University of Connecticut
Computer Science and Engineering
CSE 4402/5095: Network Security

Security of the Domain Name System (DNS)

Version of Fall '24

Domain Name System (DNS) Security

- **Introducing DNS (or a quick recap)**
- DNS poisoning
  - Motivation: the hacker's swiss knife
  - Method 1 (historical): by Gratuitous `glue' RR
  - Method 2: send from corrupt NS
  - Method 3: send spoofed DNS response
  - Method 4: dangling DNS records
- DNSSEC: Cryptographic security for DNS
- DNS Privacy issues and defenses

Domain Names and IP Addresses

- IP packets contain source, dest IP addresses
  - 32 bits, e.g. 128.33.44.223
- Routers use IP Addresses
  - To deliver packets to their destinations
- Users use Domain Names, e.g. www.foo.edu
- Domain Names are hierarchical, and:
  - Meaningful: *.edu: university, www.*: web server
  - Easier to manage, remember and use
  - The Domain Name System (DNS) maps domain names to IP addresses

Benefits of using domain names, not IPs

- Usability: easier to remember, meaningful
- Abstraction
  - Fixed domain-name mapped to sometimes changing IP
  - IP changes for mobility, optimization (close to customer), business (CDN/cloud, change provider)
  - Multiple IPs for the same domain: redundancy, IPv4 & IPv6, localization (for efficiency)
  - Organization: hierarchy, wildcards
- Map domain names to different **record types**:
  - IPv4 and IPv6 addresses
  - And many more types of **resource record (RR)** types

Some Domain Name System Record Types

- A: IPv4 address, AAAA: IPv6
- 'Alias' name: CNAME
- Mail server: MX
- Name server: NS
- Server for service: SRV
- Policies and PKs: CAA, SPF, DKIM,…
- Blacklists
- Any text: TXT
- Reverse-DNS (`pointer') records, map an IP to a domain: PTR

IP of domain

Name of email (SMTP) server of domain

Name of name server of domain

Servers authorized to send email for domain

CAs used by BoA

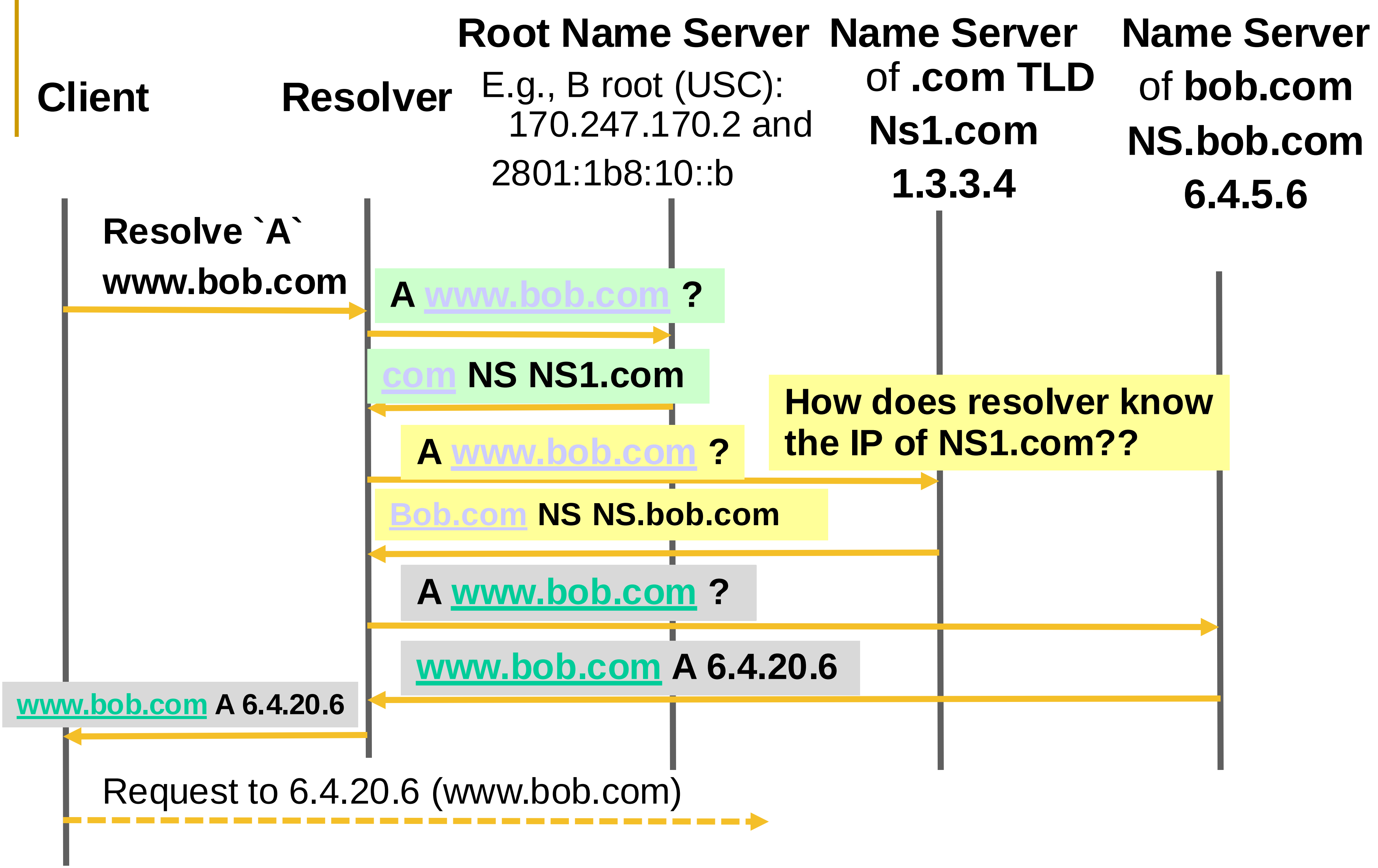
Indication if IP is spammer

Domain name of owner of IP

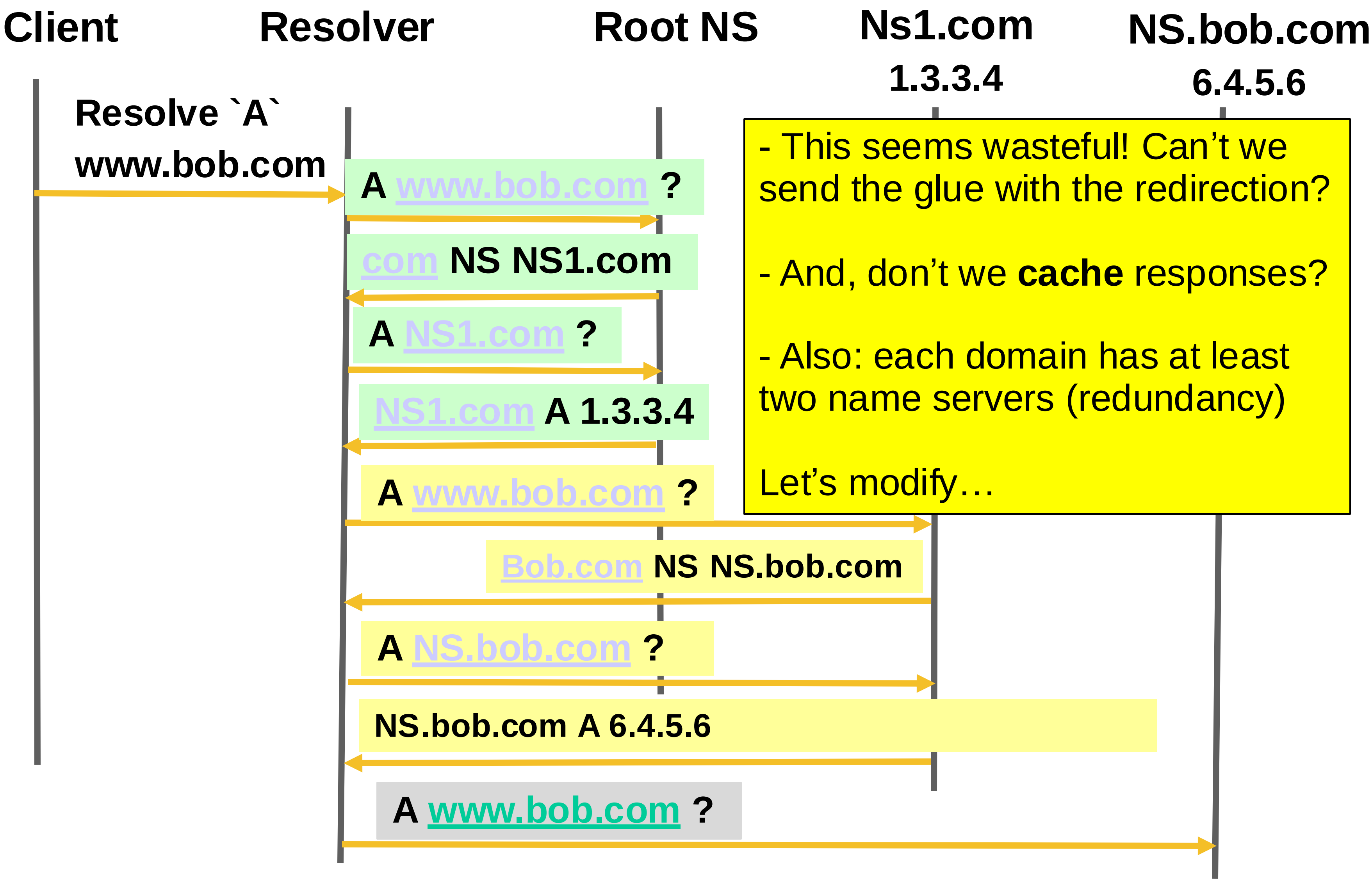
DNS Entities

- DNS Client (aka DNS stub resolver)
  - Provides DNS resource records (RRs) to applications
  - If requested RR (type, domain) not in cache, asks resolver
  - Contact resolver by IP address (set manually or by DHCP)
- Resolver (aka recursive resolver, DNS cache, or local NS)
  - Returns resource records (RRs) requested by clients
  - Requests RRs from Name Servers, if not in resolver's cache
    - Selects order of name servers to try based on past performance: failures and delays
- Name servers (aka authoritative NS)
  - Provides the RRs of one or more domains
  - Hierarchical, e.g., NS of .com identifies NS of bob.com

DNS Resolution Process (simplified)



Glue query to find IPs of name servers



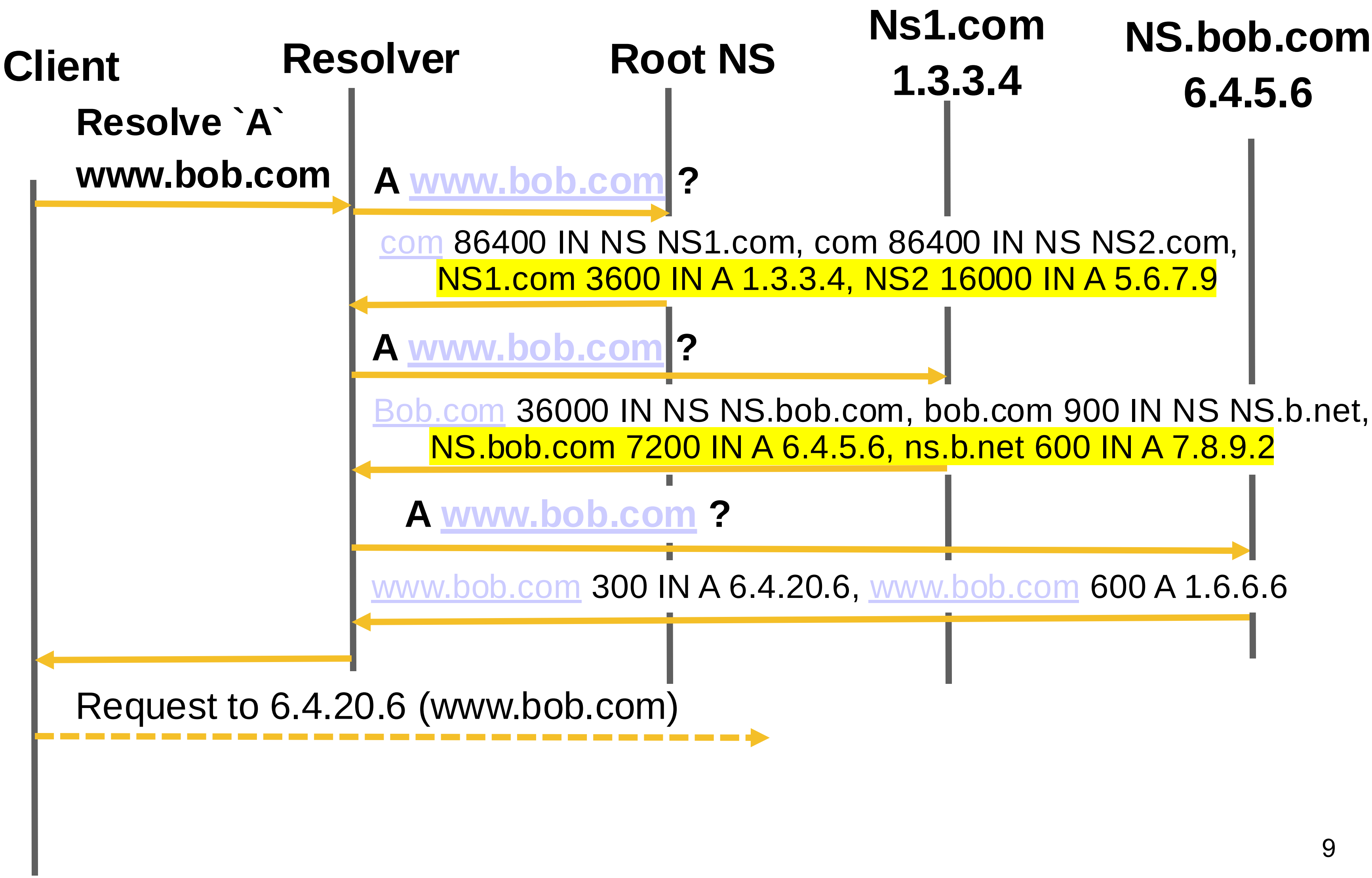
- This seems wasteful! Can’t we send the glue with the redirection?

- And, don’t we **cache** responses?

