https://brown-csci1660.github.io

# CS1660: Intro to Computer Systems Security Spring 2025

## Lecture 8: Authentication


Co-Instructor: **Nikos Triandopoulos**


February 20, 2025

BROWN

# CS1660: Announcements

- Course updates
  - Project 1 is due today – let us know if you have any issues (need extension, etc.)
  - Homework 1 is due in a week from today (Thu, Feb 27)
  - Project 2, new dates: Out Feb 25 – Due Mar 11
  - Where we are
    - **Part I: Crypto** – wrap up today, transitioning to Web security…
    - **Part II: Web**
    - Part III: OS
    - Part IV: Network
    - Part V: Extras

# Today

- Cryptography
  - Wrap up
- Authentication
  - User
  - System
  - Data

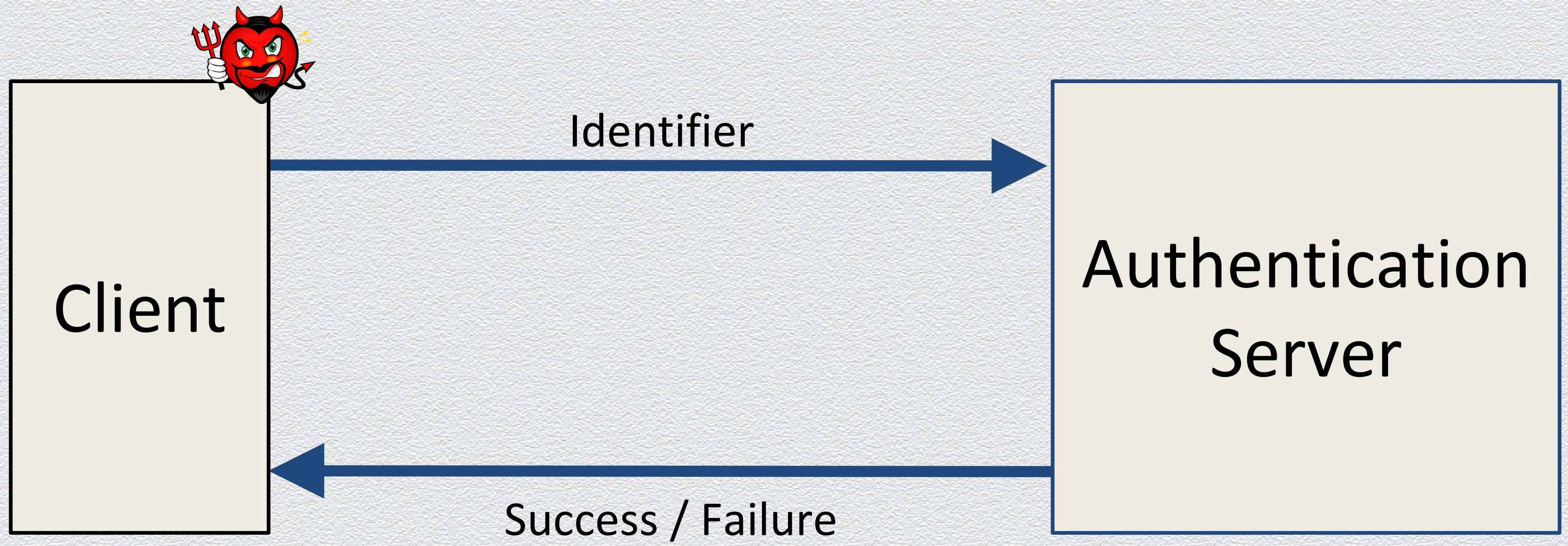
# Crypto recap through Discrepancies…

# Discrepancies

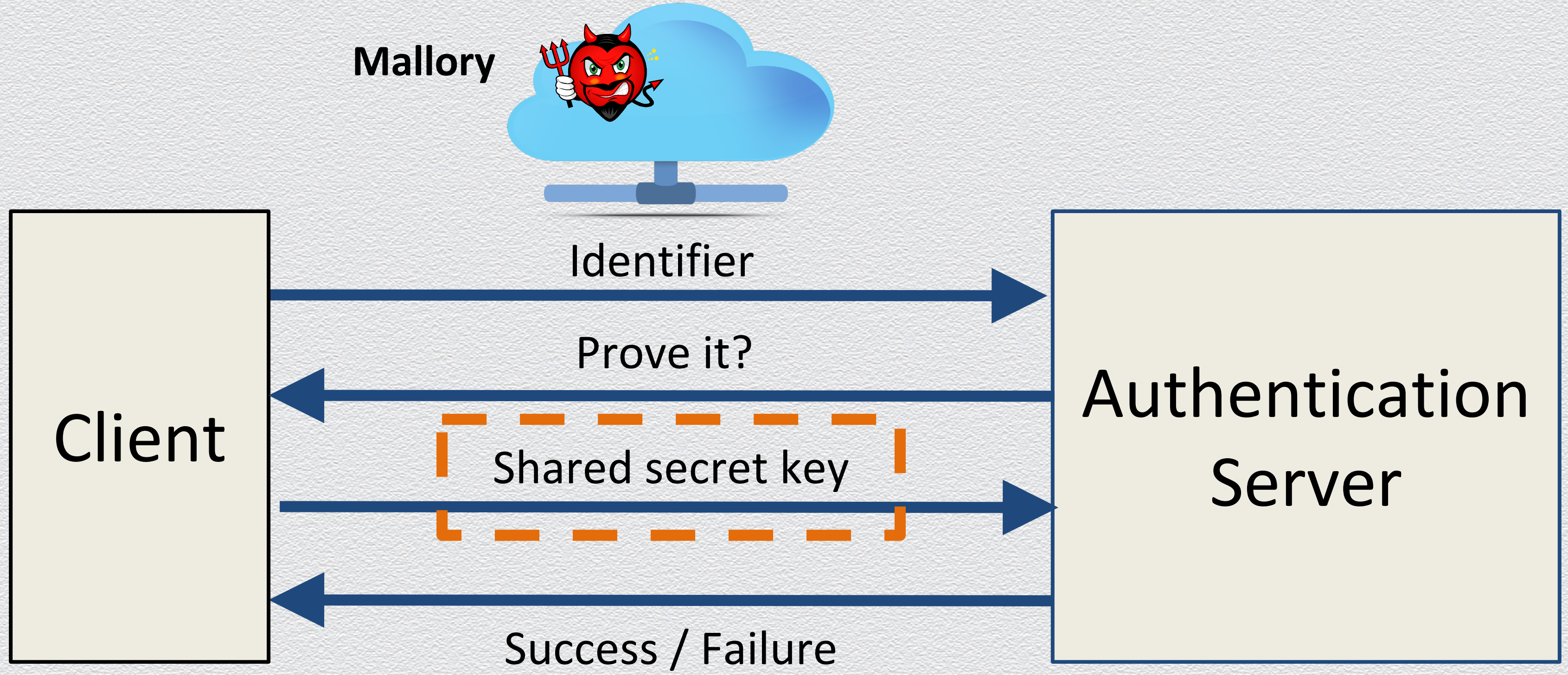
- Security Vs. cryptography
- Guarantees Vs. threat model
- Confidentiality Vs. integrity
- Prevention Vs. detection
- Old Vs. modern cryptography
- Perfect Vs. computational security
- Modelled Vs. practical attacker
- Crypto Vs. non-crypto security
- Truly Vs. pseudo random
- Secret Vs. public
- Theory Vs. practice
- Ideal model Vs. implementation
- Open Vs. closed design
- Symmetric Vs. asymmetric crypto
- Block Vs. all-length designs
- Data Vs. user authentication
- Set-up Vs. real-world assumptions
- Good hygiene Vs. arbitrary practices
- Random Vs. non-random

# Authentication protocols

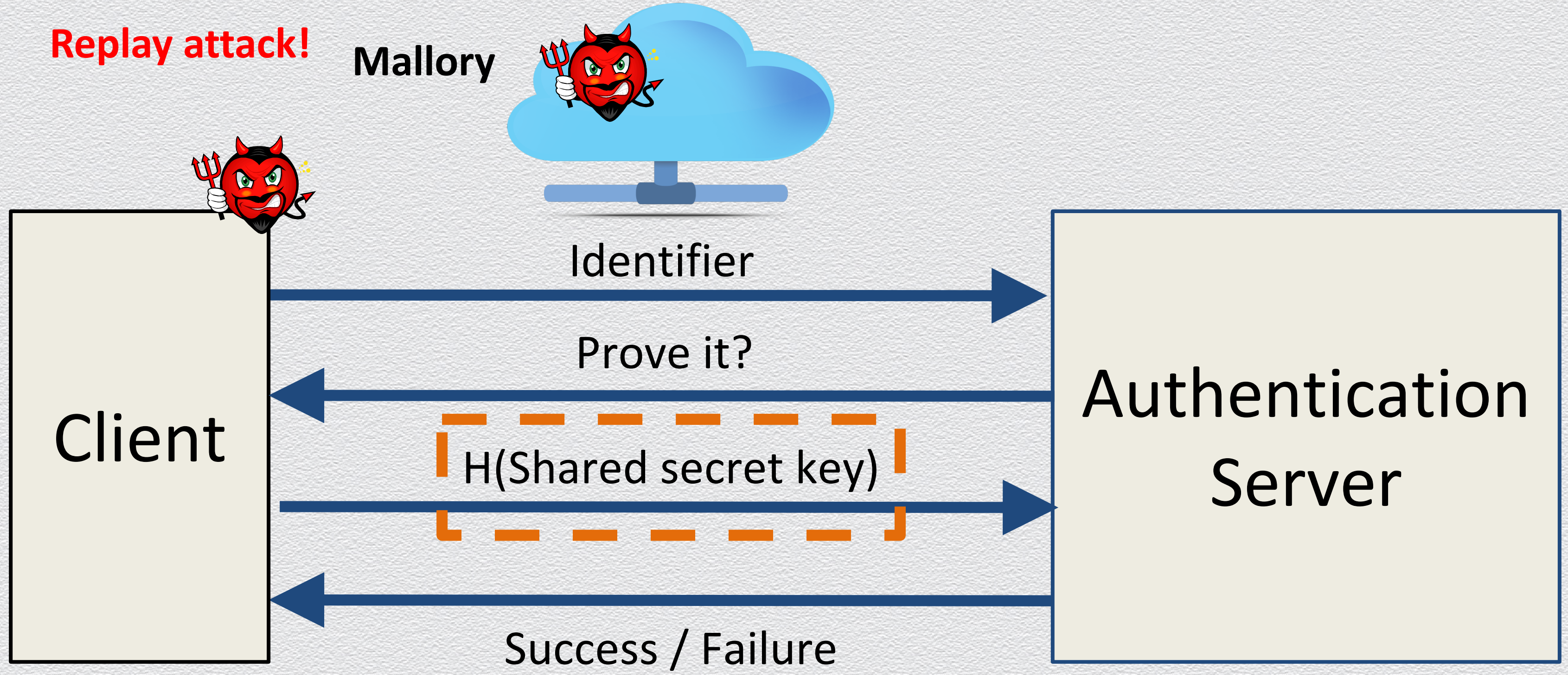
# How to authenticate two systems?



