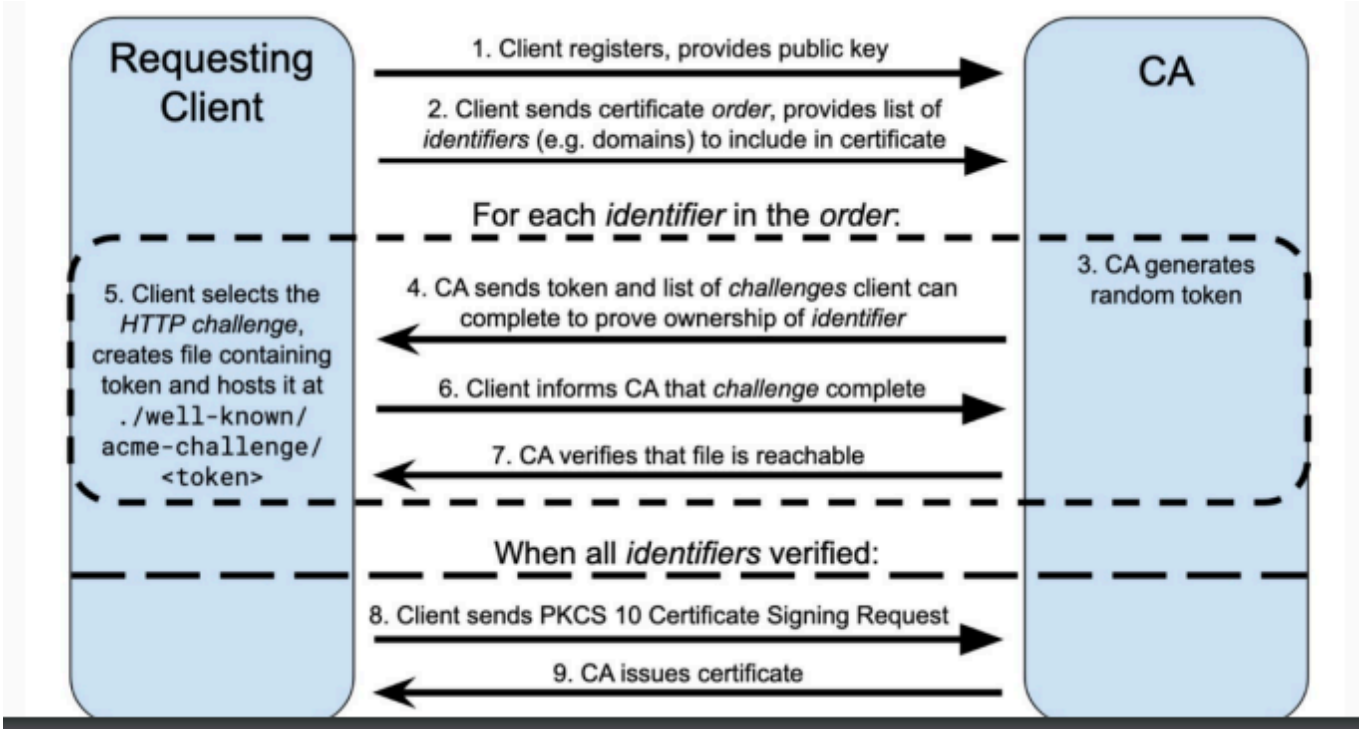


Let's Encrypt

fully automated/free CA (as most CAs have to pay)

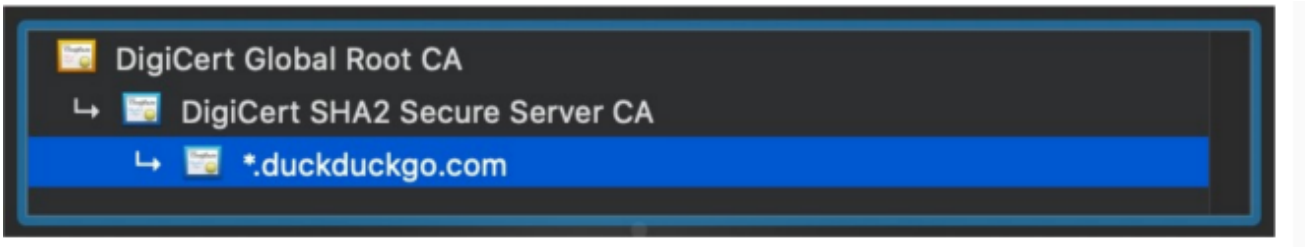
- performs only domain validation
- automated validation via ACME protocol

how does let's encrypt work:



X.509 Certificates

- defines structure of certificates and the concept of a certifciate chain
- root certifcate in the cahain is treated as as **trust anchor**



example of root certificate:

Name	Kind	Expires	Keychain
AAA Certificate Services	certificate	31 Dec 2028 at 23:59:59	System Roots
AC RAIZ FNMT-RCM	certificate	1 Jan 2030 at 00:00:00	System Roots
Actalis Authentication Root CA	certificate	22 Sep 2030 at 12:22:02	System Roots
Admin-Root-CA	certificate	10 Nov 2021 at 07:51:07	System Roots
AffirmTrust Commercial	certificate	31 Dec 2030 at 14:06:06	System Roots
AffirmTrust Networking	certificate	31 Dec 2030 at 14:08:24	System Roots
AffirmTrust Premium	certificate	31 Dec 2040 at 14:10:36	System Roots
AffirmTrust Premium ECC	certificate	31 Dec 2040 at 14:20:24	System Roots
Amazon Root CA 1	certificate	17 Jan 2038 at 00:00:00	System Roots
Amazon Root CA 2	certificate	26 May 2040 at 01:00:00	System Roots
Amazon Root CA 3	certificate	26 May 2040 at 01:00:00	System Roots
Amazon Root CA 4	certificate	26 May 2040 at 01:00:00	System Roots
ANF Global Root CA	certificate	5 Jun 2033 at 18:45:38	System Roots
Apple Root CA	certificate	9 Feb 2035 at 21:40:36	System Roots
Apple Root CA - G2	certificate	30 Apr 2039 at 19:10:09	System Roots
Apple Root CA - G3	certificate	30 Apr 2039 at 19:19:06	System Roots

Public Key Cryptography

Secrecy without shared acecss

- anyone can encrypt to Bob (or many websites) important in huge open environment like the internet
  Integrity without key exchange
- use digital signatures
- small number of distributed keys
  small key distribution
- restricted to CAs
  distadvantages:
- complicated set-up
- slow
- uses strong assumptions

Week 5: Availability and Malware

Availability: the ability to use the system as anticipated

Thinking about how:

- attacks on avaialbility work
- botnets make money
- viruses spread and get prevented

Threats to avaiilability:

- hardware failures (nah)
- malware
- denial of service (DoS)

5.1 Introduction and Denial of Service (DoS)

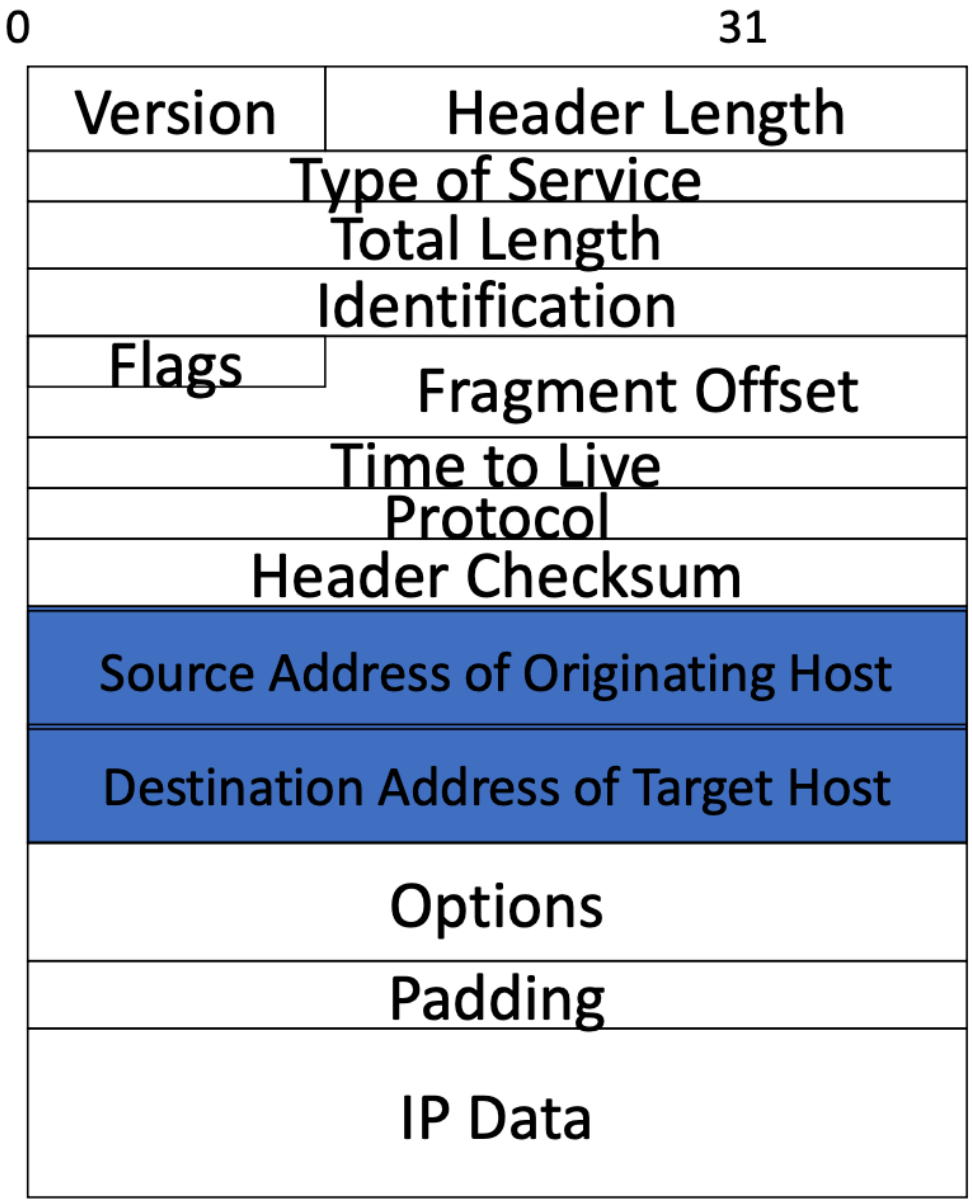
DoS

- Denial of Service
- Informal goal: take out a large site with little computing work
- how? amplication
  - use small number of packets => obtain big effect
- two types of amplication attack:
  - **DoS bug**
    - design flow on machine allowing one machine to disrupt a service
  - **Dos flood:**
    - command botnet to generate a lot of requests (to flood a server)
- DoS can happen in any layer
- unfortunately, Internet is not designed to handle DoS or DDoS (distributed DoS)

IP Header Format

- Connectionless (established only in high layers - TCP/UDP)
  - unreliable

- best effort

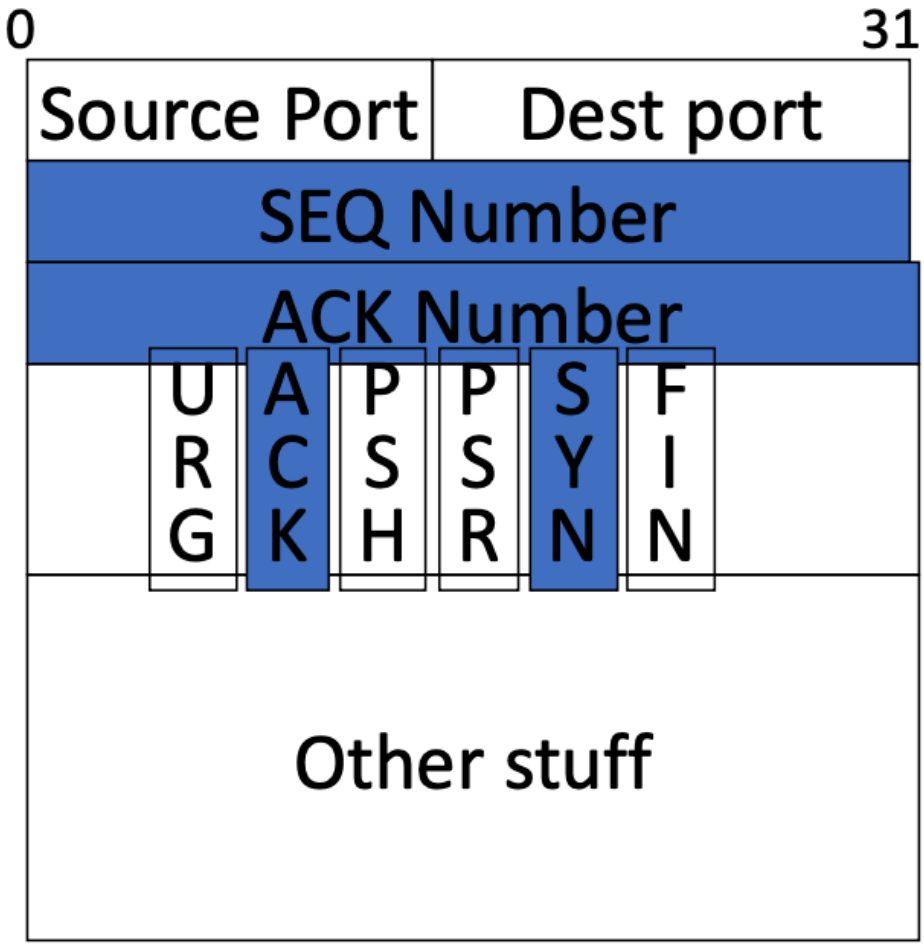


TCP Header format

TCP:

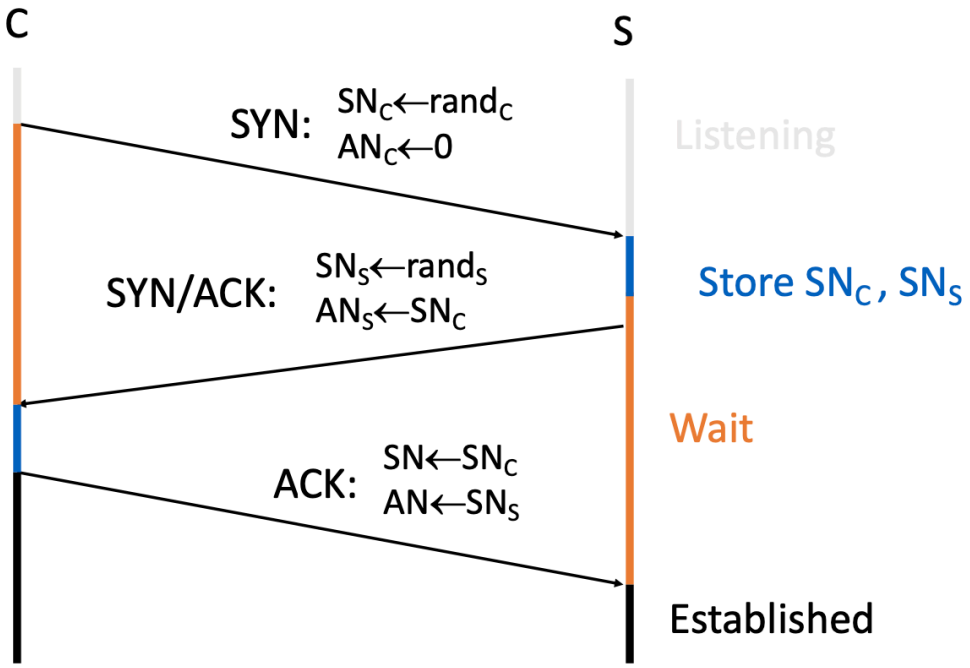
- session based (to establish a TCP connection to browser from client to send/receive data)
- congestion control (protocol ensures packets are not lost when there is congestion)

- in order delivery (ensure packets are put in right order)



TCP Handshake

- Used to establish a connection between client-server



2 Messages are sent from client to server, one is sent from server to client:

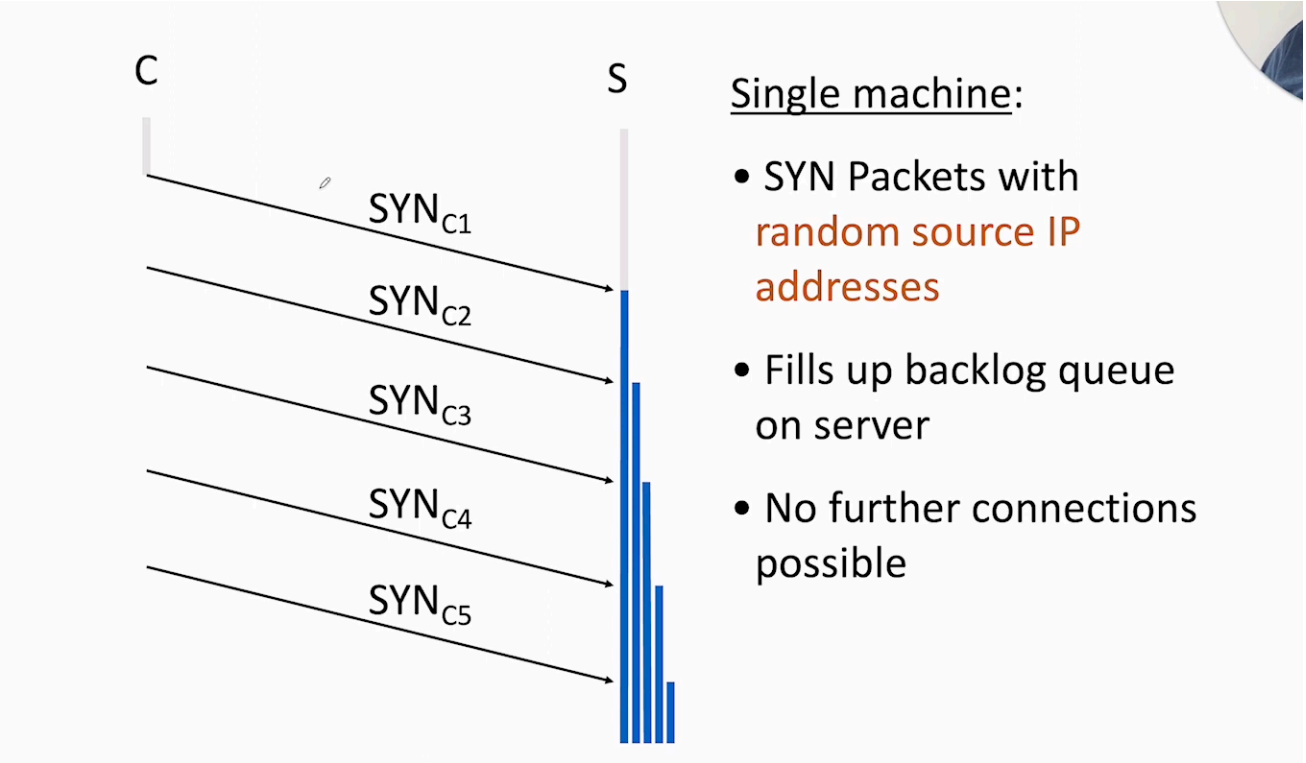
- SYN Packet (client $\rightarrow$ server)
  - $SN_c = rand_c$
  - sequence number of client that is picked by random
  - $AN_c = 0$
  - set to 0

  Server has to store SN_c, SN_s where:
- SN_c from client
- SN_s which is generated at random by server
- SYN/ACK (server $\rightarrow$ client)
  - $SN_s = rand_s$
    - sequence number of the server - generated as random number
  - $AN_s = SN_c$
    - server sends back the client's sequence number
- ACK message
  - $SN = SN_c$
    - identifies which client
  - $AN = SN_S$
    - acknowledgement number

Once the client sends back the SN_c and SN_s, a TCP connection is established

How to DoS bug a TCP handshake?

- TCP SYN Flood I: low rate
- Server waits for the the handshake to make - hence if more SYN packets are sent, there'll be a queue



Low rate SYN flood defences

- non-solution
  - increase backlog queue size / decrease timeout (doesn't solve/defend that problem)
- correct solution (when under atack)
  - syncookies: remove state from server
  - small performance overhead

Syncookies

- use secret key and data in packet to gen. server SN
- server responds to client with SYN-ACK cookie:
  - T = 5-bit counter incremented every 64 secs
  - L = MAC_{key}(SAddr, SPort, DAddr, DPort, SN_C, T) [24 bits]
    - SAddr - source address
    - SPort - source port
    - DAddr - destination address
    - DPort - destination port
    - key: picked at random during boot
  - $SN_S = (T.mss.L)$ ($|L|$ = 24 bits)

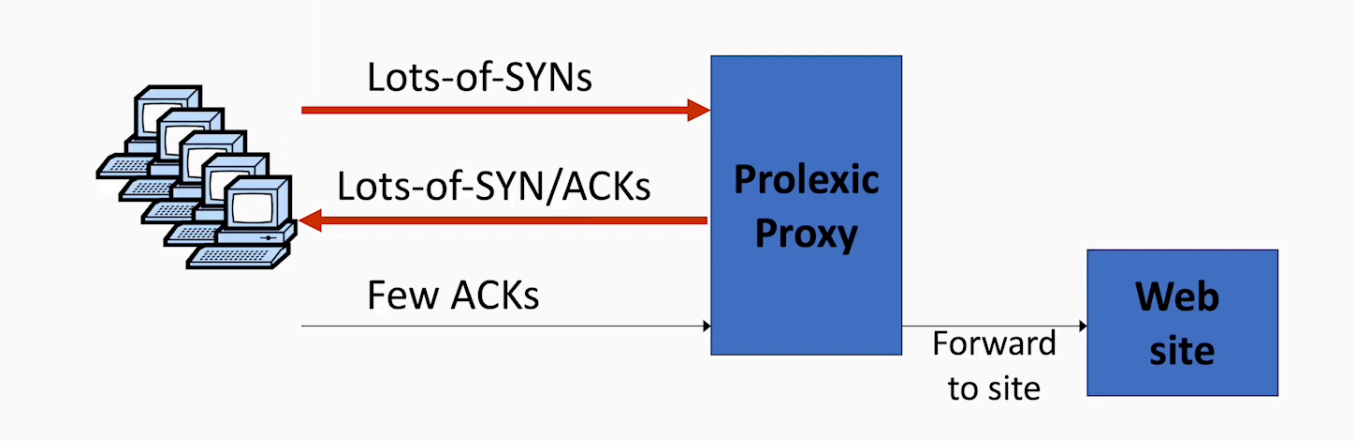
    - server does not save state
- honest client responds with ACK ($AN = SN_S, SN = SN_C + 1$)
    - server allocates space for socket only if valid SN_S

SYN Floods II

- bot army to flood specific target (DDoS)
    - 20k bots can generate 2GB/sec of SYns
    - at website:
        - saturates network uplink or network router
        - random source IP => attack SYNs look the same as real SYNs

Prolexic/Verisign Design

Idea: you only forward established TCP connections to site



Stronger attack example: TCP Con flood

- command bot army to:
    - complete TCP connection to web site
    - send short HTTP HEAD request
    - repeat
- will bypass SYN flood protection proxy
- but:
    - attacker can no longer use random source IPs
        - reveal location of bot zombies
    - proxy can now block or rate-limit bots

5.2 DoS Examples

Examples:

- DNS DDoS (take out DNS servers)

- DoS via route hijacking (e.g. YouTube route hijacking resolved in two hours but shows a huge vulnerability in DoS with no solution)
- Mirai (2016) - blocked assess within US to twitter, reddit, netflix, airbnb, etc
- GItHub (2018) - achieve amplification without broadcast via a memcached attack - generated a lot of traffic
- Dyn DDoS (2016) - using a botnet targeting provider Dyn, also taking down a whole chunk of Internet (IoT - botnet included millions of poorly secured embedded devices)

Summary of DoS attacks in various layers:

Attack	Affected Area	Example	Description
Network Level Device	Routers, IP Switches, Firewalls	Ascend Kill II, "Christmas Tree Packets"	Attack attempts to exhaust hardware resources using multiple duplicate packets or a software bug.
OS Level	Equipment Vendor OS, End-User Equipment.	Ping of Death, ICMP Echo Attacks, Teardrop	Attack takes advantage of the way operating systems implement protocols.
Application Level Attacks	Finger Bomb	Finger Bomb, Windows NT RealServer G2 6.0	Attack a service or machine by using an application attack to exhaust resources.
Data Flood (Amplification, Oscillation, Simple Flooding)	Host computer or network	Smurf Attack (amplifier attack) UDP Echo (oscillation attack)	Attack in which massive quantities of data are sent to a target with the intention of using up bandwidth/processing resources.
Protocol Feature Attacks	Servers, Client PC, DNS Servers	SYN (connection depletion)	Attack in which "bugs" in protocol are utilized to take down network resources. Methods of attack include: IP address spoofing, and corrupting DNS server cache.