- Also: each domain has at least two name servers (redundancy)

Let’s modify…

DNS Resolution Process: with Glue, multiple name servers and TTL (caching)



DNS Caching

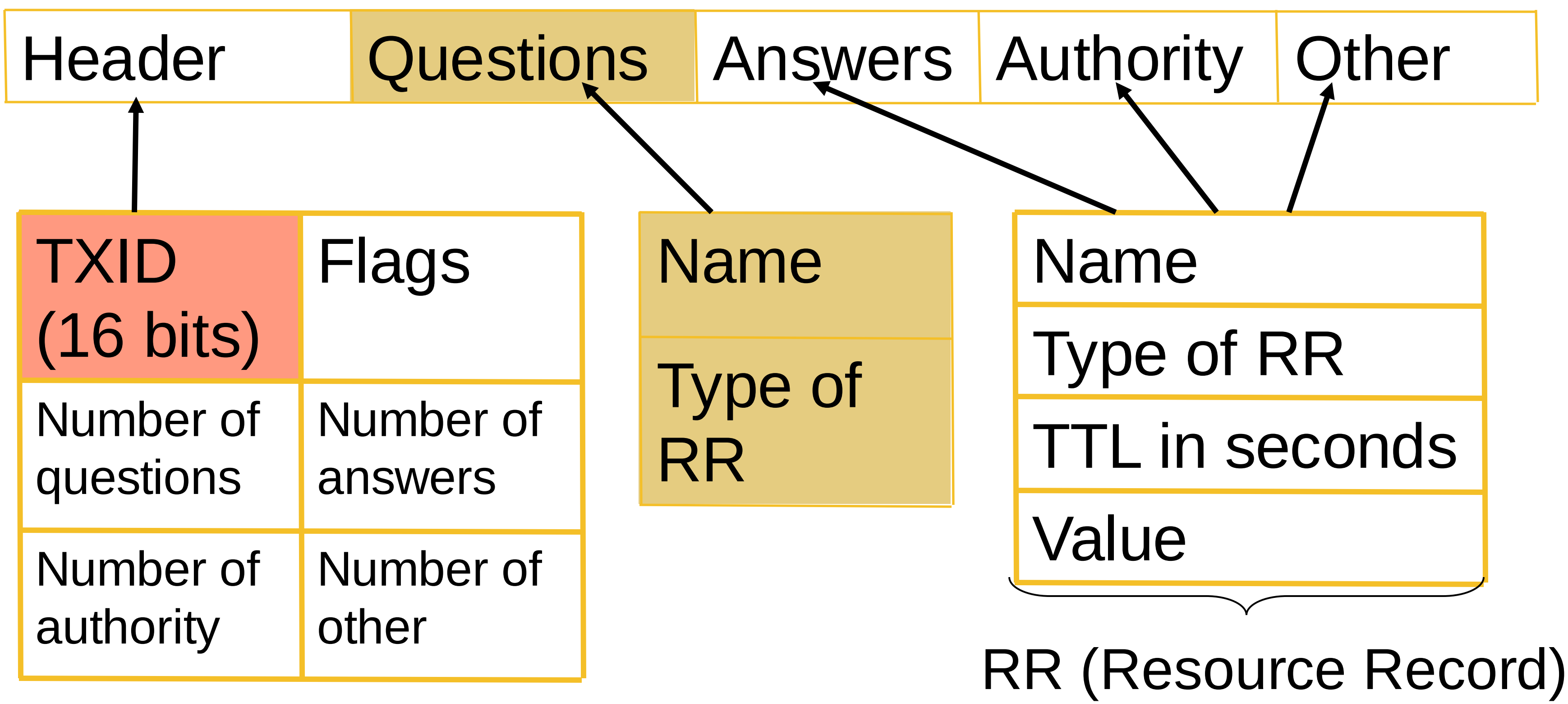
- Caching of RRs is critical for performance
  - An RR is cached for TTL value (specified in RR)
  - Non-existing domain (Nxdomain) responses cached for period set by resolver
- Each DNS client maintains a cache
  - And sends requests to resolver (which keeps a cache too)
- Resolver cache
  - Sends query to NS of most specific domain in cache
    - E.g., send query to NS of Bob.com if known; if not, to NS.com (or root)
  - Often nested (univ. resolver → ISP's resolver)
  - 'Open resolver': provides service to any client

Examples of Domain Name System Records

- A: IPv4 address, AAAA: IPv6 address
  - Bob.com A 1.2.3.4, bob.com AAAA 2024:1111:2222:333:4444:FFFF
- ‘Alias’ name: CNAME
  - Bob.com CNAME bob.com.cdn.net
- Mail server: MX
  - Bob.com MX 1 smtp.bob.com, bob.com MX 5 backup.mail.bob.com
- Name server: NS
  - Bob.bom NS ns1.bob.com
- Server for service (IMAP, SIP, XMPP, LDAP,…): SRV
  - Service, protocol, priority, weight, port, server-domain
  - _imap._tcp.bob.com. 86400 IN SRV 10 4 143 imap.bob.com
- Policies and PKs: CAA, SPF, DKIM,…
  - CAA (CA authorization) defines allowed issuing CA, policy, etc.
- Reverse-DNS (`pointer’, rDNS): PTR
  - 6.6.6.6.in-addr.arpa PTR 666.org, 2024:::::666.ip6.arpa PTR 666.org

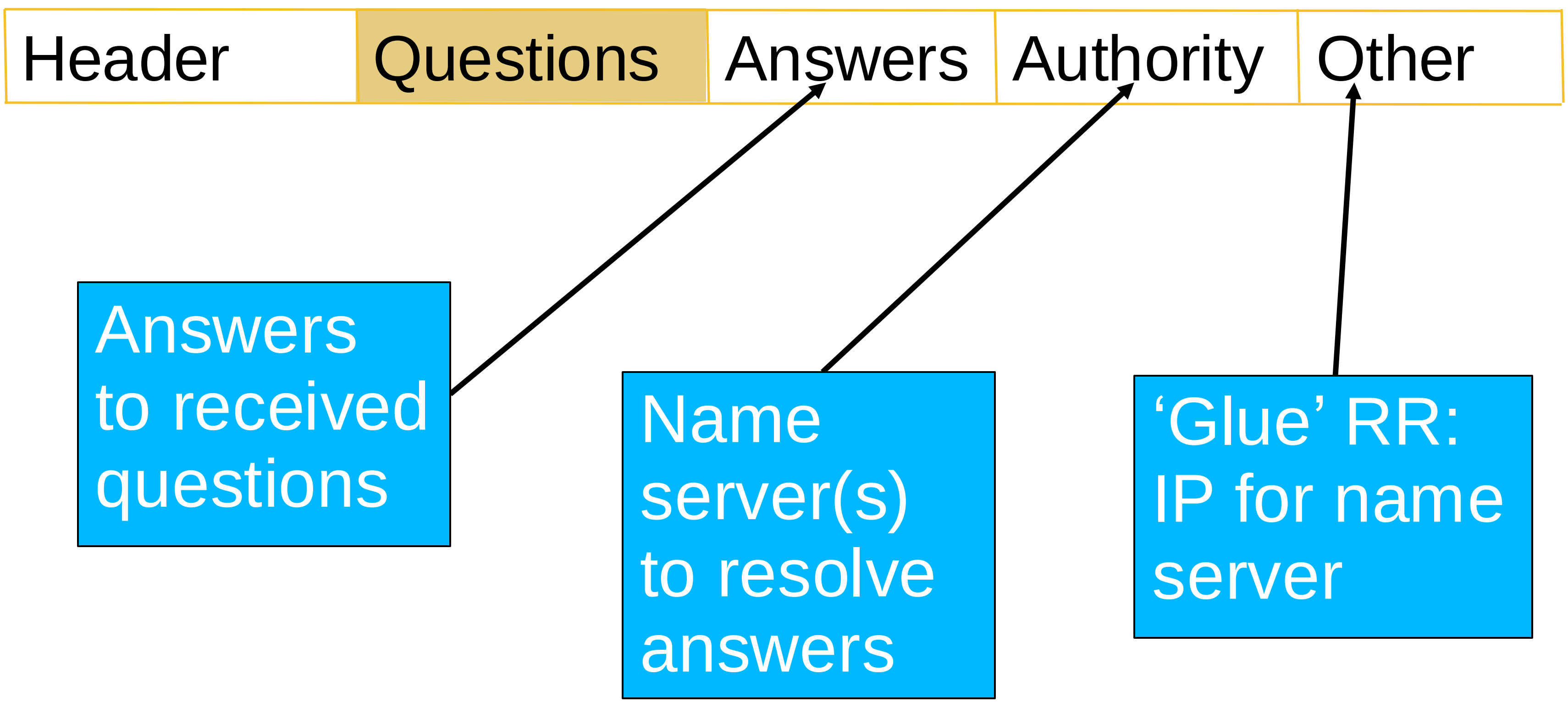
DNS Messages

- DNS protocol: send request, receive reply
- Single format for requests & replies



DNS Messages

- DNS protocol: send request, receive reply
- Single format for requests & replies
  - Typically, requests contain only header & questions



DNS Messages

- DNS protocol: send request, receive reply
- Single format for requests & replies

Header	Questions	Answers	Authority	Other

Answers to received questions:
Bob.com A 1.2.3.4

Name server(s) to resolve answers:
b.com NS NS.b.net

'Glue' RR:
IP for name server
ns.b.net A 7.8.9.2

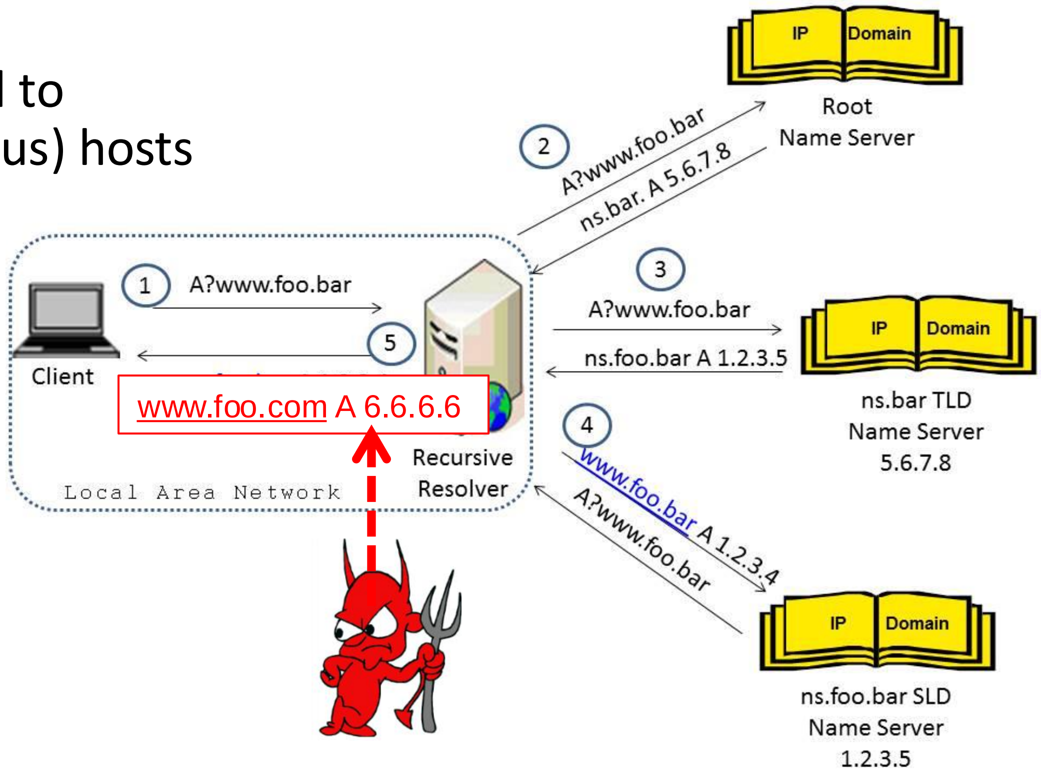
DNS Security: Goals

- Authenticity
  - Owners should control mappings (name →IP)
  - Challenge-response defence against **off-path attacker**
  - Or: DNSSEC to protect against **MitM attacker**
    - DNSSEC uses signatures and cryptographic hashing, see later
- Availability
  - Prevent Denial of Service (DoS) attacks
- Privacy / Confidentiality ??
  - Privacy for server/data from user: not a goal
    - Protocol allows any client to query any server
    - Undesirable: \`what's there?\` query
  - Privacy for user's queries ?
    - Your resolutions tell a lot about you! Motivation to offer open resolvers ☺
    - Defense from MitM/ISP: DNS Over HTTPS / TLS (DOH, DOT)
    - Defense from resolver: NAT ?

Threat: Cache Poisoning

Problem: no authentication of DNS responses

- Attacker can provide spoofed records
- Resolver caches
- Clients redirected to incorrect (malicious) hosts



DNS Poisoning: the Hacker's Knife

Circumvent: Blacklists, SOP, SPF, DKIM, CAA

Phishing

Cookies theft

Malware Distribution

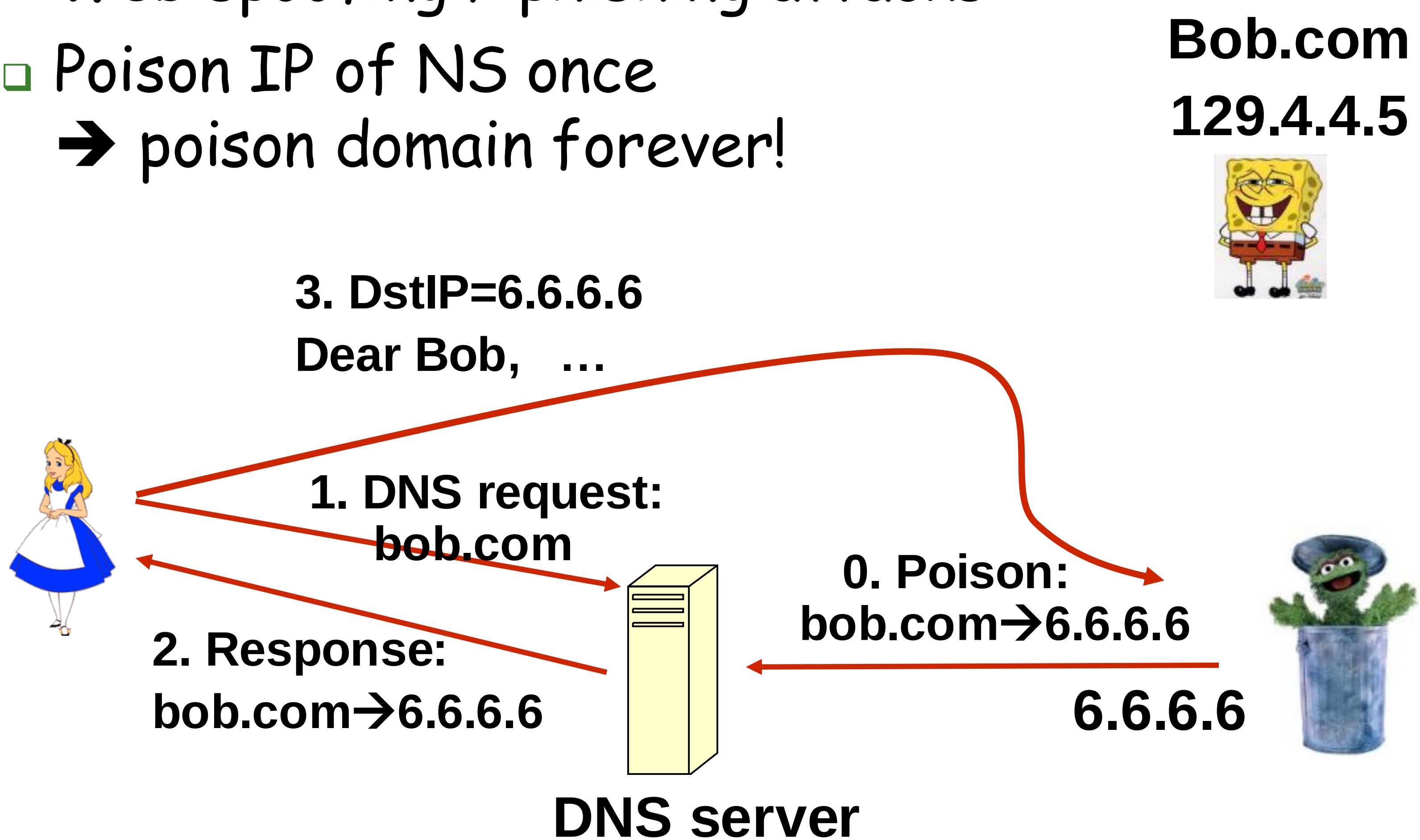
DoS

Fake certificates, Passwords, more….

Block updates

MITM via DNS Poisoning

- Allows offpath attacker to become MITM
  - Web spoofing / phishing attacks
  - Poison IP of NS once
    ➔ poison domain forever!