# But…


Mallory
Identifier
Prove it?
Shared secret key
Success / Failure
Client
Authentication Server

# Even better method…



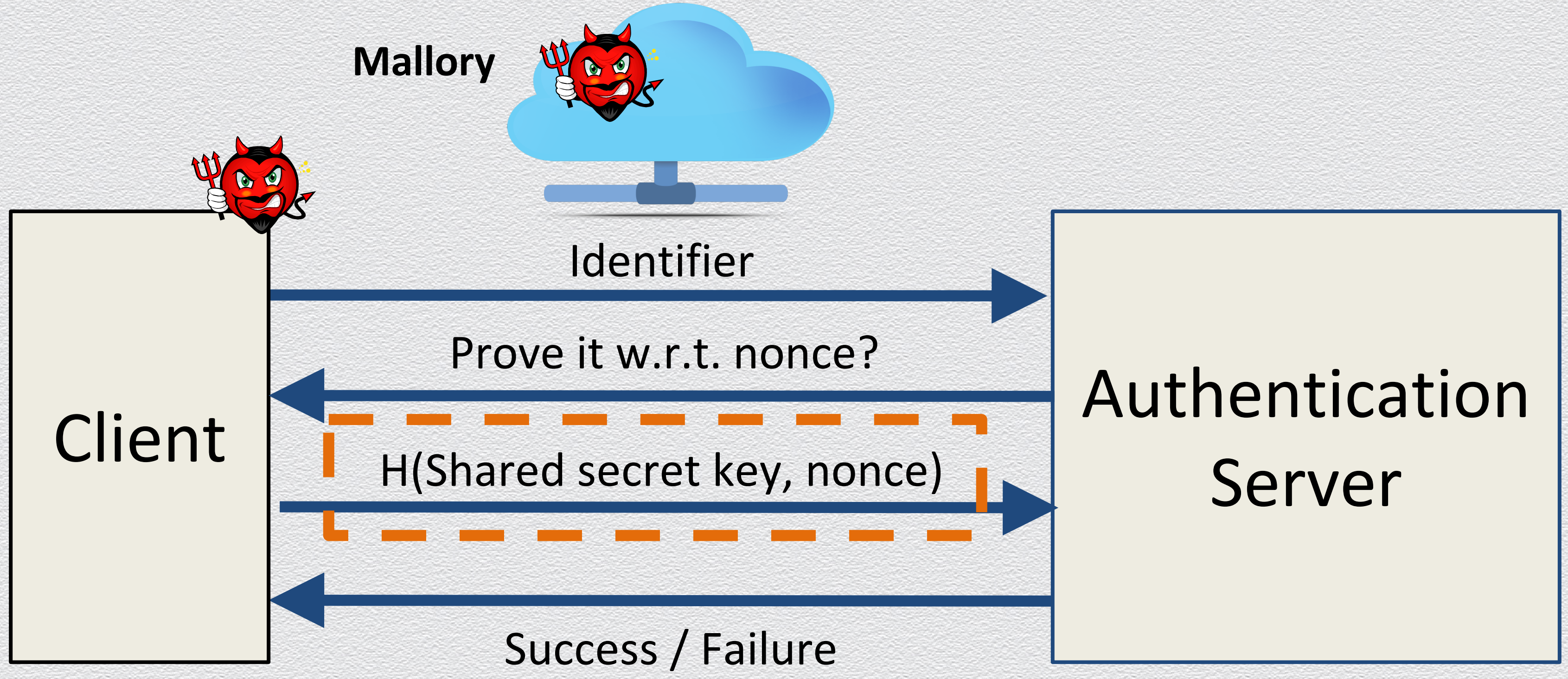
# Challenge-response

- Use **challenge-response**, to prevent replay attack
  - Goal is to avoid the reuse of the same credential
- Suppose Client wants to authenticate Server
  - **Challenge** sent from Server to Client
- Challenge is chosen so that…
  - Replay is not possible
  - Only Client can provide the correct **Response**
  - Server can verify the response

# Nonces

- To ensure "freshness", can employ a **nonce**
  - Nonce == **n**umber used **once**
- What to use for nonces?
  - A **unique** random string
- What should the Client do with the nonce?
  - Transform the nonce using the shared secret
- How can the Server verify the response?
  - Server knows the shared secret and the nonce, so can check if the response is correct

# Challenge-Response authentication method



# Authentication protocols

- Challenge response mainly relies on nonce
- What if nonce wasn't random?
- Harder to authenticate humans, more on that later…

# Summary of message-authentication crypto tools

| | Hash (SHA2-256) | MAC | Digital signature |
|---|---|---|---|
| Integrity | Yes | Yes | Yes |
| Authentication | No | Yes | Yes |
| Non-repudiation | No | No | Yes |
| Crypto system | None | Symmetric (AES) | Asymmetric (e.g., RSA) |

# Entropy

- Amount of uncertainty in a situation

- Fair Coin Flip
  - Maximum uncertainty
- Biased Coin Flip
  - More bias $\rightarrow$ Less uncertainty

# Entropy (cont.)

- Computers need a source of uncertainty (entropy) to generate random numbers.
  - Cryptographic keys.
  - Protocols that need coin flips.
- Which are sources of entropy in a computer?
  - Mouse and keyboard movements or thermal noise of processor.
  - Unix like operating systems use dev/random and dev/urandom as randomness collector

# Random numbers in practice

- We need random numbers but...

  "*Anyone who considers arithmetical methods of producing random numbers is, of course, in a state of sin.*" - John von Neumann

- Bootup state is predictable and entropy from the environment may be limited:
  - Temperature is relatively stable
  - Oftentimes the mouse/keyboard motions are predictable
- Routers often use network traffic
  - Eavesdroppers.
- Electromagnetic noise from an antenna outside of a building
- Radioactive decay of a 'pellet' of uranium
- Lava lamps…

# Lava lamps

- Cloudflare company uses lava lamps as an entropy source

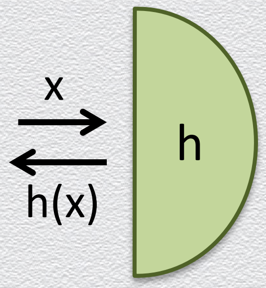
# Provable security: Idealized models

- challenge in proving security of scheme S that employs scheme S'
  - no reasonable assumption on S' or $\mathcal{A}$ can provide a security proof for S
- naïve approach: look for other schemes or use scheme S (if S' looks "secure")
- middle-ground approach: fully rigorous proof Vs. heuristic proofs
  - employ **idealized** models that **impose** assumptions on S', $\mathcal{A}$
  - formally prove security of S in this idealized model
  - better than nothing…
- <u>canonical example</u>: employ the **random-oracle model** when using hashing
  - a cryptographic hash function h is treated as a **truly random** function

# The random-oracle model

treats a cryptographic hash function h as a “black box” realizing a **random** function

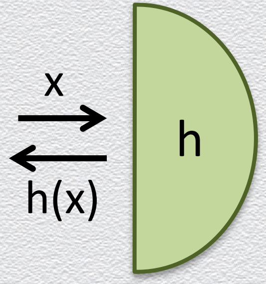
- models h as a “secret service” that is publicly available for querying
  - anyone can provide input x and get output h(x)
  - nobody knows the exact functionality of the “box”
  - queries are assumed to be private




- interpretation of internal processing
  - if query x is new, then record and return a **random** value h(x) in the hash range
  - otherwise, answer **consistently** with previous queries on x

# Using a random oracle h: Properties

- models h as a “secret service” that is publicly available for querying
  - **black-box access**: information leaks only via its API
  - **consistent** & **private querying**
  - **random hashing**
- in proofs by reduction (reduction $\mathcal{A}'$ using adversary $\mathcal{A}$)
  - probability is taken (also) over random choice of uniform h
  - in simulating oracle h (accessed by $\mathcal{A}$) $\mathcal{A}'$ can exploit the above properties
    - if x has not been queried before, h(x) is uniform (cf. PRG value G(x))
    - if $\mathcal{A}$ queries h on x, $\mathcal{A}'$ learns x (**extractability**)
    - $\mathcal{A}'$ can select answer h(x) to query x as long as it’s uniform (**programmability**)