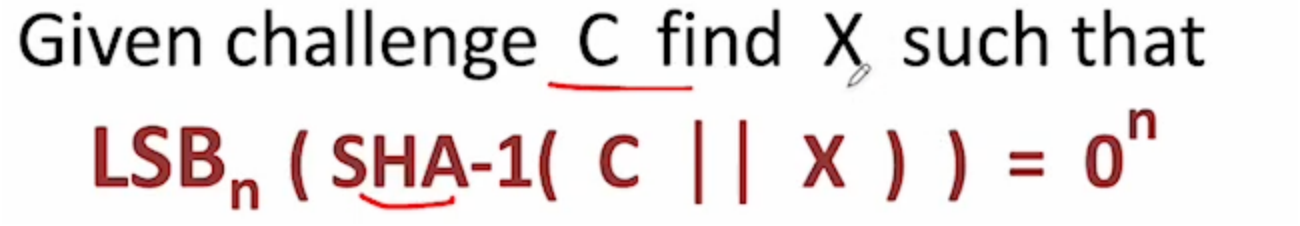
5.3 DoS Mitigation

Arms race between attack/defence of DoS attacks

Ideas of how to prevent DoS:

1. Client Puzzles

Idea - slow down attackers
Moderately hard problem (expensive work for adversary - slow rate of requests)



- trying 2^n comibintions (expected) time to solve
- checking puzzle solution is easy

During DoS attack:

- everyone must submit puzzle solution with requests
- introduce the puzzle when suspicion of a DOS attack happens (otherwise no puzzle needed)

Benefits:

- hardness of challenge : n
  - decided based on DoS attack volume
  Limitations:
- requires changes to both clients and servers
- hurts low power legitimate clients during attack:
  - clients on cell phones and tablets cannot connect
  CPU power ratio:
- high end server/low end cell phone = 8000
- impossible to scale too hard puzzles
- main memory access time ratio:
  - high end server/low end cell phone = 2

Better puzzles:

- solution requires many main memory accesses

2. CAPTCHAs

Idea: verify connection is from a human - user type a word (easy for user) but hard for a system to recognise the word
applies to application layer DDoS

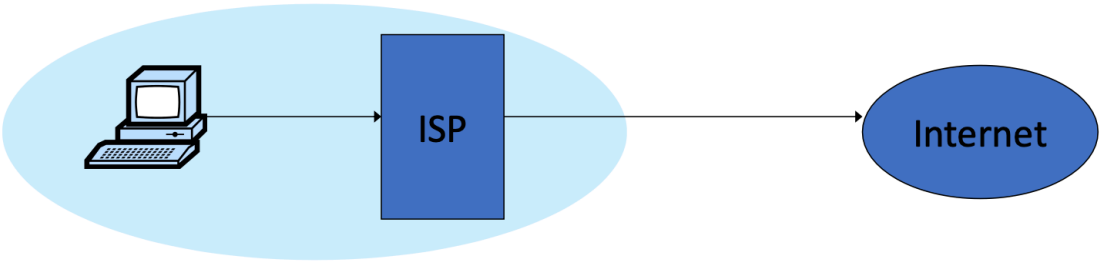
- during attack: generate CAPTCHAs and process request only if valid solution
- present one CAPTCHA per source IP address

3. Source identification

Goal: identify packet source
ultimate goal: block at the source

3a. Ingress filtering

- how to find packet origin during a DDoS with spoofed source IPs



- Ingress filtering policy: ISP only forwards packets wioth legitimate source IP

Imlementation problems:

- all ISPs must do this - requires global trust
- if 10% of ISP do not implement => there is no defence

3b. Traceback

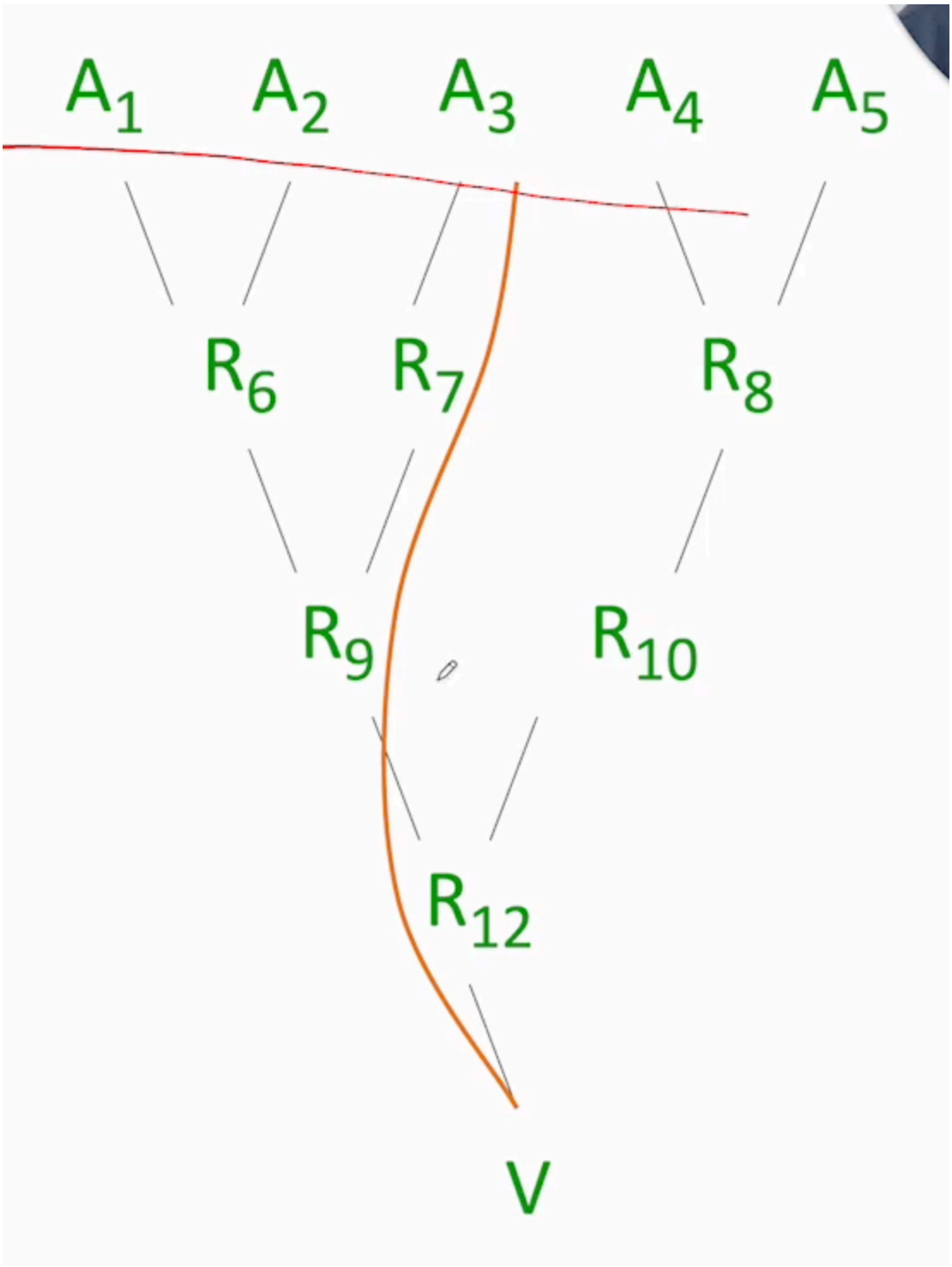
- goal: given set of attack packets, determine path to source
- how: change routers to record info in packets
- assumptions:
  - most routers remain uncompromised
  - Attacker sends many packets
  - Route from attacker to victim remains relatively stable

simple method:

- write path into network packet
  - each router adds its own IP address to packet
  - victim reads path from packet
- problem:
  - requires space in packet
  - path can be quite long
  - no extra fields in current IP format
  - changes to packet format too much to expect

  better idea:
- as DDoS involves many packets on same path

- store one link in each packet
  - each router probabilistically stores own address
  - fixed space regardless of path length

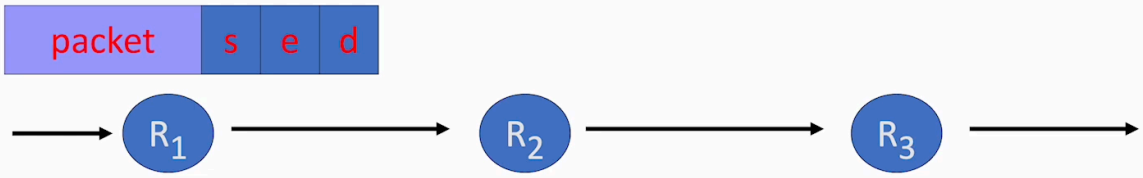


  edge sampling
- data fields written to packet
  - edge: start/end IP addresses
  - distance: number of hops since edge stored

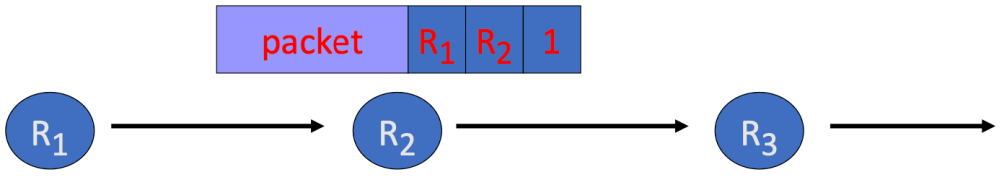
- marking procedure for router R:

```
if coin turns up heads (with probability p) then
    write R into start address
    write 0 into distance field
else
    if distance == 0 write R into end field
    increment distance field
```

- Packet received
  - R_1 receives packet from source or another router
  - Packet contains space for start, end, distance



- Finish writing edge
  - –R2 chooses not to overwrite edge
  - –Distance is 0
    - Write end of edge, increment distance to 1



- Increment distance
  - R3 chooses not to overwrite edge
  - Distance >0
    - Increment distance to 2

Path reconstruction

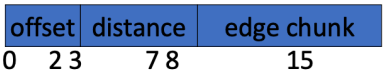
- Extract information from attack packets
- Build graph rooted at victim
  - Each (start,end,distance) tuple provides an edge
- # packets needed to reconstruct path

$$E(X) < \frac{\ln(d)}{p(1-p)^{d-1}}$$

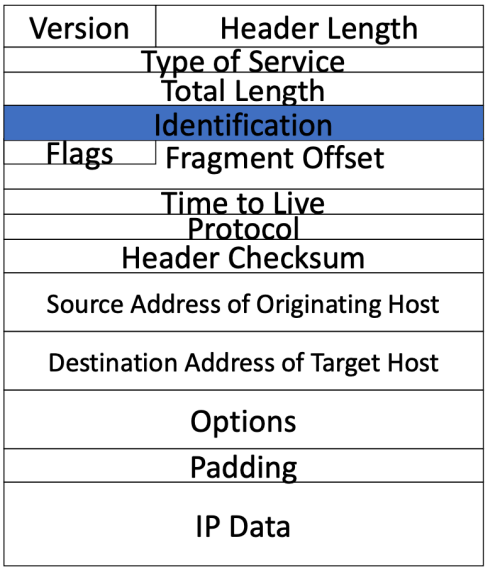
where p is marking probability, d is length of path

But where to store edge?

- Identification field
  - Used for fragmentation
  - Fragmentation is rare
  - 16 bits
- Store edge in 16 bits?



  - Break into chunks
  - Store start + end



conclusion:

- denial of service
  - attempt to defeat avaialbility
- flooding/overload
  - presenting commands moreq uickly than a server can handle them
    - targeting applications or resources
- blocked access / access failure
  - preventing a service from functioning

5.4 Botnets

Bots

A bot (aka zombie/drone) is a **compromised machine** that can be **controlled** by an attacker remotely

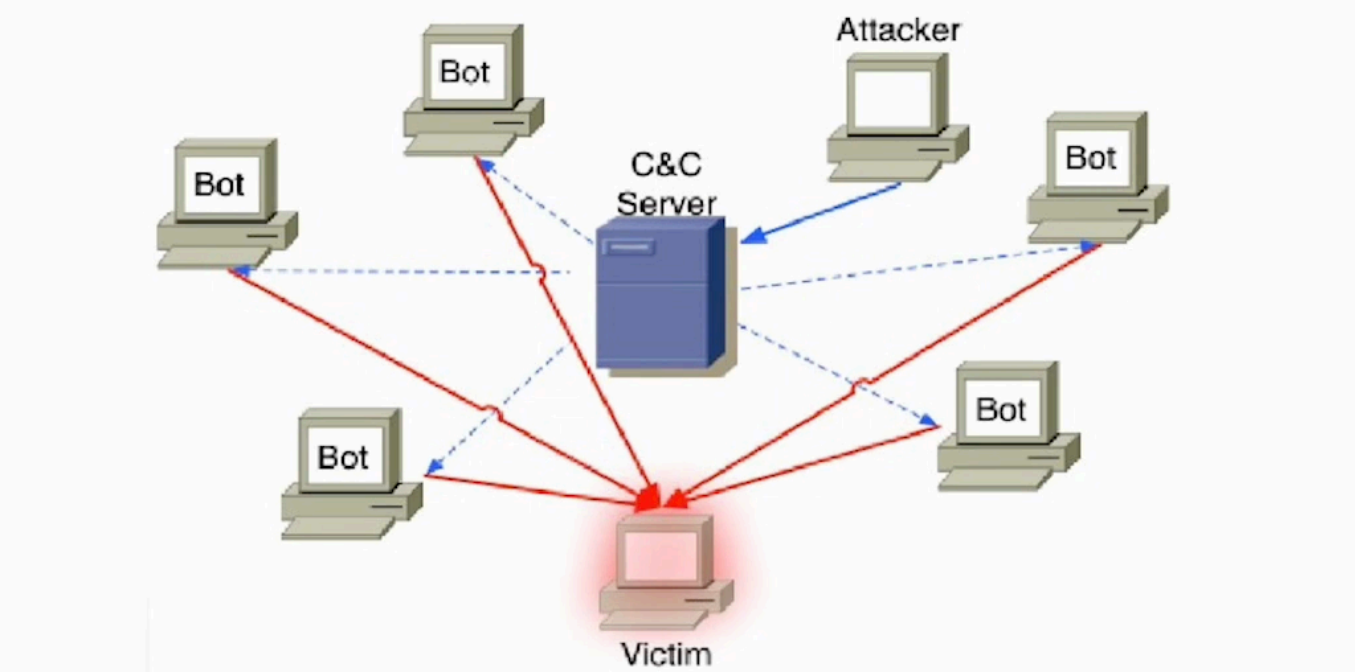
- initially used to be automated programs to provide useful services on IRC (internet Relay Chat) channels

Distinguishing features:

- easy to control remotely
- implementation of different commands
- infected machines are incorportated in large newtorks (botnets) that are controlled by botmaster
- loaded on a computer after compromise (e.g. by a trojan or a worm)

Botnets

Attacker controls a **Command and Control (C&C)** server
All bots contact C&C to report their status or receive malicious commands to perform



C&C Mechanisms

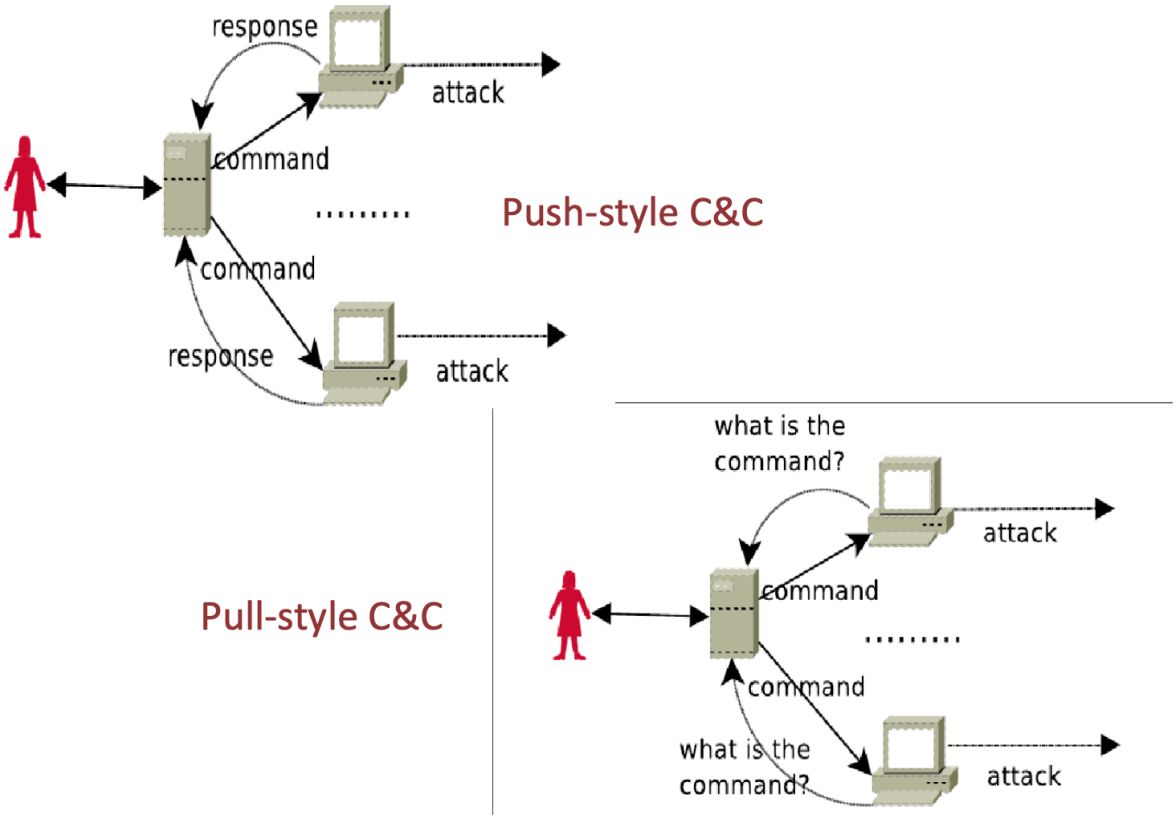
- Most distinguishing and powerful feature of botnets:
  - as long as there is an update command defined for the bots the botmaster can change the command set by updating their bots
  - C&C brings great flexibility to the activities that can be performed by bots
- however it can also be the weakest link:
  - single point of "failure"
  - so real botnets don't have a single C&C server
- types of C&C:
  - centralised
  - peer-to-peer

Centralised C&C

- two approaches:
  - push style (push commands to bots)
  - pull style (recieve updaets)
- multiple C&C servers:
  - multiple layers of hosts between bot and botmaster
- address of C&C server(s) must be available to each boty

    - in binary, config file, etc
    - frqeuently updated
- large botnets are often **partitioned* into smaller ones
    - each bot knows only a few C&C servers

push vs pull



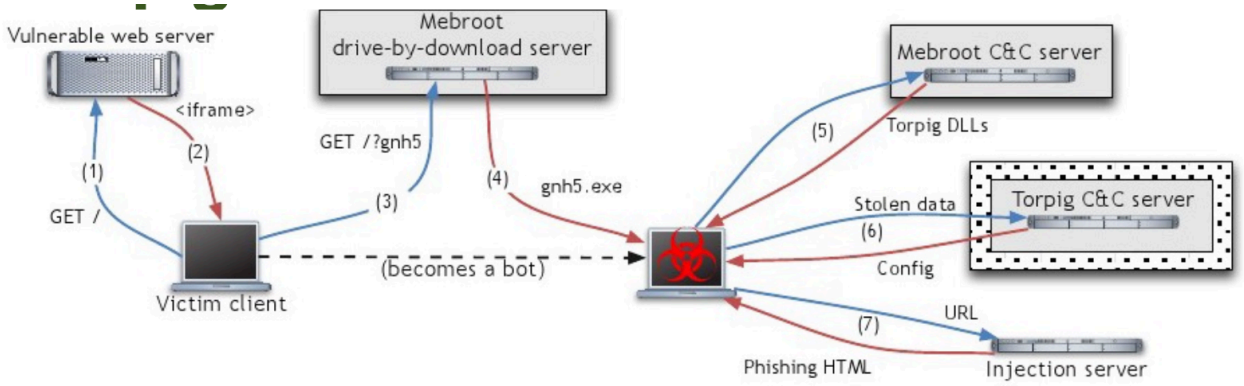
Peer-to-Peer C&C

- bot commands propagate in a P2P network
- every bot is a peer (e.g. like in BitTorrent protocol)
- more robust
    - more difficult to cartch botmaster
    - even if some nodes are down, netowkr continues its activities
- custom protocol:
    - new,unknown traffic can be "noisy"
    - a bot needs a way to find new neighbours
- standard protocol:
    - allow C&C traffic to blend in with legitimate P2P traffic
    - legitimate P2P blocked by many ISPs

Bullet-Proof Hosting

- sometimes C&C is hosted by ISPs that are completely unresponsive to abuse complaints
  - take down the entire ISP
- de-peering: other ISPs rescind their peering agreements with malicious ISPs
  - not easy as it involves law enforcement

Torpig Botnet



Mebroot: rootkit that takes cntrol of a machine by replacing the system's Master Boot Record (MBR)

- executed at boot time, undetected by most anti-virus
- has no malicious capability, but provides a generic platform to manage (install, uninstall and activate) other maliciuous modules
  - e.g. torpig botnet

Bots Defence

Do not compromise the bots

- attack C&C infrastructure
  - take IRC channel off-line
  - when dynamic DNS used for central command server, route traffic to blackhole
- Honeypots
  - actively join the botnet - to study the behaviour of the botnet before taking it down
  - vulnerable computer that serves no purpose other than to attract attackers + study their behaviour in controlled environments
  - when honeypot is compromised, bot logs into botnet
  - allows defenders to study actions of botnet owners

5.5 Malware

- Short form for **maliciuous code**
  - software that fulfills author's malicious intent

    - intentionally written to cause adverse effects
    - comes in many flavours but all share a common characteristic: perform some **unwanted activity**
- the term usually used equivalent to a virus by media
    - virus is JUST a type of malware (there are many types of malware)

Nowadatys people hack for profit (underground enconomy) -> have to monitor underground channels / understand economic structure / find and disrupt weak links, not just understanding technical side

Taxonomy

	NEED AN HOST PROGRAM	SELF-CONTAINED PROGRAM
SELF SPREADING	Computer Viruses	Worms
NON-SPREADING	Trojan Horses Rootkit	Keyloggers Spyware Dialers

71

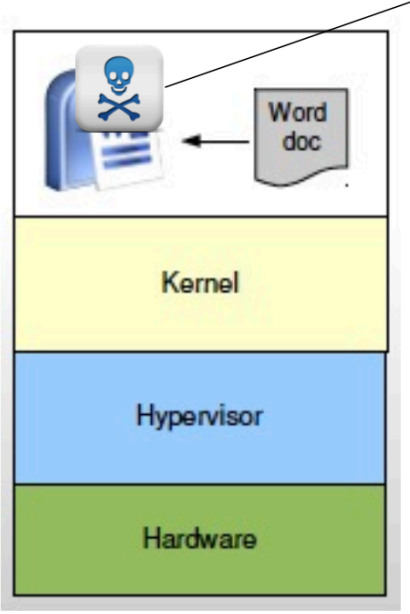
Virus

- a program that can infect (spread) other programs by modifying them to include a, possibly evolved, version of itself

Features:

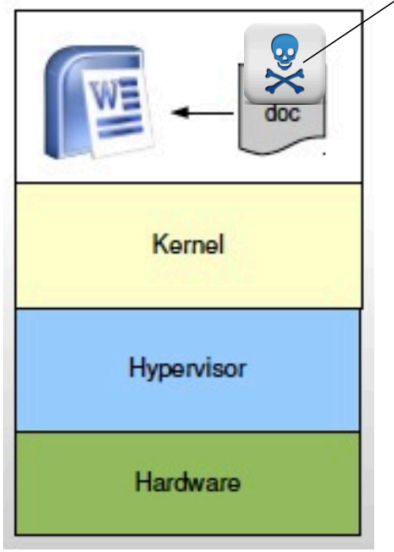
- reproduce its own code
- attach itself to other files (as virus cannot "survive" by itself)\
- get executed when infected executable file is executed

Where can a virus be found at?
Firsty level - file/macro infection



File infection

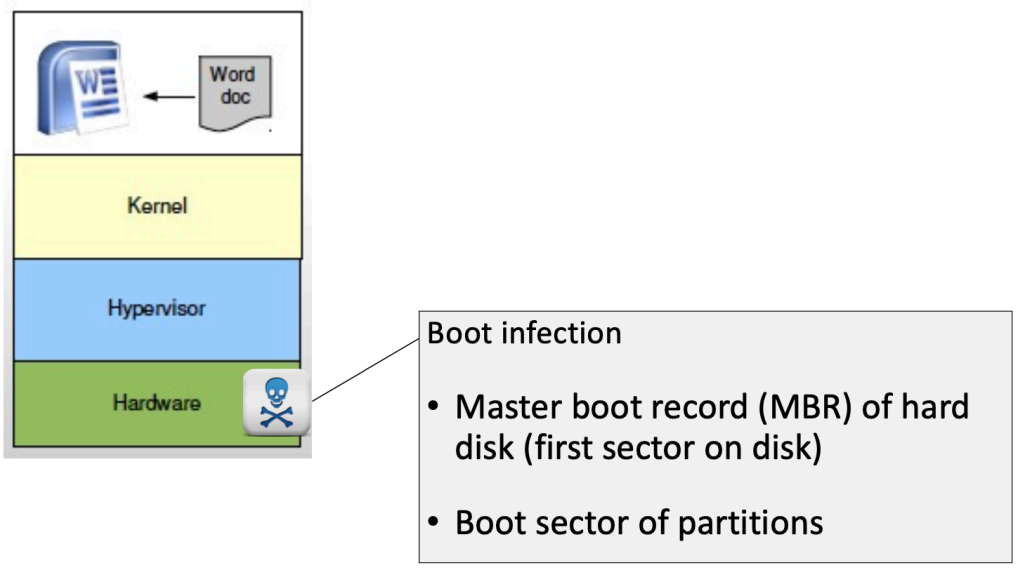
- Overwrite virus
  - Substitute the original program
- Parasitic virus
  - Append virus code before or after the program
  - Modify program entry point
- Cavity virus
  - Install inside of the file it is infecting
  - Usually in empty gaps inside the program binary



Macro infection

- Many modern applications support macro languages
  - Microsoft Word, Excel, Outlook
  - Macro languages are powerful
- Embedded macros automatically executed on load
- Infection through an exploit
- MS Word and Acrobat are common victims

itll work its way down to the hardware



Virus Defence

Via **antivirus software**

- signature based detection
  - database of byte-level or instruction-level signatures that match virus
  - wildcards/regular expressions (regex) can be used
  - hash of known malicious programs
- heuristics (check for signs of infection)
  - code execution starts in last section
  - incorrect hjeader size in PE header
  - suspicious code section names
  - patched import address table
- behavioral signatures

Alternative way: Sandboxing

- run untrusted applications in restricted environment

Worms

- a self-replicating computer program that uses a network to send copies of itself to other nodes

Features

- autonomous spread over network
- it is self-contained (doesn't need to be embedded in a file like a virus)
- speed of spreading may increase over time

worm propagation

- **email harvesting**
  - consult address books
  - files that might contain email addresses
    - inbox of email client
    - internet explorer cache and personal directories
- **network share enumeration**
  - windows discovers local computers, which can be attacked
  - some worms attack everything incl network printers
- **scanning**
  - randomly generate ip addresses and send probes
  - some worms use hit-list with known targets

worms mail propagation

- social engineering techniques to get executed
  - fake from address
  - promise interesting pictures or applications
  - hide exec extensions (.exe) behind harmless ones (.jpg)
- attempt to hide from virus scanners
  - zipped, sometimes even with password-encrypted (ask user to unpack)
- exploit IE bugs when HTML content is rendered
- speed of spread is limited, humans are in the loop
  - patterns correspond to time-of-day

exploit-based propagation

- rqeuire no human iunteraction
  - exploit well-known network services
  - spread much faster
- propagation speed limited either
  - by network latency
    - worm thread has to establish TCP connection
  - by bandwidth

        - worm can send UDP packets as fast as possible
- spread can be modelled with classic disease model
    - slow start
    - worm grows exponentially
    - final phase where only few uncompromised machines left

worms defence

- **virus scanners**
    - effective against email-based worms
- **host-level** defence
    - protect software from being exploited
    - stack protection techniques
    - aslr (address space layout randomisation)
    - attempts to achieve diversity to increase protection
    - not effective when users execute malware themselves
- **network-level** defence
    - intrusion detection systems
        - fast/auto signature generation is active research/innovation area
    - limit number of outgoing connections
        - help to contaion worms that perform scanning
    - personal firewall
        - block outgoing SMTP connections from unknown applications

Trojan Horses (TH)

- class of malware that often appears to perform a desirable function but also performs **undisclosed malicious activities**

Features:

- require user to explicitly run the program
- unable to make copies of themselves / self-replicate
- can be used to perform any kind of malicious activity

How do Trojan horses get in

- through social engineering
- user opens suspicious email attachment
- user follow link to an infected site

- user downloads/executes an unknown binary (contain trojan horse)

Spyware

- software that secretly monitors user's behaviour and collects private information without user knowledge

Features:

- threat to user privacy
- hidden from user and difficult to dertect
- leak sensitive information (e.g. credit acrd numbers, visited webpages, passwords) to someone else over internet
- often implemented as browser plugin/extension

KEyloggers

- record key strokes typed by user
  - runs as a process in background
- used to capture interesting user data
  - passwords, information for social engineering attacks (emails)
  - successful independent of any application-level protection or network-level protection (e.g. encryption)
- easy to write
  - easy to re-implement
  - difficult to recognise by antivirus sacnners

Rootkits

- Disguise existence of malicious program in a system and enable continued privilege access to it

Features

- installed by an attacker after a system has been compromised
- hide the presence of attacker/ allow him to return on a later time
- difficult to detect

User-space rootkits

- replace system programs with trojaned versions
  - system browsing (ls, find)

    - system logging (syslogd)
    - system monitoring (ps, top, netstat)
    - user authentication (login, sshd)

Kernel rootkits

- modify kernel data structures to hide processes, files and network activites
- installed by
    - kernel patch
    - loadable kernel module
    - kernel memory patching (/dev/kmem)

Rootkit defence

- user-space
    - tripwire
    - chkrootkit
- kernel-space
    - kstat (kernel security therapy anti-trolls)

Week 6: Human-centred security

Human centred approach

About thinking **socio-technically**

- recognising/considering interaction between people, workplace, technology

Think of users as your **partner**, not your enemy (when following a human-centred approach when designing secure systems). Users care about security, but lack awareness of threats (security departments can lack awareness of users)

Security policy should be compatible with working practices where they should account for people's limitations (e.g. memory)

Users should be more aware of security related threats

What does it mean by human-centered?