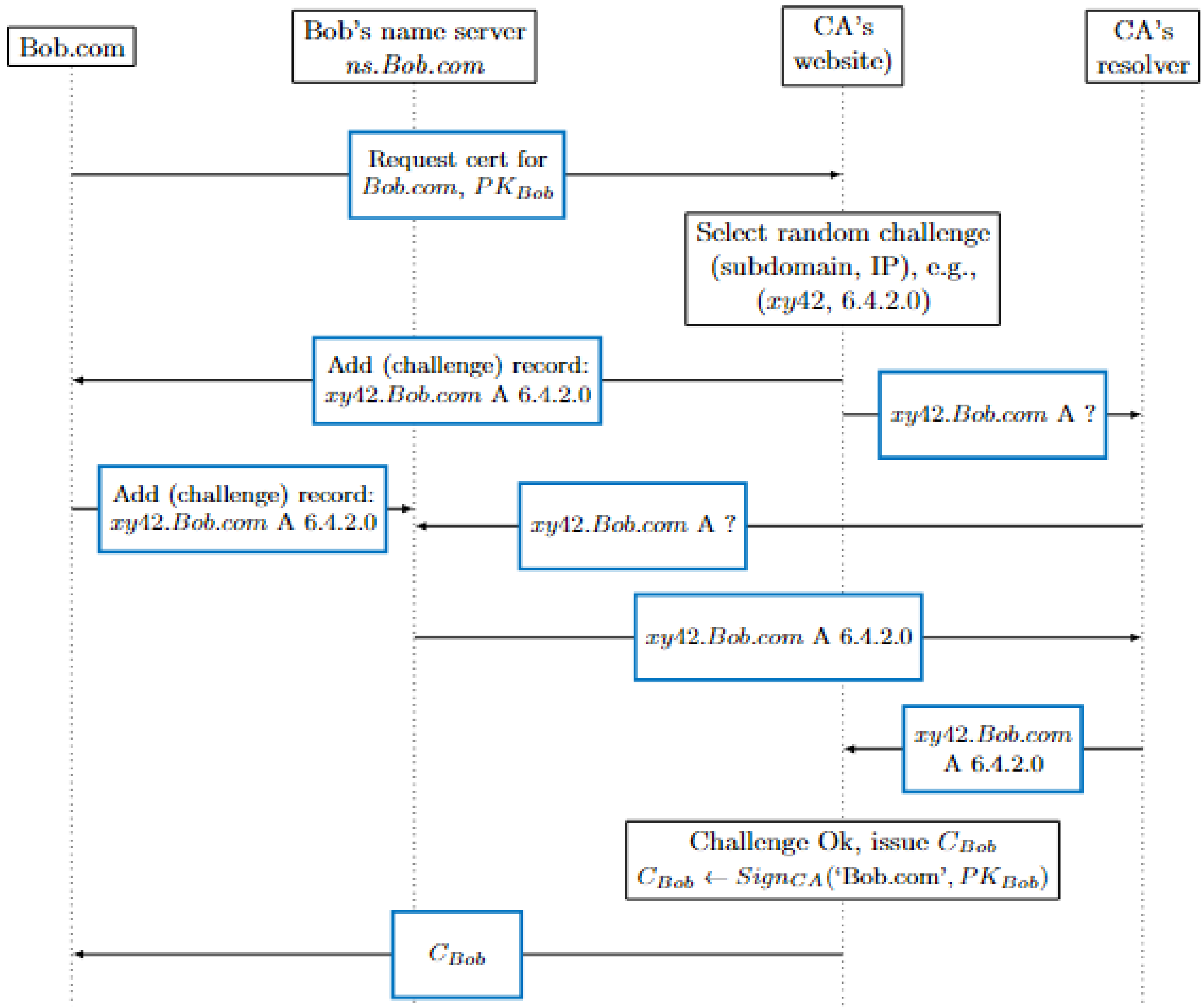
Steal Password by Poisoning

- Password recovery procedure exploit
  - Attacker poisons MX record used by webserver

Poisoning to get a rogue certificate certified

- A CA signs a certificate authenticating pk for domain
- The CA should verify that the request is valid
- Domain validation: validate control over the domain
- Poison the CA's resolver circumvents domain validation
- Let's first see how domain validation works…

Domain Validation

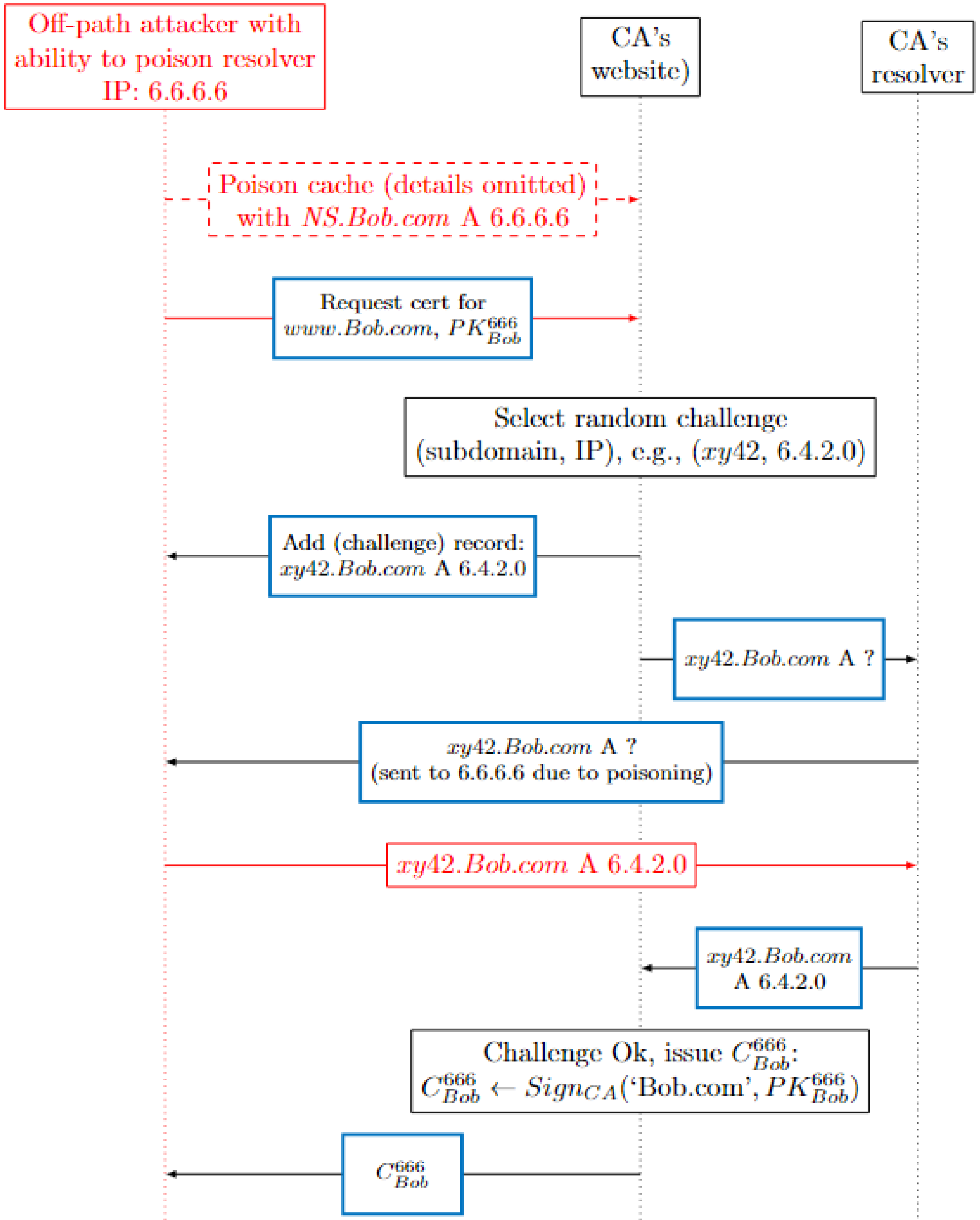


Attack

Tricking a domain-validating CA to issue a fake certificate for www.Bob.com by DNS poisoning, against domain-validating CA.

We show later **how** the poisoning is done.

Exercise: draw a sequence diagram showing how the off-path attacker can exploit C_{Bob}^{666}. You may assume that the attacker can DNS-poison additional resolvers.

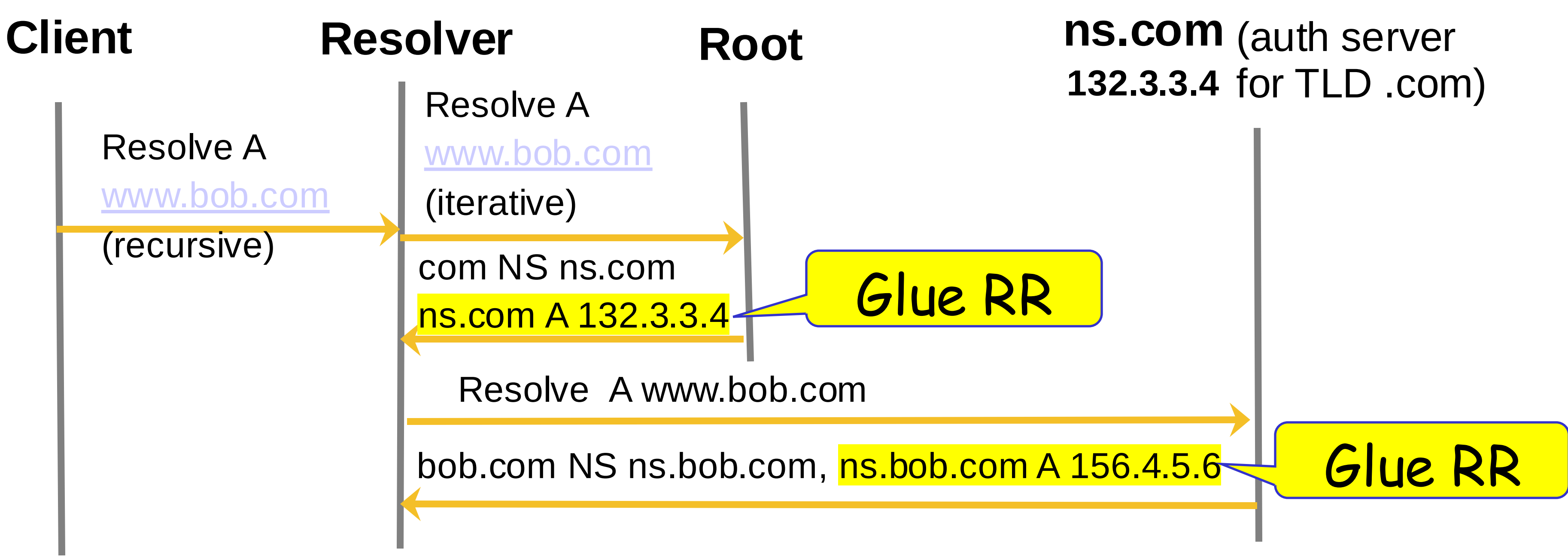


DNS Poisoning: How?

- DNS Poisoning: `inject' fake DNS RR
- **Method 1 (historical):** by Gratuitous `glue' RR
  - e.g. query A www.eve.com, response: eve.com NS download.com and download.com A 6.6.6.6
  - **Bailiwick Rule**: allow answers only for subdomains
    - **ns**.eve.com can't answer for download.com
    - Some resolvers cache combo: eve.com NS-IP 6.6.6.6
- Method 2: send from corrupt NS
  - Some referral-chains are very long
    - Transitive trust relationships
  - Some (many?) NSs run vulnerable versions!
- Method 3: send spoofed DNS response
- Method 4: dangling DNS records

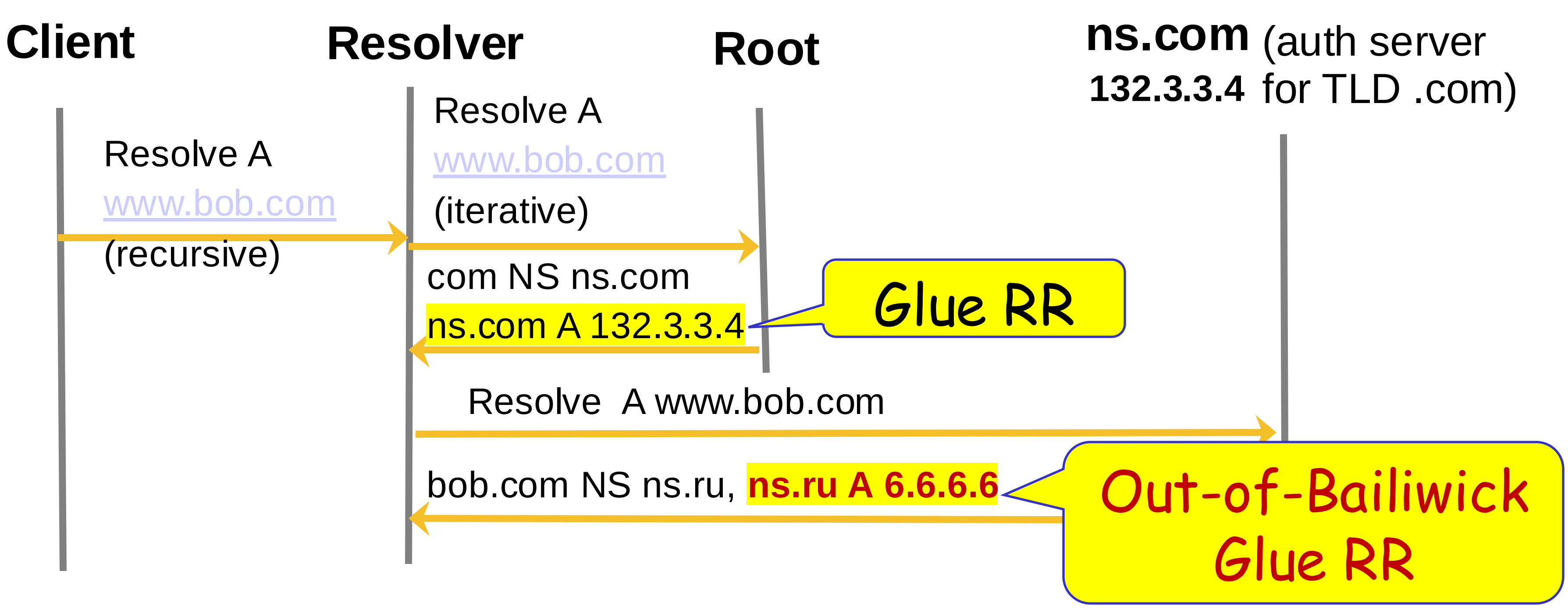
Gratuitous glue RR in Responses

- Normally: RR is received to fulfil request
- Gratuitous RR: received **without** request
  - In response to different request, or appended to a request
- Main use: glue (A) RR , sent with a referral (NS)

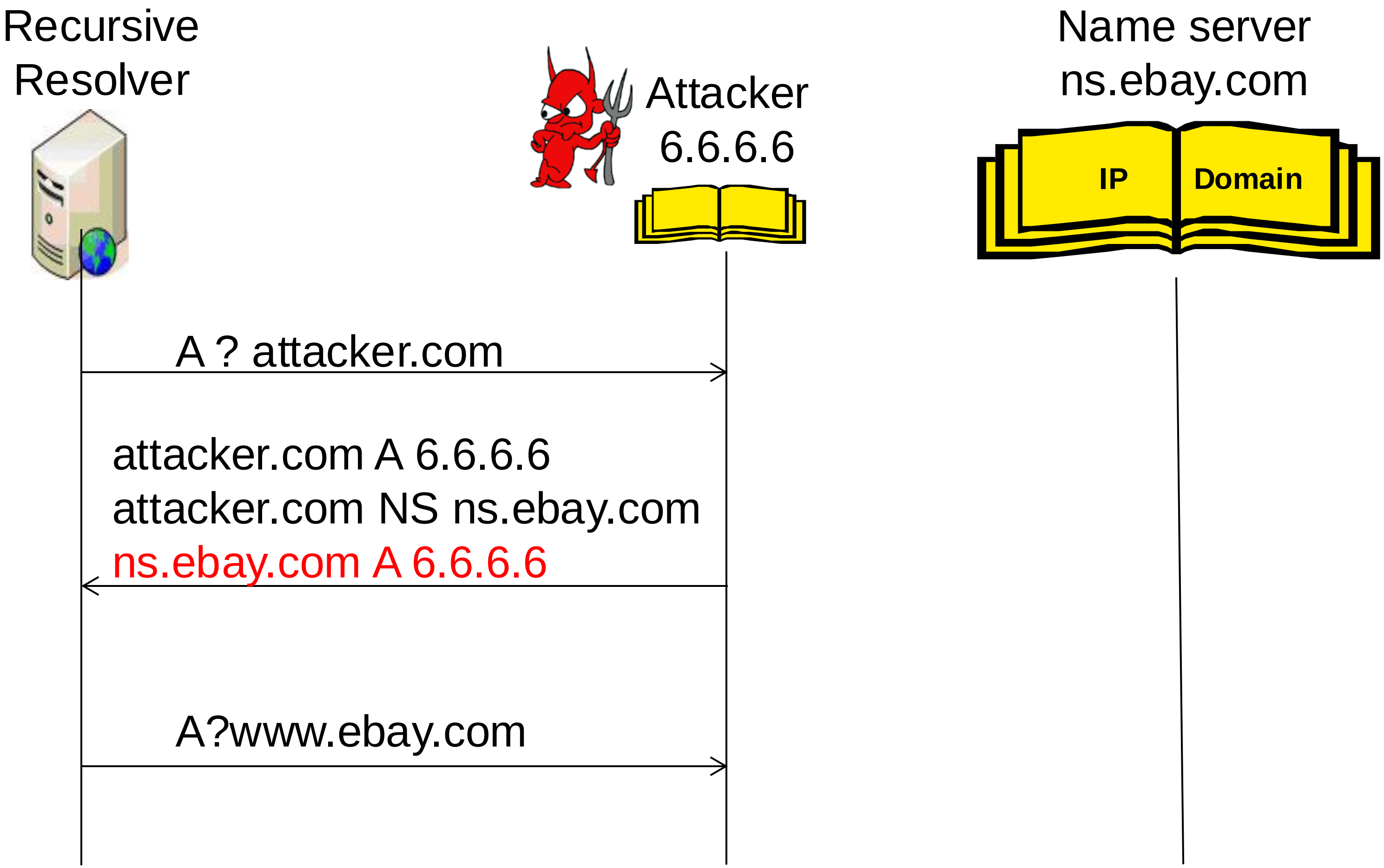


Out-of-Bailiwick Gratuitous Glue RR

- **Out-of-Bailiwick Gratuitous RR**: a RR for one domain, received from a name server for another
- **Can be abused for DNS cache poisoning!**
  - Out-of-bailiwick RRs ignored since ~1997