# Recall: PRF – security

b = 0 when $\mathcal{D}$ thinks that its oracle is f()

b = 1 when $\mathcal{D}$ thinks that its oracle is $F_k()$

security parameter $1^n$


**case 0** $\mathcal{T}$

O() is uniform f

**case 1**

O() is $F_k()$, $k \in \{0,1\}^n$

**Y = random string**

X

Y

**Y= pseudorandom string $F_k(X)$**

$b \in \{0,1\}$

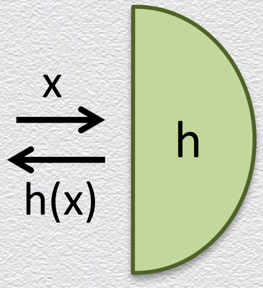
$\mathcal{D}^{O()}$

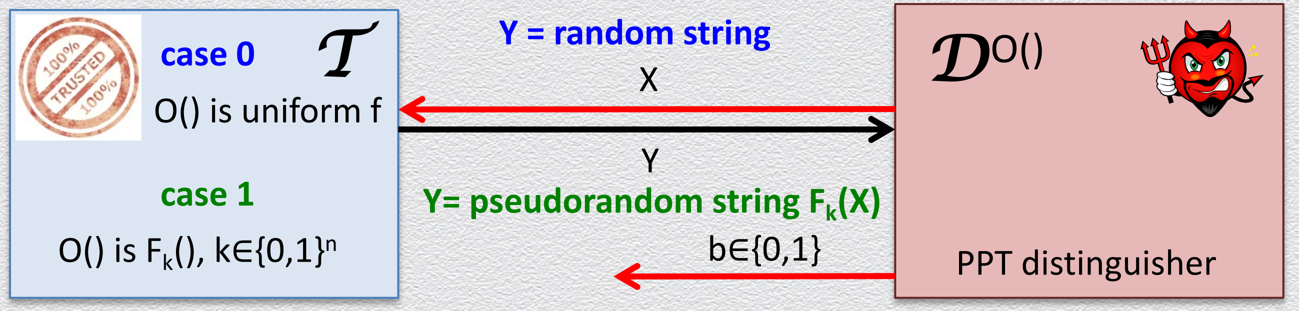
PPT distinguisher

$$| \Pr[ \mathcal{D}^{F(k,\cdot)}(1^n) = 1] - \Pr[ \mathcal{D}^{f()}(1^n) = 1] | \le negl(n)$$

$\mathcal{D}$ behaves the same no matter what its oracle is!

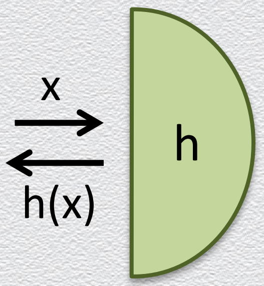
# Random-oracle model Vs. PRF




- random-oracle model
  - models publicly-known & deterministic cryptographic hashing
  - used as black box in constructions (& analysis)
  - in practice, instantiated by a concrete scheme
- PRF
  - models keyed functions that produce pseudorandom values if keys are secret
  - oracle access to a uniform f is used as a means to define security of PRFs
  - PRFs are generally not random oracles

# Power of random oracles




consider a random oracle h

- h can be used as a PRG (assuming h expands its input)
  - | Pr[ D(h(s)) = 1] – Pr[ D(r) = 1] | $\leq$ negl(n)
  - querying for h(s) happens with negligible probability
- h is a CR hash function (assuming h compressed its input)
  - why?
- h can provide a PRF (assuming inputs and outputs of 2n and n, respectively)
  - $F_k(x) = h(k||x)$
  - | Pr[ $D^{h(),F(k,\ )}(1^n) = 1$] – Pr[ $D^{h(),f()}(1^n) = 1$] | $\leq$ negl(n)
  - why?

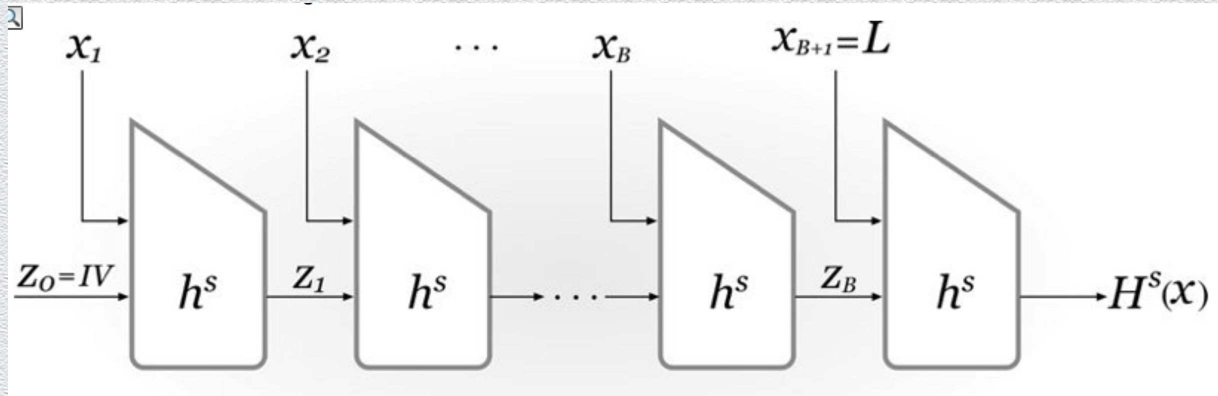
# Random-oracle methodology

1. design & analyze using random oracle h; 2. instantiate h with specific function h’

- how sound is such an approach? on-going debate in cryptographic community
- pros (proof in random-oracle model better than no proof at all)
  - leads to significantly **more efficient** (thus practical) schemes
  - design is **sound**, subject to limitations in instantiating h to h’
  - at present, only **contrived** attacks against schemes proved in this model are known
- cons (proofs in the standard model are preferable)
  - random oracles **may not exist** (cannot deterministically realize random functions)
  - real-life $\mathcal{A}$s see the code of h’ (e.g., may find a shortcut for some hash values)
  - can construct scheme S, s.t. S is proven secure using h, but is insecure using h’
  - note: “h’ is CR” Vs. “h’ is a random oracle”

# Constructing hash functions in practice

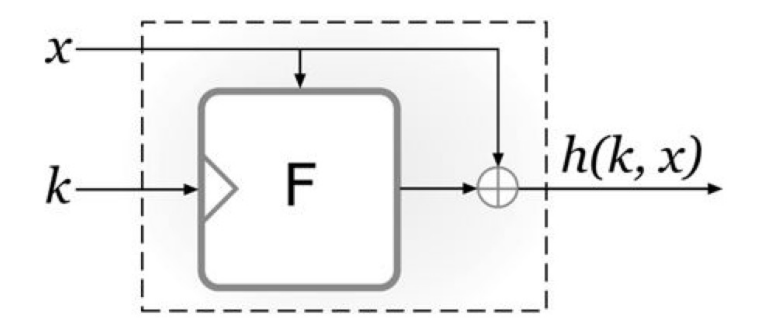
- typically, using the Merkle-Damgård transform




- (this precludes practical schemes being random oracles!)
- reduces problem to design of CR compression functions
- generic PRF-based compression schemes exist

# The Davies-Meyer scheme

- assume PRF w/ key length n & block length $l$
- define h: $\{0,1\}^{n+l} \rightarrow \{0,1\}^{l}$ as **h(k||x) = $F_k$(x) XOR x**
- h is CR, if F is an **ideal cipher**
  - idealized model that treats a PRF as a **random keyed permutation**
  - stronger than random oracle
  - some known block ciphers e.g., DES and triple-DES, are known not to be ideal ciphers!

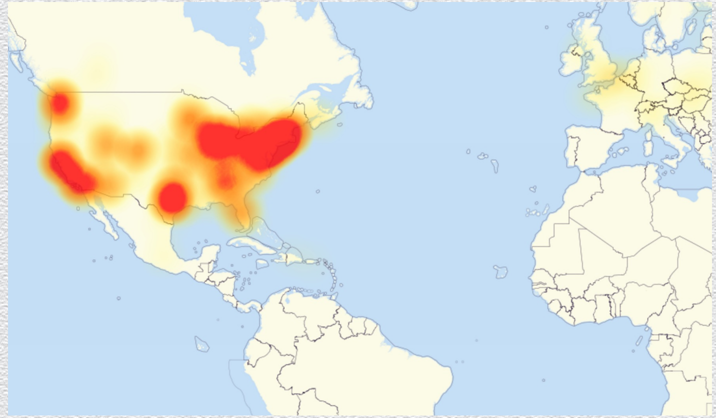


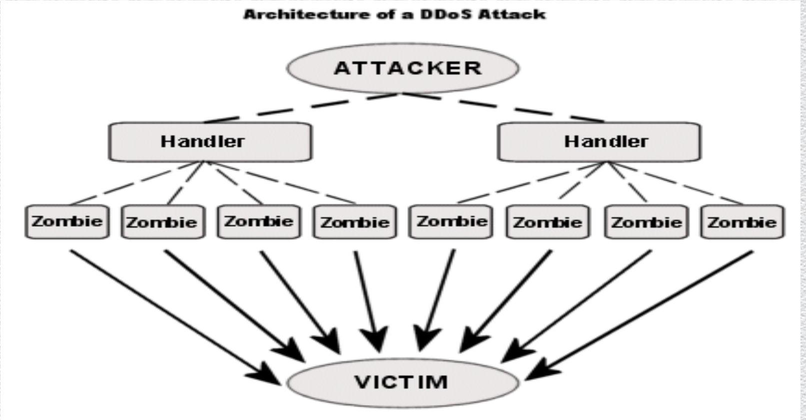
# The Dyn DDoS attack

# It’s unfair! – I had no class but couldn’t watch my Netflix series!

On October 21, 2016, a large-scale cyber was launched

- it affected globally the entire Internet but particularly hit U.S. east coast
- during most of the day, no one could access a long list of major Internet platforms and services, e.g., Netflix, CNN, Airbnb, PayPal, Zillow, …
- this was a **Distributed Denial-of-Service (DDoS)** attack





# DoS: A threat (mainly) against availability

Which main security property does a Denial-of-Service (DoS) attack attempt to defeat?