- **understand** how people interact with technology
- identify the **limitations** of users to avoid asking people to do what they are unable to do
- identify **opportunities** to incorporate people into security systems

- **identify** human constraints and vulternabilities (e.g. behavioural biases, memory capacity)
- **consider** where the constraints could be
- **design** a system that manages human constraints to reduce risk

Where can things go wrong?

Unintentional

- human-error/mistakes
- poor digital hygeine
- non-compliance with security system/policy

Intentional

- malicious user

Information Security Policy

- security policies are often complex and difficult for users to follow
- they lack an understanding of limitations of the people that are being asked to follow the policy
- developed by SM (seniro managers), without consultation and consideration of the users
- difficult to udnerstand and comply with policy/processes, result in non-secure workarounds

Benefits of usable systems

- **Effectiveness**
  - support users in completing actions accurate
- **Efficiency**
  - users can perform tasks quickly through the easiest process
- **Engagement**
  - users find it pleasant to use/appropriate for its industry/topic
- **Error Tolerance**
  - support a range of user actions and only shows an error in genuine erroneous siuations. You achieve this by finding out the number, type and severity of common errors users make, as well as how easily users can recover from those errors
- **Ease of Learning**
  - new users can accomplish goals easily and even more easily on future visits

How can we help users with security

- understand the process "in the wild", not in "ideal world"
- provide users with simple, low cost actions e.g. WhatsApp
- identify barriers/obstacles
- make systems/processes usable
- consider affordances within design of systems
- signifiers are indicators (e.g. a mark or sound) that communicates appropriate behaviour to the user
- feedback communicates the result of an action

Human-centered design

- People should be at the **center of the design**
- we need a good understanding of people, their needs, and context of thier work
- rapid test of ideas, observation, ideation
- many beneifts including improved usability, fewer user errors and faster learning of system
- critique: could the process improve a system for those involved in the design processl, but make it worse for those that are not?

Human limitations

Designing security systems/processes with **human limitations** in mind

Memory limitations

- people have short-term/long-term memory
- consider this limitations in authentication proccesses e.g. provide 1st, 4th, and 10th character of a complex passcode (hard!!)
- long-term memory such as procedural memory (e.g. rememberinh wot o drive) - helps develop habits (both good/bad ones)
    - design systems that encoruage learning of good habits

Cognitive Biases

Cognitive bias:

- mental shortcuts that are more practical
- can result in security issues e.g. underassessing risk

Types of bias that impact on security:

- **optimism bias** (overly optimistic about their personal risk e.g. unsecure wifi)

- **anchoring** - people tend to stick with their first choice/decisions as they did that before (will do that again)
- **consensus bias**
- **hyperbolic time discounting** - they care about security but doesn't act in a way that they care about security (place lower value on longer term risk/higher priority on shorter-term rewards)

Example: unsecure wifi

- people are too optimisitc about thier own risk (optimism bias)
- people tend to use wifi to avoid using mobile data (save money)
- short-term goal: need access to something but doesn't think about thier risk (underestimation)

Optimistic Security Execs

- SMs are **quite optimistic** as they perceive their firms to have higher degree of control for their information security than other firms have (illusion of control)
- the illusion increases their optimism

Anchoring

- people tend to stick with their first decision/choice where their future decisions are **anchored** to this initial choice/decisions

Limitations around security warnings
security warnings, notifications and "nudges" are used to increase user awarenss and improve security behaviour

- users become desensitised to security warnings
- warnings occur too frequently
- false-positives are too frequent
- lack awareness of the threat

people's attention is a resourdce - a lack of attention can result in security threats

- develop warning mechanisms that take into account the gorilla experiment
- some people focus on attention on other parts (instead of the security warnings)

Security awareness

Awareness considerations

- how much control do users perceive they have
- cultural factors influence effectiveness of awareness intervnetions/campaigns
- people may know what to do (be aware) but feel unable to act due to low usability of systems
- users should be self-motivated to act, ideally
- qwuestions to ask:
  - are users motivated to do the right thing
  - are users able to do the right thing
  - are users effectively prompted to do the right thing
  - how long will user motivation last
  - will users continue to have the ability to do the right thing
  - how effective are the prompts

should we sanction poor behaviour? - discouraged

Inclusive security

Consider **accessibility** (usability) in designing security systems/processes

- accessible/universal design allows systems/processes to be accessible to all, even those with disabilities (those with complex needs)

Existing accessibility features in devices can be problematic for security (e.g. spellcheckers are ineffective for password fields, features that use OCR technology can fail to read CAPTCHAs, etc)

Age-related factors

- can result in security related usability problems (physical dexterity/cognitive capability)
- engagement in cybersecurity protective behaviours may be dependenty on competence levels, support from others and demand

Example: older adults and 2FA

- considered a more effective mechanism but older adults are reluctant to use 2FA
- lack of technical understanding
- uncertianty around data privacy
- imcompatible hardware/software
- difficulties with small hardware components due to physical dexterity

Week 7: Authentication

Authentication is **what you know**, **what you have** and **what you are**
What you know

- text passwords
- graphical passwords
- personal details
  what you have
- RFID card
- cryptographic tokens
  what you are
- biometrics

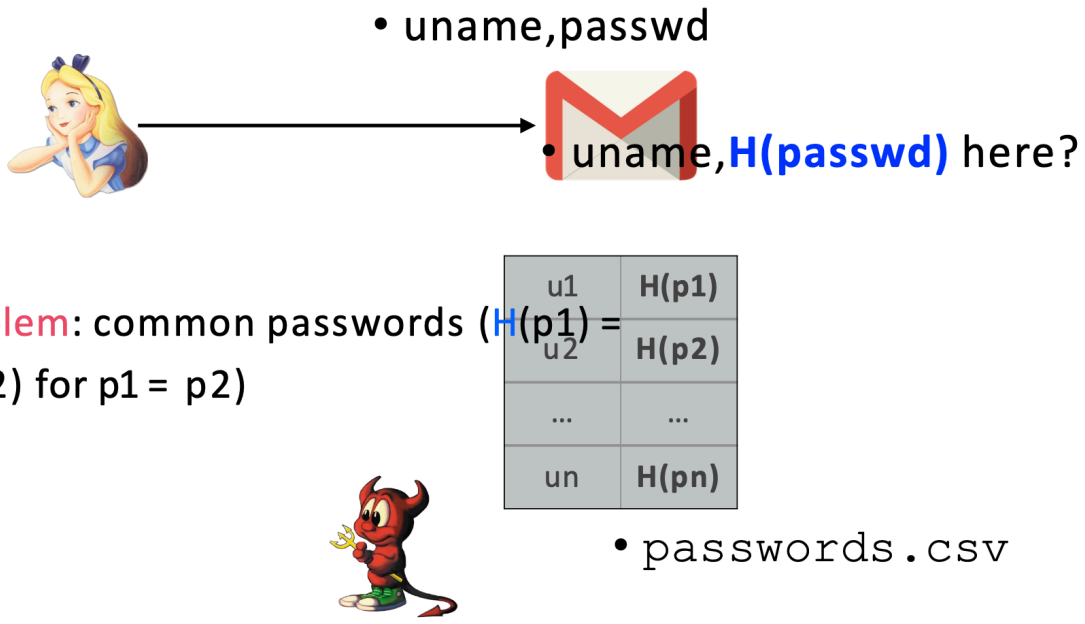
Passwords and PINs

Passwords

Storing passwords

- problem: password is revealed
- solution: use a hash function??

DO NOT store the actual passwords in csv file (anyone can use them if they have access) - store the hash values of the passwords instead (in hash format)



Why not encrypt?

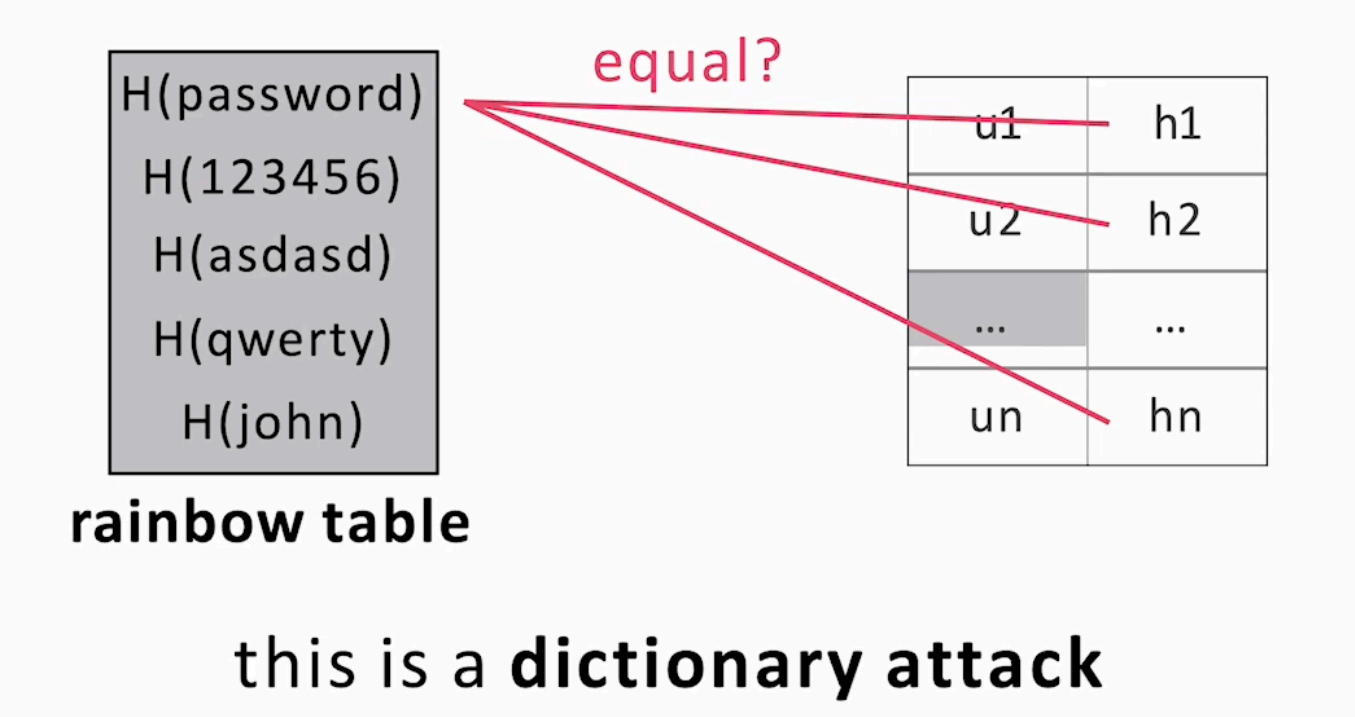
- not feasible for gmail to store the encrypted key
- key have to be stored (hard to import/memorise)

you can make educated guesses on passwords as some passwords are common where the hash values are the same, if passwords before hashing are same

Common Passwords
them they dumb af

general purpose	service specific
password(1)	hotmail
123456	gmail
asdasd	dreamweaver
qwerty(123)	macromedia
<names>	linkedin

Cracking passwords
Goal - recover common passwords, given a password file (containing hash values of passwords)



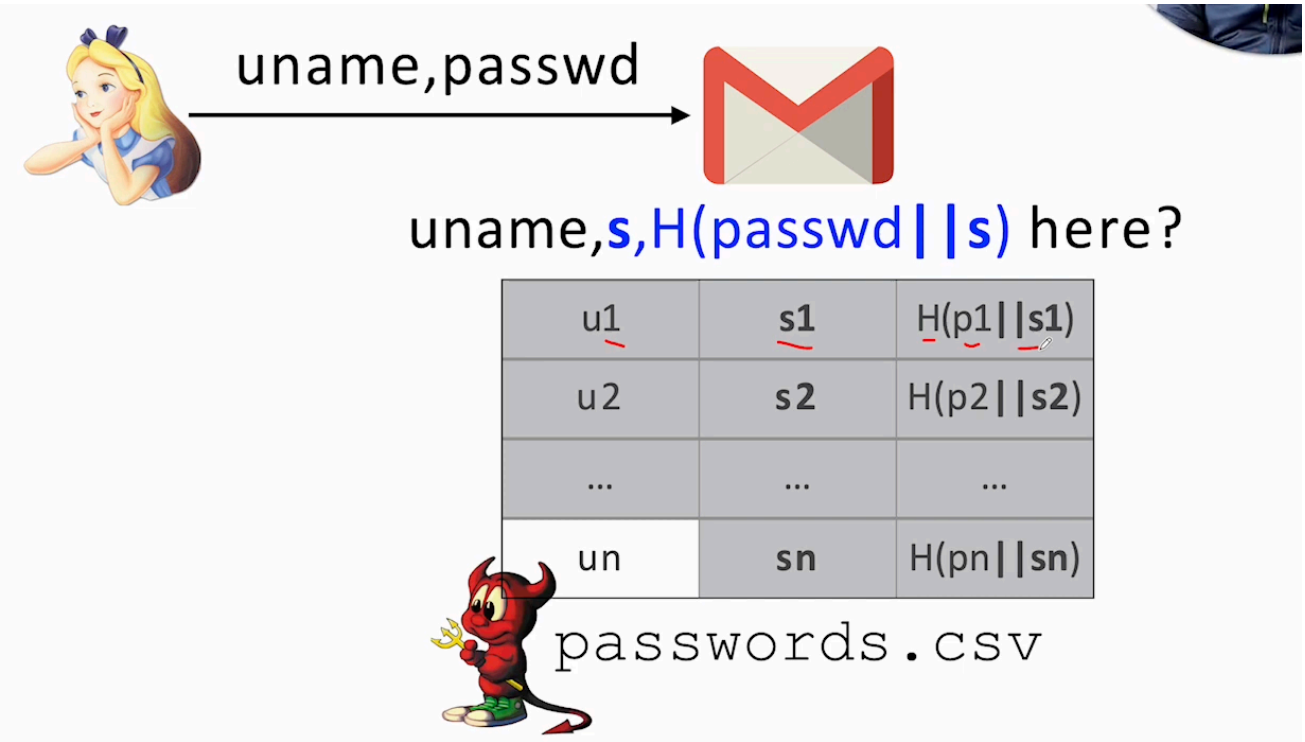
use a **rainbow table** - contains hash values of most common passwords
check if the output correspond to any of the hashes in the passwords file

known as a **dictionary attack**

you can crack passwords within hours (gpu clusters can make 350 billion guesses per second :0)

Solution

- do not store passwords in hash format
- instead store the salt where its unique per user, and store the hash result form hashing the concatenation of the password and the salt



problem: common passwords
solution: use a salt

even if passwords p_1 and p_2 are the same, having a different salt will get you different hash values

Unix passwords

- password stored in /etc/passwd
- hashing: instead of password, store $H(password)$
  - when user enter password, compute hash and compare with entry in password file
- hash function H must be:
  - **one-way**: given $H(password)$, hard to find password
    - no known algorithm better than trial and error

Old-school unix passwords

- uses DES encryption as hash function

    - encrypt NULL string using password as key
    - truncates passwrods to 8 characters
    - run DES 25 times - to slow down brute force attacks
- problem: as metnioned passwords are not truly random
    - 52 upper and lower-case letters, 10 digits and 32 punctionation symbols: there are $94^8 \approx 6 \times 10^{15}$ possible 8-character passwords
    - humans like to use dictionarty words, human and pet names $\approx$ 1 million common passwords

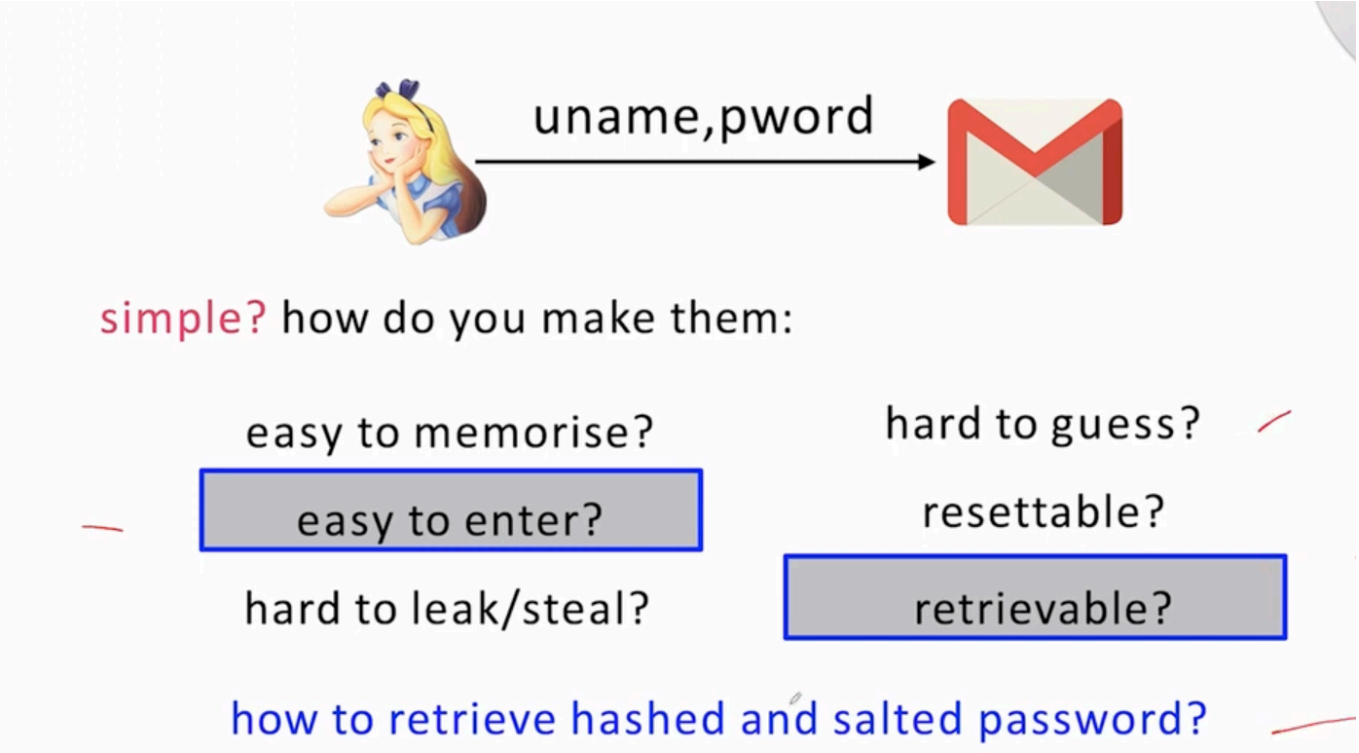
Salting

- /etc/shadow
    - emidec:6/Tj902wM$XasSkj1qs46FZRc3RDMAUK0AsUE3OJSChgZNp/edPBNhm7yuR5PvKe3.yT7RsbmJXtT1nrgMyh28Qi/06pmDd/:15297:0:99999:7:::
    - username, password, days since last change, min #days to change password, max #days password validity, #days warn, #days account disabled after password expires, #days account disable (field delimiter is the : symbol)
- "password" is `$ID$salt$hash`
    - ID denotes encryption method used (6 = SHA-512)
    - salt is a random number, different for every user, chosen when password is first set
    - Hash is H(salt, password)

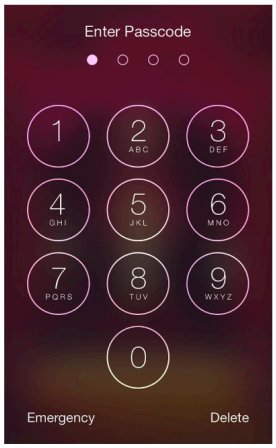
why salting
without salt:

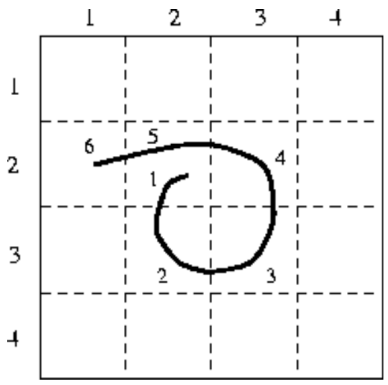
- attacker can pre-compute hashes of all dictionary words once for all password entries
  - same hash function used on all machines where identical passwords hash to identical values
  - one table of hash values works for all password files
  with salt:
- attacker mujst compute hashes of all dictionary words once for each combination of salt value and password
    - with a 12-bit random salt, same password can hash to 4096 different hash values

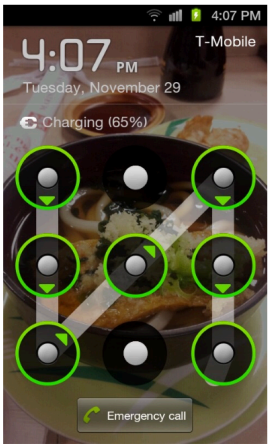
Retrieving hashed and salted passwords



Graphical passwords







- makes passwords easier to memorise

Security Questions

questions about what you already know

What is your pet's name?
In what year was your father born?
In what county where you born?
What is the color of your eyes?

What is the first name of the person you first kissed?
What is the last name of the teacher who gave you your first failing grade?
What is the name of the place your wedding reception was held?
In what city or town did you meet your spouse/partner?
What was the make and model of your first car?