Out of Bailiwick Attack [historical]



Bailiwick Rule: Accept Records Only within Bailiwick

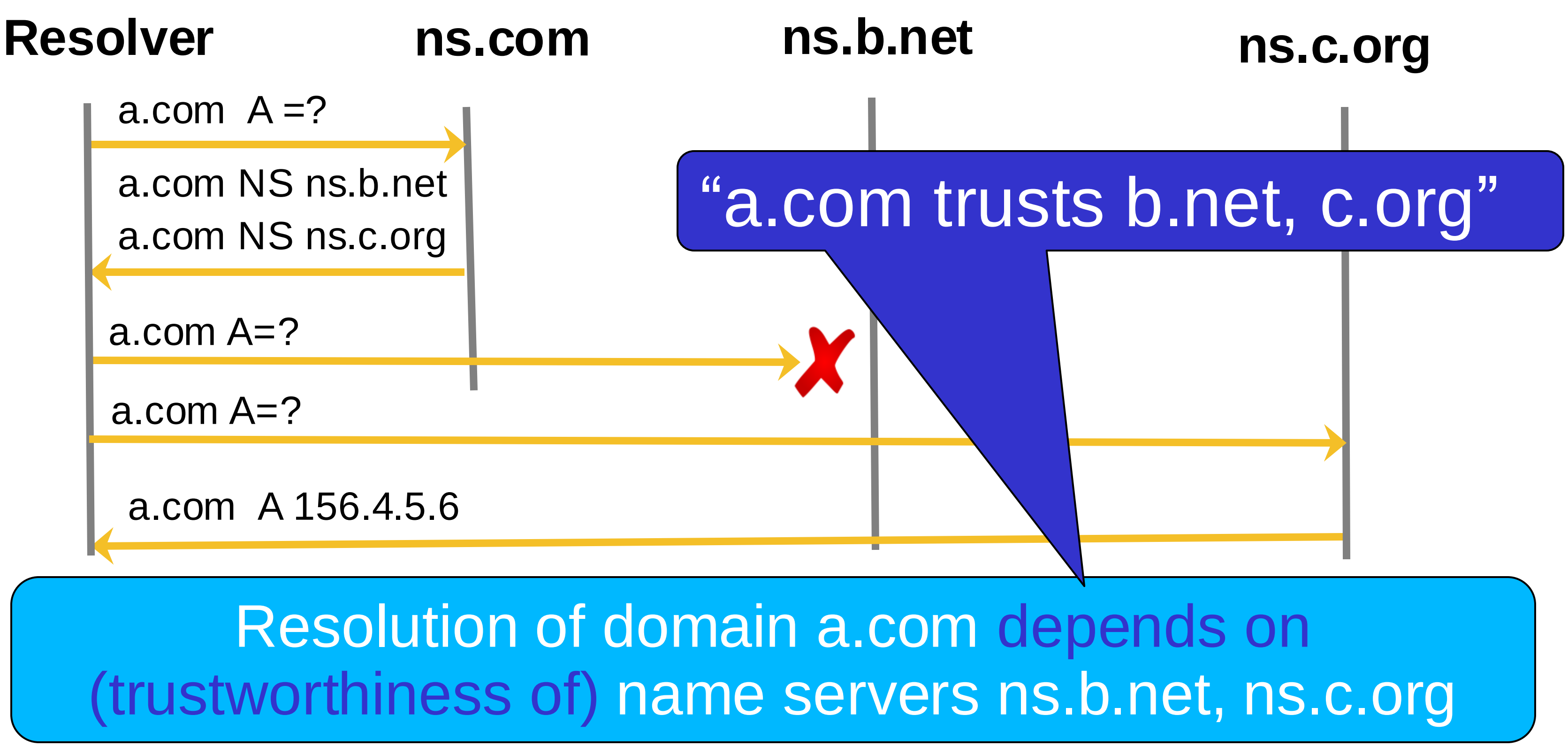
- Prevents out-of-Bailiwick attack
- A Name Server can respond for records within its domain
  - Root servers can return any record
  - TLD server can return anything within that TLD
  - ns.bank.com can return anything for bank.com
- Resolvers ignore out-of-bailiwick responses
  - Some resolvers use out-of-bailiwick for glue (resolution of NS), and do not cache it (no impact on other queries)
  - Or, cache combo, e.g.: eve.com NS-IP 6.6.6.6

DNS Poisoning: How?

- DNS Poisoning: `inject' fake DNS RR
- Method 1 (historical): by Gratuitous `glue' RR
  - e.g. query *A* www.eve.com, response: eve.com NS download.com and download.com *A* 6.6.6.6
  - **Bailiwick Rule**: allow answers only for subdomains
    - **ns**.eve.com can't answer for download.com
    - Some resolvers cache combo: eve.com NS-IP 6.6.6.6
- **Method 2: send from corrupt NS**
  - Some referral-chains are very long
    - **Transitive trust** relationships
  - Some (many?) NSs are vulnerable (e.g., unpatched)
- Method 3: send spoofed DNS response
- Method 4: dangling DNS records

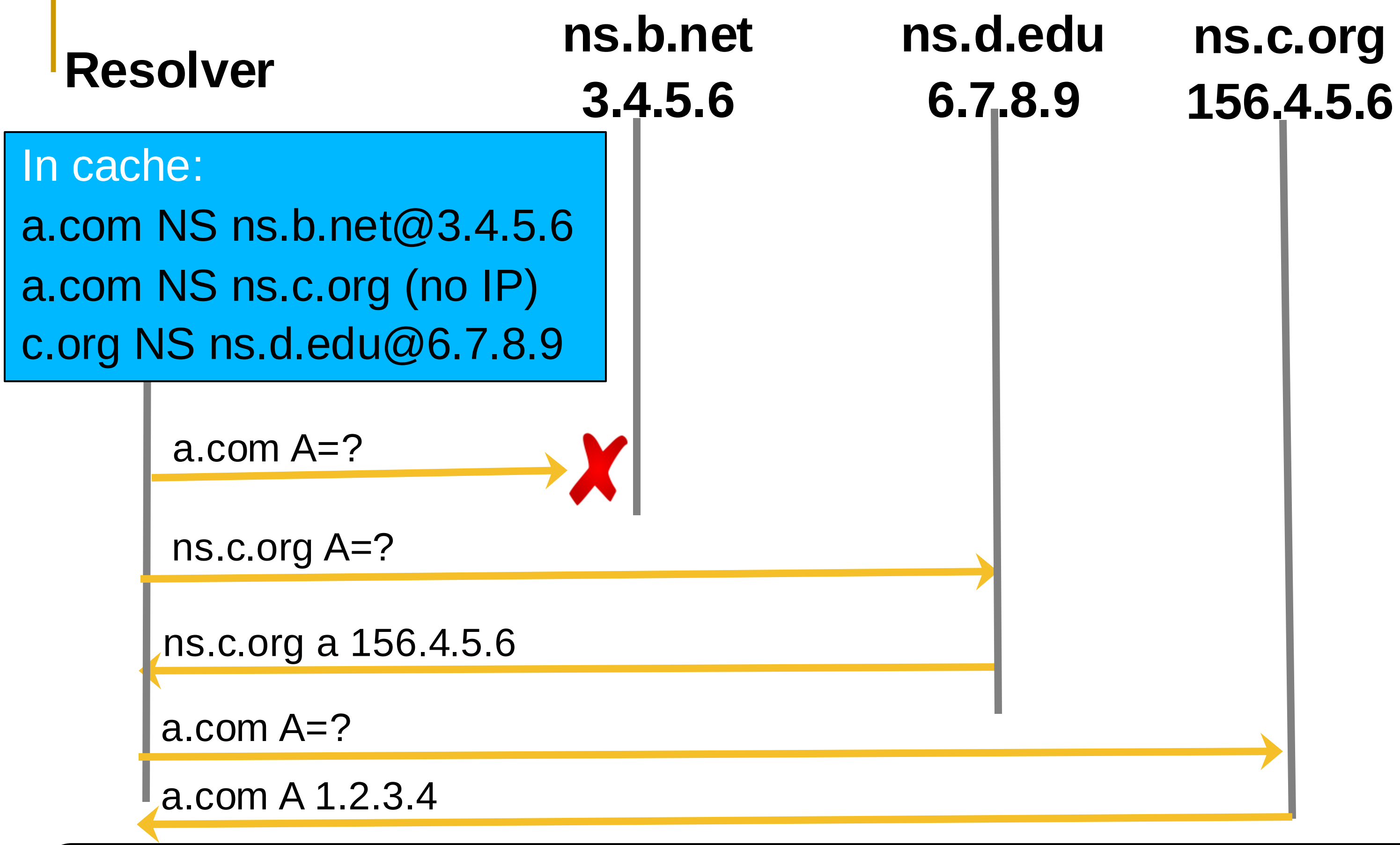
DNS Trust Relationships

- Often, some name servers of a domain are from other domains (for backup, performance)
  - a.com has name server (also) in b.net
  - Resolvers identify & use the 'best performing' name server



DNS: Transitive Trust

- Resolver may need to find IP of name servers, too



a.com trusts b.net, c.org and c.org trusts d.edu
➔ a.com also trusts d.edu

DNS: Transitive Trust Attack

- Off-path attacker, can drop packets [how?]
- Controls 'transitively-trusted' NS, e.g., ns.666

Resolver | ns.b.net | ns.d.edu | ns.666 6.6.6.6

In cache:
a.com NS ns.b.net + glue
a.com NS ns.c.org (no IP)
c.org NS ns.d.edu + glue
c.org NS ns.666@6.6.6.6

a.com A=?

ns.c.org A=?

ns.c.org A=?

ns.c.org a 6.6.6.6

a.com A=?

a.com A 6.6.6.6

Often, domains `trust' other domains, creating (often long) 'chains of trust'…

Attacker may lead to a vulnerable name server, e.g., by DoS on non-vulnerable name servers

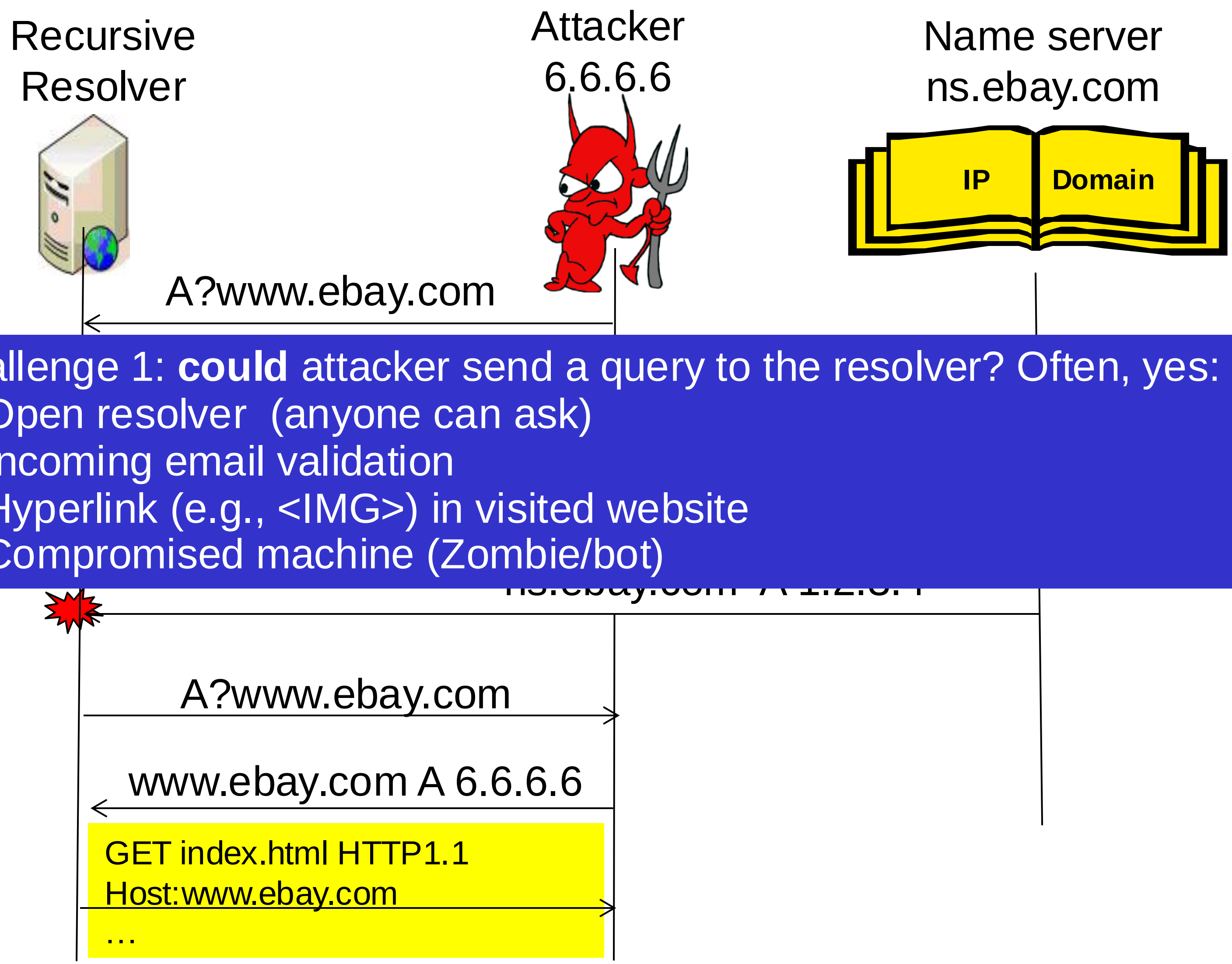
Domain Name System (DNS) Security

- DNS: quick recap
- DNS poisoning
  - Motivation: the hacker's swiss knife
  - Method 1 (historical): by Gratuitous `glue' RR
  - Method 2: send from corrupt NS
  - **Method 3: send spoofed DNS response**
  - Method 4: dangling DNS records
- DNSSEC: Cryptographic security for DNS
- DNS Privacy issues and defenses

Off-path DNS poisoning attacks

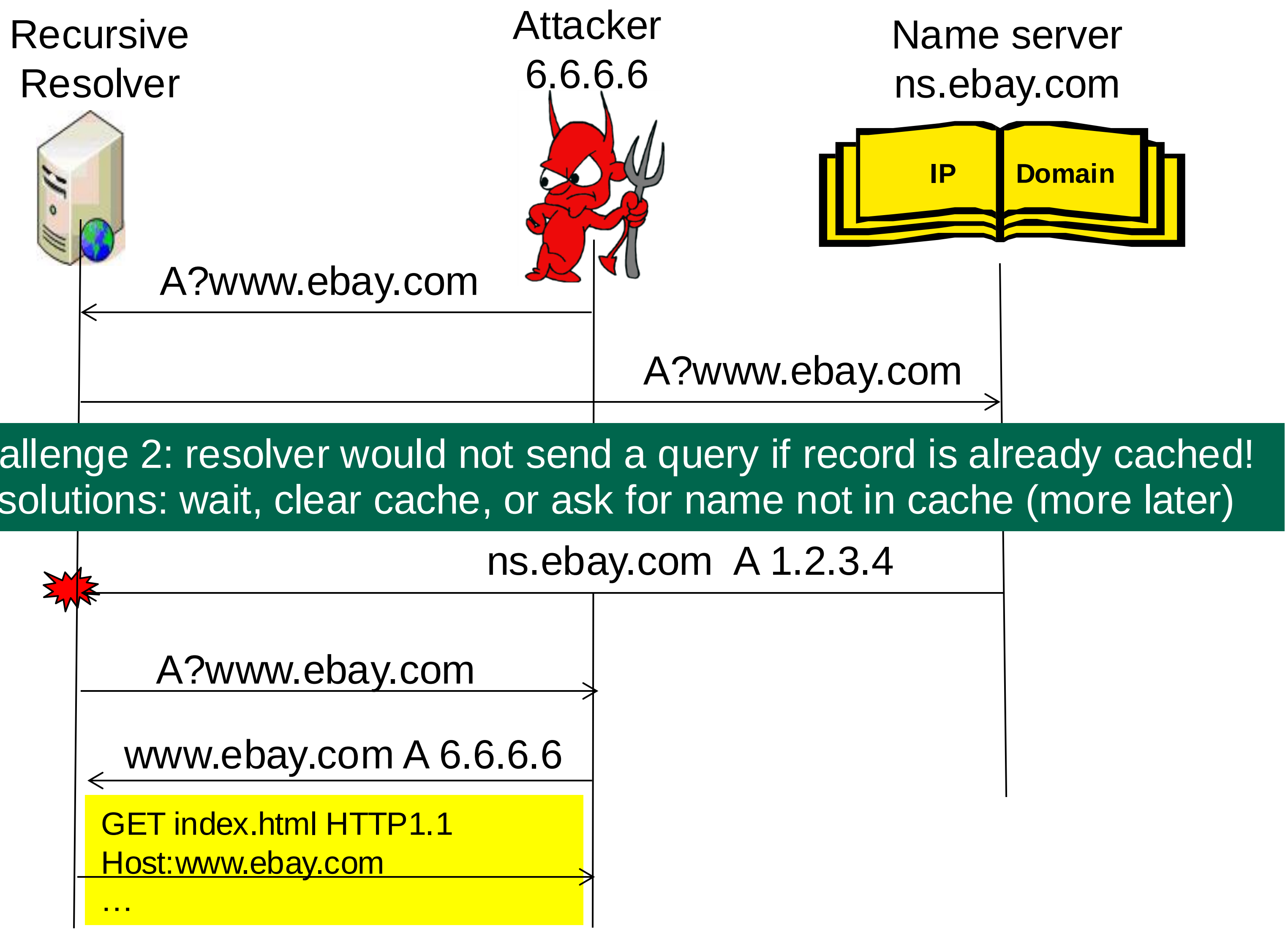
- DNS is clearly vulnerable to MitM attacks
  - Against MitM, use DNSSEC (crypto-defense)
  - Or… **assume attacker is only off-path**
- But… DNS vulnerable to off-path attacks too!
  - Motivating DNSSEC: use signatures, hashing (against MitM!)
- We present **Spoofed-response attacks**:
    - Concept and challenges
    - Kaminisky's attack
    - Improved DNS security ('post-Kaminsky' defenses)
    - Source port de-randomization attacks ('post-Kaminsky' attacks)