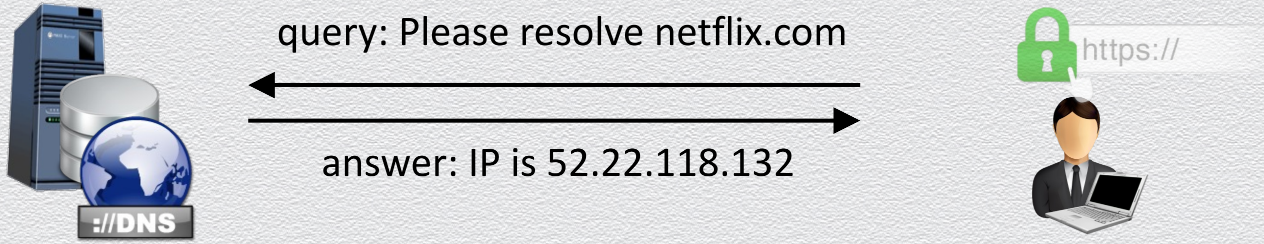
- availability; a user is denied access to authorized services or data
    - availability is concerned with preserving authorized access to assets
    - a DoS attack aims against this property; its name itself implies its main goal
- integrity & confidentiality; services or data are modified or accessed by an unauthorized user
    - elements of a DoS attack may include breaching the integrity or confidentiality of a system
    - but the end goal is disruption of a service or data flow; not the manipulation, fabrication or interception of data and services

# DNS

# The Domain Name Service (DNS) protocol

Resolving domain names to IP addresses

- when you type a URL in your Web browser, its IP address must be found
  - e.g., domain name “netflix.com” has IP address “52.22.118.132”
  - larger websites have multiple IP responses for redundancy to distributing load
- at the heart of Internet addressing is a protocol called DNS
  - a database translating Internet names to addresses



# DNS name resolution is a critical asset – a target itself!

What main security properties must be preserved in such an important service?

- all properties in CIA triad are relevant!
- resolving domain names to IP addresses is a service that
  - must critically be available during all times – availability
    - or else your browser does not know how to connect to Netflix…
  - must critically be trustworthy – integrity
    - or else connections to malicious sites may occur (e.g., DNS-spoofing attacks)
  - must also protect database entries that are not queried – confidentiality
    - or else an attacker may find out about the structure of a target organization (e.g., zone-enumeration attacks)

# Recursive name resolution: hierarchical search

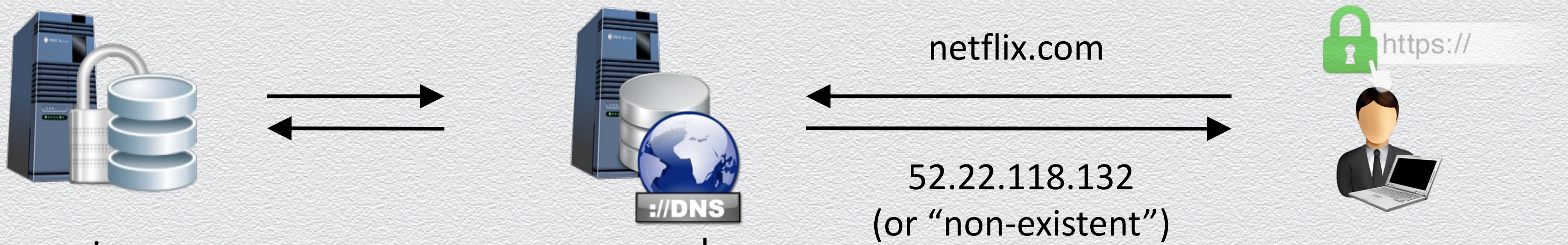
Search is performed recursively and hierarchically across different type of DNS resolvers

- application-level (e.g., Web browser), OS-level (e.g., stub resolver): locally managed
- recursive DNS servers: query other resolvers and cache recent results

**DNS entries:**
<netflix.com, 52.22.118.132>

**subset of cached queried entries**
(or information of other resolvers)

**locally cached IP addresses**
(at Web browser and OS)



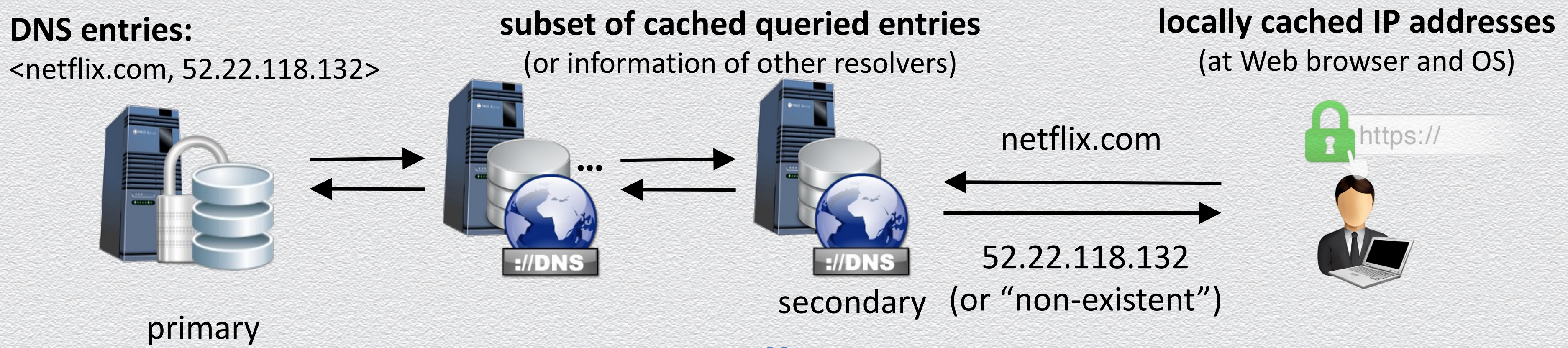

primary

secondary

# Recursive name resolution: hierarchical search

Search is performed recursively and hierarchically across different type of DNS resolvers

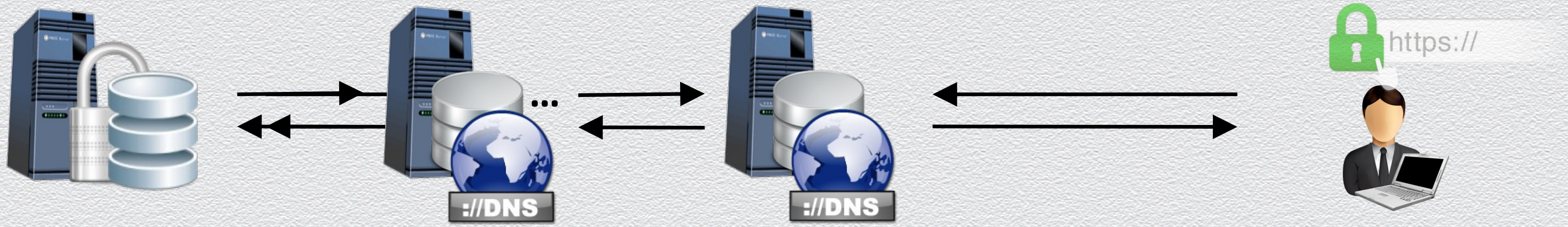
- application-level (e.g., Web browser), OS-level (e.g., stub resolver): locally managed
- recursive DNS servers: query other resolvers and cache recent results
- root name servers: refer to appropriate TLD (top-level domain) server
- TLD servers: control TLD zones such as .com, .org, .net, etc.



# Recursive name resolution: flexibility

Infrastructure allows for different configurations

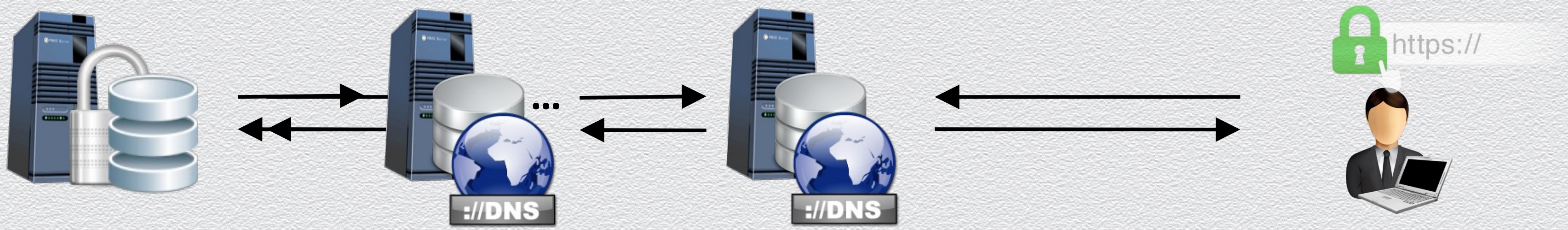
- authoritative-only servers: answer queries on zones they are responsible for
  - fast resolution, no forwarding, no cache
- caching / forwarding servers: answer queries on any public domain name
  - recursive search / request forwarding, caching for speed, first-hop resolvers
- primary / secondary servers: authoritative servers replicating DNS data of their domains
- public / private servers: control access to protected resources within an organization



# Recursive name resolution: benefits

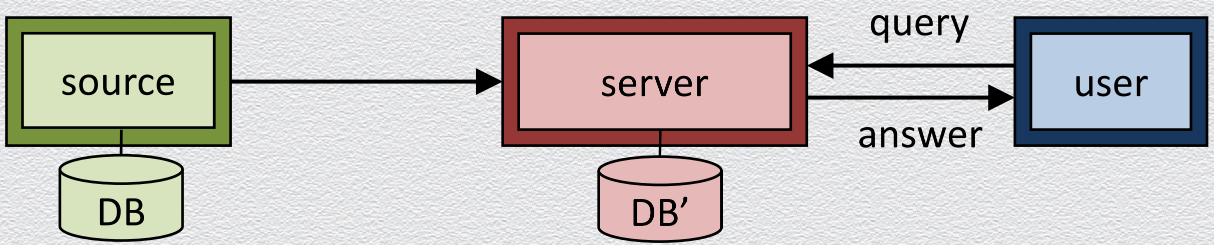
Why DNS uses non-authoritative name servers (that is, recursive resolution)?