Coping Strategies

Try to enforce:		**Users will**
different passwords		use same password
long passwords		add '123' (or other padding)
random passwords	⟶	write passwords down
letters and numbers		add '1' (or '123' or other)
regular passwords changes		add, e.g., 'spring' or 'june'
more complicated stuff		use reset functionality

Authentication as user task

properties of password-based authentication:

- unaided recall
- recall and entry have to be 100% correct
- no corrective feedback on failure

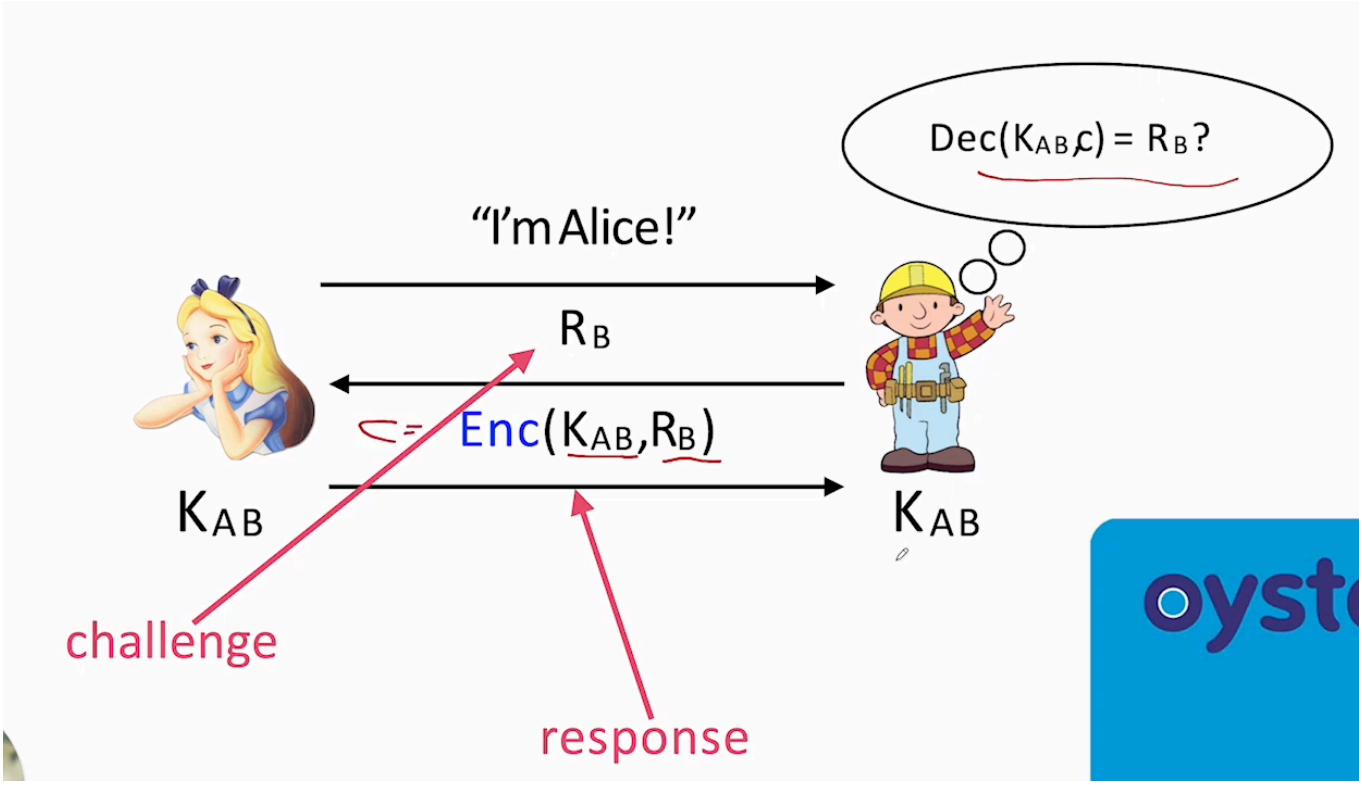
coping strategies:

- re-using same password (or small set of passwords)
- writing passwords down
- relying on password reset

Passwords:
security - want long and random passwords
usability - people can't generate/remember these

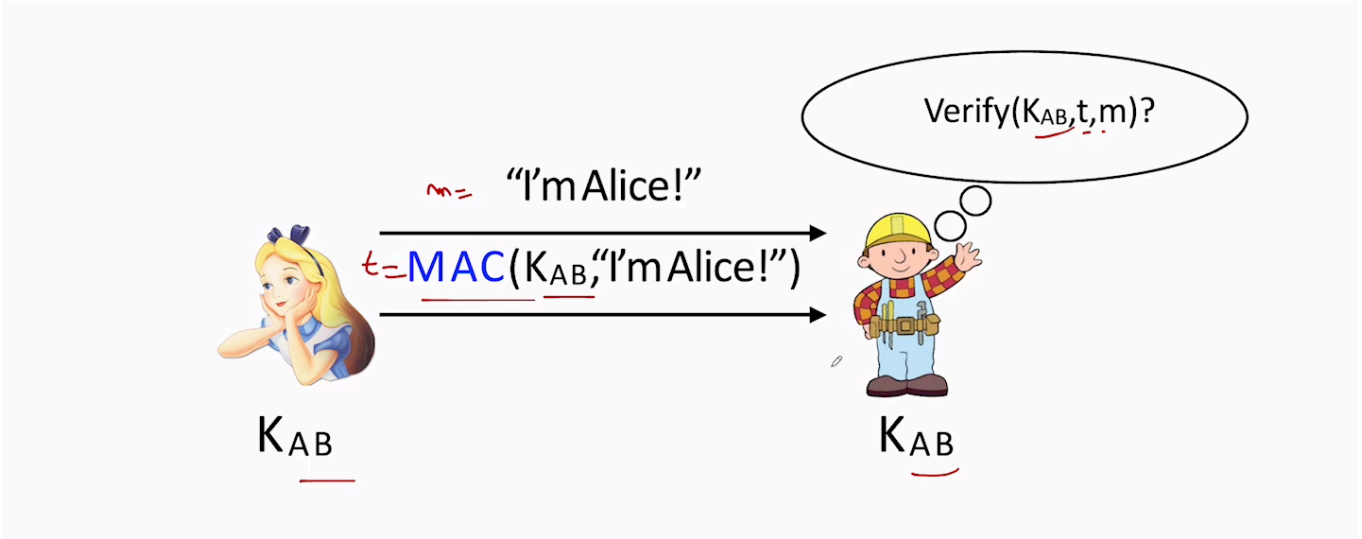
Security tokens

Challenge response

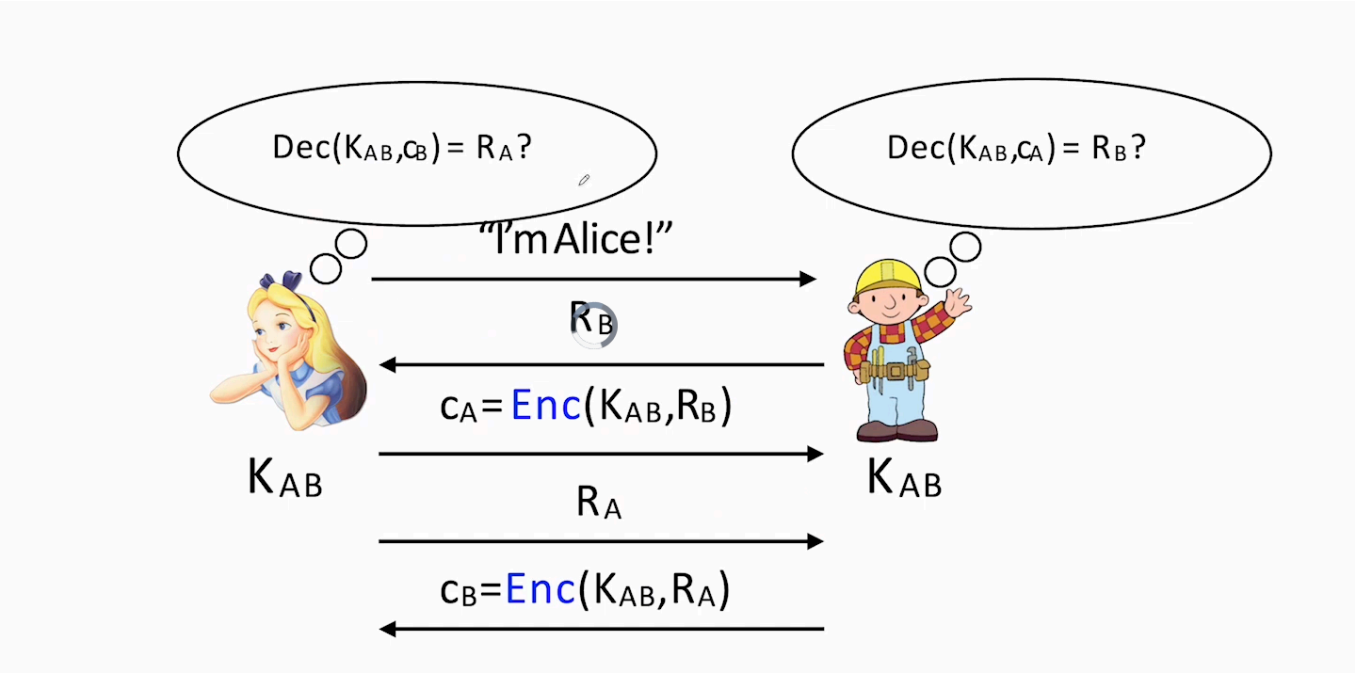


Seen in oyster card (there is a key inside where key is used in authentication to authenticate to TfL system)

Computing MAC on Alice and Bob's key won't work in really coxnstraints environemnts - Bob will have to keep track of m and t. Eavesdropper can use t to get the message

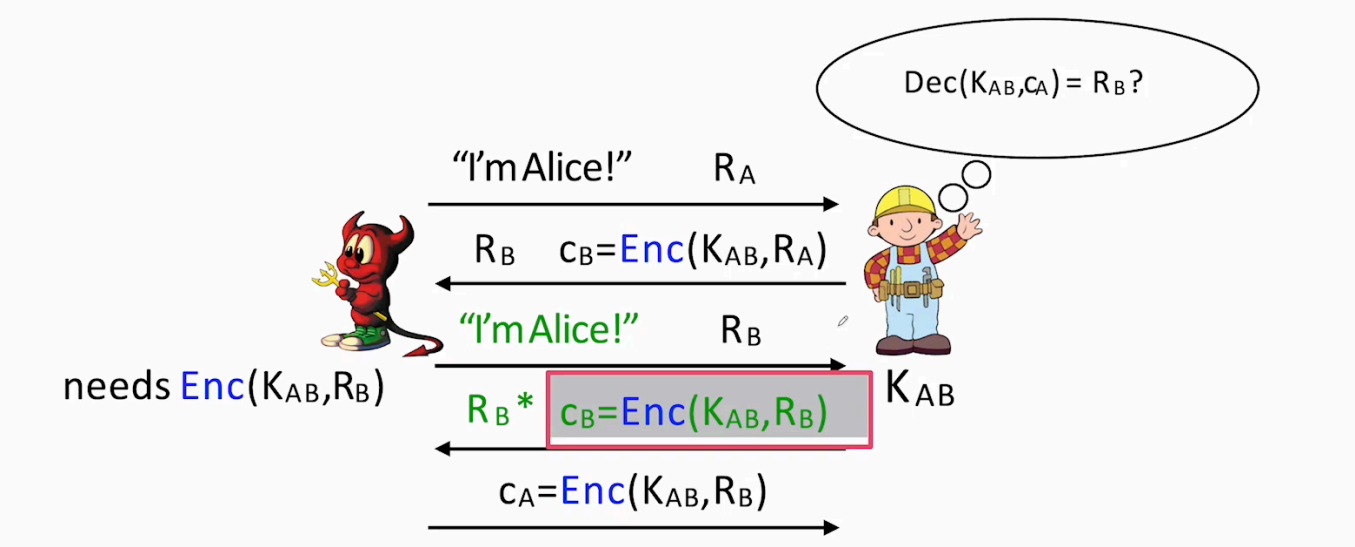


How about a **two-way challenge response**



Bob generates a challenge R_B and Alice encrypts R_B using the shared key
Bob has to check if decrypting C_A with K_{AB} will return the same challenge he generated R_B
Alice will send a challenge R_A (works the same way but inverse as Bob)

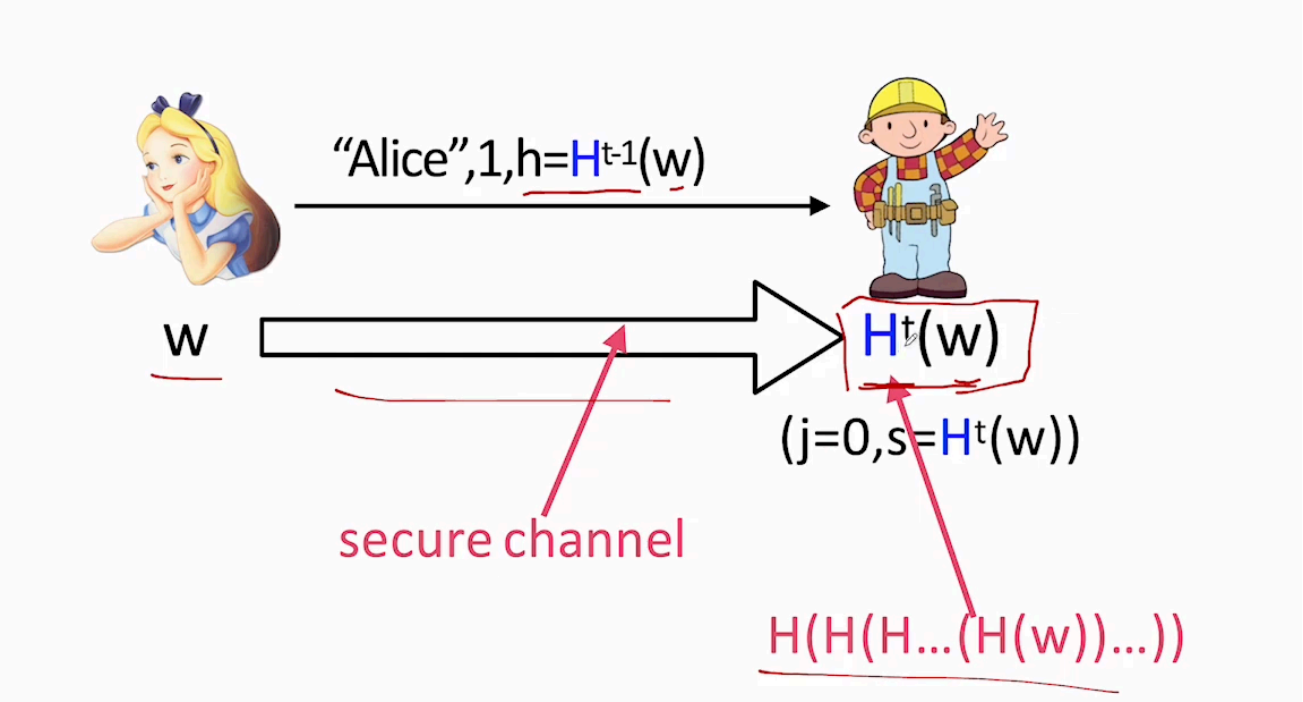
Adversary can pretend to be Alice (exploit the two-way response)



Don't design your own crypto as you need to consider all possible attackers

One-Time Password (OTP)

- could be based on time (requires synchroniastion) or math

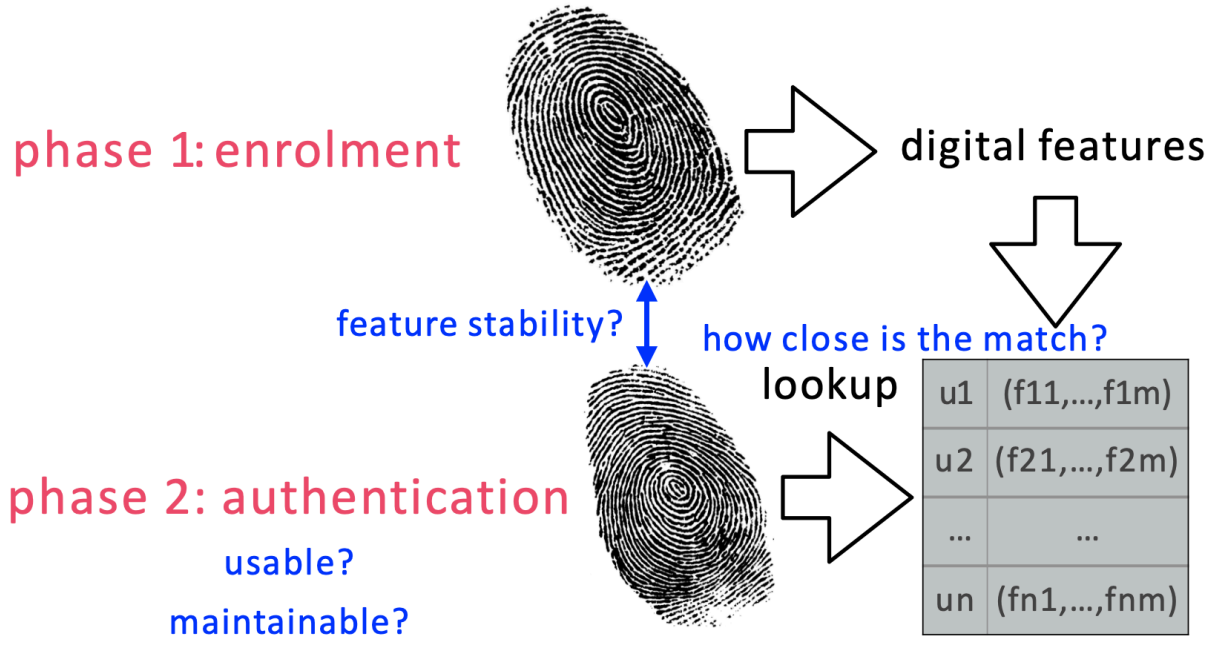


property:
pre-image resistance: given h, hard to find m such that $H(m) = h$

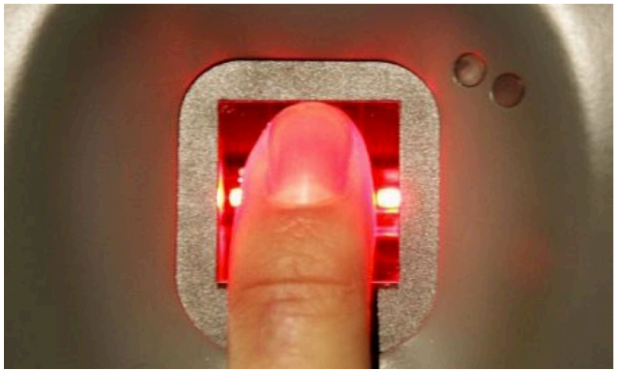
Biometrics

- A biometric is a property of an individual that can be measurted and from which **distinguishing** and **repeatable** features can be extracted to automatically recognise individuals
- biometric sample - raw data obtained from a capture device
- biometric template - stored biometric features

how is data stored?



Fingerprint scanners



challenges	how does it do?
usable?	okay
maintainable?	okay
how close is the match?	good
feature stability?	good

check for "liveness" to avoid amputation

Hand scanners

challenges	how does it do?	
usable?	good	
maintainable?	good	
how close is the match?	okay	hard to train
feature stability?	good	

hard to train scanner on how to determine closeness

Retina scanners

challenges	how does it do?	
usable?	okay	
maintainable?	good	think contact lenses
how close is the match?	good	
feature stability?	okay	

Technical considerations of Biometrics

Advantages:

- nothing to remember
- passive
- can't share

- unique to one person

DIsadvantages:

- Revocation is difficult
- Invasive
- Private but not secret (easy to sell off glass, door handle, ... - can make fake fingerprints to fool fingerprint scanners)
- if sale acceptance rate is 1 in million, chance of false match is above 50% with only 1609 samples (birthday paradox)

User Acceptance

- Threat to **privacy** in terms of stored templates
  - who is using the data and for what purpose?
  - implicit identification
- doubts about **reliability**
  - sophisticated attackers can bypass system
  - successful attacks erode confidence

Multi-factor authentication

Use different kinds of combination of authentication

One time password is obtained from different channel (text message to user)

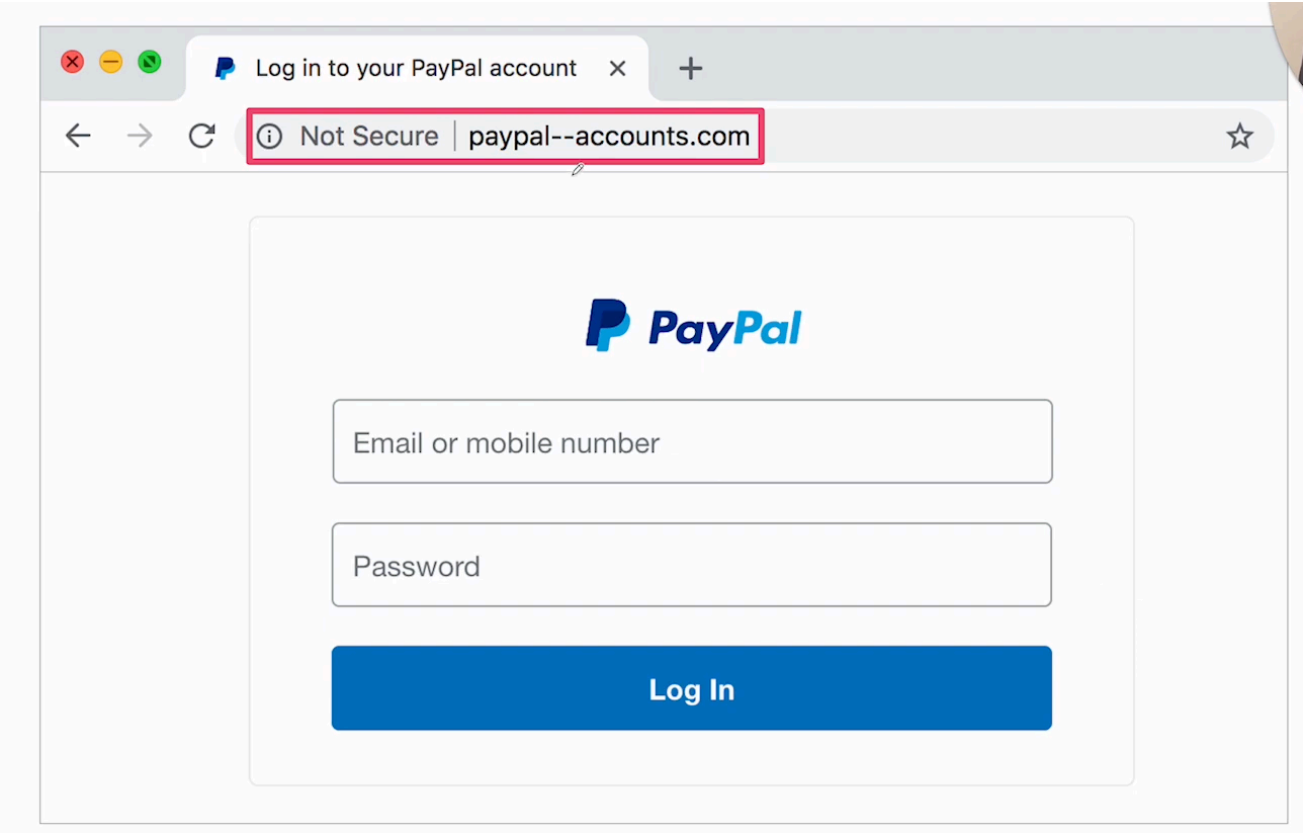
Having multiple factors do help in preventing account takeovers

Attacks on authentication

Phishing

- **happens via email** - impersonate a legitimate source - contains a link that redirect to fake website
- (Vishing happens over phone - "voice phishing" via deepfakes for voice impersonation)
- (Smishing - happens over SMS)
- (Pharming - technique to enchance phishing)
- used to steal credentials by redirecting users to a fake website to enter their credentials
- can be hard to distinguished between phishing emails and real emails

- e,g,

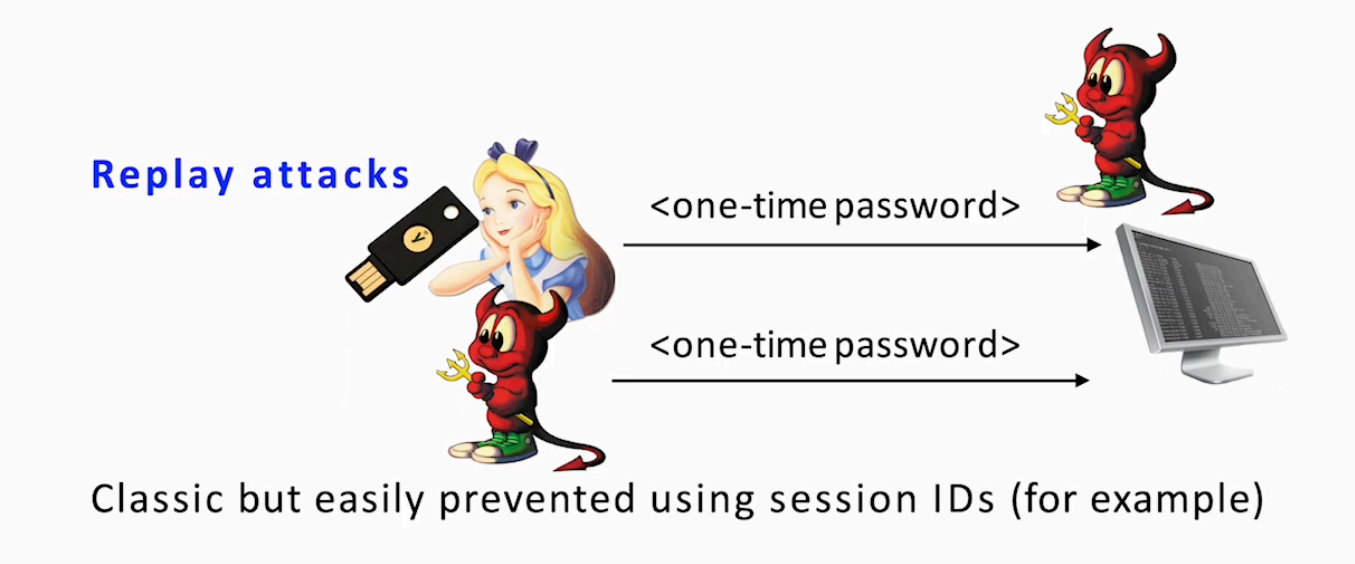


examples of fake urls (typosquatting):

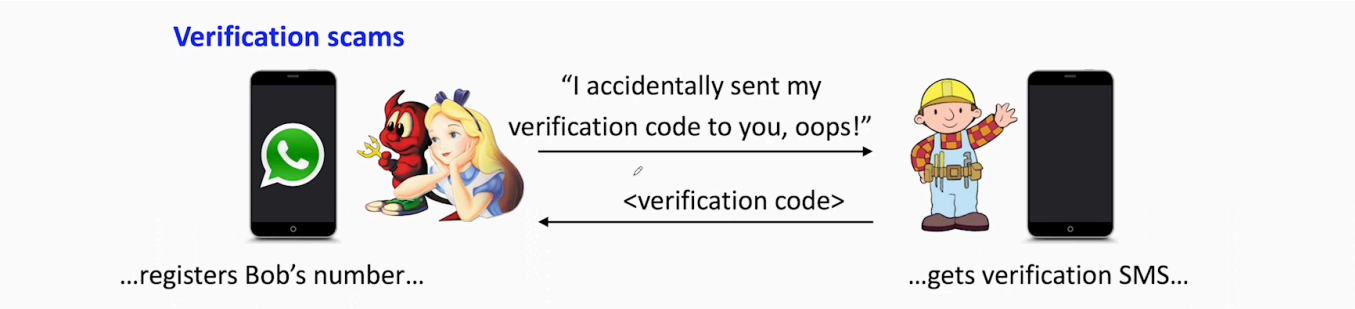
other attacks on 'what you know':

- capture attacks
  - skimming (for PINs)
  - keylogging
  - packet sniffing (unencrypted web traffic)
- intimate partner violence (adversary is a partner)
- observation attacks (shoulder-surfing)
- side-channel attacks (keyboard emanations, finger grease)
- coercion ("rubber-hose") attacks

attacks on 'what you have'