DNS Poisoning by Spoofed Response



Challenge 1: **could** attacker send a query to the resolver? Often, yes:
- Open resolver (anyone can ask)
- Incoming email validation
- Hyperlink (e.g., <IMG>) in visited website
- Compromised machine (Zombie/bot)

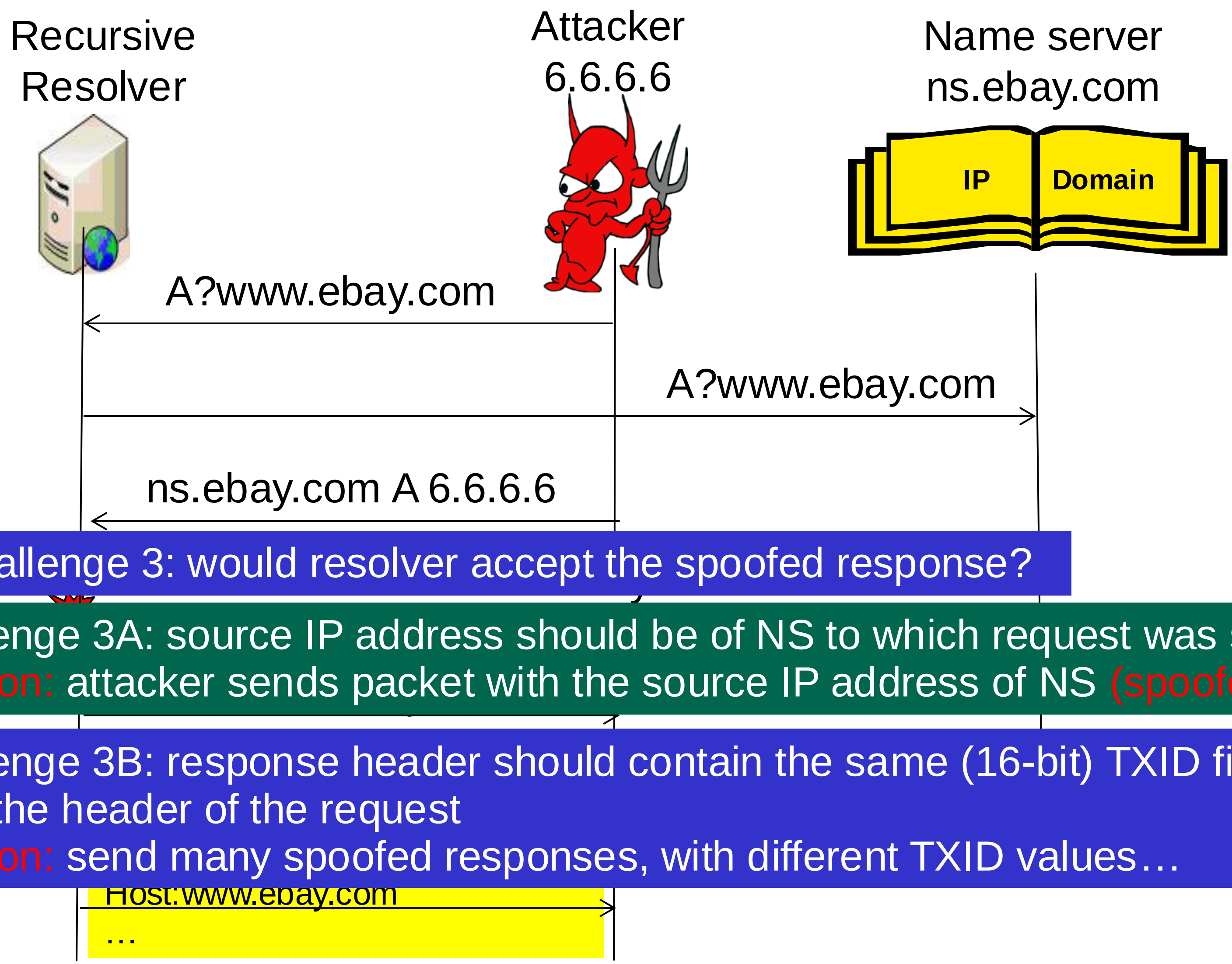
DNS Poisoning by Spoofed Response



Challenge 2: resolver would not send a query if record is already cached!

- solutions: wait, clear cache, or ask for name not in cache (more later)

DNS Poisoning by Spoofed Response



Challenge 3: would resolver accept the spoofed response?

Challenge 3A: source IP address should be of NS to which request was sent!
Solution: attacker sends packet with the source IP address of NS (spoofed)

Challenge 3B: response header should contain the same (16-bit) TXID field as in the header of the request
Solution: send many spoofed responses, with different TXID values…

DNS Poisoning by Spoofed Response

- Attacker sends many responses with different TXID values
- But, **must win the race**:
  - Legit response surely has correct TXID
  - Would the legit response arrive before the 'lucky' spoofed response (with correct TXID)?
  - First correct response is accepted <u>and cached</u>
    - For TTL time (minute, hour, day,...)
      - TTL is the duration of validity of a given mapping
  - Subsequent, incorrect D are <u>ignored</u>

➔ Winning the race considered impractical…
Until Kaminsky's attack [2008]
RIP Dan Kaminsky: 1979-2021

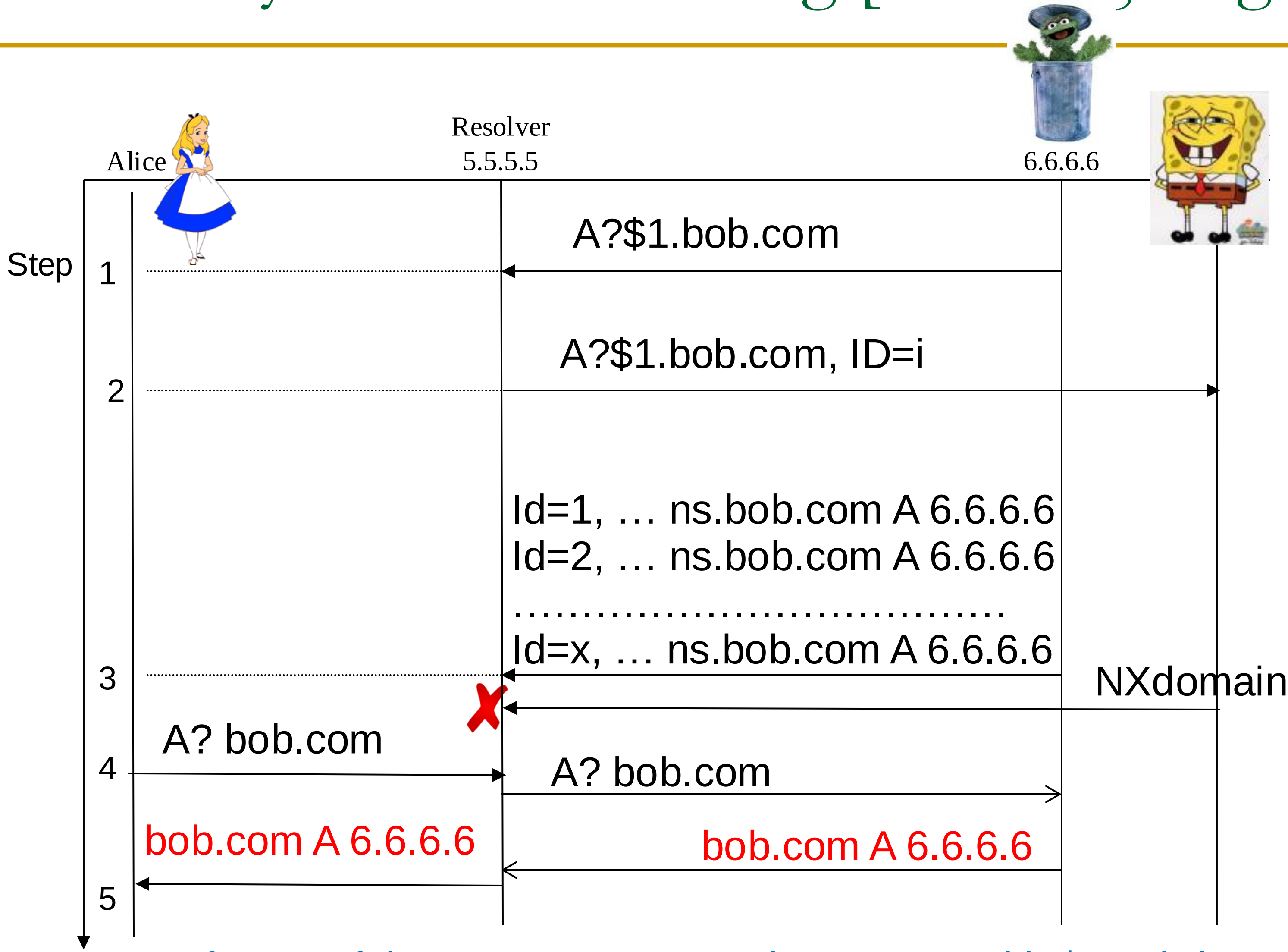
Kaminisky's attack: four ideas

- Challenge 1: ensuring response not in cache
- Idea 1: query is for a non-existing domain ➔ surely not in cache!
- Challenge 2: only a limited number of spoofed response can arrive before the legit response
- Idea 2: no problem, just try again with a different non-existing domain!
- Challenge 3: how to exploit poisoning response to a non-existing domain?
  - Exercise: steal cookies, even with 'secure' attribute (get a cert)
  - Kaminsky's (3rd) idea: **poisoned glue**

Kaminsky's 3rd Idea: poisoned glue (or NS)

- Query: 'A' record for (say) 585.bank.com
- Spoofed response:
  - Option 1, poisoned glue:
    Answer section: empty (or an answer)
    Authority section: bank.com NS ns.bank.com
    Glue (additional section): ns.bank.com A 6.6.6.6
  - Option 2, poisoned NS:
    Answer section: empty (or an answer)
    Authority section: bank.com NS ns.666.com
    Additional section: (none or a glue)
- Caching ns.bank.com A 6.6.6.6 or bank.com NS ns.666.com ➔ entire bank.com domain poisoned
- Would they cache? Depends on resolver
  - Cautious resolvers don't cache referral glue and NS RRs
  - Other spoofing payloads, e.g., use CNAME [KSW'17]

Kaminsky's DNS Poisoning [showing single query]

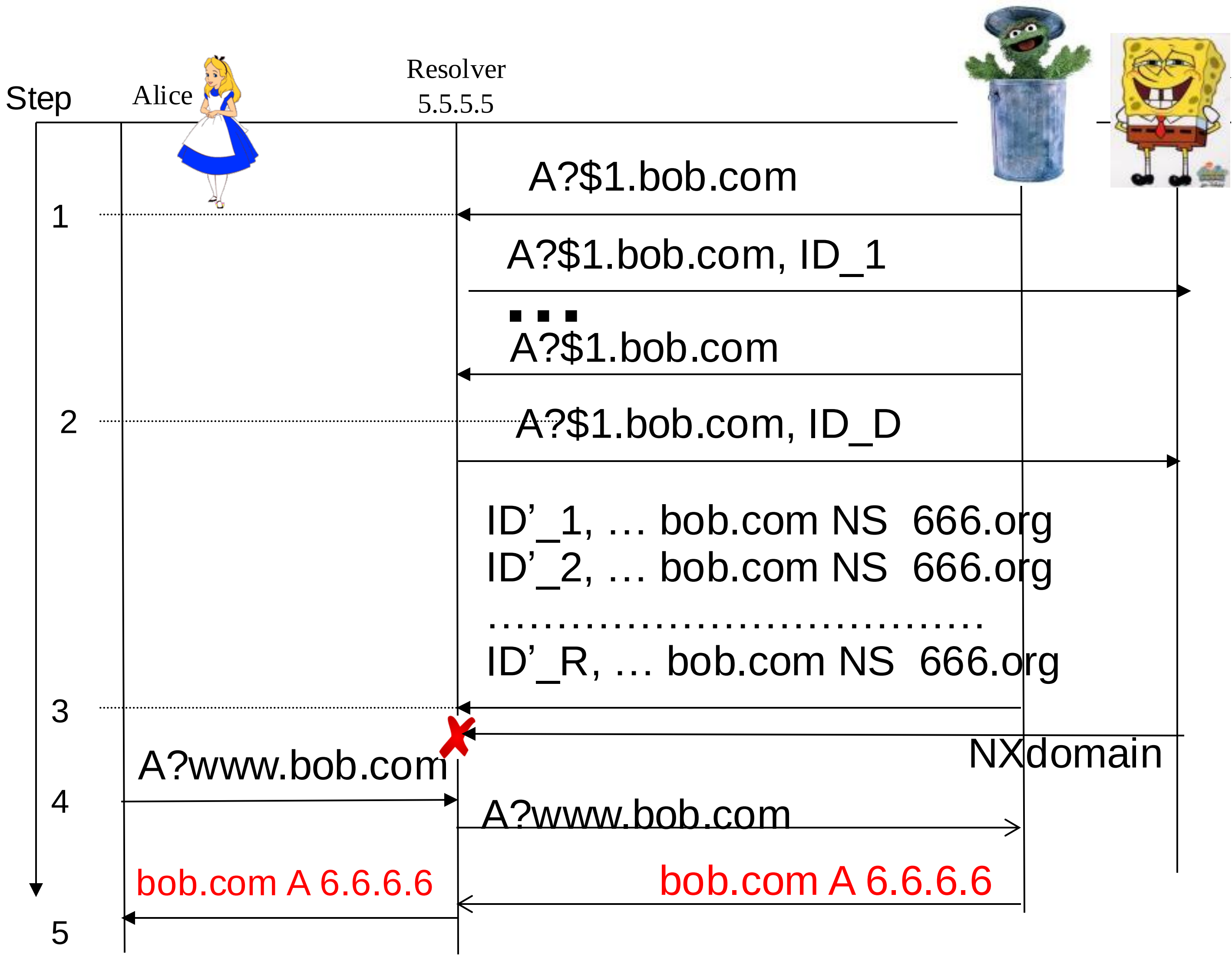


If none of the x responses match… repeat with $2 subdomain!
Actually, Kaminsky had one more idea!

4^{th} idea: duplicate requests → birthday!