- for more scalability & locality
  - high query loads can saturate the response capacity of primary servers
  - secondary do not have to store large volumes of DNS entries
  - cached recently queried domain names speed up searches due to locality of queries
- for added security / locality / scalability alone – not quite
  - e.g., non-authoritative name servers are untrusted and thus possibly compromised



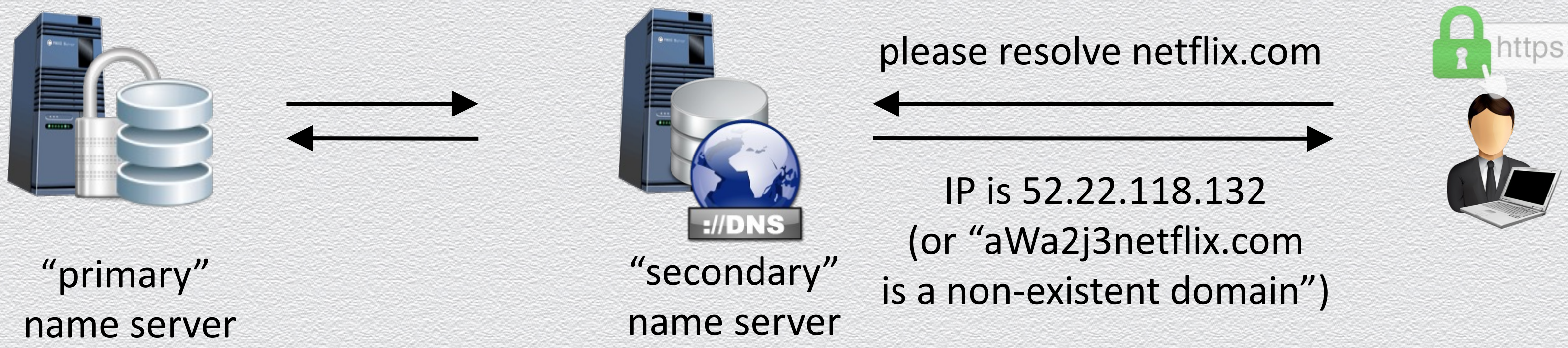
# DNS integrity: Protocols DNSSEC & NSEC

# DNS as a (distributed) database-as-a-service

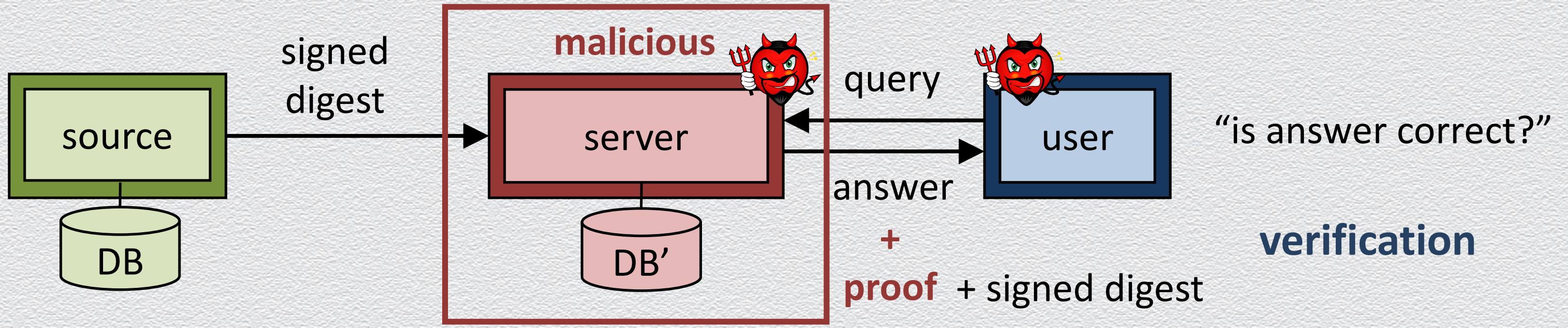


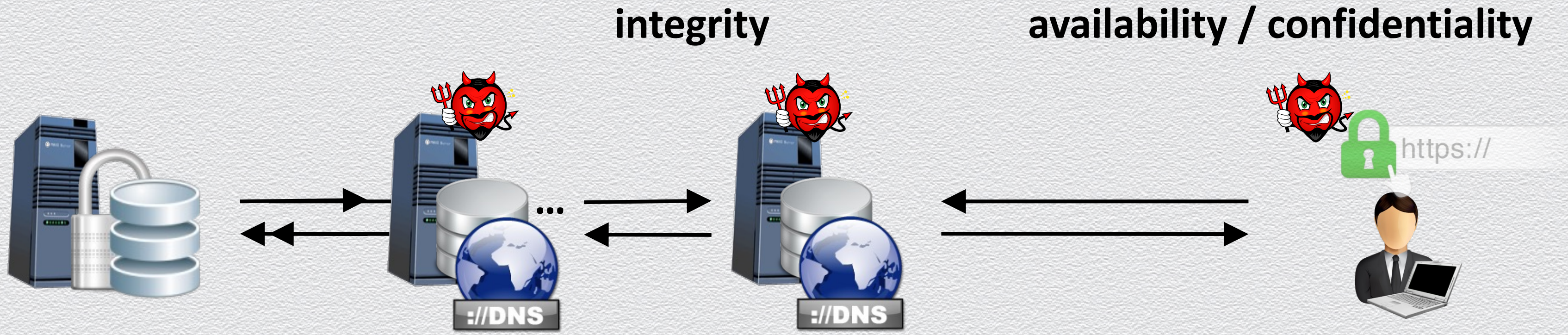

**DNS entries:**
<netflix.com, 52.22.118.132>

**subset of cached queried entries**
(or information of other resolvers)



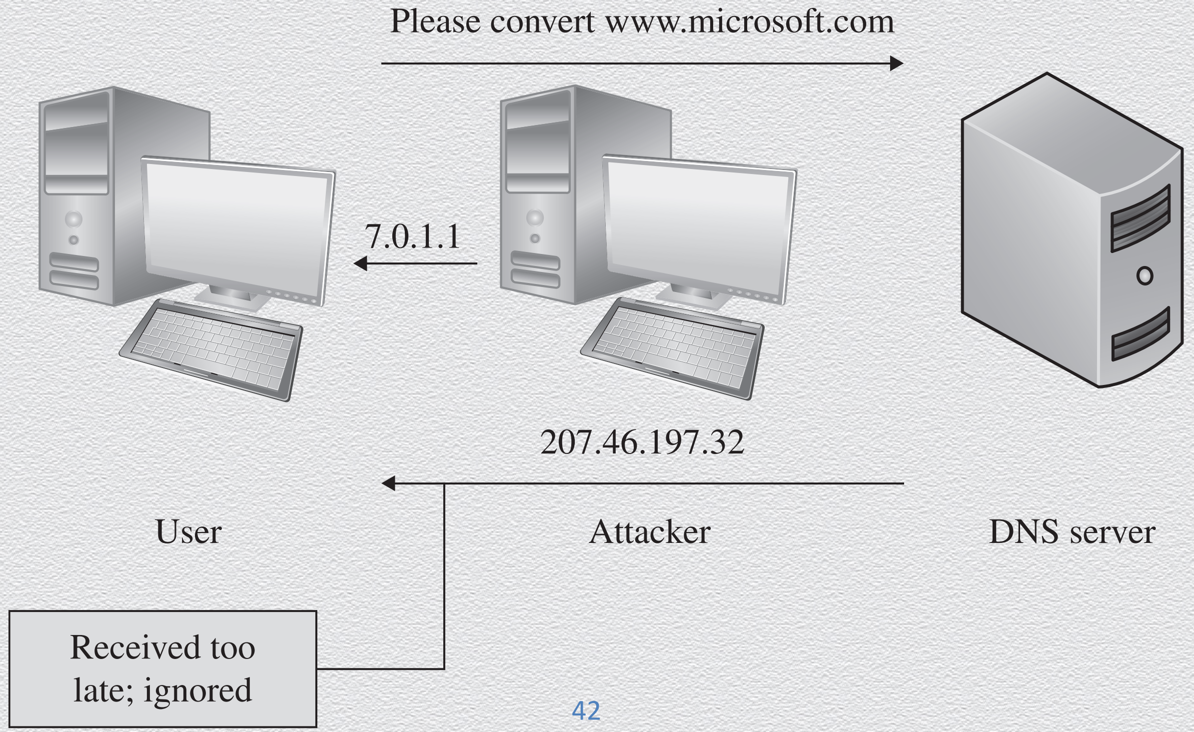
# A critical asset prone to attacks



# DNS spoofing (or cache poisoning)

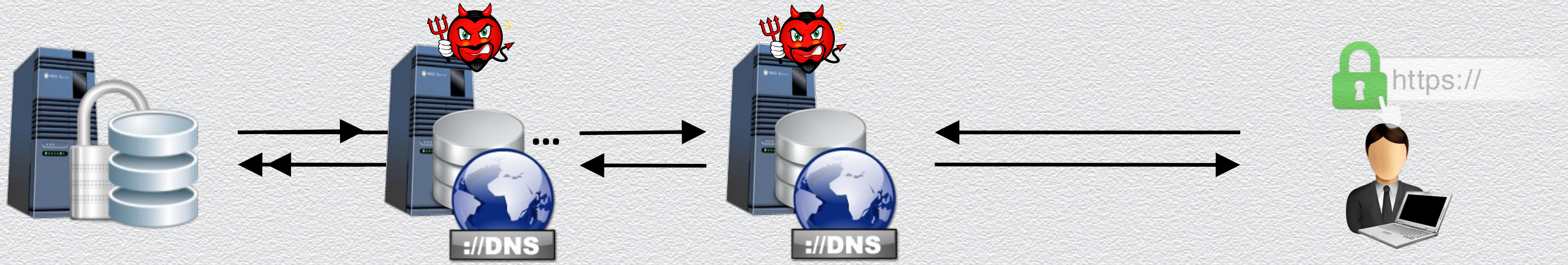
The attacker acts as the DNS server in order to redirect the user to malicious sites