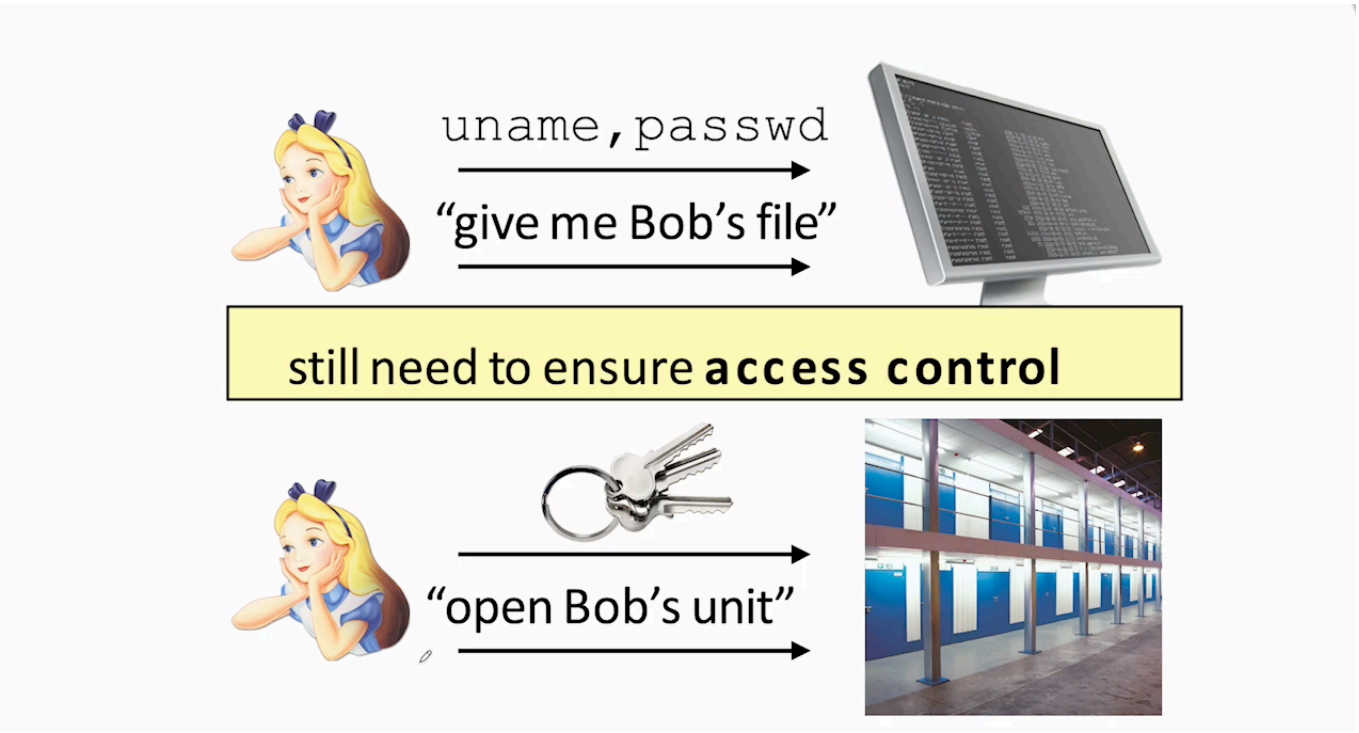
verification scams:



Password reuse
Number of accounts N, number of different passwords M
normally its $N > M$ as people tend to reuse their passwords
password managers ensure that $N = M$

Week 8: Access Control

Access Control



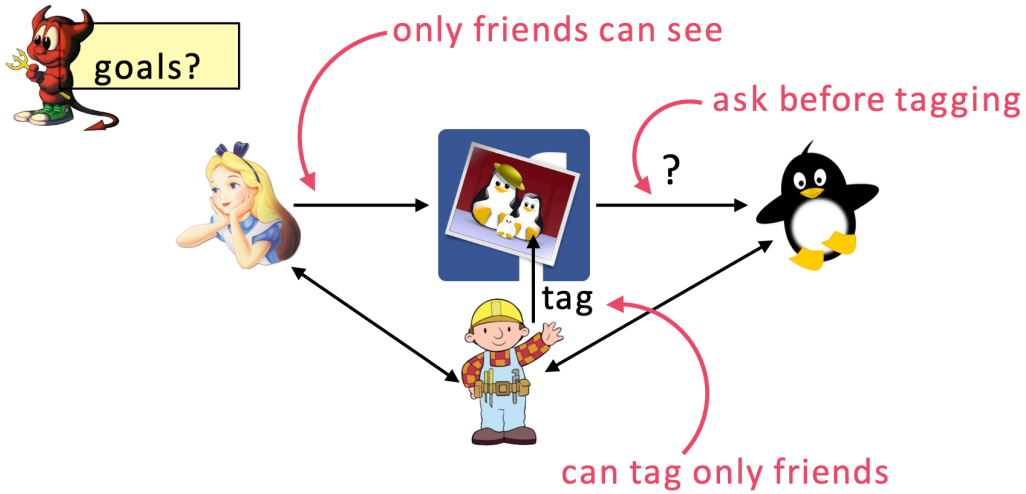
Once user is authenticated to server, user is only given the information that is needed (access control)

What is Access Control?

- **The ability of one entity to permit or deny the use of a particular source to another**
- authentication is already a (coarse) form of access control

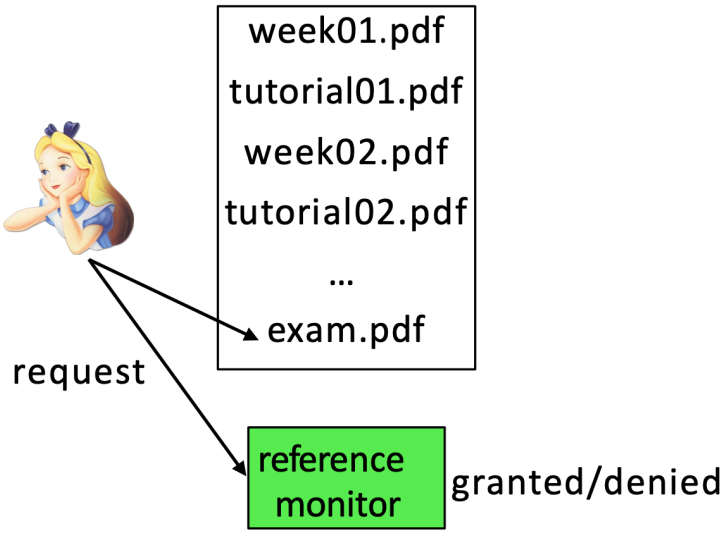
example of design principles used: least privilege, complete mediation (need a module to access reference monitor - OS has to check permissions), fail-safe default, open design, psychological acceptability, economy of mechanisms

Example: Social networks



Access Rights

A way to exercise acceess control - ensure users who want to access files is either granted/denied (do they have the rights to access)



Types of FIle Accesses

	non-ALT	ALT
non-OBS	execute	append
OBS	read	write

- subjects (s)
- objects (o)
- access rights (r/p)

subjects - users of the system
objects - the different files
access rights - execute, read, write, append (some combination of ALTeration / OBServation)

- execute - cannot alter/observe
- read - can observe/cannot alter
- append - can alter/cannot observe
- write - can alter/observe

Access Control Matrix

Rows: users
Columns: files

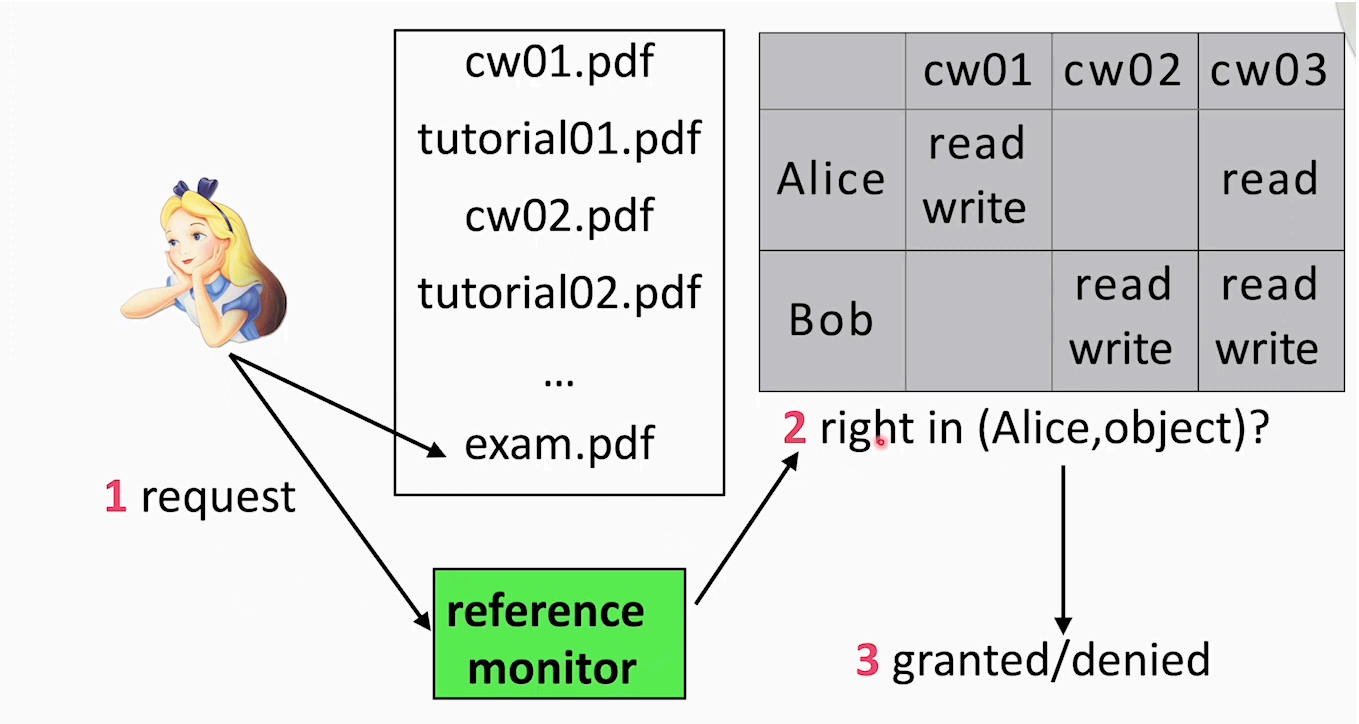
- shows access rights for particular files for specific users

S: Alice, Bob
O: cw01,cw02, cw03
R: read, write

	cw01	cw02	cw03
Alice	read write		read
Bob		read write	read write

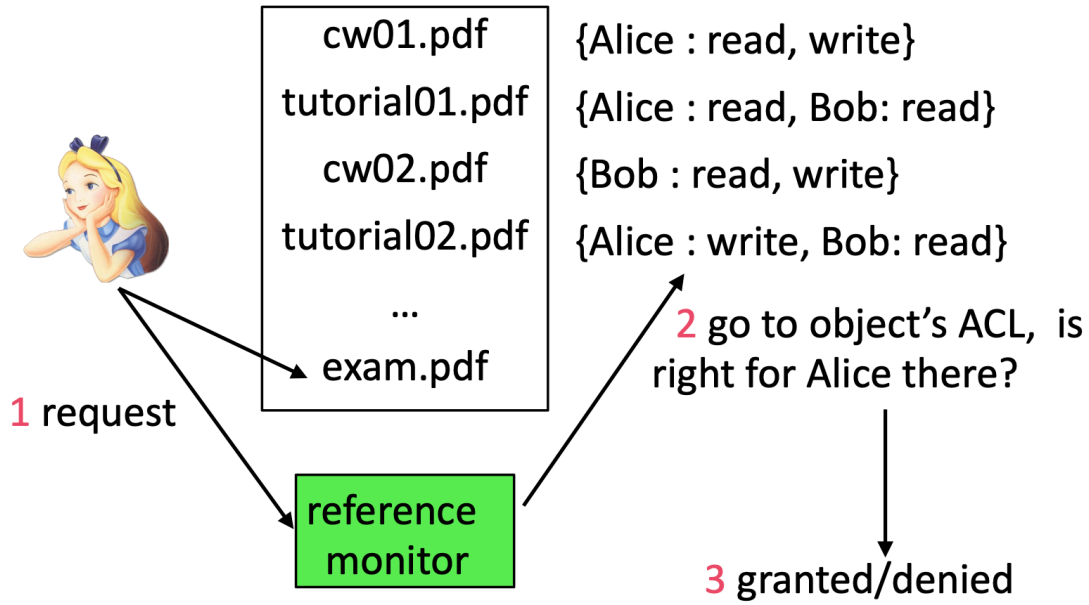
can Alice read cw01?
can Bob write cw01?

answer: Alice can read cw01 / Bob cannot write cw01



When a user wants to request a file, reference monitor checks the access rights of the user and the file and grants/denies accordingly

Access Control List



For each file, there is a list of users and their respective access rights (known as the ACL - access control list)

Reference monitor uses the ACL to see if the user making the request has the rights for that file. Itll return the result (granted/denied)

UNIX Permissions

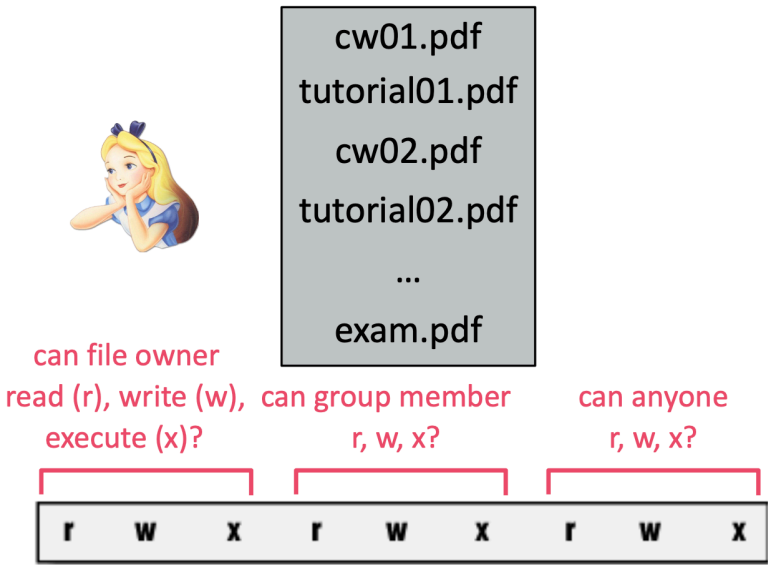
For each file, there are 3 possible permissions:

- read (r)
- write (w)
- execute (x)

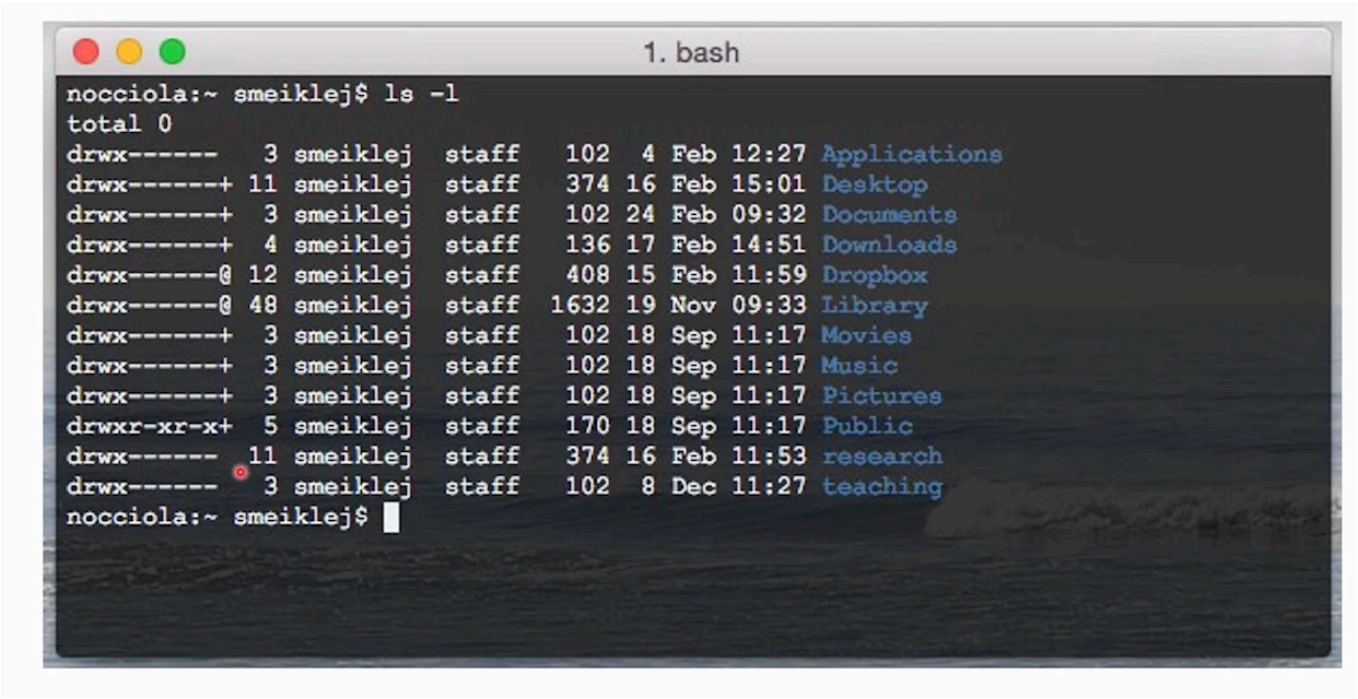
There are 3 kinds of subjects where one subject can have any of the 3 possible permissions:

- file owner
- group member
- anyone (default)

This gets you 9 possible access rights



example:

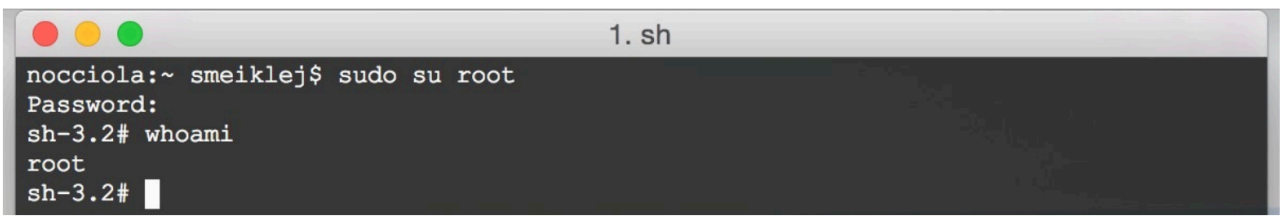


drwx----- = file owner can only read/write/execute in directory

drwxr-xr-x+ = any user can read/execute, but only file owner can read/write/execute

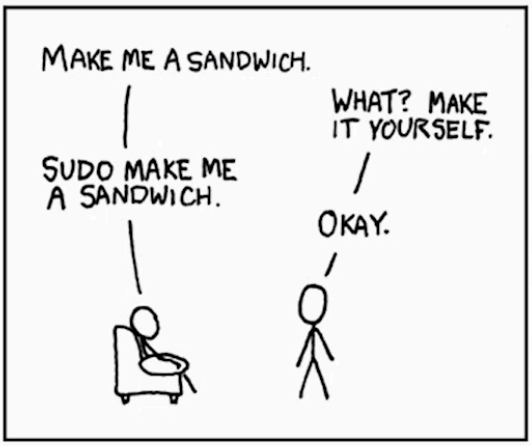
`ls -l` - gets the directory folders and their respective file permissions in UNIX

Root User



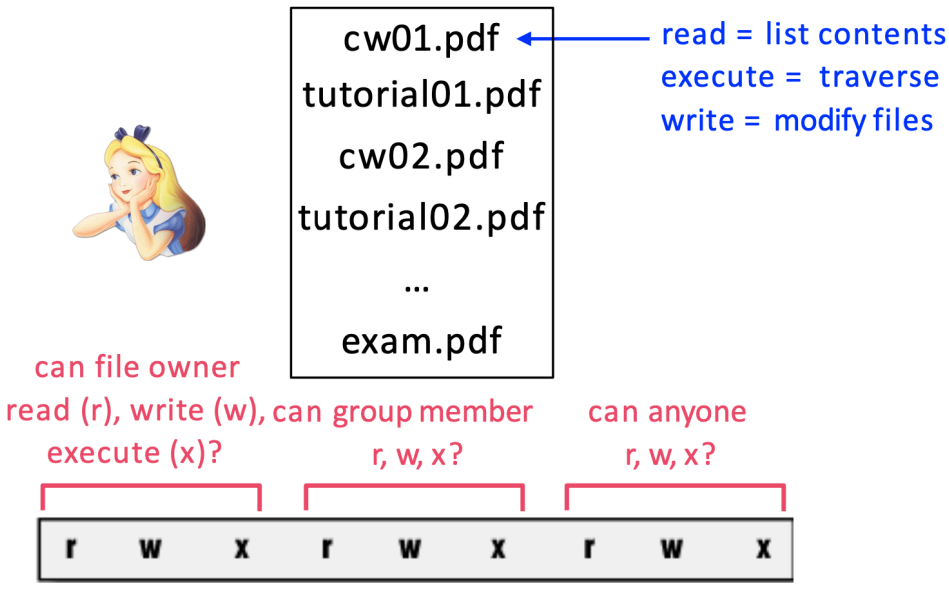
- the default owner of all system files
- protects users from themselves
- important in multi-user systems

`sudo`



- allows one user to temporarily run things with privileges of another (often `root`)
- accountability: `sudo` usage is logged
- better than creating new accounts with more permissions
- sudo elevates privilege / run things with privileges of another user

Permissions for directories



read - see what is in the directory
write - modify each file in directory
execute - traverse the directory (go inside directory and their subfolders)

Sticky Bit

Sticky bit (T) for a directory changes write privileges, can rename or delete files only if you are the owner (or root)

Can do this even if you don't have write permissions on the individual files!

The sticky bit (T) for a directory changes write privileges, can rename or delete files only if you are the owner (or root)

read = list contents
execute = traverse
write = modify files

= create, rename, or delete

```
nocciola:~ smeiklej$ chmod 1700 research
nocciola:~ smeiklej$ ls -l
total 0
drwx------+ 10 smeiklej  staff   320 12 Mar 16:29 Desktop
drwx------+  4 smeiklej  staff   128  6 Mar  2015 Documents
drwx------+  4 smeiklej  staff   128 13 Mar 09:30 Downloads
drwx------@ 18 smeiklej  staff   576 11 Mar 21:18 Dropbox
drwx------@ 74 smeiklej  staff  2368  5 Nov 11:59 Library
drwx------+  3 smeiklej  staff    96 18 Sep  2014 Movies
drwx------+  6 smeiklej  staff   192 20 Nov  2017 Music
drwx------+  6 smeiklej  staff   192 21 Jun  2017 Pictures
drwxr-xr-x+  5 smeiklej  staff   160 18 Sep  2014 Public
drwx------T 21 smeiklej  staff   672 17 Dec 10:17 research
drwx------  8 smeiklej  staff   256 20 Dec 13:58 teaching
drwx------  6 smeiklej  staff   192  5 Jan 21:11 writing
```

Example:

permissions	owner	group	filename
rwx-----x	bob	eng	week01.pdf
rwxrwxrwx	bob	eng	cw01.pdf
rwx--x--x	alice	alice	week02.pdf
rw-r-----	alice	cs	cw02.pdf
rw-r--r--	bob	cs	week03.pdf
rw--wxr--	root	cs	exam.pdf

alice,cs

which files can Alice write?

answer: `cw02.pdf` , $ `week02.pdf`

Trusted Computing Base

refers to every component of the system which the security policy relies

if something goes wrong, then security policy may be violated
needs to be kept small

example of **economy of mechanism**

UNIX Processes

Processes

- are isolated (cannot access each others' memory)
- run with userID (uid) of a specific user
  - when you run a process, its with the permissions of your `uid`
  - processes can access any files that you have access to (assuming there is access control implemented for processes)
- processes started by `root` (`uid` 0) can reduce their privileges by changing to a less privileged `uid`

PRocess User IDs

every process has 3 different user IDs:

- **effective user id (EUID)** - determines permissions for the process
- **real user id (RUID)** - determines user that started the process
- **saved user id (SUID)** - EUID prior to any changes

Changing user id

`root` can change EUID/RUID/SUID to arbitrary values
unprivileged users can change EUID to RUID or SUID

`setuid(x)` - change all of EUID/RUID/SUID to `x`
`seteuid(x)` - changes just EUID to `x`

example - SSH:

What if SSH runs as `root` and ran the following code?

```
→ if (authenticate(uid, passwd) == SUCCESS) {
      seteuid(uid);
      exec("/bin/bash");
  }
```

euid = 0
ruid = 0
suid = 0

authenticate user id and password, if successful then set the euid to the uid and execute the bash script

```
  if (authenticate(uid, passwd) == SUCCESS) {
      seteuid(uid);
→     exec("/bin/bash");
  }
```

euid = ~~0~~ uid
ruid = 0
suid = 0

Unprivileged users can change EUID to RUID or SUID

setuid(0) ← get root privileges

as unprivileged users can change EUID to RUID/SUID, user can get root privileges by setting uid to 0

instead of setting euid, you set uid (everything) to the uid

```
  if (authenticate(uid, passwd) == SUCCESS) {
→     setuid(uid);
      exec("/bin/bash");
  }
```

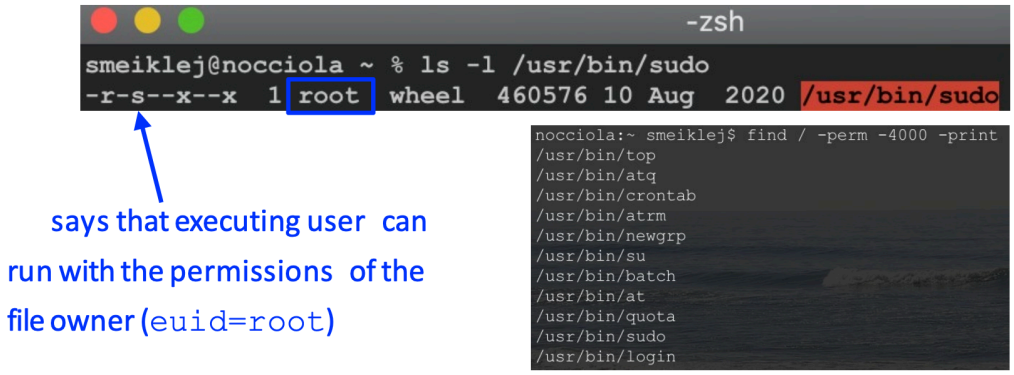
euid = ~~0~~ uid
ruid = ~~0~~ uid
suid = ~~0~~ uid

Elevating Privileges

Sometimes we need to elevate our own privileges
Example: running `passwd` modifies `/etc/shadow`, which `root` only can read/write

UNIX allows you to set EUID of an executable to be the file owner rather than executing user using the `setuid` bit

SETUID BIT



SUID (Saved user ID) is used to know which user's password can be modified, when running `passwd`

Changing Privileges

When user connects to system, it runs login process as root

- authenticate user with username/password
- change userid and groupid to be those of the user
- execute the user's shell
- so system **drops** privileges from root to regular user

does user ever need to **elevate** privileges?