- Attacker sends D duplicate requests
- All for <u>exactly the same query</u> (domain)
- Some resolvers send D <u>identical requests</u>
- Probability of a response to match: $D/_I$
  - Match: 16-bit DNS-ID, ports, IP as in a request
  - If only DNS-ID is random: $I = 2^{16}$
- Attacker sends R responses
- Probability of one of them to match?
- Birthday paradox: $p \lesssim \min\left(1, \frac{D \cdot R}{I}\right)$

Kaminsky's Birthday Attack (duplicate requests)



Defenses against Kaminsky's Attack

- Don't cache referral glue and NS RRs
- RFC 5452: Local server must validate as follows…
- Same question section as in request
- Response received within reasonable delay
- Ignore if already received valid response for query
- Same (16-bit) ID field
  - Local server must choose ID randomly
- Resp(Src.IP)=Req(Dst.IP)
  - Most domains have 1 to 3 likely-to-be-used name servers from a given resolver
- **Resp(Dest.IP,Dest.port)=Req(Src.IP,Src.port)**
  - **Main Defense: Source Port Randomization (SPR),** i.e., resolver selects random source port
  - Preferably, also randomize source IP

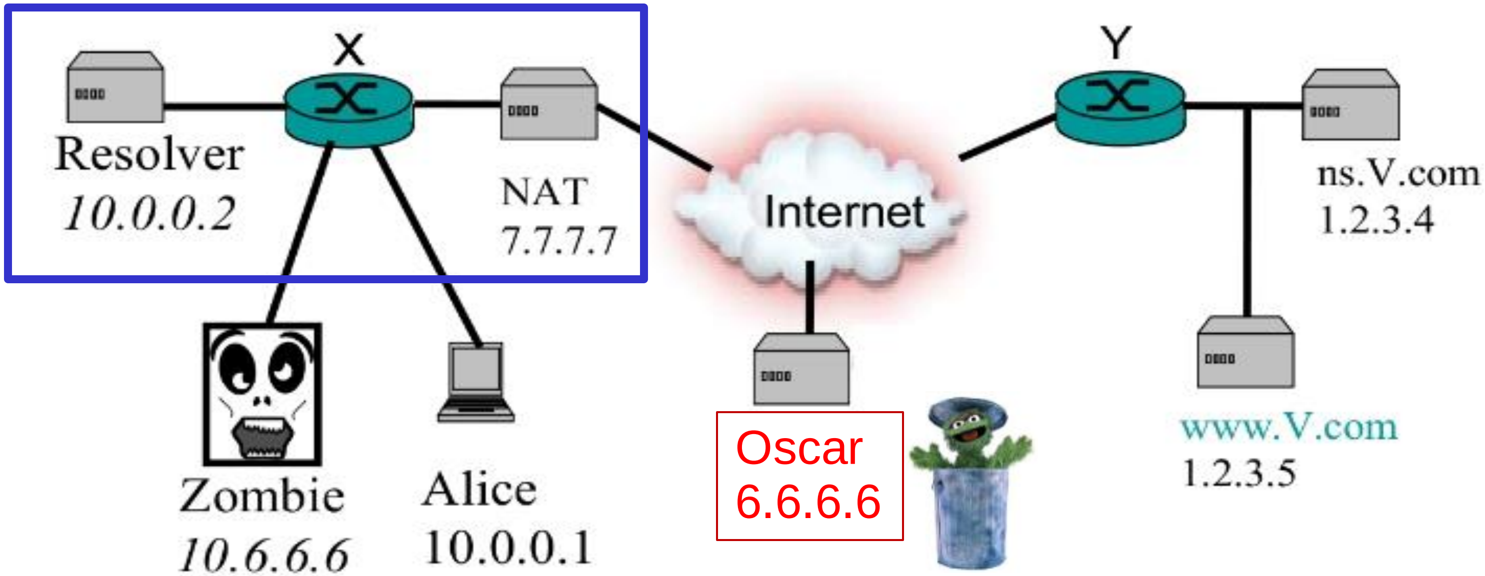
Source Port Randomization (SPR)

- Send requests from random/unpredictable ports [Bernstein2002, RFC5452]
  - Port field - 16 bits
  - Increases the search space: 16 bit ID, 16 bit port → ~ 2^{32}
- Makes Kaminsky's attack less practical
  - Birthday attack:
    2^{16} requests + 2^{16} responses → poison with prob. ~ ½
  - Many (most?) resolvers also prevent/limit birthday attack
    - Detect new query identical to pending one → don't resend

Source Port Randomization (SPR)

- Send requests from random/unpredictable ports [Bernstein2002, RFC5452]
  - Port field - 16 bits
  - Increases the search space: 16 bit ID, 16 bit port → ~ 2^{32}
- Makes Kaminsky's attack less practical
  - 2^{16} requests + 2^{16} responses → poison with prob. ~ ½
  - Many (most?) resolvers also prevent/limit birthday attack
  - Reduced motivation to deploy DNSSEC ? ☹
- Several Source Port De-Randomization attacks
  - Next: resolver behind NAT source port de-randomization
  - In IP security: fragmentation-based de-randomization
  - In papers: other source port de-randomization attacks
    - Recent, very effective: SadDNS attack(s); and others

Resolver behind NAT

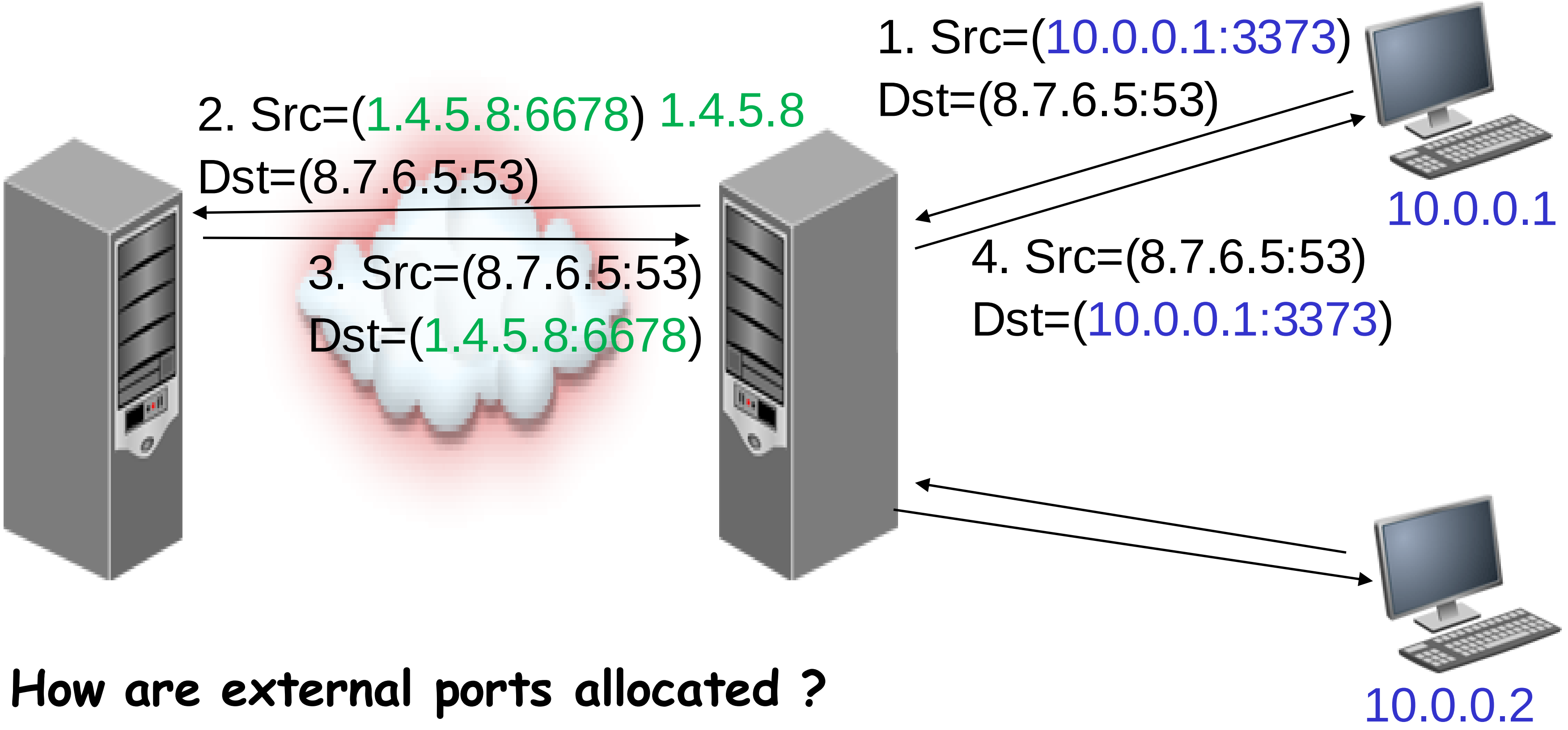


Typical home/office network:

- Includes a resolver and different hosts, some possibly `zombies' (controlled by attacker)
- Connected via a NAT device (IP: 7.7.7.7)
- Just in case, let us explain what's a NAT

NAT: Network Address Translation

Goal: share IP addresses among multiple hosts in Net
Net uses private ranges IPs (10.0/8, 172.16/12, 192.168/16)
NAT uses one `external' IP; maps internal (IP:port) pairs to an external port



How are external ports allocated ?

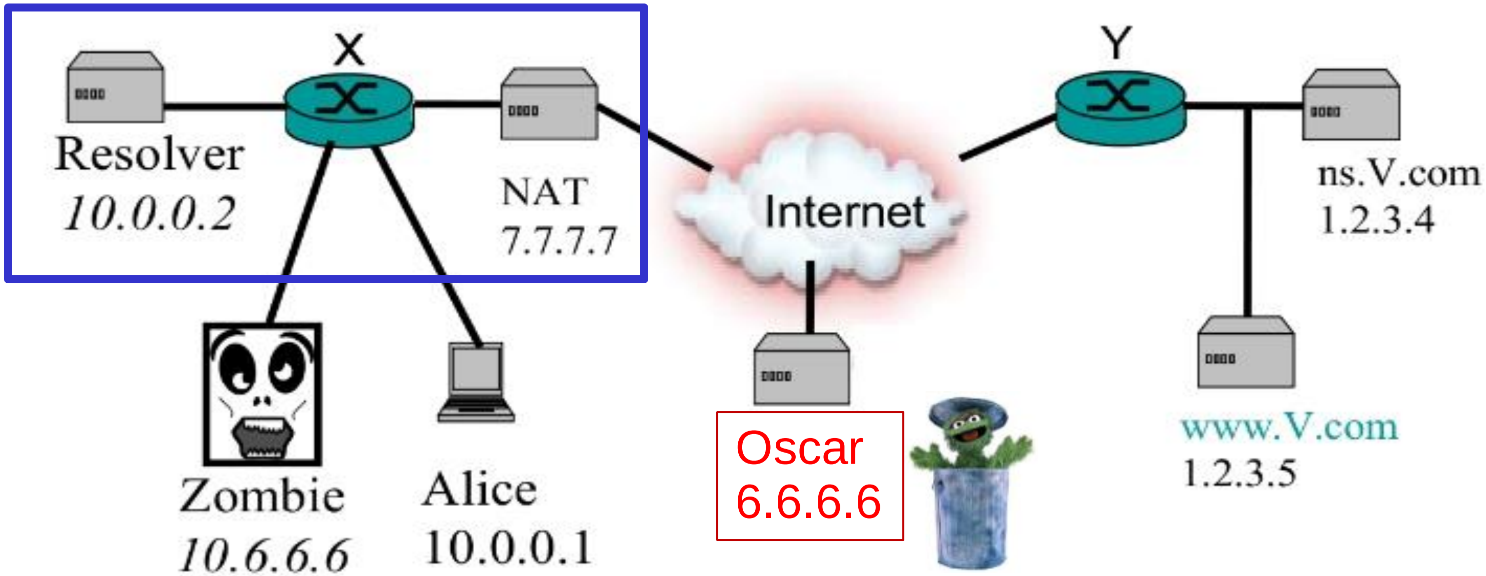
NAT: External Port Allocation Methods

- Goal: different external ports per connection
  - Assume up to $2^{16} = 65536$ concurrent connections
- Naïve solution 1: on the p_x connection to x, use port p_x
  - Where x=(IP:port) is a destination
- Disadvantages?
  - Need to maintain p_x forever: restart p_x if not used/saved
  - Security concern?
  - Attacker can guess value of p_x
  - This foils source port randomization (SPR)
    ➔ attacker can use Kaminiski's attack (against SPR resolver)
- Naïve solution 2: select a random port for each connection
  - Disadvantage?
  - Collisions (birthday paradox: after __ concurrent connections)

Per-Dest-Incrementing Port Allocation

- Goal 1: different external ports per connection
- Goal 2: off-path attacker cannot predict next port
- Naïve solution 3: select random ports, remember ports in use (avoid collisions)
  - Inefficient (remember all ports)
- Alternative: Per-Dest-Incrementing port allocation:
  - On connection to x, increment port p_x (mod $2^{16} = 65536$)
  - If no current p_x, select a random (or pseudo-random) value
    - One way to select pseudo-random $p_x \leftarrow PRF_k(x||time)$
    - A common simplification using hash: $p_x \leftarrow h(k||x||time)$
  - Very common port-allocation method (e.g., Linux)

Derandomizing SPR: Resolver behind NAT

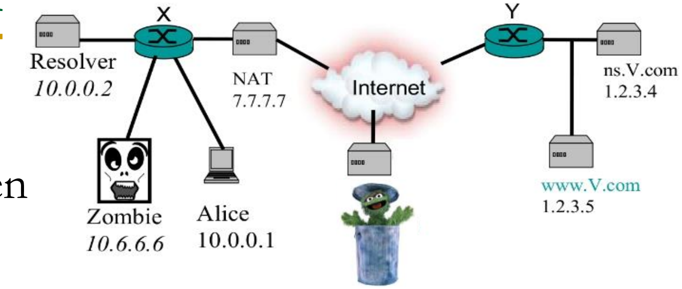


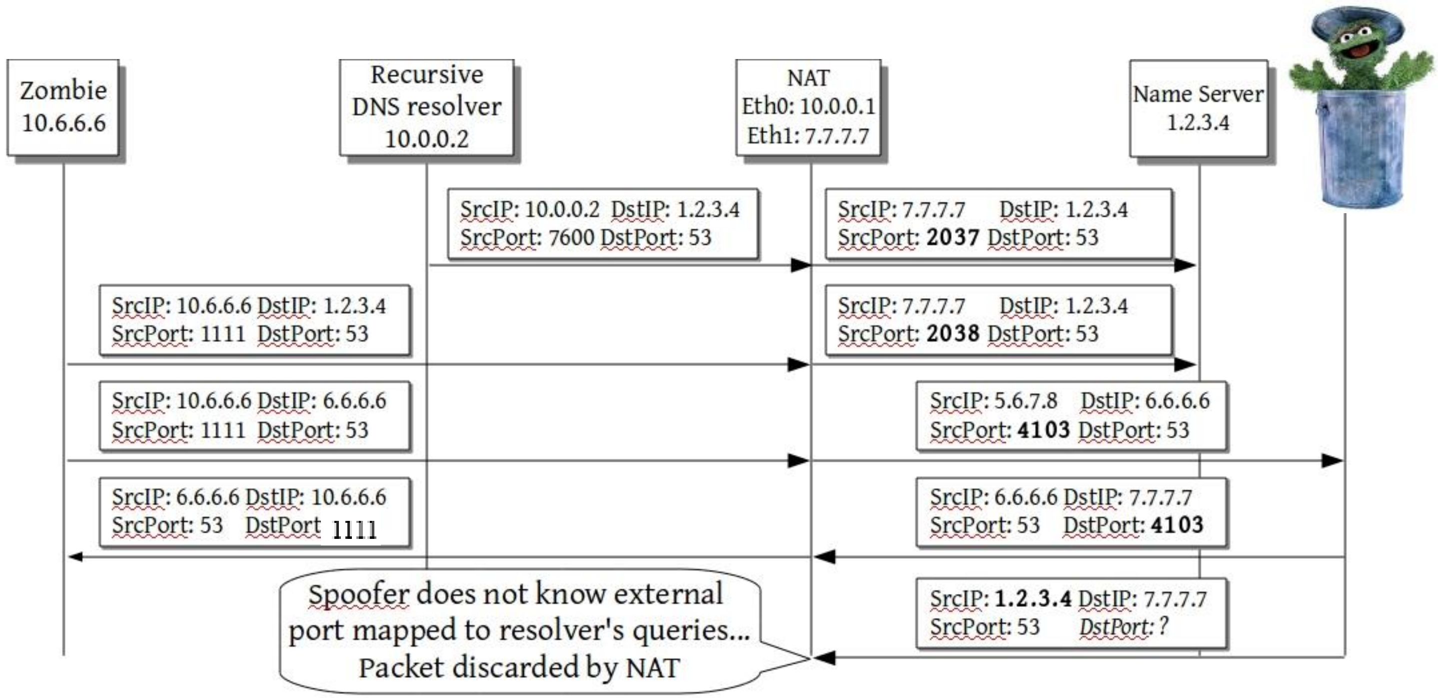
Typical home/office network:

- Connected via a NAT device (IP: 7.7.7.7)
- Attack assumptions:
  - Network includes a Zombie - compromised host
  - **NAT uses per-dest-incrementing port allocation**
    - A (less efficient) variant works without this assumption
    - Even more efficient variant for globally-incrementing ports