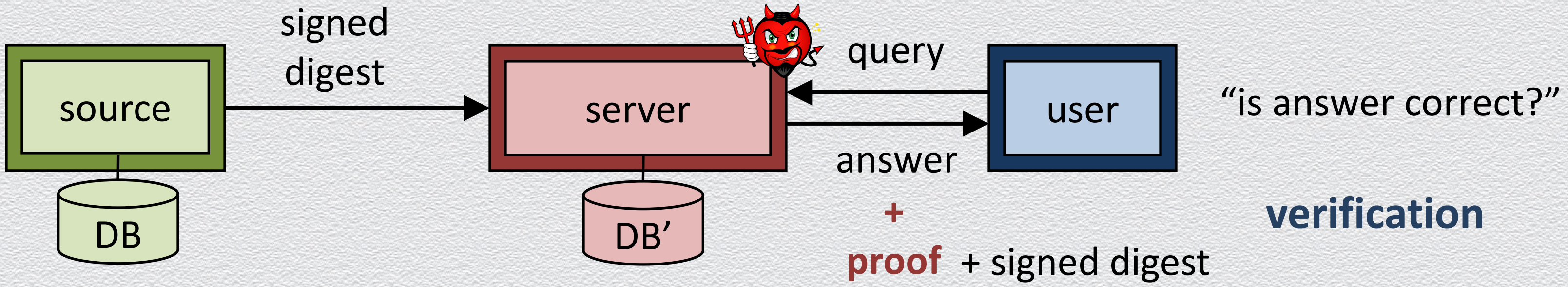
# DNSSEC & NSEC

Security extension of DNS protocol to protect integrity of DNS data

- correct resolution, origin authentication, authenticated denial of existence
- specifications made by Internet Engineering Task Force (IETF) via RFCs
  - an RFC (request for comments) is a suggested solution under peer review
- challenges: backward-compatible, simplicity, confidentiality, who signs
  - NSEC (next secure record): extension that provides proofs of denial of existence



# DNSSEC & NSEC: core idea

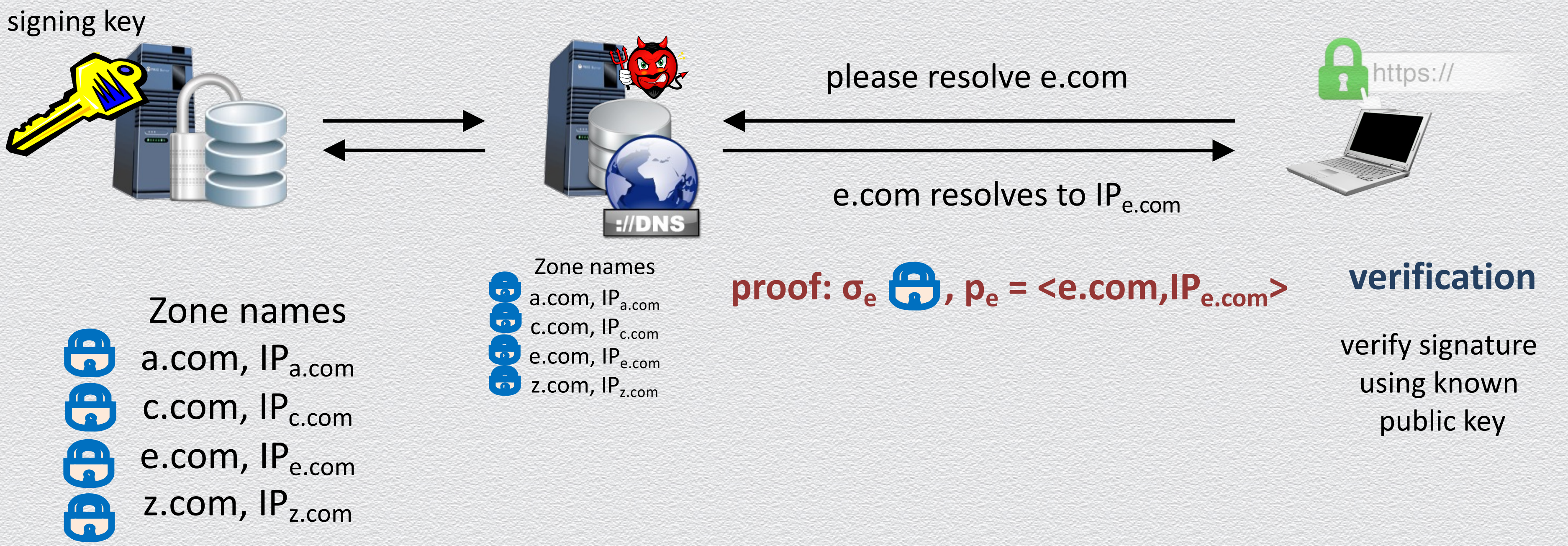



**DNSSEC protocol**: each DNS entry is pre-signed by primary name server

**NSEC protocol**:
- domain names are lexicographically ordered and then each pair of neighboring existing domain names is pre-signed by the primary name server
- non-existing names, e.g., aWa2j3netflix.com are proved by providing this pair “containing” missed query name, e.g., <awa.com, awb.com>

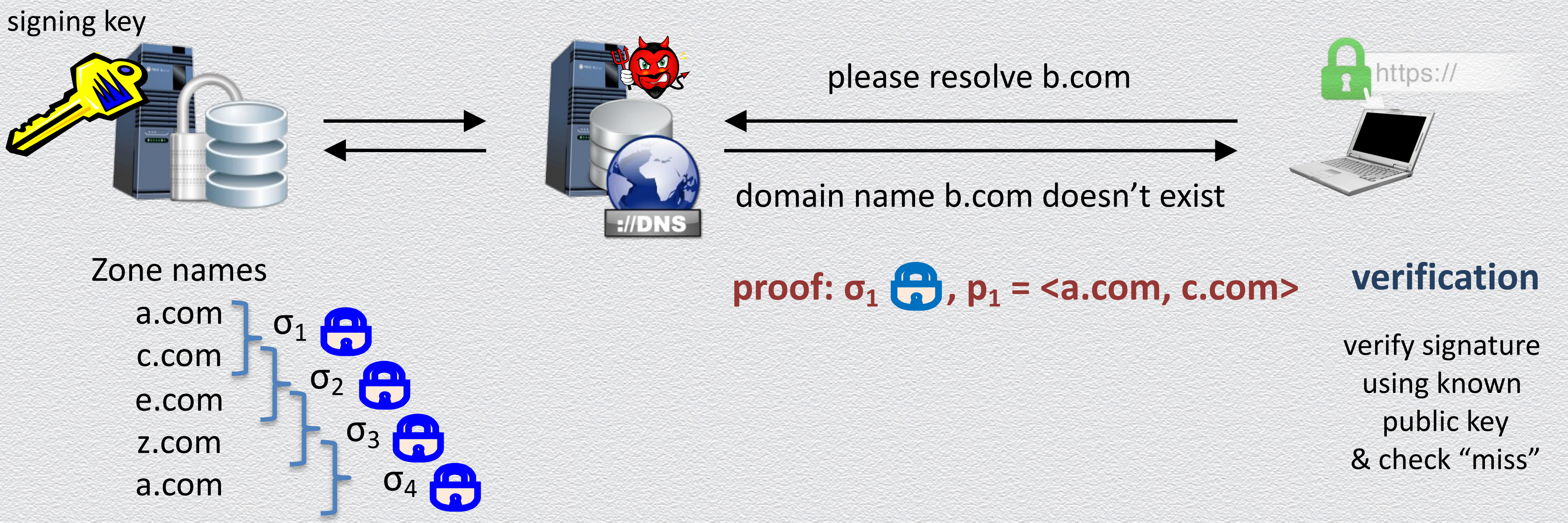
# DNSSEC: example

Each entry <domain name, IP address> in the database is individually signed by a primary DNS server and uploaded to secondary DNS servers in signed form



# NSEC: example

Additionally, pairs of consecutive (in alphabetical order) domain names are individually signed by a primary DNS server and uploaded to secondary DNS servers in signed form

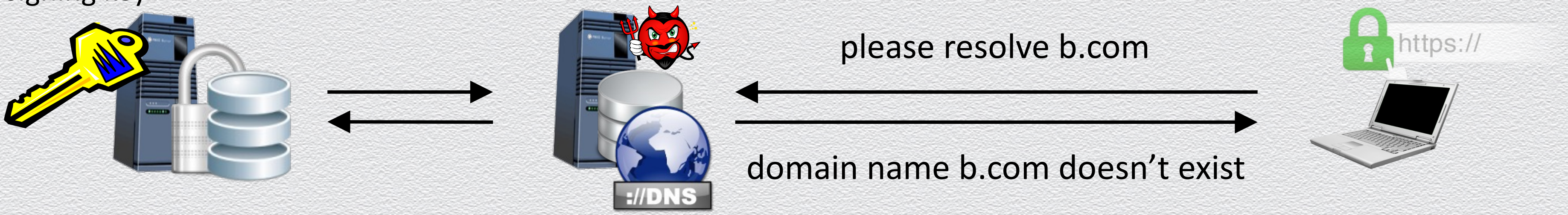


# NSEC vulnerability: Protocols NSEC3 & NSEC5

# The problem

Proofs of non-existing names leak information about other unknown domain names

signing key




Zone names

a.com
c.com
e.com
z.com
a.com

$\sigma_1$ $\sigma_2$ $\sigma_3$ $\sigma_4$

**proof: $\sigma_1$ , $p_1$ = <a.com, c.com>**

leaked information

user asked for b.com but
also learned for a.com & c.com

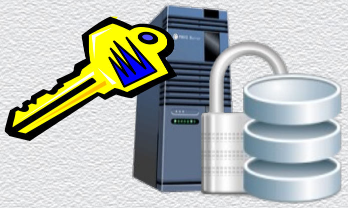
**verification**

verify signature
using known
public key
& check "miss"