- answer : yes
- e.g. changing their password as the master password file for system has to be edited
- need some authorised way to elevate privileges i.e. using setuid bit

Privileges

Other architectures (e.g. Windows) have differences but themes are same

Pros:

- simple model provides protection for most situations
- flexible enough to make most access control policies as possible

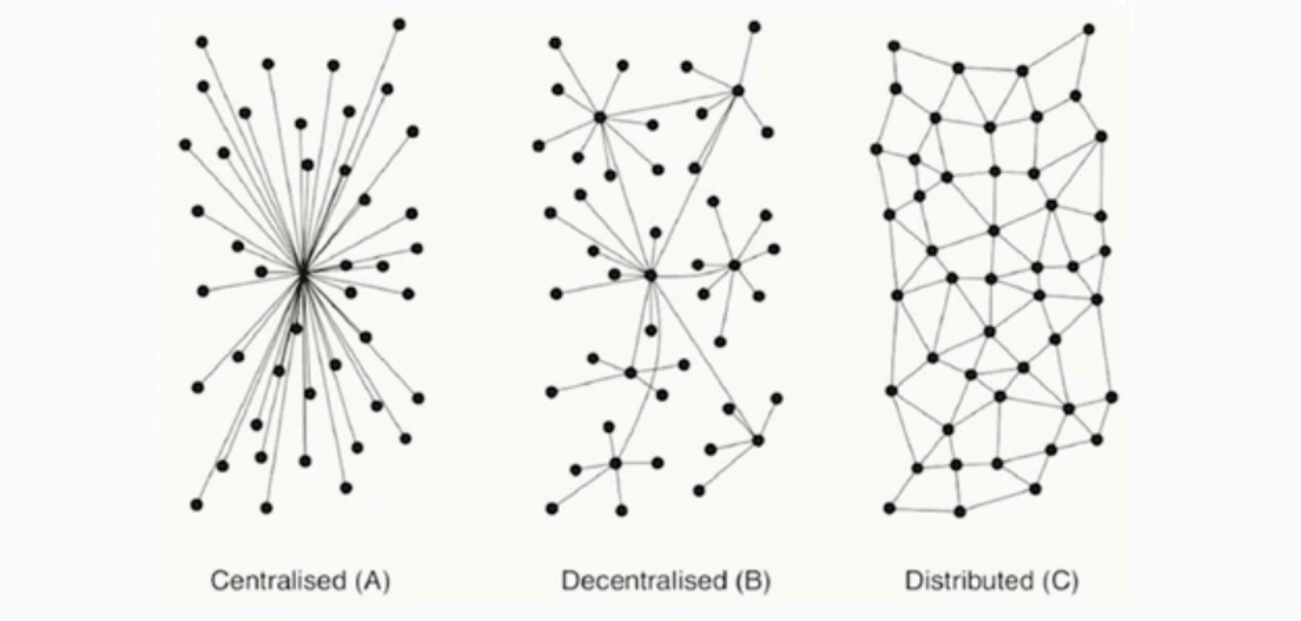
Cons:

- ACLs (access control list) are coarse-grained
- can't differentiate processes run by single user
- nearly all system operations need root access

Permissions

past (present) - centralised (one mainframe computer with many users)

- still relevant in large organisations
- model used in platforms like Moodle



  the present - currently uses many distributed personal devices where users need to make more decisions for themselves

Types of Access Control

Example - social networks

Example: Social Networks

S: Alice, Bob, penguin
O: photo
R: view, tag, auth

	photo
Alice	view tag
Bob	view tag
penguin	view tag auth

tag whom?
authorise whom?

what if Alice wants to change who can view photo?

S: Alice, Bob, penguin
O: photo, Alice, Bob, penguin
R: view, tag, auth

	photo	Alice	Bob	penguin
Alice	view	tag auth	tag	
Bob	view	tag	tag auth	tag
penguin			tag	tag auth

Access Control Policies

Mandatory (MAC)

- permissions assigned

Discretionary (DAC)

- owner sets permissions

Role-Based (RBAC)

- can implement MAC or DAC large hierarchial organisations

example:

S: Alice, Bob, penguin

O: photo, Alice, Bob, penguin R:

view, tag, auth, owner

	photo	Alice	Bob	penguin
Alice	view owner	tag auth	tag	
Bob	view	tag	tag auth	tag
penguin			tag	tag auth

Graham-Denning

Creation:

1. subject x **creates** object o

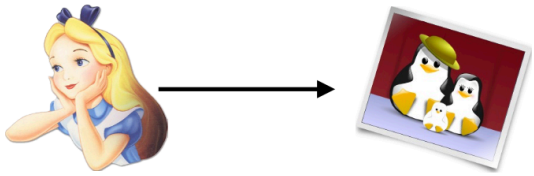
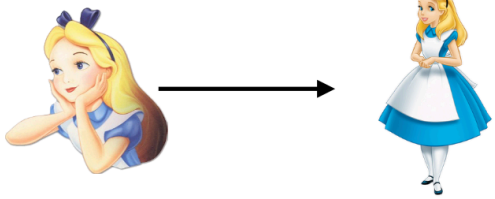


	photo
Alice	owner

2. subject x **creates** subject s



	profile
Alice	control owner

Create an object o and another subject s

Deletion:

3. subject x **deletes** object o

	photo
Alice	owner

reference monitor → (x,o,"owner") in table? then delete column o

4. subject x **deletes** subject s

	profile
Alice	control owner

reference monitor → (x,s,"owner") in table? then delete column s

Rights:

5. subject x **grants** right r/r* on o to s


	photo
Alice	owner
Bob	view

reference monitor (x,o,"owner")? then add (s,o,r/r*)

6. subject x **transfers** right r/r* on o to s

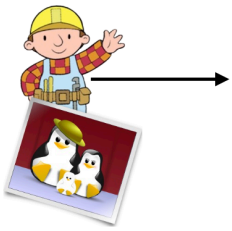
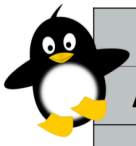



	photo
Alice	owner
Bob	view*
penguin	view

reference monitor (x,o,r*)? then add (s,o,r/r*)

7. subject x **deletes** right r/r* on o for s (revocation)

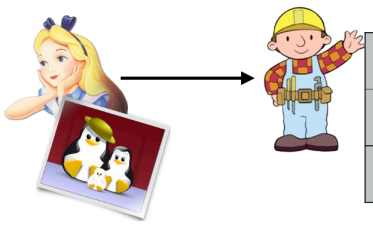


	photo
Alice	owner
Bob	view

reference monitor

(x,o,"owner") or
(x,s,"control")?
then delete (s,o,r/r*)

8. subject x **checks** rights on o for s

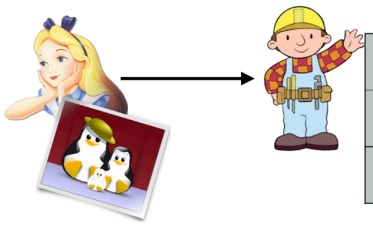


	photo
Alice	owner
Bob	view

reference monitor

(x,o,"owner") or
(x,s,"control")?
then return (s,o,*)

example:

S: Alice, Bob, penguin

O: photo, Alice, Bob, penguin R:

view, owner, control

2 4 2 4 2 4 1 3

	Alice	Bob	penguin	photo
Alice	control owner			owner
Bob		control owner		5view*8
penguin			control owner	6 view 7

RBAC

- only scalable solution
  - 10 users of 10 resources = 100 policy definitions
  - less likely to make mistakes
  - roles are already implemented in system
- already used for UNIX permissions (user, group, world)
- people would change but roles stay the same (permissions stay the same)

Access Control in Organisations

- Ensure that an access control policy is implemented correctly, where it should have:
  - no gaps
  - no conflicts
  - no unintended restrictions
- use **information asymmetry** between system administrators and system owners

Week 9: Security examples

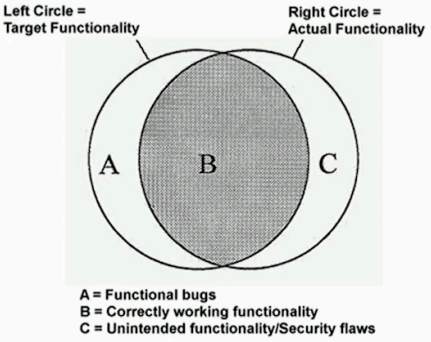
Software security

When is a program secure?

- when it does exactly what it **should**
  - not more
  - not less
- what should a program do? how do we know?
  - somebody tells us (do we trust them)
  - we write the code ourselves (how often is this true)
- program can be "secure" if it doesn't do bad things like
  - crash my system
  - delete or corrupt important files
  - send my password over the internet
- if it could do bad things - is it still secure??

Unintended Functionality

- **Exploit** is a mechanism by which an attacker triggers some unitended functionality of the system (blind spot for the developer)



- security involves understanding both intended/unintended functionalities of the system

Software Vulnerability

- A bug in a program that allows a user capabiltiies that should be denied to them
- example of a common type of vulnerability is the ones that violate control flow integrity (CFI)

Buffer overflow

- program variables get allocated some regions of physical memory in the form of buffers
- buffer overflows happenw hen program writes data beyond its allocated buffers
- ubiquitous in systems-level languages (e.g. C/C++), made worse by the fact that many standard library functions make it easy to go beyond array bounds
- string functions e.g. `strcpy()` and `strcat()` write to destination buffer until they encounter `\0` in input (end of string), so user providing the input (who can easily be the attacker) can control how much gets written

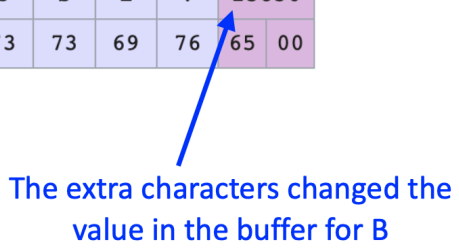
e.g. - `strcpy`

```
char A[8] = "";
unsigned short B = 1979;
```

variable name	A								B	
value	[null string]								1979	
hex value	00	00	00	00	00	00	00	00	07	BB

```
strcpy(A, "excessive");
```

variable name	A								B	
value	'e'	'x'	'c'	'e'	's'	's'	'i'	'v'	25856	
hex	65	78	63	65	73	73	69	76	65	00



only 8 characters are allocated, last character is overflow to B

example of checking password

```
int check_auth(char *password) {
    int auth_flag = 0;
    char pass[16];
    strcpy(pass, password);
    if (strcmp(pass, "abc123") == 0)
        auth_flag = 1;
    return auth_flag;
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        printf("Need to provide a password\n");
    }
   if (check_auth(argv[1]))
        printf("you're logged in\n");
    else
        printf("incorrect password\n");
```

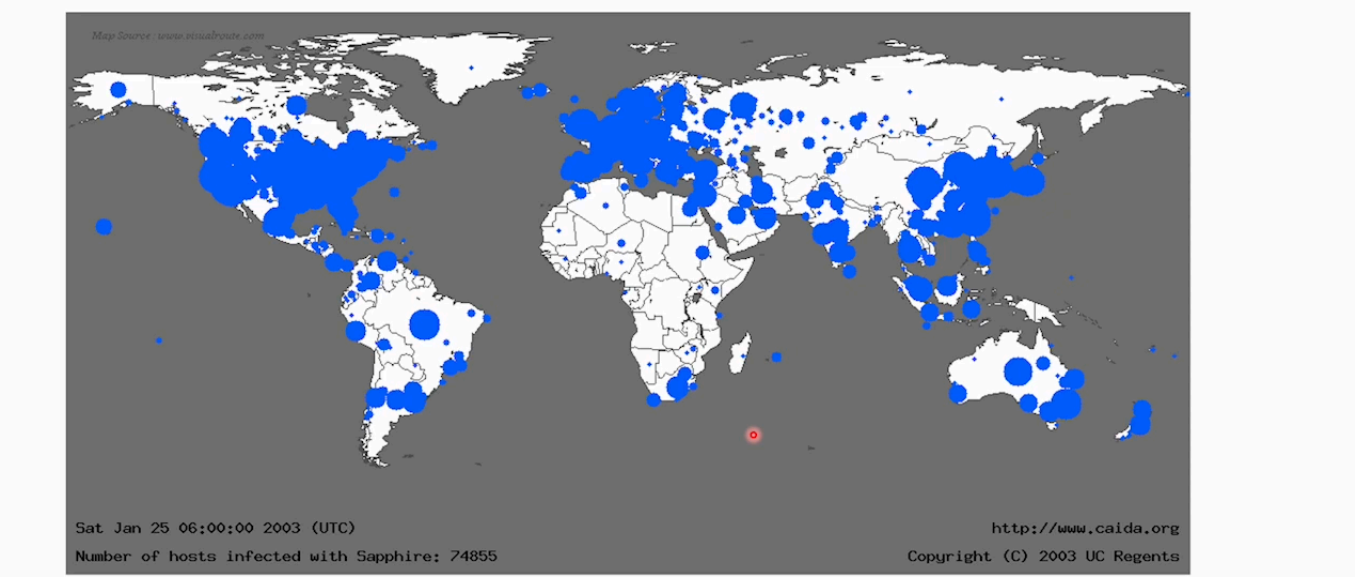
```
}
```

```
➜  ~ gcc -g -fno-stack-protector -D_FORTIFY_SOURCE=0 -o auth auth.c
➜  ~ ./auth emidec
Nope, that's an incorrect password
➜  ~ ./auth emidec123
Nope, that's an incorrect password
➜  ~ ./auth AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Welcome! You're logged in
[1]    6565 segmentation fault  ./auth AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
➜  ~ ./auth AAAAAAAAAAAAAAA
Nope, that's an incorrect password
➜  ~ ./auth AAAAAAAAAAAAAAAAAAA
Welcome! You're logged in
➜  ~
```

password is max 15 characters (passed from command line) - user/adversary can overwrite the password (by overflowing the buffer)

you can authenticate via buffer overflow without knowing the correct password

Worms

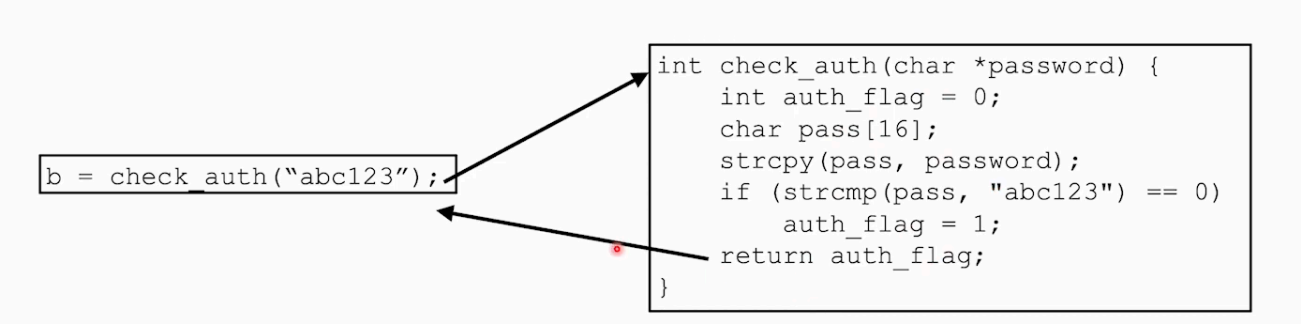


- spread autonomously by exploiting vulnerabiltiies
- spread quickly/unpredictably, easy to detect
- e.g. slammer worm infected 75k within 10 mins

Buffer overflows

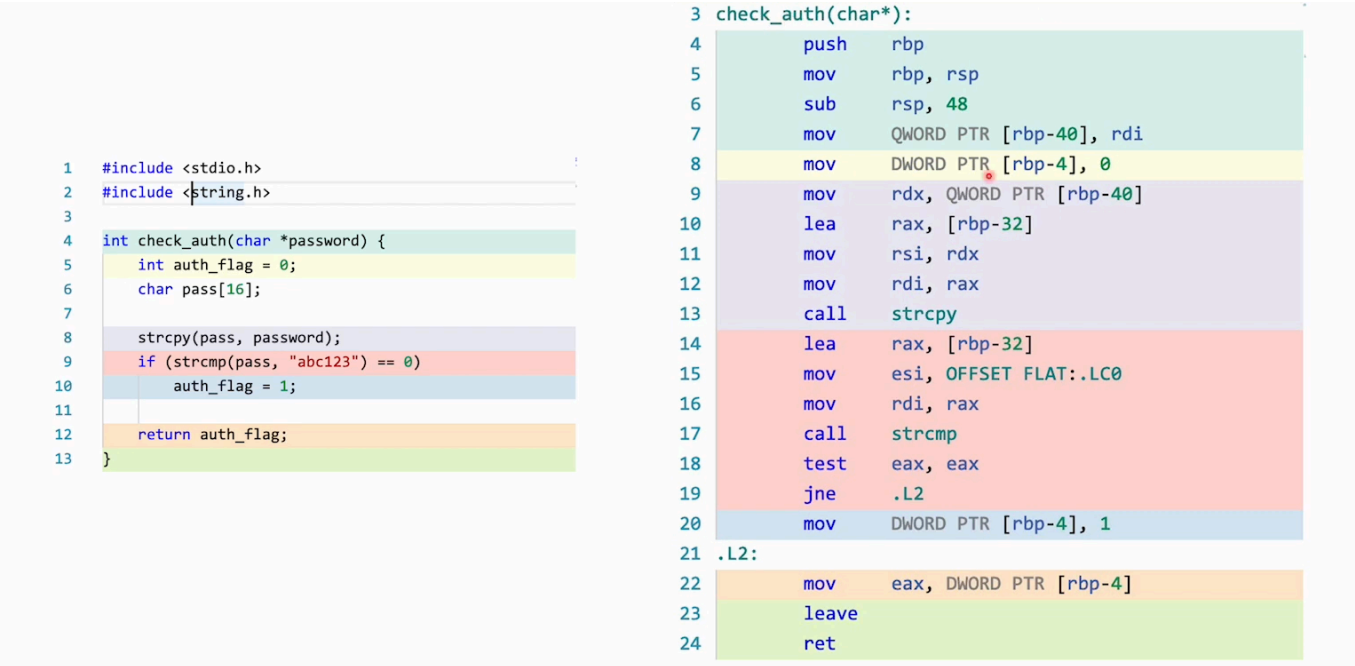
Function calls

- How do function calls work?



  how does called function know where to return to? / where is the retur address stored?
- answer: computer keeps track by using a **stack**

example:



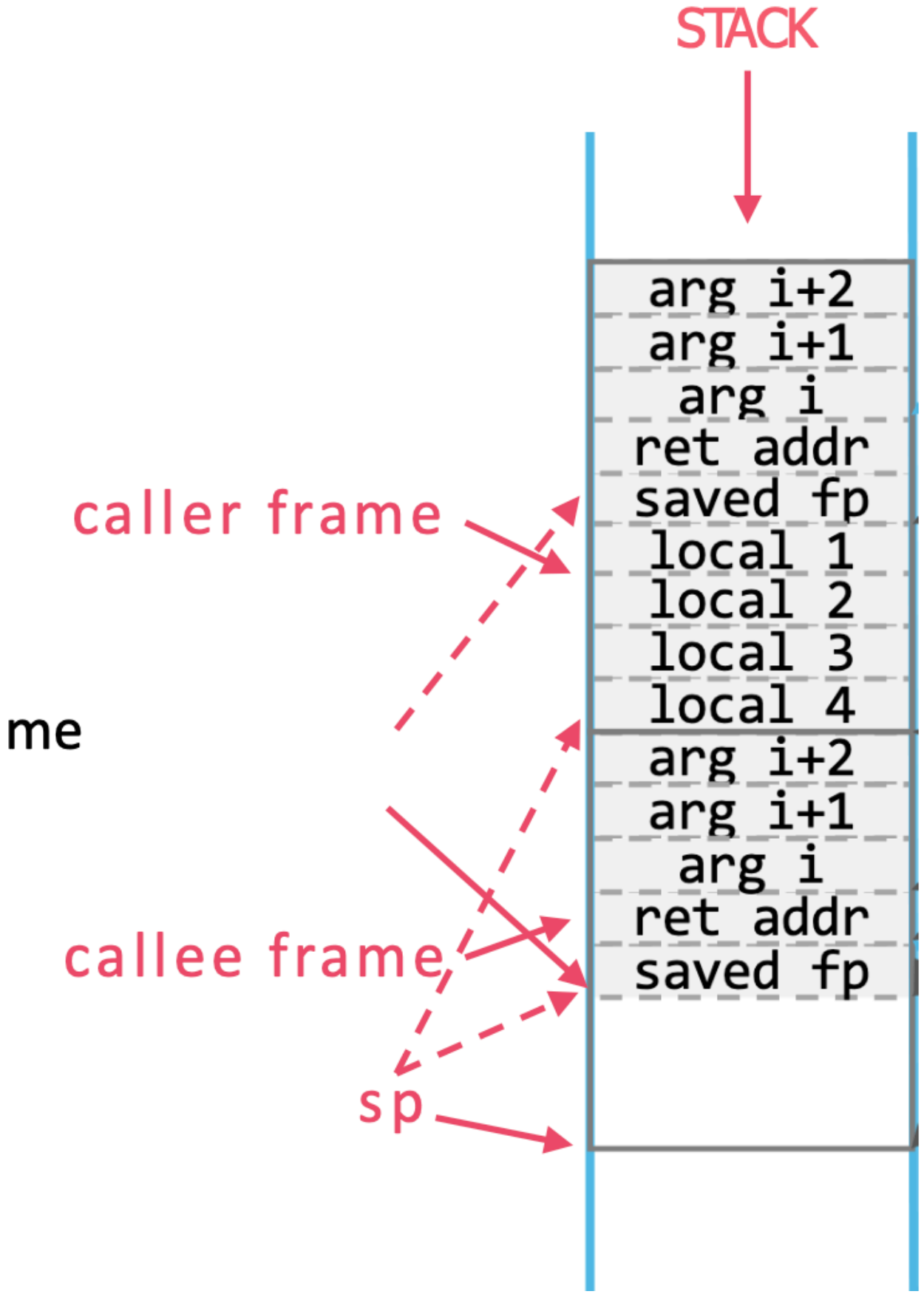
```c
#include <stdio.h>
#include <string.h>

int check_auth(char *password) {
    int auth_flag = 0;
    char pass[16];

    strcpy(pass, password);
    if (strcmp(pass, "abc123") == 0)
        auth_flag = 1;

    return auth_flag;
}
```

```asm
check_auth(char*):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 48
        mov     QWORD PTR [rbp-40], rdi
        mov     DWORD PTR [rbp-4], 0
        mov     rdx, QWORD PTR [rbp-40]
        lea     rax, [rbp-32]
        mov     rsi, rdx
        mov     rdi, rax
        call    strcpy
        lea     rax, [rbp-32]
        mov     esi, OFFSET FLAT:.LC0
        mov     rdi, rax
        call    strcmp
        test    eax, eax
        jne     .L2
        mov     DWORD PTR [rbp-4], 1
.L2:
        mov     eax, DWORD PTR [rbp-4]
        leave
        ret
```

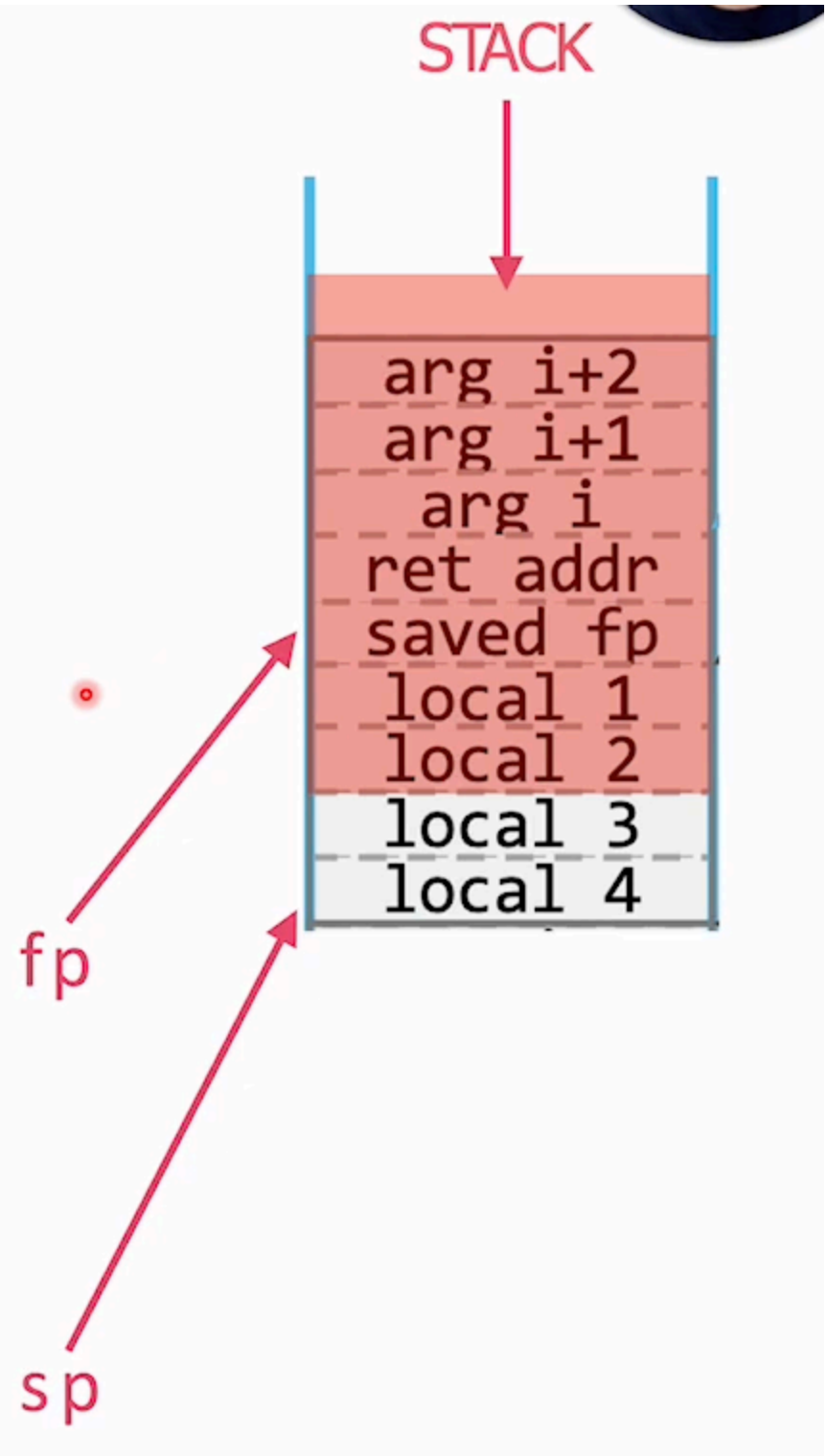
Call Frame

- what happens when a function is called?
  - allocate new frame for callee
  - caller pushes arguemnts/return address
  - calle:
    - pushes old frame pointer (fp) - fp points to bottom of frame of currently executing function
    - sets fp = sp (stack pointer) - sp points to top of stack
- what happens when function ios returned>?
  - callee pops local storage and sets sp = fp