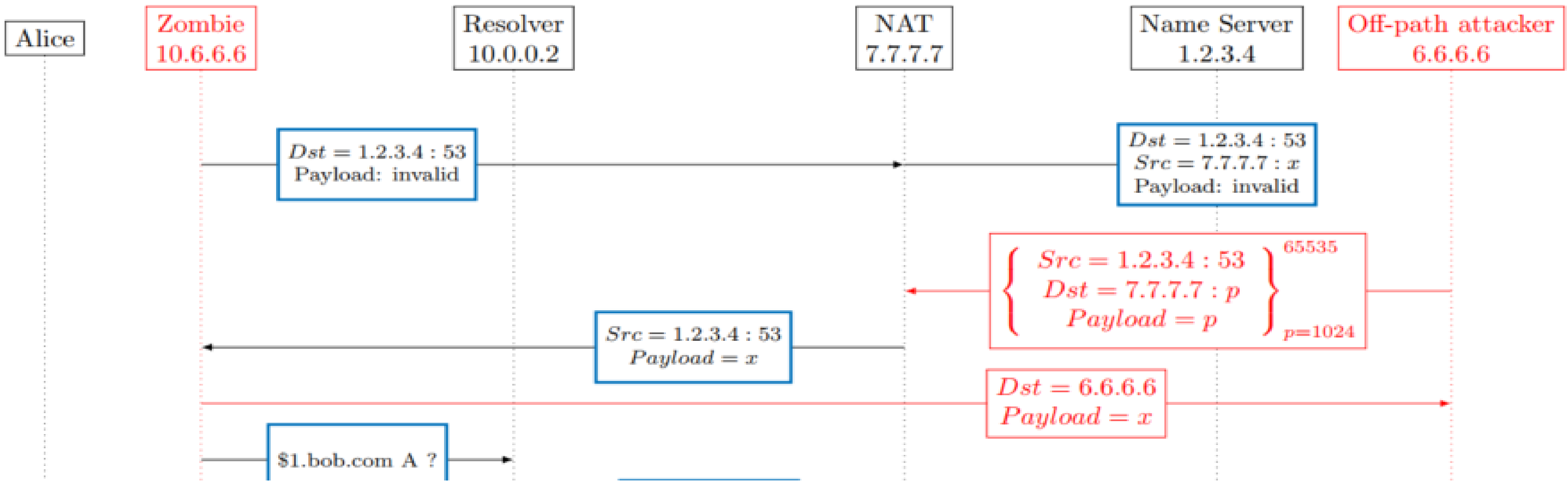
Ports for queries via NAT

- For per-dest incrementing ports NAT
  - For each destination: select first port at random, then choose next available port
- Can attacker predict port assigned by NAT?

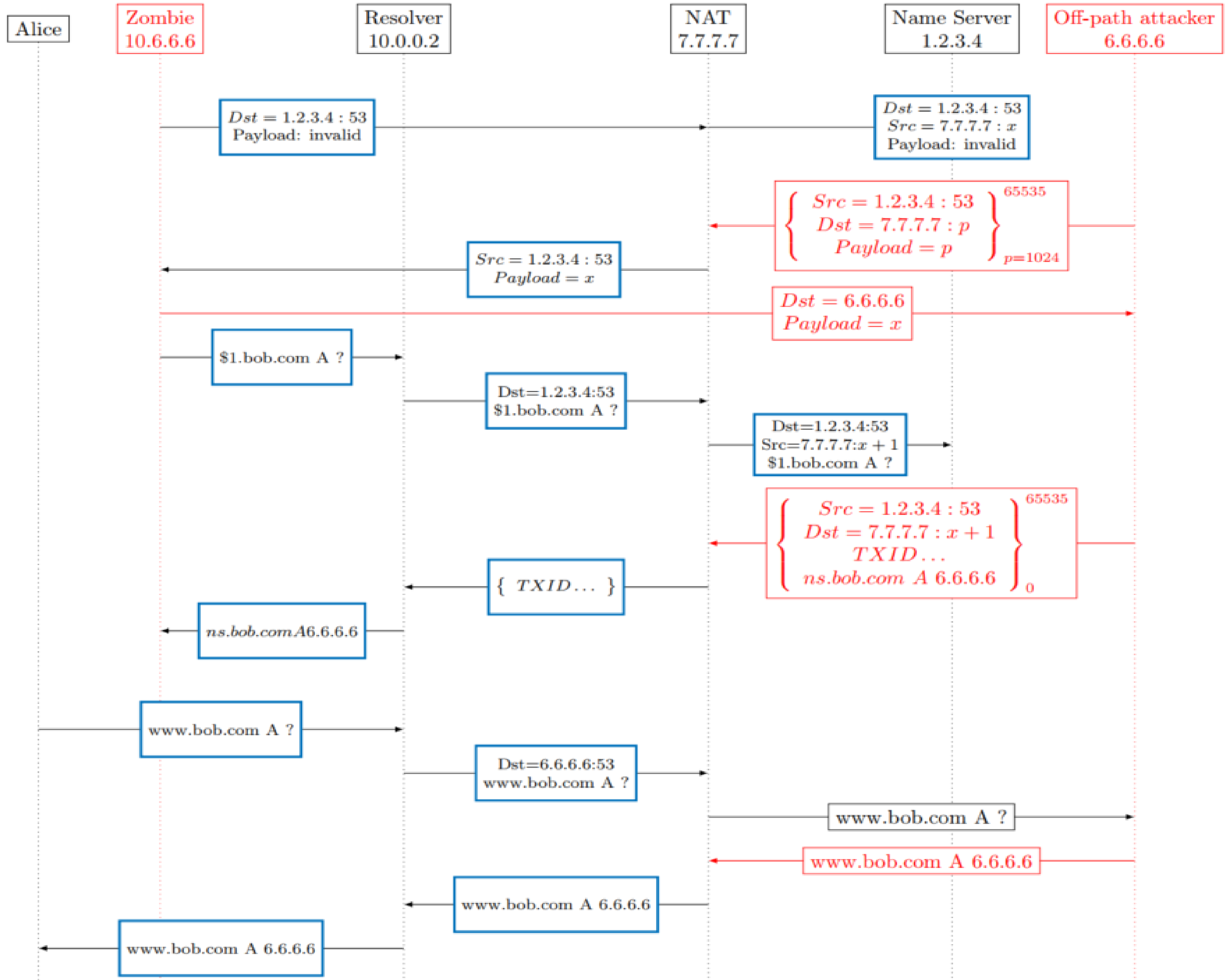




Source Port deRandomization



- Zombie sends a packet to port 53 of the name server
- NAT allocates port x to port 53 of the name server
- Spoofer sends a packet with payload p to each port $p \in \{1, \ldots, 65536\}$
- Only the one sent to port x gets through to zombie (with payload x)
- Zombie echoes current port (x) to attacker
- Attacker continues with Kaminsky's attack (and known source port!)



Alice
Zombie 10.6.6.6
Resolver 10.0.0.2
NAT 7.7.7.7
Name Server 1.2.3.4
Off-path attacker 6.6.6.6
Dst = 1.2.3.4 : 53
Payload: invalid
Dst = 1.2.3.4 : 53
Src = 7.7.7.7 : x
Payload: invalid
Src = 1.2.3.4 : 53
Dst = 7.7.7.7 : p
Payload = p
65535
p=1024
Src = 1.2.3.4 : 53
Payload = x
Dst = 6.6.6.6
Payload = x
$1.bob.com A ?
Dst=1.2.3.4:53
$1.bob.com A ?
Dst=1.2.3.4:53
Src=7.7.7.7:x + 1
$1.bob.com A ?
Src = 1.2.3.4 : 53
Dst = 7.7.7.7 : x + 1
TXID ...
ns.bob.com A 6.6.6.6
65535
0
{ TXID ... }
ns.bob.com A6.6.6.6
www.bob.com A ?
Dst=6.6.6.6:53
www.bob.com A ?
www.bob.com A ?
www.bob.com A 6.6.6.6
www.bob.com A 6.6.6.6
www.bob.com A 6.6.6.6

Domain Name System (DNS) Security

- DNS: quick recap
- DNS poisoning
  - Motivation: the hacker's swiss knife
  - Method 1 (historical): by Gratuitous `glue' RR
  - Method 2: send from corrupt NS
  - Method 3: send spoofed DNS response
  - **Method 4: dangling DNS records**
- DNSSEC: Cryptographic security for DNS
- DNS Privacy issues and defenses

Dangling DNS records

- Recall: several DNS records map resources to domain names, including CNAME, NS and MX, e.g.:
  - foo.com NS ns1.CDN.net
  - www.fee.com CNAME server765.cloud.com
- Dangling DNS record: domain freed, record remains
  - E.g., using a new NS: foo.com NS ns.foo.com
  - But forgot to remove old NS record to ns1.CDN.net
    - Or old NS records is still in cache of resolvers
- Attacker may get control of old domain (ns1.CDN.net)
  - E.g., when domain name allocated by CDN/cloud to customers
  - Allows to map resource to attacker-controlled IP !

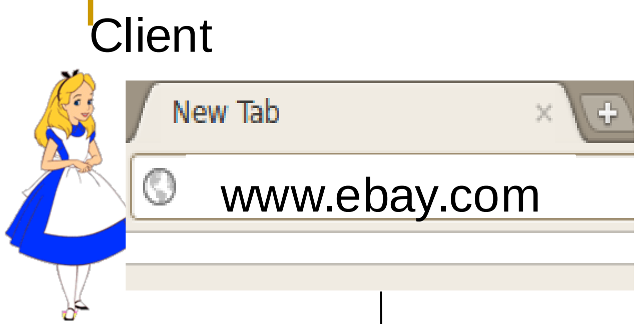
Domain Name System (DNS) Security

- DNS: quick recap
- DNS poisoning
  - Motivation: the hacker's swiss knife
  - Method 1 (historical): by Gratuitous `glue' RR
  - Method 2: send from corrupt NS
  - Method 3: send spoofed DNS response
  - Method 4: dangling DNS records
- **DNSSEC: Cryptographic security for DNS**
- DNS Privacy issues and defenses

DNS Security (DNSSEC) [RFCs 4033 to 4035]

- Goal: prevent MitM (and off-path) attacks
  - Also defends against exploits of vulnerable name servers
- Main idea: **sign DNS RRsets**
  - RRset: set of Resource Records with same name and type
    - E.g., all 'A' records of foo.com
  - Signature in a separate DNS RR, of type RRSIG
  - Private signing key of domain owner [offline?]
    - authoritative DNS server
  - Hierarchical certification of domains' public key
- → prevents cache poisoning attacks
- → secures use of DNS-based policies, public keys
- Includes a (very limited) `DNS-specific' PKI

Domain Name System with DNSSEC



DNSSEC validation at **client**?
- Essential if resolver isn't trusted
- But 'breaks' recursive service
- Requires support from OS, resolver
- Supported by some clients, incl. popular browsers
- Some resolvers `break' DNSSEC
  - -Esp. : NXDomain → Ad

www.ebay.com

1.2.3.4

DNSSEC: Public Keys and Record Types

- Typical domains have two public keys: KSK and ZSK
- Key Signing Keys (KSK)
  - Authenticated by parent domain, or
    by operator/software ('trust anchor', e.g., for root domain)
- Zone Signing Keys (ZSK)
  - Authenticated (signed) by KSK of domain
- Domain may use multiple ZSKs/KSKs, to phase-in a new key or for different signing algorithms
- Basic DNSSEC Record types:
  - DNSKEY: public keys (mainly, KSKs and ZSKs)
  - RRSIG: signatures (over an RRset, e.g., DNSKEY RRset)
  - DS: hash to authenticate KSK's DNSKEY (Hash-then-Sign)

The RRSIG, DS and DNSKEY RRs

Root Zone (KSK embedded in browsers, tag=3861)
. 86400 IN DNSKEY 256 3 5 (… …) ; ZSK1 (tag=1127)
. 86400 IN DNSKEY 256 3 5 (… …) ; ZSK2
. 86400 IN RRSIG DNSKEY 5 0 86400 (… **3861** …)
com. 86400 IN RRSIG DS 5 1 86400 (… 1127 …)
com. 86400 IN DS **50322** 5 1 (2BB18 … A53B0A)

Signature over both DNSKEYs using KSK **3861**

Signature of… (using ZSK 1127)

Hash of KSK **50322**

com Zone
com. 6400 IN DNSKEY 257 3 5 (… …) ; KSK (**50322**)
com. 6400 IN DNSKEY 256 3 5 (… …) ; ZSK1 (2623)
com. 6400 IN DNSKEY 256 3 5 (… …) ; ZSK2 (14672)
com. 6400 IN RRSIG DNSKEY 5 1 6400 (… **50322** …)
www.com. 3600 IN A 1.2.3.4 ; A RR
www.com. 3600 IN RRSIG A 5 2 (… 2623 …) ; Signs A RRset

Signature over all DNSKEYs using KSK 50322.

Signature over type-A RRset using ZSK 2623

DNSSEC Resource Record (RRs): RRSIG

- RRSIG: signature over an **RRset** and validity period
  - Signature over the concatenation of all RRs of given name, type
  - Sent automatically with response to request for RRset, if request requests DNSSEC records (DNSSEC Ok (DO) bit)
  - Example: signature over set of RRs of type *A* of www.foo.com:
    www.foo.com. 6400 IN **RRSIG** *A* 5 3 6400
    20230218173103 (20220218083059 2624 foo.com.
    oJB1W6WNGv+ldv.........DQfsS3Ap3o=)
  - TTL of signature, RRset is 6400s (1hour) [usually identical]
  - Signing algorithm is '5' (RSA/SHA1)
  - Number of 'labels' in signed name is 3 (www.foo.com).
  - Expiration time is 17:31:03 at 2/18/2023;
    signing time is 08:30:59 at 2/18/2022
  - 2624 is the 'key tag' – identifies signer's key (may have few). For most algs: 16-bits sum of the data of the DNSKEY data
  - Signer is foo.com and oJB1... is the Base64-encoded signature

DNSSEC Resource Record (RRs): RRSIG

- RRSIG: signature over an **RRset** and validity period
  - Signature over the concatenation of all RRs of given name, type
  - Sent automatically with response to request for RRset, if request requests DNSSEC records (DNSSEC Ok (DO) bit)
  - Example: signature over set of RRs of type *A* of www.foo.com:
    www.foo.com. 6400 IN **RRSIG** *A* 5 3 6400
    20230218173103 (20220218083059 2624 foo.com.
    oJB1W6WNGv+ldv.........DQfsS3Ap3o=)
  - TTL of signature, RRset is 6400s (1hour) [usually identical]
  - Expiration time is 17:31:03 at 2/18/2023;
    signing time is 08:30:59 at 2/18/2022
- Actually, something is weird here! What?
  - Expiration should be longer than TTL, but by so much??
  - Attacker can resend the signature anyway
  - To ensure updates, validity shouldn't be much more than TTL

The **RRSIG**, DS and DNSKEY RRs

Root Zone (KSK embedded in browsers, tag=3861)
. 86400 IN DNSKEY 256 3 5 (… …) ; ZSK1 (tag=1127)
. 86400 IN DNSKEY 256 3 5 (… …) ; ZSK2
. 86400 IN **RRSIG** DNSKEY 5 0 86400 (… **3861** …)
com. 86400 IN **RRSIG** DS 5 1 86400 (… 1127 …)
com. 86400 IN DS **50322** 5 1 (2BB18 … A53B0A)

Signature over both DNSKEYs using KSK **3861**

Signature of… (using ZSK 1127)

Hash of KSK **50322**

Why??

com Zone
com. 6400 IN DNSKEY 257 3 5 (… …) ; KSK (**50322**)
com. 6400 IN DNSKEY 256 3 5 (… …) ; ZSK1 (2623)
com. 6400 IN DNSKEY 256 3 5 (… …) ; ZSK2 (14672)
com. 6400 IN **RRSIG** DNSKEY 5 1 6400 (… **50322** …)
www.com. 3600 IN A 1.2.3.4 ; A RR
www.com. 3600 IN **RRSIG** A 5 2 (… 2623 …) ; Signs A RRset

Signature over all DNSKEYs using KSK 50322.