# Zone enumeration attack: Main idea

An attacker can simply act as a “querier” to learn target organization’s network structure!

signing key




please resolve b.com

domain name b.com doesn’t exist


Zone names

a.com
c.com
e.com
z.com
a.com

$\sigma_1$
$\sigma_2$
$\sigma_3$
$\sigma_4$

**proof: $\sigma_1$, $p_1$ = <a.com, c.com>**

exploit the “leak-domain-names” vulnerability of NSEC to learn the domain names of an entire zone

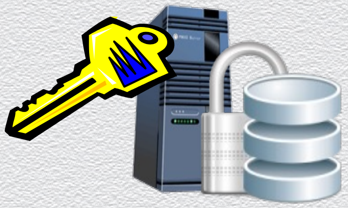
**verification**

verify signature using known public key & check “miss”

# Zone enumeration attack: Example

An attacker can simply act as a “querier” to learn target organization’s network structure!

signing key




resolve b$.com, d#.com, e%.com

none exists


Zone names

a.com
c.com
e.com
z.com
a.com

$\sigma_1$ $\sigma_2$ $\sigma_3$ $\sigma_4$

**proof: $\sigma_1$, $p_1$ = <a.com, c.com>**
**proof: $\sigma_2$, $p_2$ = <c.com, e.com>**

**proof: $\sigma_3$, $p_3$ = <e.com, z.com>**

ask for non-existing names
to get all possible proofs

**verification**

verify signature
using known
public key
& check “miss”

# Zone enumeration attack: Result

An attacker can simply act as a "querier" to learn target organization's network structure!


Zone names

a.com $\sigma_1$
c.com $\sigma_2$
e.com $\sigma_3$
z.com $\sigma_4$
a.com

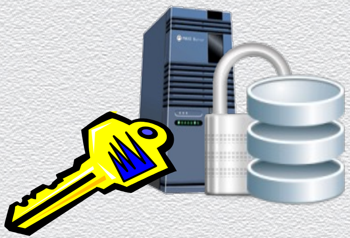
ask for non-existing names
to get all possible proofs

This attack may expose private device names (e.g., IoT devices which can be toehold for other attacks) or reveal other private data that many registries may have legal obligations to protect

Zone names

a.com
c.com
e.com
z.com
a.com

# NSEC3: NSEC in the hash domain




please resolve b.com

b.com is a non-existent domain

**proof: $\sigma_3$ , $p_3$ = <dde45,zrit5>**


Zone names

a.com
c.com
e.com
z.com

hash h →

a1bb5
23ced
zrit5
dde45

sort →

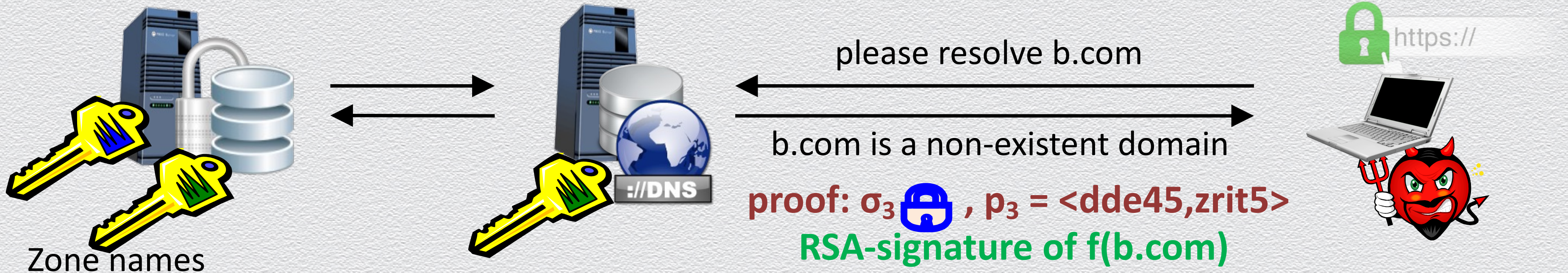
23ced
a1bb5
dde45
zrit5
23ced

$\sigma_1$
$\sigma_2$
$\sigma_3$
$\sigma_4$

asked for b.com but
learned h(e.com) & h(z.com)

h(b.com) = ntwo4
e.g., h is SHA-256

# NSEC5: A secure solution




Zone names

a.com
c.com
e.com
z.com

hash h' →

a1bb5
23ced
zrit5
dde45

sort →

23ced
a1bb5
dde45
zrit5
23ced

$\sigma_1$ $\sigma_2$ $\sigma_3$ $\sigma_4$

asked for b.com but
learned h'(e.com) & h'(z.com)

h'(b.com) = ntwo4
h: as in NSEC3
f: "message transformation" hash

**h'(x) = h( RSA-Sign( [key], f(x)) )**

# The RSA algorithm

# The RSA algorithm (for encryption)

## General case

Setup (run by a given user)

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with $\mathbf{p}$ and $\mathbf{q}$ primes
- $\mathbf{e}$ relatively prime to $\boldsymbol{\phi}(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- $\mathbf{d}$ inverse of $\mathbf{e}$ in $\mathbf{Z}_{\boldsymbol{\phi}(\mathbf{n})}$

Keys

- public key is $K_{PK} = (\mathbf{n}, \mathbf{e})$
- private key is $K_{SK} = \mathbf{d}$

Encryption

- $\mathbf{C} = \mathbf{M}^{\mathbf{e}} \bmod \mathbf{n}$ for plaintext $\mathbf{M}$ in $\mathbf{Z}_{\mathbf{n}}$

Decryption

- $\mathbf{M} = \mathbf{C}^{\mathbf{d}} \bmod \mathbf{n}$

## Example

Setup

- $\mathbf{p} = 7$, $\mathbf{q} = 17$, $\mathbf{n} = 7 \cdot 17 = 119$
- $\mathbf{e} = 5$, $\boldsymbol{\phi}(\mathbf{n}) = 6 \cdot 16 = 96$
- $\mathbf{d} = 77$

Keys

- public key is (119, 5)
- private key is 77

Encryption

- $\mathbf{C} = 19^5 \bmod 119 = 66$ for $\mathbf{M} = 19$ in $\mathbf{Z}_{119}$

Decryption

- $\mathbf{M} = 66^{77} \bmod 119 = 19$

# Another complete example

- Setup
  - **p** = 5, **q** = 11, **n** = 5 · 11 = 55
  - **ϕ**(**n**) = 4 · 10 = 40
  - **e** = 3, **d** = 27 (3·27 = 81 = 2·40 + 1)
- Encryption
  - **C** = **M**$^3$ mod 55 for **M** in $\mathbf{Z}_{55}$
- Decryption
  - **M** = **C**$^{27}$ mod 55

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $M$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| $C$ | 1 | 8 | 27 | 9 | 15 | 51 | 13 | 17 | 14 | 10 | 11 | 23 | 52 | 49 | 20 | 26 | 18 | 2 |
| $M$ | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| $C$ | 39 | 25 | 21 | 33 | 12 | 19 | 5 | 31 | 48 | 7 | 24 | 50 | 36 | 43 | 22 | 34 | 30 | 16 |
| $M$ | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| $C$ | 53 | 37 | 29 | 35 | 6 | 3 | 32 | 44 | 45 | 41 | 38 | 42 | 4 | 40 | 46 | 28 | 47 | 54 |

# Correctness of RSA

## Given

**Setup**

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with **p** and **q** primes
- **e** relatively prime to $\phi(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- **d** inverse of **e** in $\mathbf{Z}_{\phi(n)}$ **(1)**

**Encryption**

- $\mathbf{C} = \mathbf{M}^e \bmod \mathbf{n}$ for plaintext **M** in $\mathbf{Z}_n$

**Decryption**

- $\mathbf{M} = \mathbf{C}^d \bmod \mathbf{n}$

**Fermat's Little Theorem** **(2)**

- for prime p, non-zero x: $x^{p-1} \bmod p = 1$

## Analysis

Need to show

- $M^{ed} = M \bmod p \cdot q$

Use **(1)** and apply **(2)** for prime p

- $M^{ed} = M^{ed-1} M = (M^{p-1})^{h(q-1)} M$
- $\mathbf{M}^{ed} = 1^{h(q-1)} M \bmod p = \mathbf{M \bmod p}$

Similarly (w.r.t. prime q)

- $\mathbf{M^{ed} = M \bmod q}$

Thus, since p, q are co-primes

- $\mathbf{M^{ed} = M \bmod p \cdot q}$

# A useful symmetry

**[1] RSA setting**

- modulo $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, p & q are **primes**, public & private keys (e,d): $\mathbf{d \cdot e = 1 \bmod (p-1)(q-1)}$

**[2]** RSA operations involve **exponentiations**, thus they are **interchangeable**

- $\mathbf{C} = \mathbf{M}^{\mathbf{e}} \bmod \mathbf{n}$ (encryption of plaintext **M in $\mathbf{Z_n}$**)
- $\mathbf{M} = \mathbf{C}^{\mathbf{d}} \bmod \mathbf{n}$ (decryption of ciphertext **C in $\mathbf{Z_n}$**)

Indeed, their order of execution does not matter: $\mathbf{(M^e)^d = (M^d)^e \bmod n}$