- callee pops frame pointer
- callee pops return address and returns to next instruction in caller frame



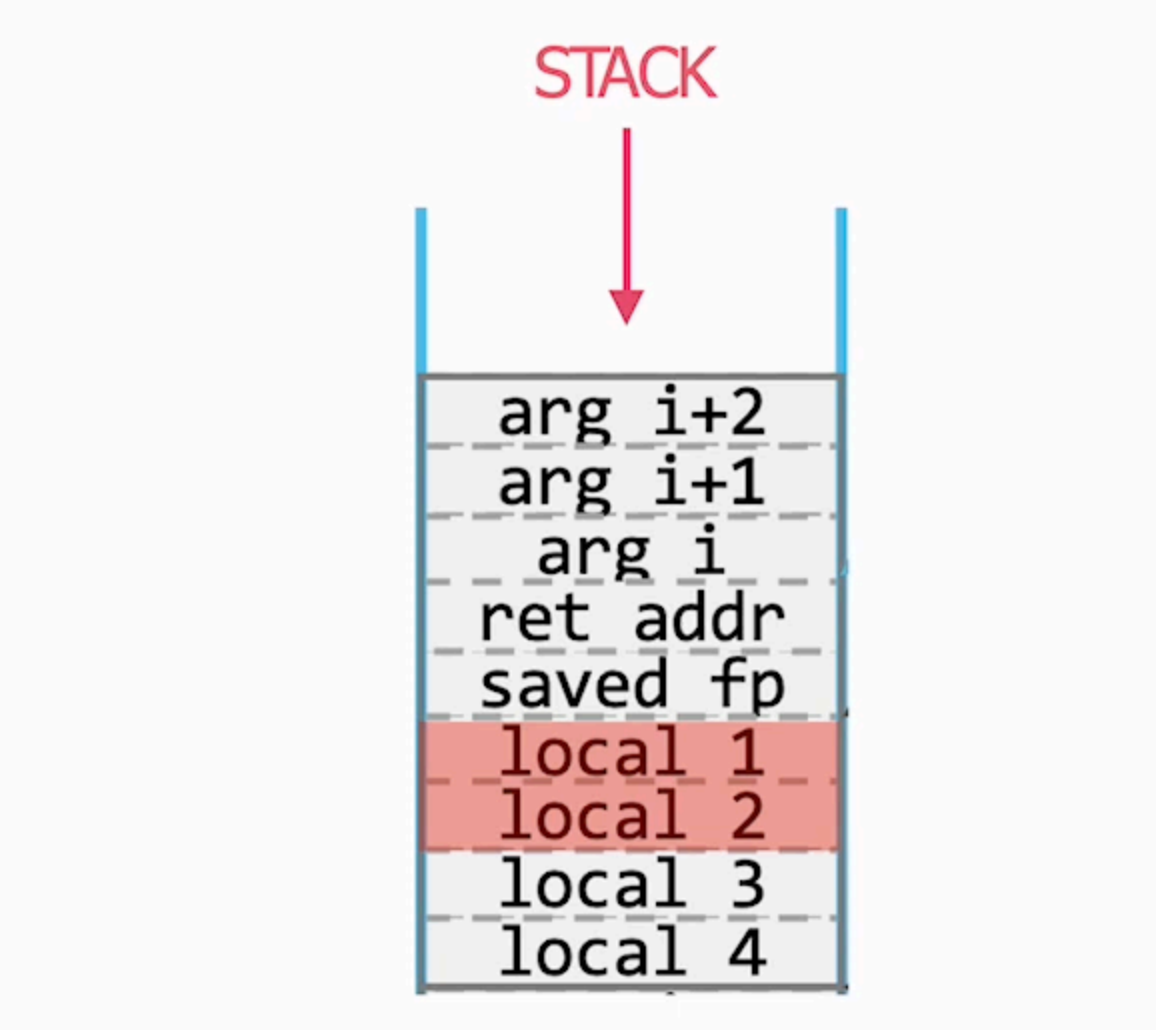
Smashing the Stack



- when overwriting a malicious value past the bounds of a local variable
- could transfer control to an address of your choice
- it can overwrite:
  - another local variable
  - saved fp
  - return address
  - function argument

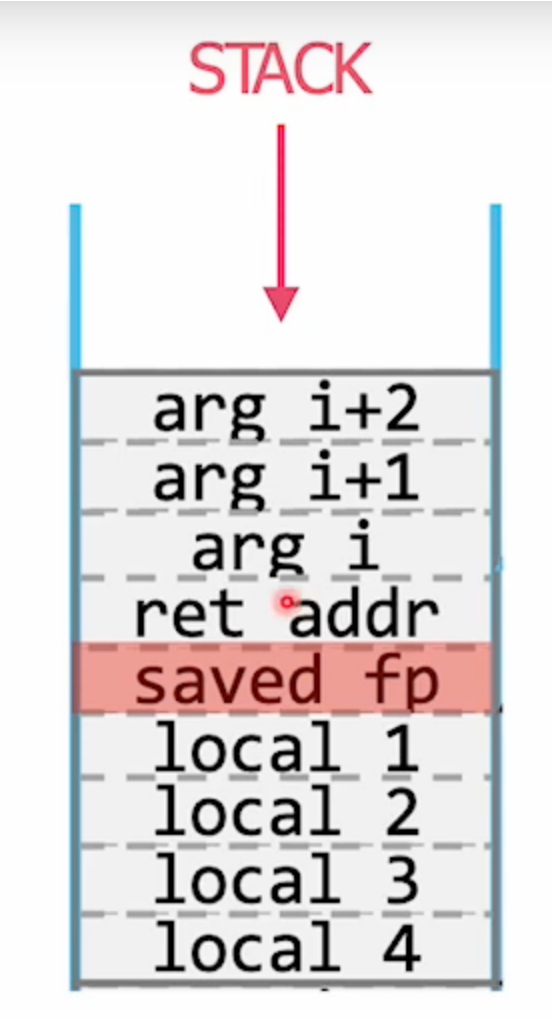
- deeper stack frames
- exception control data

Local Variables



- what happens when you overwrite another local variable
  - depends!
- it is bad if ovewriting:
  - results of a security check (isValid)
  - variable used in security check (buff_size)
  - data pointer (potential for further corruption)
  - function pointer (direct transfer of control when called)

Saved fp (frame pointer)

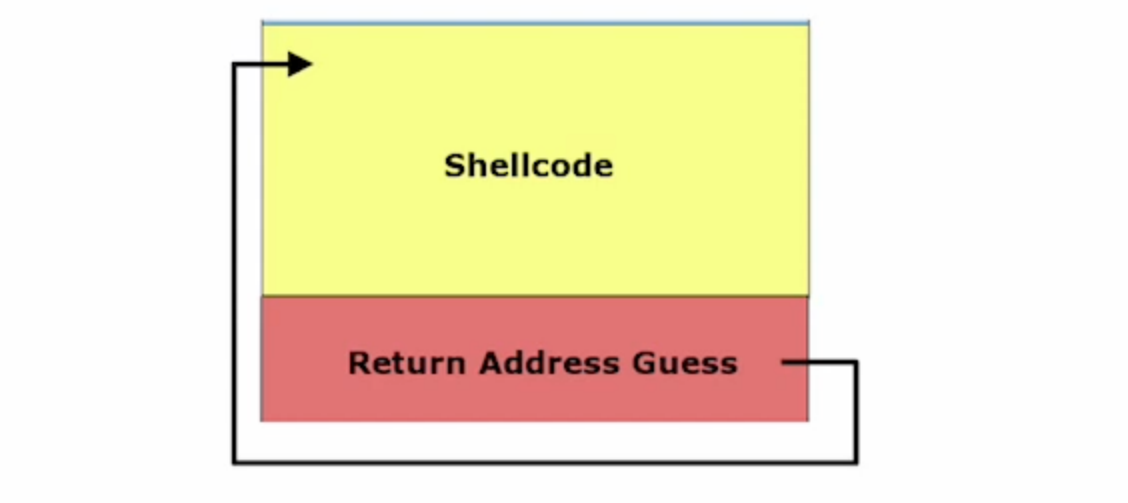


- overwriting a saved frame pointer
- when function returns, the stack moves to an attacker-supplied address -> complete control of execution
- even a single byte may be enough for this attack

Return Address

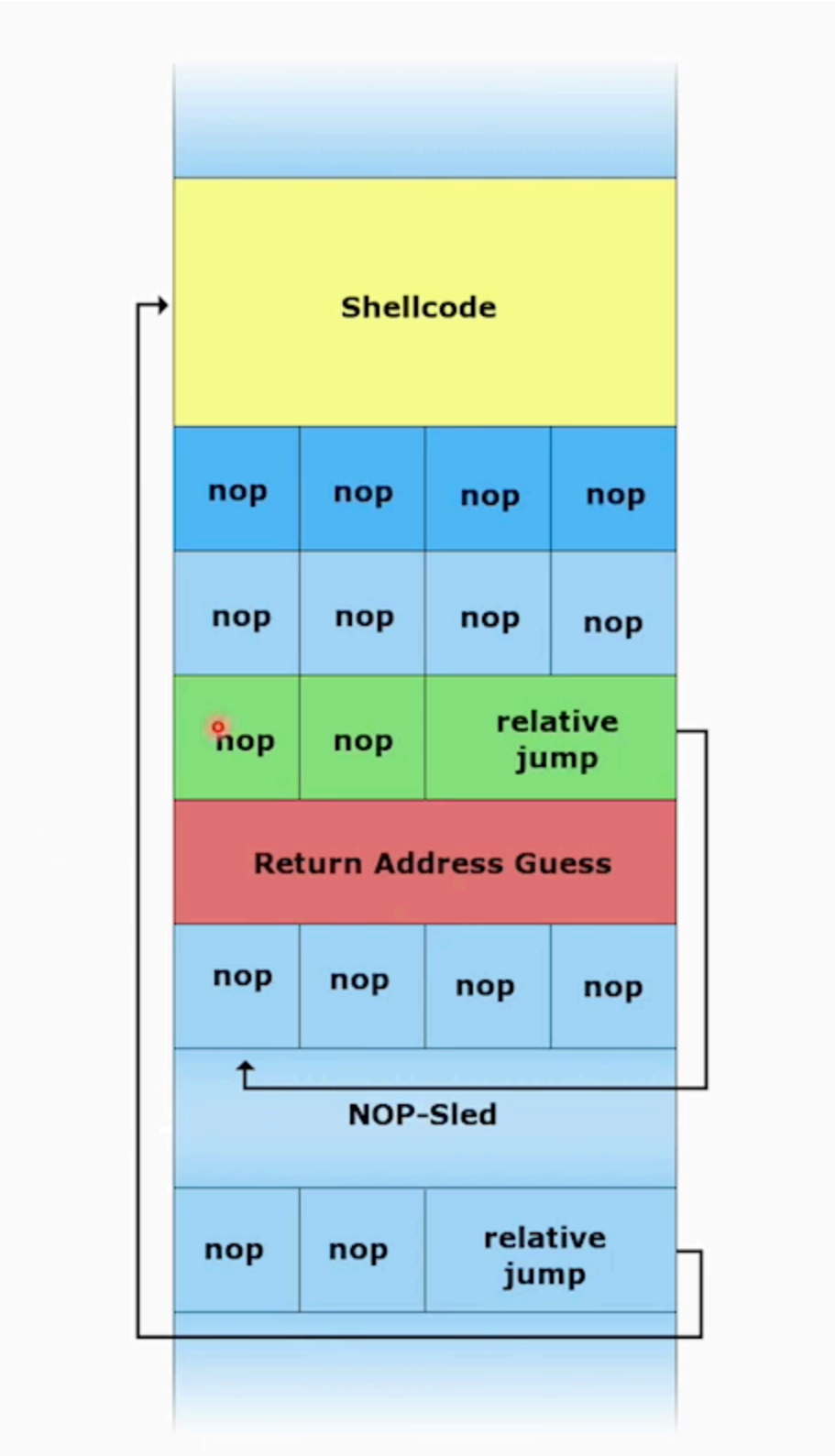
- terrible to overwrite the return address
- when function returns, control is transferred to an attacker-supplied address => complete control of arbitrary code execution (re-direct to their own code)
- called **return-oriented programming**

Shellcode



- attacker can do is launch the shell, as that allows them to execute arbitrary code (with higher privileges)
- payload is often called **shellcode**
- attacker ensures shellcode is somewhere in the stack before overwriting return address but they might not know exactly where it is

NOP slides



- attacker can rely on NOP ("no-op") instructions to create a **NOP sled** (or NOP slide)
- as long as attacker's guess lands somewhere in this sequence of NOPs they can jump at the end to the start of the shellcode as desired

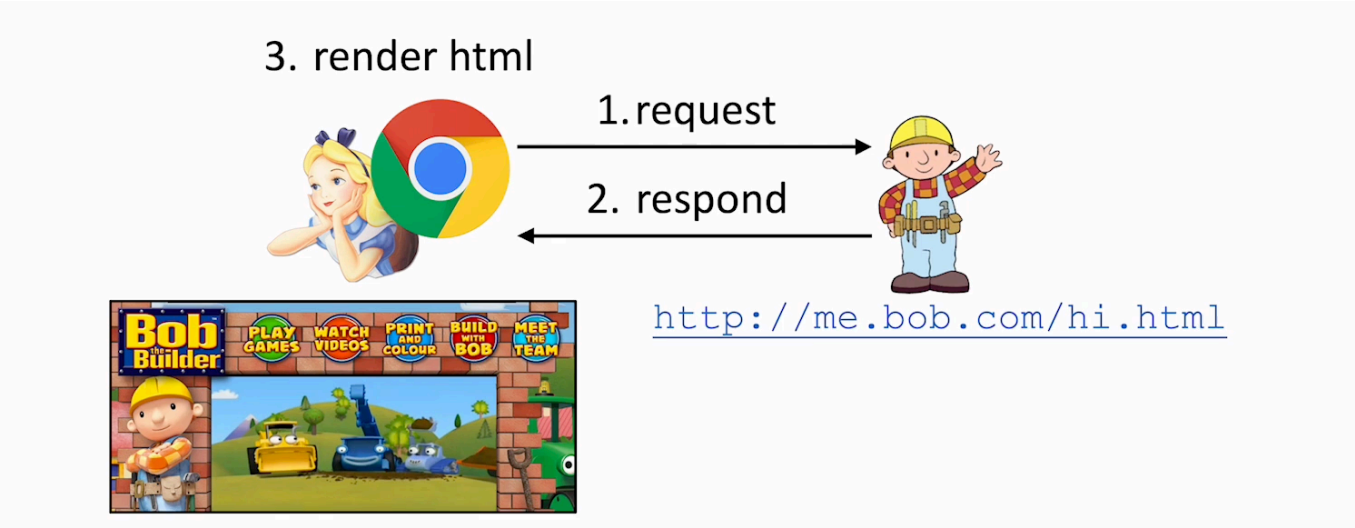
Mitigations

- think like an attacker

    - does the code check for bounds on memory address
    - is the test invoked along every path leading up to actual access (complete mediation)?
    - is the test correct? can the test itself be attacked?
- investigate security aspects of tools, frameworks, libraries, APIs that you use/uinderstnd hopw to use them safelty - defualt way of doing something is often insecure
- use `strlcpy` instead of `strcpy`
- lots of techniques on stack operations (not covered in module) - you can disable ASLR to exploit buffer overflows in C programs

Web security

Web Architecture



How is the request done?

- alice typed a url
- alice clicked a link
- alice reloaded a page
- web server responded with redirect
- web page embedded another page
- script within web page issued a request

How is the response returned?

- returns static file
- invokes a script and returns output
- invokes a plugin

Websites are **programs**: HTML + CSS + JavaScript + plugins

- html - text with markup/hyperlinks
- css - cascading stylesheet
- js - client side program

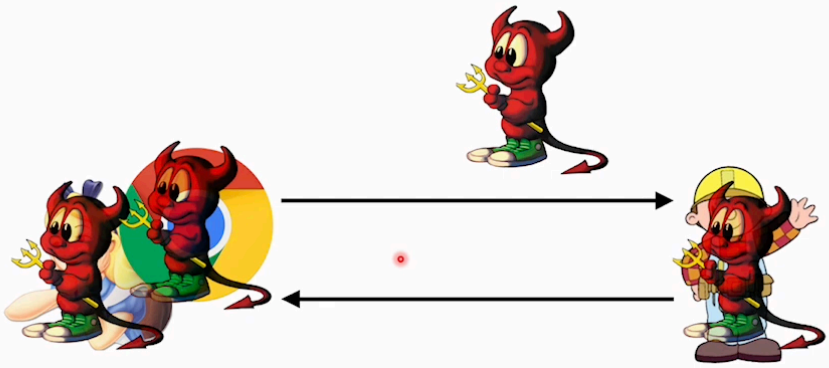
plugins e.g. Java, Flash, etc

- partially executed by client (HTML, JS, plugins, etc)
- partially executed by server (PHP, Ruby, SQL, etc.)

Web Server

- serves static content (HTML+CSS pages)
- generate dynamic content
  - cgi: php, python, etc
  - web server modules: rails, etc
  - database backend: SQL

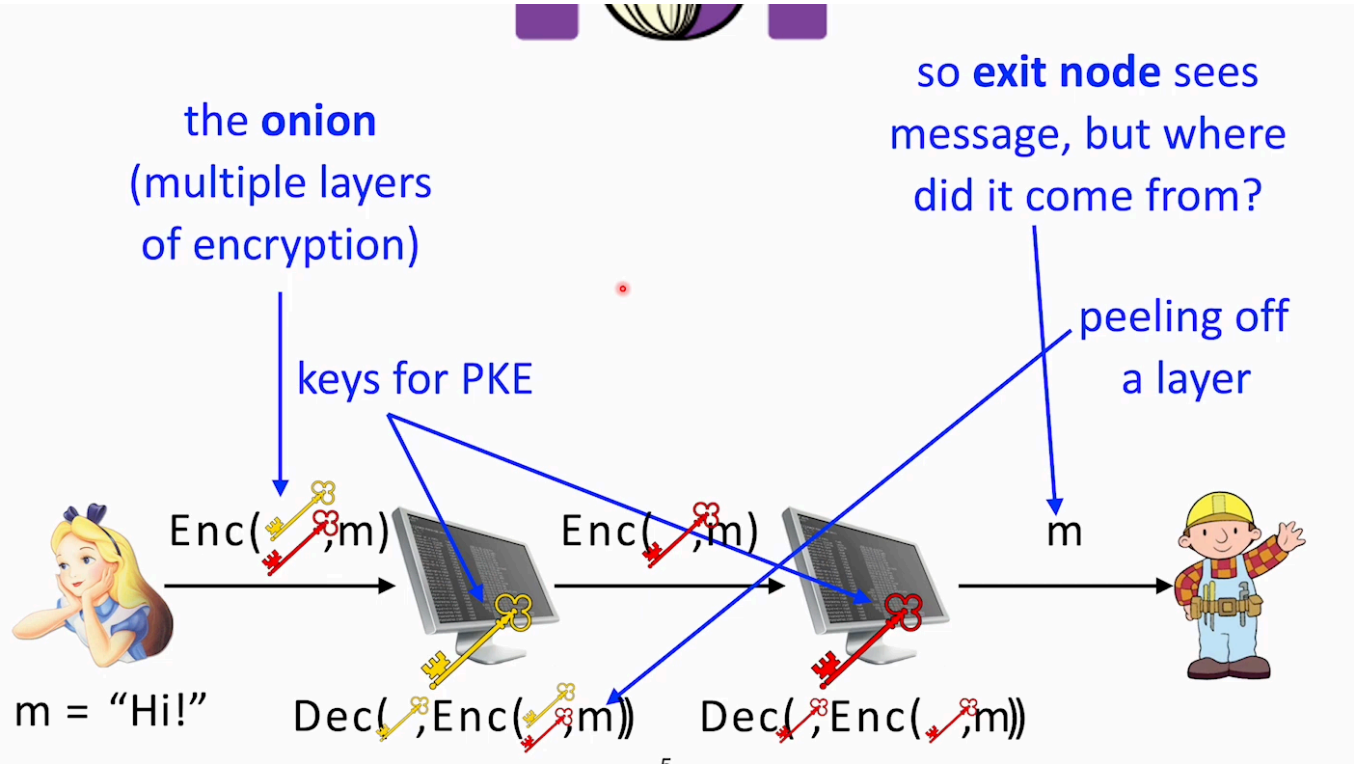
Threat Model



- Is the server trusted by the browser? or the user?
- Is the user trusted by the server? or the browser?
  Is the browser trusted by the user? or the server?
- Is there an eavesdropper spying on your web traffic?

Anonymity

TOR (The Onion Router)



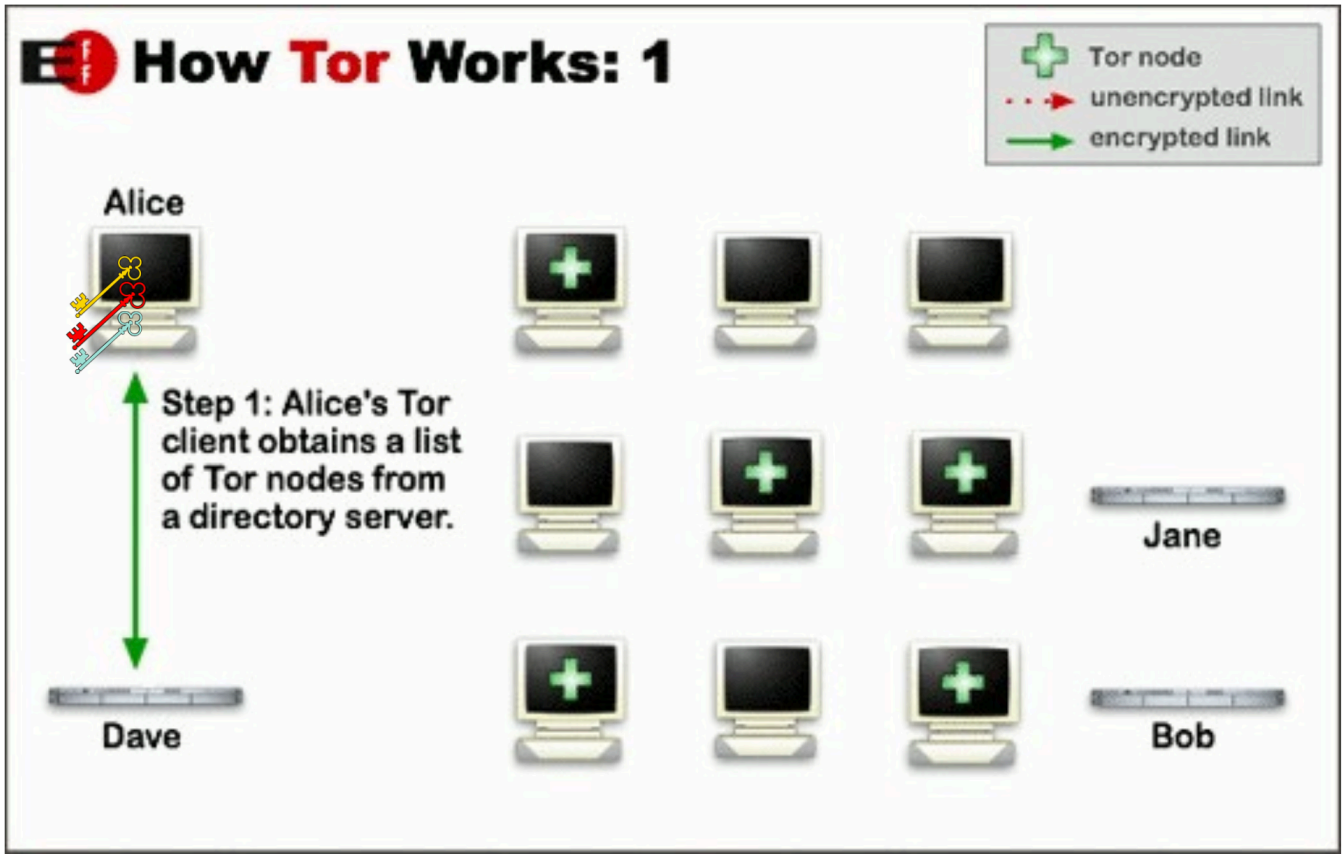
Standard way to provide anonymity for a user in the web

Have multiple layers of encryption where you peel one layer at a time to find where to route the message to

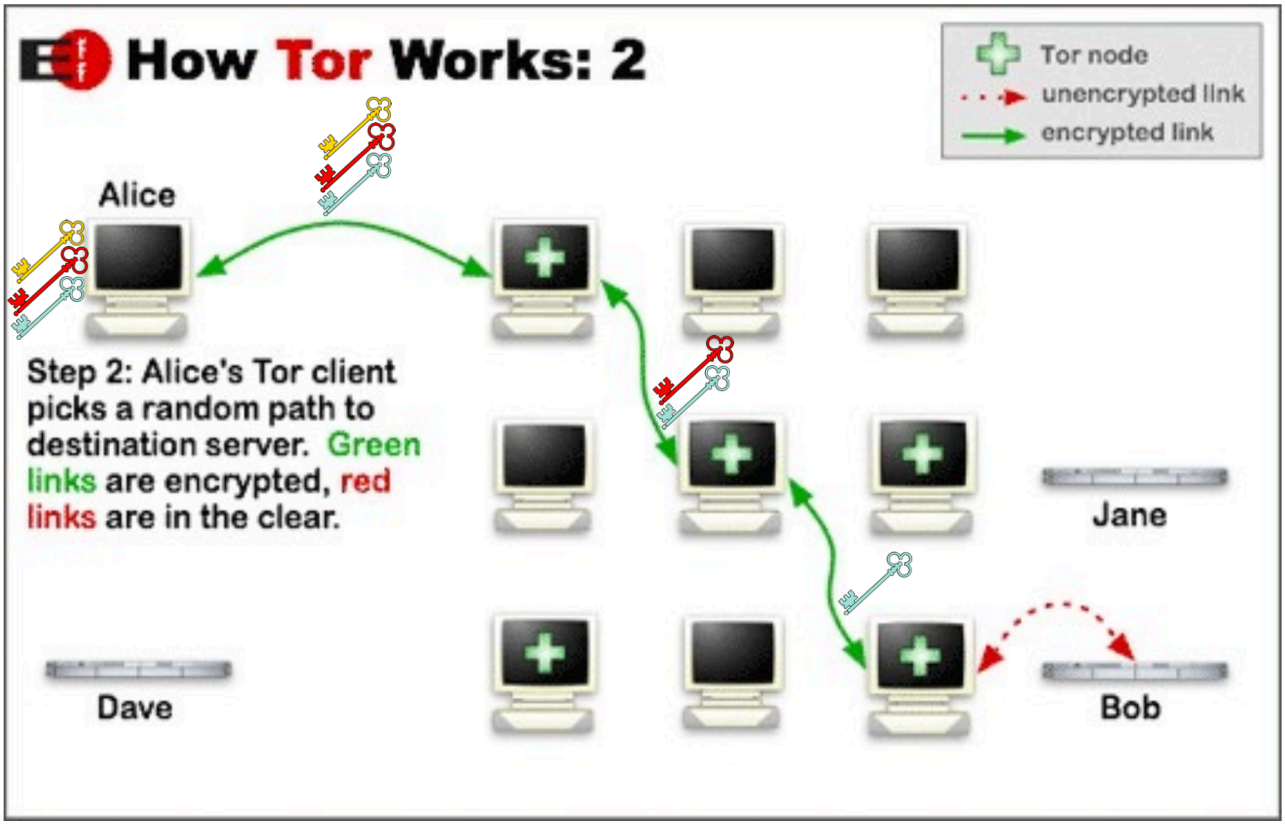
e.g.