**[3]** RSA operations involve exponents that **"cancel out",** thus they are **complementary**

- $\mathbf{x}^{\mathbf{(p-1)(q-1)}} \bmod \mathbf{n} = \mathbf{1}$ (Euler's Theorem)

Indeed, they invert each other: $\mathbf{(M^e)^d = (M^d)^e = M^{ed} = M^{k(p-1)(q-1)+1} \bmod n}$

$\mathbf{= (M^{(p-1)(q-1)})^k \cdot M = 1^k \cdot M = M \bmod n}$

# Signing with RSA

RSA functions are complementary & interchangeable w.r.t. order of execution

- **core property**: $M^{ed} = M \bmod p \cdot q$ for any message $M$ in $Z_n$

RSA cryptosystem lends itself to a **signature scheme**

- 'reverse' use of keys is possible : $(M^d)^e = M \bmod p \cdot q$
- signing algorithm **Sign(M,d,n)**: $\sigma = M^d \bmod n$ for message $M$ in $Z_n$
- verifying algorithm **Vrfy(σ,M,e,n)**: return $M == \sigma^e \bmod n$

# The RSA algorithm (for signing)

## General case

Setup (run by a given user)

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with **p** and **q** primes
- **e** relatively prime to $\boldsymbol{\phi}(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- **d** inverse of **e** in $\mathbf{Z}_{\boldsymbol{\phi}(\mathbf{n})}$

Keys (same as in encryption)

- public key is $\mathbf{K_{PK}} = (\mathbf{n}, \mathbf{e})$
- private key is $\mathbf{K_{SK}} = \mathbf{d}$

Sign

- $\boldsymbol{\sigma} = \mathbf{M}^{\mathbf{d}}$ mod **n** for message **M** in $\mathbf{Z_n}$

Verify

- Check if $\mathbf{M} = \boldsymbol{\sigma}^{\mathbf{e}}$ mod **n**

## Example

Setup

- $\mathbf{p} = 7$, $\mathbf{q} = 17$, $\mathbf{n} = 7 \cdot 17 = 119$
- $\mathbf{e} = 5$, $\boldsymbol{\phi}(\mathbf{n}) = 6 \cdot 16 = 96$
- $\mathbf{d} = 77$

Keys

- public key is (119, 5)
- private key is 77

Signing

- $\boldsymbol{\sigma} = 66^{77}$ mod 119 = 19 for **M** = 66 in $\mathbf{Z_{119}}$

Verification

- Check if $\mathbf{M} = 19^{5}$ mod 119 = 66

# Digital signatures & hashing

Very often digital signatures are used with hash functions

- the hash of a message is signed, instead of the message itself

**Signing message M**

- let h be a cryptographic hash function, assume RSA setting (n, d, e)
- compute signature σ on message M as: $\sigma = h(M)^d \bmod n$
- send σ, M

**Verifying signature σ**

- use public key (e, n) to compute (candidate) hash value $H = \sigma^e \bmod n$
- if $H = h(M)$ output ACCEPT, else output REJECT

# Security of RSA

Based on difficulty of **factoring** large numbers (into large primes), i.e., $n = p \cdot q$ into $p$, $q$

- note that for RSA to be secure, both $p$ and $q$ must be large primes
- widely believed to hold true
  - since 1978, subject of extensive cryptanalysis without any serious flaws found
  - best known algorithm takes exponential time in security parameter (key length $|n|$)
- how can you break RSA if you can factor?

Current practice is using 2,048-bit long RSA keys (617 decimal digits)

- estimated computing/memory resources needed to factor an RSA number within one year

| Length (bits) | PCs | Memory |
|---|---|---|
| 430 | 1 | 128MB |
| 760 | 215,000 | 4GB |
| 1,020 | $342 \times 10^6$ | 170GB |
| 1,620 | $1.6 \times 10^{15}$ | 120TB |

# RSA challenges

## Challenges for breaking the RSA cryptosystem of various key lengths (i.e., |n|)

- known in the form RSA-`key bit length’ expressed in bits or decimal digits
- provide empirical evidence/confidence on strength of specific RSA instantiations

## Known attacks

- RSA-155 (**512-bit**) factored in **4 mo**. using 35.7 CPU-years or 8000 Mips-years (**1999**) and 292 machines
  - 160 175-400MHz SGI/Sun, 8 250MHz SGI/Origin, 120 300-450MHz Pent. II, 4 500MHz Digital/Compaq
- RSA-**640** factored in **5 mo**. using 30 2.2GHz CPU-years (**2005**)
- RSA-220 (**729-bit**) factored in **5 mo**. using 30 2.2GHz CPU-years (**2005**)
- RSA-232 (**768-bit**) factored in **2 years** using **parallel** computers 2K CPU-years (1-core 2.2GHz AMD Opteron) (**2009**)

## Most interesting challenges

- prizes for factoring RSA-**1024**, RSA-**2048** is $100K, $200K – estimated at 800K, 20B Mips-centuries

# Deriving an RSA key pair

- public key is pair of integers (e,n), secret key is (d, n) or d
- the value of n should be quite large, a product of two large primes, p and q
- often p, q are nearly 100 digits each, so n ~= 200 decimal digits (~512 bits)
  - but 2048-bit keys are becoming a standard requirement nowadays
- the larger the value of n the harder to factor to infer p and q
  - but also the slower to process messages
- a relatively large integer e is chosen
  - e.g., by choosing e as a prime that is larger than both $(p - 1)$ and $(q - 1)$
  - why?
- d is chosen s.t. $e \cdot d = 1 \bmod (p - 1)(q - 1)$
  - how?

# Discussion on RSA

- Assume **p** = 5, **q** = 11, **n** = 5 · 11 = 55, **ϕ(n)** = 40, **e** = 3, **d** = 27
  - why encrypting small messages, e.g., **M** = 2, 3, 4 is tricky?
  - recall that the ciphertext is **C** = $\mathbf{M}^3$ mod 55 for **M** in $\mathbf{Z}_{55}$

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***M*** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| ***C*** | 1 | 8 | 27 | 9 | 15 | 51 | 13 | 17 | 14 | 10 | 11 | 23 | 52 | 49 | 20 | 26 | 18 | 2 |
| ***M*** | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| ***C*** | 39 | 25 | 21 | 33 | 12 | 19 | 5 | 31 | 48 | 7 | 24 | 50 | 36 | 43 | 22 | 34 | 30 | 16 |
| ***M*** | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| ***C*** | 53 | 37 | 29 | 35 | 6 | 3 | 32 | 44 | 45 | 41 | 38 | 42 | 4 | 40 | 46 | 28 | 47 | 54 |

# Discussion on RSA

- Assume **p** = 5, **q** = 11, **n** = 5 · 11 = 55, **ϕ(n)** = 40, **e** = 3, **d** = 27
  - why encrypting small messages, e.g., **M** = 2, 3, 4 is tricky?
  - recall that the ciphertext is $\mathbf{C} = \mathbf{M}^3 \bmod 55$ for **M** in $\mathbf{Z}_{55}$
- Assume n = 2043439438435553434354542894348343435 6091 = p · q
  - can e be the number 4343253453434536?
- Are there problems with applying RSA in practice?
  - what other algorithms are required to be available to the user?
- Are there problem with respect to RSA security?
  - does it satisfy CPA (advanced) security?

# Algorithmic issues

The implementation of the RSA cryptosystem requires various algorithms

- Main issues
  - representation of integers of arbitrarily large size; and
  - arithmetic operations on them, namely computing modular powers
- Required algorithms (at setup)
  - generation of **random numbers** of a given number of bits (to compute candidates **p**, **q**)
  - **primality testing** (to check that candidates **p**, **q** are prime)
  - computation of the **GCD** (to verify that **e** and **ϕ(n)** are relatively prime)
  - computation of the **multiplicative inverse** (to compute **d** from **e**)

# Pseudo-primality testing

Testing whether a number is prime (**primality testing**) is a difficult problem

An integer $\mathbf{n} \geq 2$ is said to be a base-$\mathbf{x}$ **pseudo-prime** if

- $\mathbf{x}^{\mathbf{n}-1} \bmod \mathbf{n} = 1$ (Fermat's little theorem)

- Composite base-$\mathbf{x}$ pseudo-primes are rare
  - a random 100-bit integer is a composite base-2 pseudo-prime with probability less than $10^{-13}$
  - the smallest composite base-2 pseudo-prime is 341
- Base-$\mathbf{x}$ pseudo-primality testing for an integer $\mathbf{n}$
  - check whether $\mathbf{x}^{\mathbf{n}-1} \bmod \mathbf{n} = 1$
  - can be performed efficiently with the repeated squaring algorithm

# Security properties

- Plain RSA is deterministic
  - why is this a problem?
- Plain RSA is also homomorphic
  - what does this mean?
  - multiply ciphertexts to get ciphertext of multiplication!
  - $[(m_1)^e \bmod N][(m_2)^e \bmod N] = (m_1 m_2)^e \bmod N$
  - however, not additively homomorphic

# Real-world usage of RSA

- Randomized RSA
  - to encrypt message M under an RSA public key (e,n), generate a new random session AES key K, compute the ciphertext as [$K^e$ mod n, $AES_K(M)$]
  - prevents an adversary distinguishing two encryptions of the same M since K is chosen at random every time encryption takes place
- Optimal Asymmetric Encryption Padding (OAEP)
  - roughly, to encrypt M, choose random r, encode M as $M' = [X = M \oplus H_1(r)\ ,\ Y = r \oplus H_2(X)\ ]$ where $H_1$ and $H_2$ are cryptographic hash functions, then encrypt it as $(M')^e$ mod n

# Summary of message-authentication crypto tools

| | Hash (SHA2-256) | MAC | Digital signature |
|---|---|---|---|
| Integrity | Yes | Yes | Yes |
| Authentication | No | Yes | Yes |
| Non-repudiation | No | No | Yes |
| Crypto system | None | Symmetric (AES) | Asymmetric (e.g., RSA) |