"HI" gets encrypted through multiple layers of encryption (1st node peels off one layer, and so on)

steps:



there are public keys for the Tor nodes



ALice encrypted with 3 keys where as the message goes along the nodes, they get decrypted oine by one (peel layer of onion)

Last node (Exit node) will route the message to Bob (HTTPS can protect the message received to Bob)

How to ban TOR users from accessing my site?

- block exit nodes

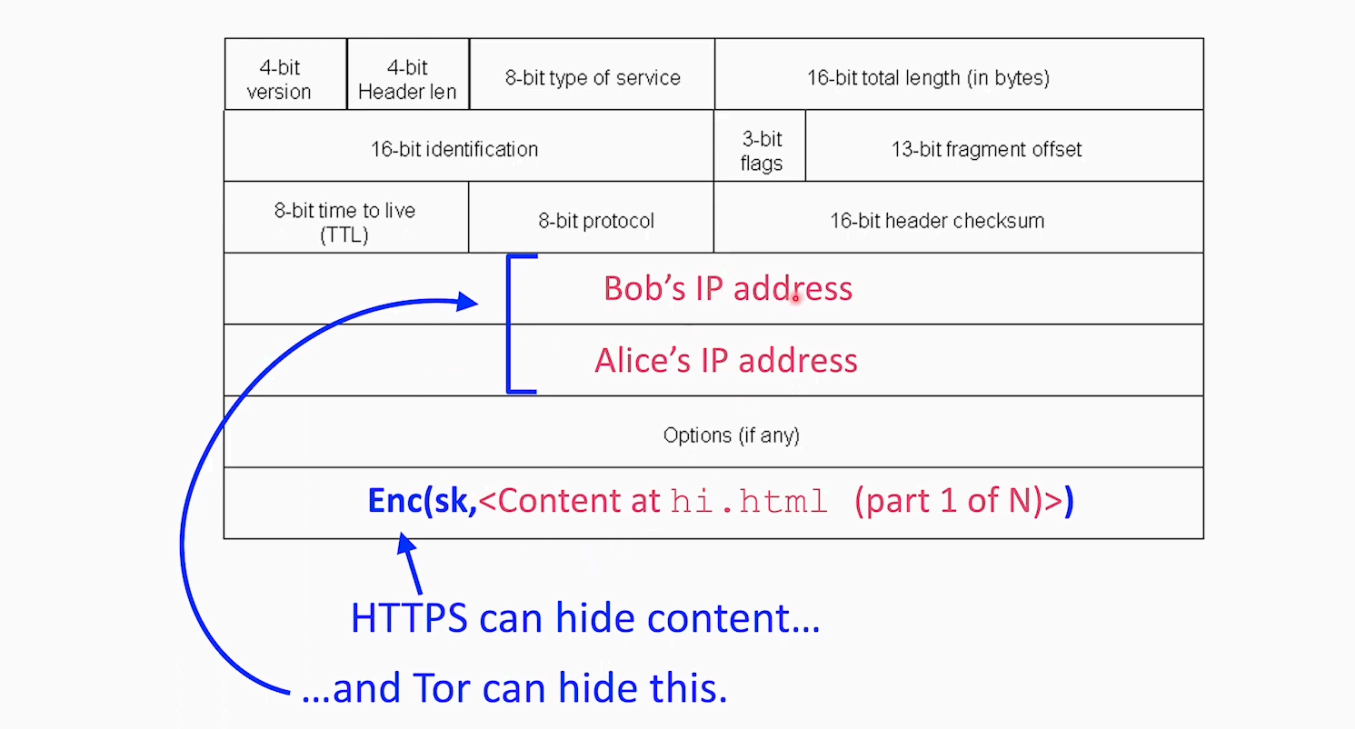
Does TOR also hide content of web traffic?

- no, it only just does routing. Use HTTPs to hide/receive content

Does TOR let me visit normal websites anonymously?

- no - hidden services exist only within Tor network

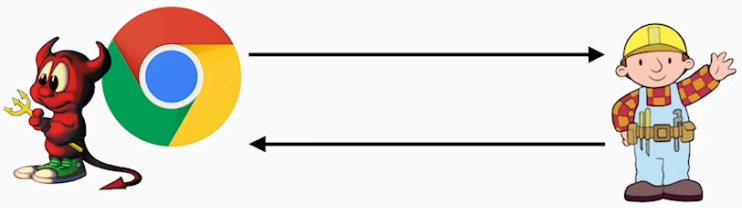
Encrypted web traffic



TOR hidden service (put website at TOR node - no one can take down the web server)

Attacks on Integrity

examples: sql injection, click fraud, xss/csrf



Is the user trusted by the server? or the browser?

- SQL injection
- Click fraud

SQL Injection

SQL - structured query language

- server-side applications generate SQL queries based on arbitrary user input
- care more about the content - a query is created based on user input (via SQL)

You can fool **sql syntax** to execute malicious queries (not supposed to run)

```
"user" = alice' ;--
query = "SELECT count(*) FROM users WHERE
user_name = 'alice' ;-- ..."

"user" = alice
"pass" = foo' OR 1=1 ;--
query = "SELECT count(*) FROM users WHERE
user_name = 'alice' AND user_pass = 'foo' OR 1=1;"
```

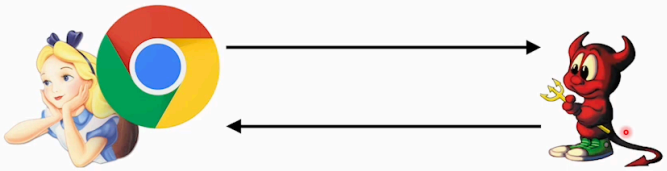
- 1=1 (will always evaluate true)
- condition clause is true - sidestep user/password authentication
- you can execute any sqwl command

Mitigations:

- solution? dont accept arbitrary user input
- parameterised queries
  - pre-compiled queries that separate commands from input
- input sanitisation

- make sure only safe input is accepted
- what is unsafe? single quote? dashes? - could be legit
- sanitise on client or server?
- use proper escaping/encoding?

Threat Model



Is the server trusted by the browser? or the user?

- Browser fingerprinting
- Forward secrecy / revocation
- Typosquatting / pharming
- Clickjacking

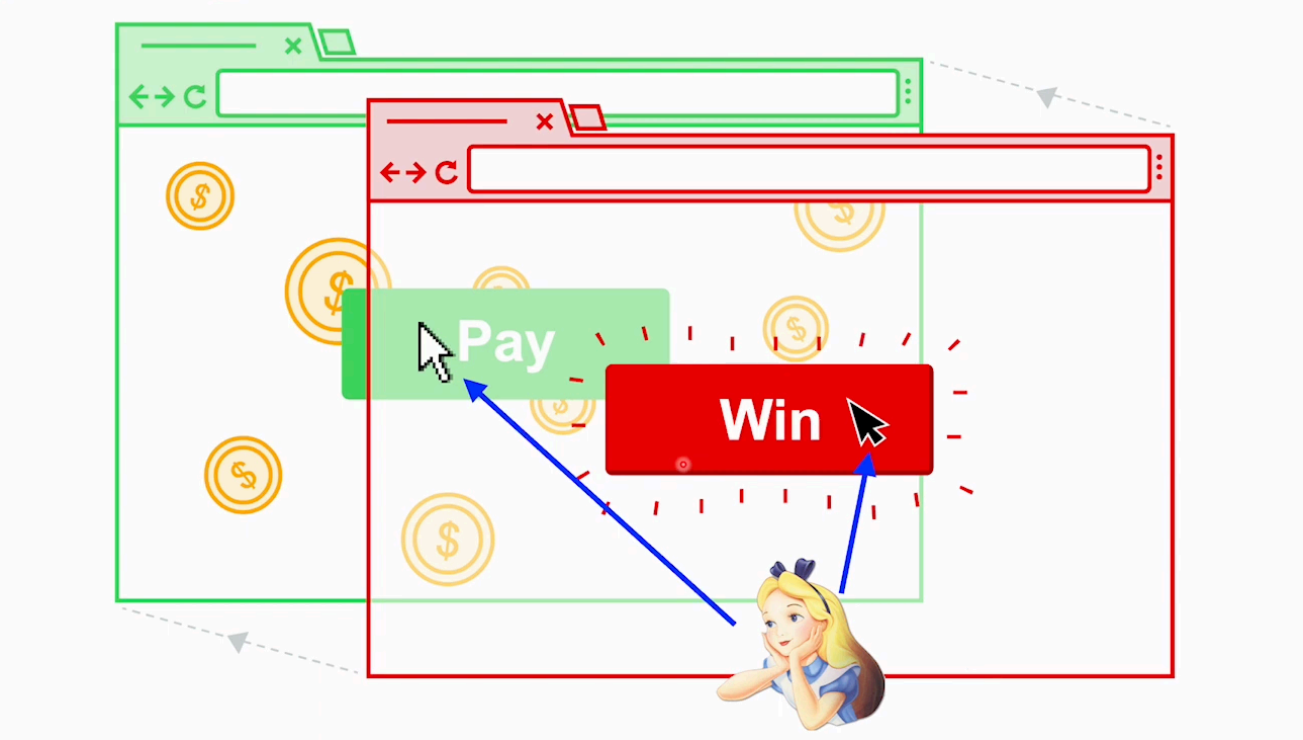
iframes

- content from one site embedded into another using **iframes**
- outer page can set width/height of frame
- only inner page can draw within its frame

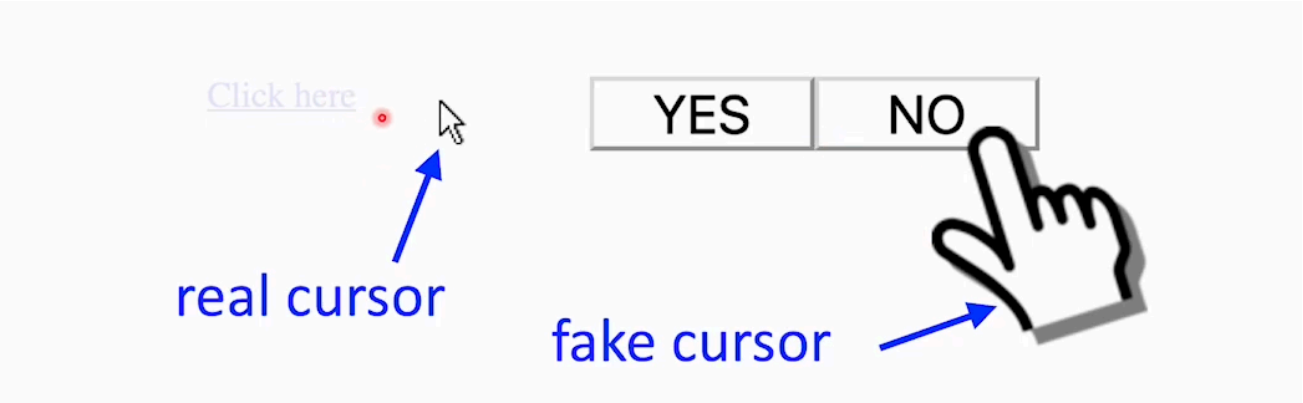
iframes can allow **clickjacking**
attacker site frames good site and covers part of it to look different/create unintended interactions

- **likejacking** - facebook users tricked to click like button

- often achieved by cursorjacking



cursorjacking:



XSS and CSRF

Web applications are session-based (stay logged in until you log out or enough time passes)

- using cookies

Cookies

Web server provides token in response that looks like:
`Set-Cookie: <cookie-name>=<cookie-value>`
attached to every future request sent to the server

examples:

- userid

- sessionid
- isauthenticated
- preferences
- shopping cart contents
- shopping cart prices

Types of cookies;
Session

- only exist during current browser session
- deleted when browser is shut down
- expiration property is not set

Persistent

- saved until some server-defined expiration

Threat model

- who is trusted and who is a potential attacker
- dopes web browser have to provide cookies
- how hard is it for user to modify their cookies

JavaScript

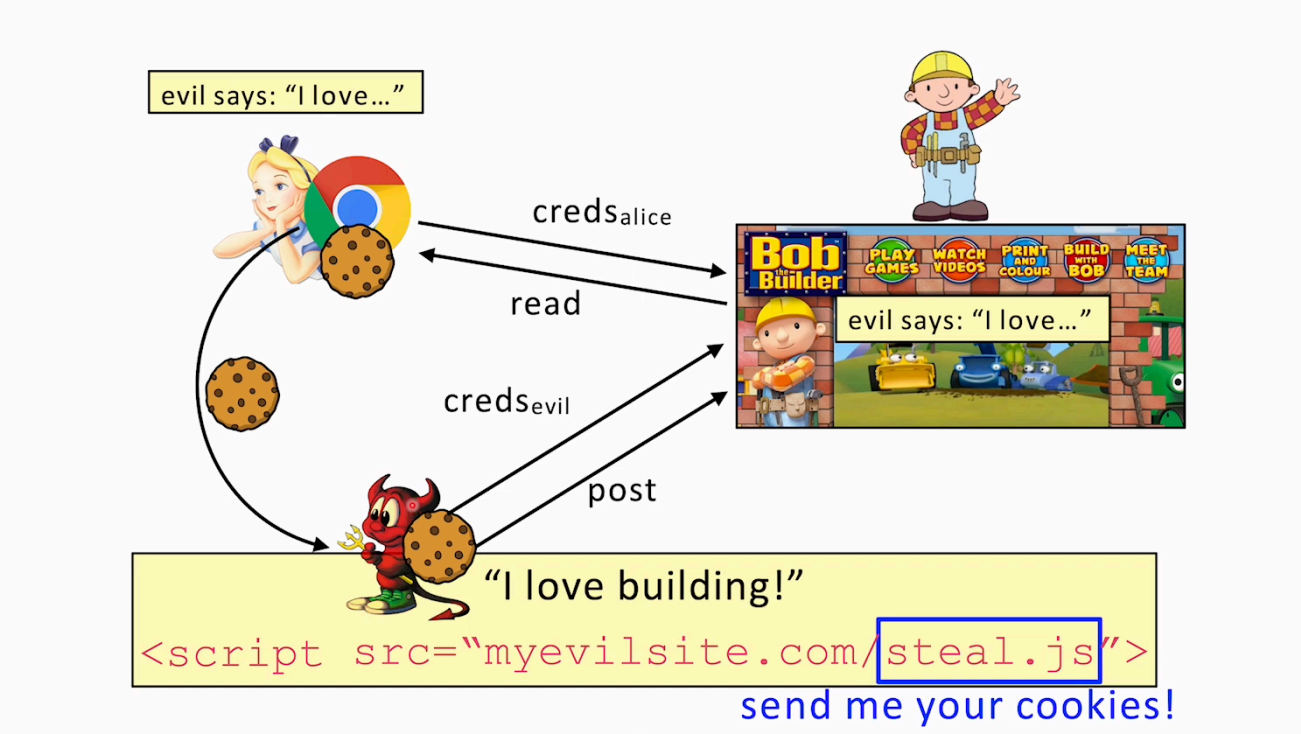
- designed as scripting language
- scripts embedded in web pages using `<script>` tag get the browser to execute some linked script
- computer is executing code (scripts) that it finds on the internet

JavaScript security

- script runs in "sandbox": no driect file access/restricted network access
- same origin policy: script can read properties of documents only from the same server, protocol and port
- same-origin policy doesn't apply to scripts loaded from arbitrary site, so the script runs as if it were loaded from site that provided the page
- server can explicitly tell brwoser other domains that are allowed using `Access-Control-Allow-Origin` header

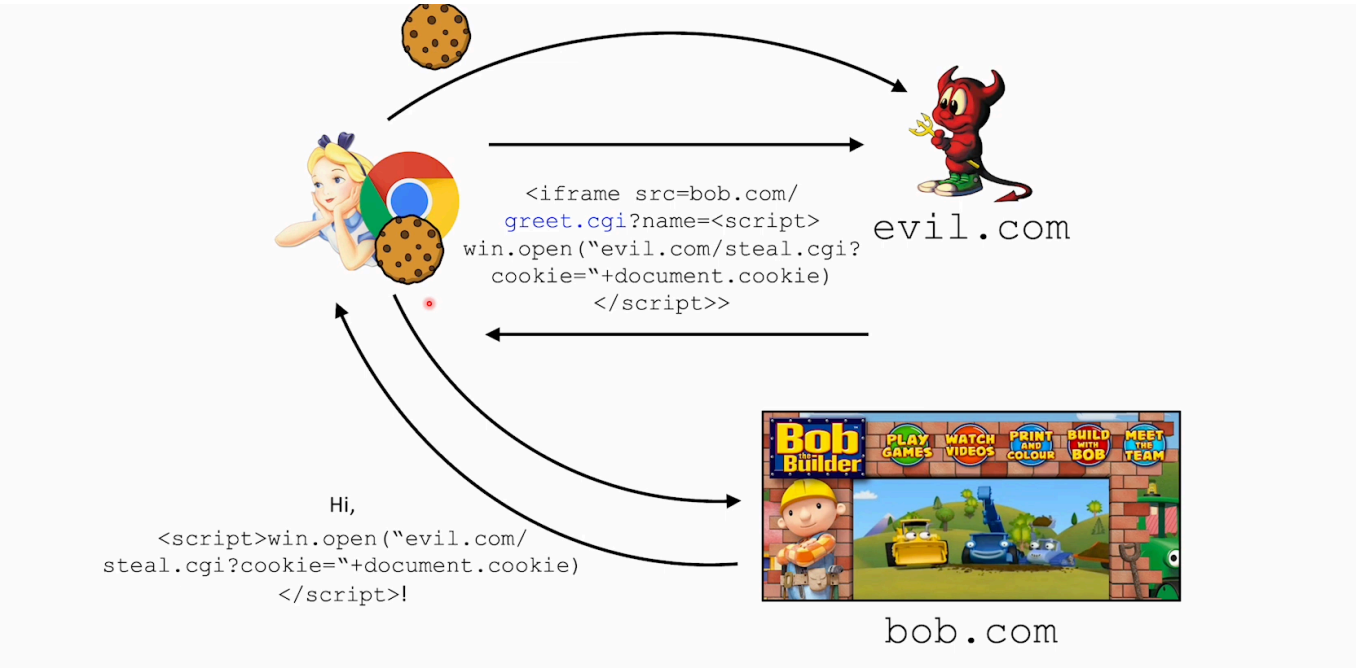
XSS (Cross-SIte Scripting)

- exploits the trust a user has in a site (form of **session hijacking**)
- allows an attacker to do anything a legitimate client-side script from that server could do
  - show false information
  - request sensitive information
  - trigger HTTP requests from the client
- how to prevent?
  - hard to prevent injection of scripts (not enough to block "<" or ">" or allow only simple HTML tags)
  - partial fix: `httpOnly` cookies cant be accessesd by script (still doesn't stop XSS attacks, but only stops cookie theft)



Adversary wants to steal the cookie and then can impersonate Alice
Alice can be tricked to send cookie to the adversary through a javascript script

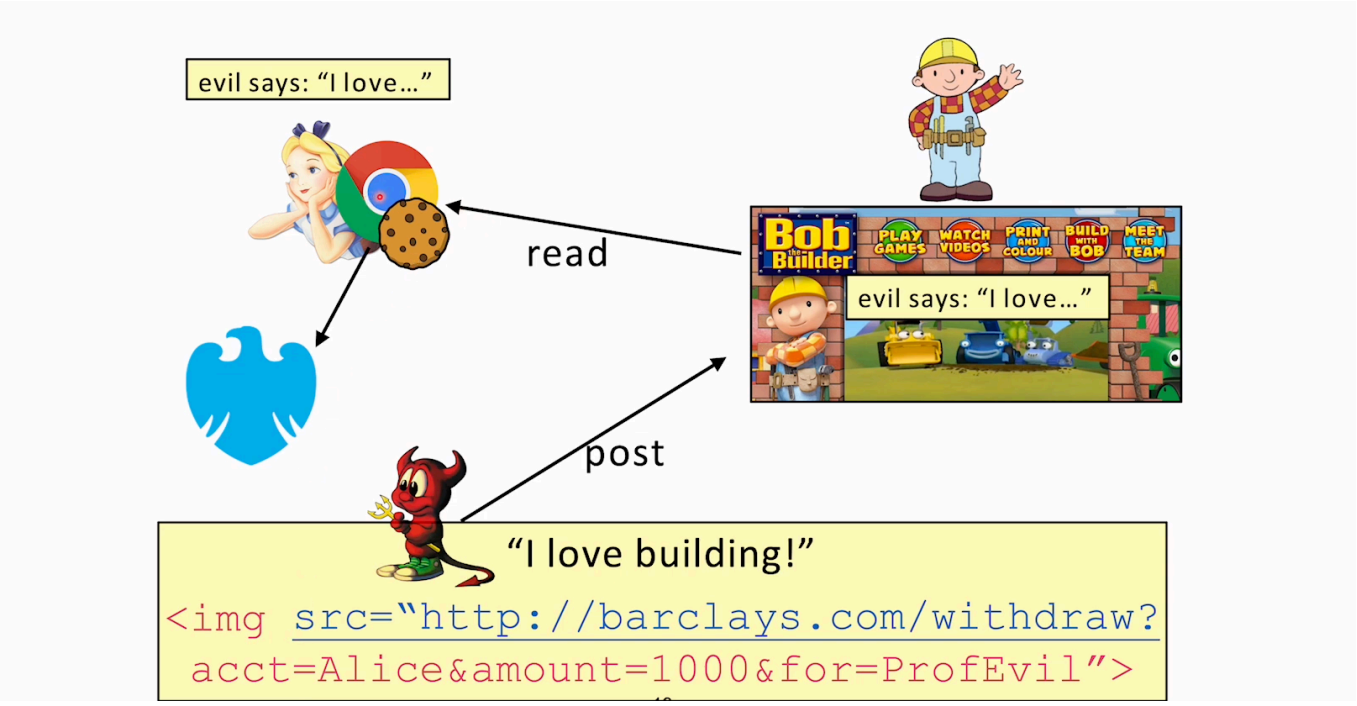
Reflected XSS



Script is triggered from `evil.com` which makes Alice to give her cookie to adversary

CSRF (Cross-Site Request Forgery)

Execute commands as if its coming from victims



If attacker controls website, it can embed image that loads script that triggers a comamnd

CSRF exploits the trust a site has in a browser (**confused deputy** attack)

When browser issues a `GET` request, it attaches all cookies it has from target site

the target sees the cookies but has no way of knowing the rqeuerst was really authorised by the (human) user

how to prevent?

- secret tokens visible only by same-origin content (client needs to include these tokens in state-altering requests)
- don't alter state based on `GET` requests
- same-origin cookies (Chrome)

XSS vs CSRF

XSS

- server-side vulnerability
- attacker injects a script into trusted website
- trusting browser executes attacker's script

CSRF

- server-side vulnerability
- attacker gets trusted browser to issue requests
- trusting website executes attacker's requests

How to mitigate?

For websites:

- use same-origin policy / content secuirity policy
- sanitise HTML
- require additional authentication

For users (partial mitigation as we still rely on browsers):

- don't run scripts!
- don't "stay signed in"