Signature over type-A RRset using ZSK 2623

DNSSEC Resource Record (RRs): DNSKEY

- RRSIG: signature over an **RRset** and validity period
- DNSKEY: DNS public verification key(s), e.g.:

foo.com. 86400 IN DSNSKEY 256 3 5 (Abl4…z3w==)
(domain) (TTL) (flags) (prot) (alg) (public key)

  - 16-bit flags field; only bits 7 and 15 used (other: zeros)
    - Details in next slide
  - 3 is 'protocol' field (only value allowed is 3)
  - 5 is 'signature algorithm' field; 5 is RSA-SHA1. Other options defined (by IANA), e.g., ECDSA
  - Public key is BASE-64 encoded
- Note: keytag is not specified; it is computed as a (known) function of the public key
  - If multiple keys have the same tag, try all of them [KeyTrap]

DNSSEC Resource Record (RRs): DNSKEY

- RRSIG: signature over an **RRset** and validity period
- DNSKEY: DNS public verification key(s), e.g.:

  foo.com. 86400 IN DSNSKEY 256 3 5 (Abl4…z3w==)

  (domain) (TTL) (flags) (prot) (alg) (public key)

  - 16-bit flags field; only bits 7 and 15 used (others must be zero)
    - Bit 7 signals key validates RRsigs
    - Bit 15 signals Secure Entry Point (SEP): key validated by parent zone or 'pre-trusted' by resolver; one KSK must have SEP set
    - Flag bits are numbered from left (MSb) to right (LSb: bit 15)
    - Values: 256 (bit 7 – RRSIG key), 257 (also SEP)
  - Typically, we use two type of keys:
    - Key Signing Keys (KSK), Secure Entry Point (SEP), flags: 257, used to validate DNSKEY RRSIGs;
    - Zone Signing Keys (ZSK), flags: 256, validate (other) RRSIGs, validated by KSK

DNSSEC Resource Record (RRs): DNSKEY

- RRSIG: signature over an **RRset** and validity period
- DNSKEY: DNS public verification key(s)), e.g.:

  foo.com. 86400 IN DSNSKEY 256 3 5 (Abl4…z3w==)

  - 16-bit flags field; only bits 7 and 15 used (other: zeros)
    - Bit 7 signals key validates RRsigs
    - Bit 15 signals Secure Entry Point (SEP)
  - Key Signing Keys (KSK), flags=257, for DNSKEY RRSIGs
  - Zone Signing Keys (ZSK), flags=256, for other RRSIGs
  - Why separate KSK from ZSK?
  - ZSK used more → more exposed → change often
  - ZSK roll-over: keep new ZSK in parallel to old ZSK
  - Changing KSK requires re-validation by parent zone
  - KSK is typically longer, kept offline, changed rarely
  - Parent domain validates SEP keys (KSKs) using the DS RR

The RRSIG, DS and **DNSKEY** RRs

Root Zone (KSK embedded in browsers, tag=3861)		
.	86400 IN **DNSKEY** 256 3 5 (… …)	; ZSK1 (tag=1127)
.	86400 IN **DNSKEY** 256 3 5 (… …)	; ZSK2
.	86400 IN RRSIG DNSKEY 5 0 86400 (… **3861** …)	
com.	86400 IN RRSIG DS 5 1 86400 (… 1127 …)	
com.	86400 IN DS **50322** 5 1 (2BB18 … A53B0A)	

Signature over both DNSKEYs using KSK **3861**

Signature of… (using ZSK 1127)

Hash of KSK **50322**

com Zone		
com.	6400 IN **DNSKEY** 257 3 5 (… …)	; KSK (**50322**)
com.	6400 IN **DNSKEY** 256 3 5 (… …)	; ZSK1 (2623)
com.	6400 IN **DNSKEY** 256 3 5 (… …)	; ZSK2 (14672)
com.	6400 IN RRSIG DNSKEY 5 1 6400 (… **50322** …)	
www.com.	3600 IN A 1.2.3.4	; A RR
www.com.	3600 IN RRSIG A 5 2 (… 2623 …)	; Signs A Rrset

Signature over all DNSKEYs using KSK 50322.

Signature over type-A RRset using ZSK 2623

DNSSEC Resource Record (RRs): DS

- RRSIG: signature over an **RRset** and validity period
- DNSKEY: public key of zone, mainly:
  - Zone Signing Key (ZSK): validates non-key RRsets
  - Key Signing Key (KSK): validates ZSKs
- DS (Delegation Signer): hash of SEP(KSK) of subdomain
  - Essential to allow separate owner / zone file for subdomain
  - Example of DS for foo.com (in zone file of .com):
    foo.com 86400 IN DS 50322 5 1 (2BB18 … A53B0A)
  - TTL of key is 86400s (24hours)
  - 50322 is the 'key tag' – identifies signer's key (may have few).
    For most algs: 16-bits sum of the data of the DNSKEY data
  - Signing algorithm is '5' (RSA/SHA1) and hashing is '1' (SHA1).
  - 2BB18 … A53B0A … is the Base64-encoded hash, over owner name concatenated to DNSKEY data
  - DNSKEY should specify Secure Entry Point (flags=257)
  - h

The RRSIG, DS and **DNSKEY** RRs

Root Zone (KSK embedded in browsers, tag=3861)
. 86400 IN **DNSKEY** 256 3 5 (... ...) ; ZSK1 (tag=1127)
. 86400 IN **DNSKEY** 256 3 5 (... ...) ; ZSK2
. 86400 IN RRSIG DNSKEY 5 0 86400 (... **3861** ...)
com. 86400 IN RRSIG DS 5 1 86400 (... 1127 ...)
com. 86400 IN DS **50322** 5 1 (2BB18 ... A53B0A)

Signature over both DNSKEYs using KSK **3861**

Signature of... (using ZSK 1127)

Hash of KSK **50322**

com Zone
com. 6400 IN **DNSKEY** 257 3 5 (... ...) ; KSK (**50322**)
com. 6400 IN **DNSKEY** 256 3 5 (... ...) ; ZSK1 (2623)
com. 6400 IN **DNSKEY** 256 3 5 (... ...) ; ZSK2 (14672)
com. 6400 IN RRSIG DNSKEY 5 1 6400 (... **50322** ...)
www.com. 3600 IN A 1.2.3.4 ; A RR
www.com. 3600 IN RRSIG A 5 2 (... 2623 ...) ; Signs A RRset

Signature over all DNSKEYs using KSK 50322.

Signature over type-A RRset using ZSK 2623

DNSSEC: a DNS Public Key Infrastructure?

- DNSSEC is basically a DNS-specific Public Key Infrastructure (PKI)
  - No revocation mechanism (only expiration)
  - No transparency mechanism, no extensions…
  - But: naming constraints are built-in
- Cipher agility in DNSsec ?
  - DS records may use different hashing algs; all are sent to the client, who can validate using only some of them
    - Attacker cannot drop any since DS RRset is signed (RRSIG)
  - Offer multiple signatures using different algorithms, keys
- Downgrade attack : attacker causing client (resolver) to use weaker algorithm than supported by server
  - Leaving only weaker-security records / signatures
- Question: is DNSSEC vulnerable to downgrades??
- x

DNSSEC Downgrade Prevention

- Downgrade attack 1: send only DNSKEYs using weak algs
- Prevention:
  - Domain must have DNSKEY for each algorithm indicated by the domain's DS RRset in the parent domain
    - With valid hash as per the DS RR
- Downgrade attack 2: send only RRSIGs using weak algs
- Prevention:
  - For every signing alg *S* in any DNSKEY record of the domain, there must be an RRSIG using alg *S* (with a key from DNSKEY)
  - ➔ If given only RRSIGs using weak alg, response is invalid !
  - Concern: req' applies to servers, not validators [RFC6840 sect.5.11] ➔ several resolver implementations do not prevent such downgrades!

[Heftrig, Shulman, Waidner UsenixSec'23]

DNSSEC Downgrade Prevention

- Downgrade attack 1: send only DNSKEYs using weak algs
- Downgrade attack 2: send only RRSIGs using weak algs
- Downgrade attack 3: don't send any DNSKEYs
  - As if domain is not using DNSSEC at all

Root Zone (KSK embedded in browsers, tag=3861)		
.	86400 IN **DNSKEY** 256 3 5 (... ...)	; ZSK1 (tag=1127)
.	86400 IN **DNSKEY** 256 3 5 (... ...)	; ZSK2
.	86400 IN RRSIG DNSKEY 5 0 86400 (... **3861** ...)	
com.	86400 IN RRSIG DS 5 1 86400 (... 1127 ...)	
com.	86400 IN DS 50322 5 1 (2BB18 ... A53B0A)	

Signature over both DNSKEYs using KSK **3861**

Signature of… (using ZSK 1127)

com Zone	
www.com. 3600 IN A 1.2.3.4	; A RR
… (other non-DNSSEC records)	

- Prevention: ???

DNSSEC Downgrade Prevention

- Downgrade attack 1: send only DNSKEYs using weak algs
- Downgrade attack 2: send only RRSIGs using weak algs
- Downgrade attack 3: don't send any DNSKEYs
- Downgrade attack 4: don't send the DS from parent domain, or don't send RRSIG records for a signed RR
  - As if domain/RR is not protected using DNSSEC
- Prevention: server must send <u>proof of non-existence</u> of DNSSEC (DS or RRSIG) records
- Related attack: send NXDOMIN (non-existing domain) in response to query for an existing domain
  - E.g., blacklist, policy

Secure DNS: proof of no (signed) RR

- What if bar.com has no public key?
  - Does not yet support Secure DNS
- Can send unsigned RRs…
- But: attacker may send unsigned RRs, even if bar.com does have public key!
- Maybe "com" would sign a 'NoSec RR', indicating bar.com has no public key?
- Any concerns?
  - *Efficiency – need to sign `NoSec RR' for every unsigned domain*
  - *Worse – we have to sign in real time!*
    - *To prevent replays, and if subdomain doesn't exist*
- ***NSEC RR Solution***: "**Next SEC**ure record is <name>"
  - Use lexicographic order, sign in advance

The NSEC (Next SECure) RR

- NSEC record: al.com NSEC don.com A RRSIG NSEC
  - Means: no RR alphabetically between al.com and don.com
  - Also: al.com has only A, RRSIG and NSEC records
- Sent in response to NSEC query for <u>any</u> record from al.com to don.com (alphabetically)
- Example: COM={al.com, don.com, ed.com, jon.com}
- Request is bob.com; response=?
  - Response is: al.com. 300 IN NSEC don.com A RRSIG NSEC
  - Authenticated (signed) using RRSIG
- Request is guy.com; response=?
  - Response is ed.com. 300 IN NSEC jon.com A RRSIG NSEC
- Request is sam.com; response=?
  - Response is jon.com. 300 IN NSEC al.com A RRSIG NSEC

The NSEC RR – and Zone Enumeration

- NSEC stands for 'Next Secure'
- NSEC record: al.com NSEC don.com A RRSIG NSEC
  - Means: no RR alphabetically between al.com and don.com
  - Also: al.com has only A, RRSIG and NSEC records
- Sent in response to NSEC query for any record from al.com to don.com (alphabetically)
- Allows zone enumeration (discovery of subdomains)
- A potential security/privacy concern:
  - Directed attacks at subdomain (e.g., website)
  - Some name may identify vulnerability
    - E.g.: proxy.x.com ➔ maybe open proxy??
  - Names may be sensitive, e.g., new company name
- Challenge: authenticate non-existing-domain response, but prevent zone enumeration!

NSEC3 Hashed Non-Existence Response

- Authenticated non-existing RR response:
  <hash1> NSEC3 <alg> <opt> <n> <salt> <hash2> <RRtypes>

e.g.: F3…D.com NSEC3 1 0 9 4A…F 6C…8 A RRSIG NSEC

- Hashed: goal is to prevent zone enumeration (but…)
  - Suppose h(al.com)= F3…D and h(don.com)= 6C…8
  - Means: no RR x.com such that F3…D < h(x.com) < 6C…8
  - Proving non-existence of RRs (e.g., of x.com)
  - Also: al.com has only A RRSIG and NSEC records
- Parameters: <alg> <opt> <n> <salt>
  - <alg>: hashing algorithm, mostly 1 for SHA-1
  - <opt>: if 0, NSEC3 done for all records in domain; if 1, NSEC3 only for protected records. (see later)
  - <n>: number of iterations (of hash function), here 9
  - <salt>: salt input to hash function, here 4A…F [base64]

NSEC3 Hashed Non-Existence Response

- Authenticated non-existing RR response:
  <hash1> NSEC3 <alg> <opt> <n> <salt> <hash2> <RRtypes>
- NSEC3 hash calculation:
  - Goal: (limited) defense against dictionary attacks
  - Iterated n times (to make computations slower [note: DoS?])
  - Salted: against precomputation (same salt for entire domain)
    - Actually, two fields: salt and salt length (0 to 255 bytes)
  - Hash of *name* with *salt* after *i* iterations:

$$H_{name}^{i,salt} = h\left(H_{name}^{i-1,salt} \| salt \| i-1\right);$$

$$H_{name}^{0,salt} = h(name \| salt)$$

- Compare this salt to salt of PW file

Zone Enumeration with NSEC3 ?

- NSEC3 hash calculation:
  - Hash of $name$ with $salt$ after i iterations:
    $H_{name}^{i,salt} = h\left(H_{name}^{i-1,salt} || salt || i-1\right); H_{name}^{0,salt} = h(name||salt)$
- Each NSEC3 record contains next hash of name
  - To prove non-existence of names hashed in-between
- Bernstein's attack: collect hashed values (about one request per record), then offline dictionary attack!
- 'White lies' defense [RFC7129]: if $name$ doesn't exist, send NSEC3 for $H_{name}^{i,salt} - 1$, with next-hash $H_{name}^{i,salt} + 1$.
  - Disadvantage: online signing (with zone's private key)
  - Adding random `white lies' records in advance (offline), doesn't help much
- NSEC5 proposal: uses special private key (still, online)

Zone Enumeration with NSEC3 ?

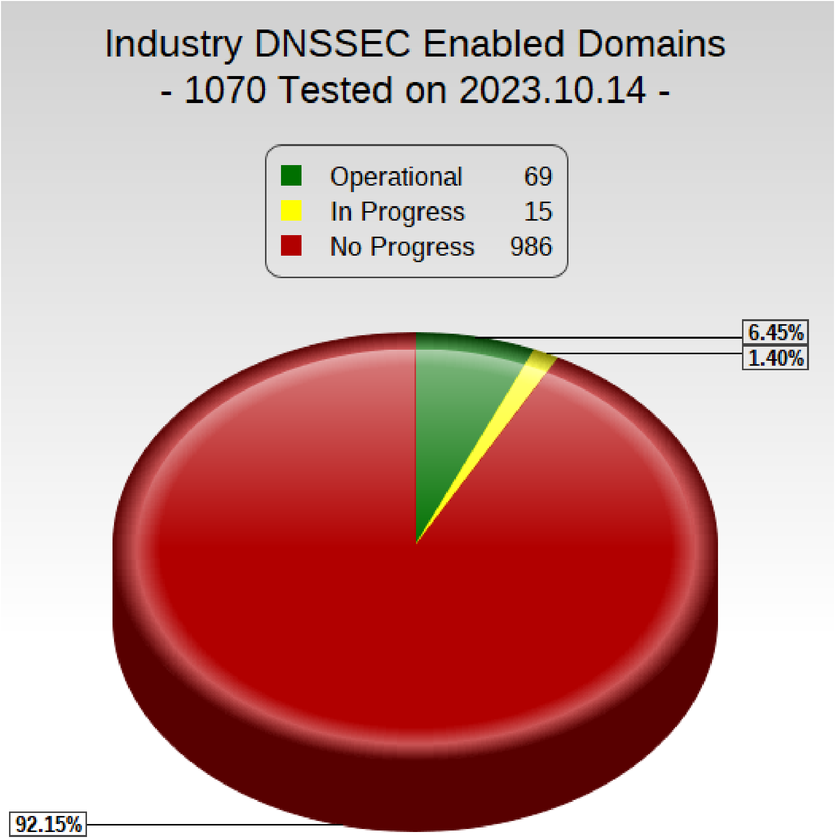
- NSEC3 hash calculation:
  - Hash of $name$ with $salt$ after i iterations:
    $H_{name}^{i,salt} = h\left(H_{name}^{i-1,salt} \| salt \| i-1\right); H_{name}^{0,salt} = h(name \| salt)$
- Each NSEC3 record contains next hash of name
  - To prove non-existence of names hashed in-between
- Bernstein's attack: collect hashed values (about one request per record), then offline dictionary attack!
- Per-name salt defense:
  - Add 'salt' record type:
    name SALT <salt>
  - NOT signed; if query to non-existing name: return random salt
  - Does this prevent attack?

The Opt-Out Flag <opt>

- Recall the <opt> Opt-Out flag of NSEC3:
  <hash1> NSEC3 <alg> <opt> <n> <salt> <hash2> <RRtypes>
- Opt-Out=0 (no Opt-Out): NSEC3 records done for <u>all</u> RRs in domain
- Opt-Out=1 (Opt-Out used): NSEC3 records done only for RRs of DNSSEC-protected subdomains
  - And <hash2> is of next DNSSEC-protected subdomain
- In many zones, most subdomains are unsigned
  - E.g., Top-Level Domains (TLDS), such as com.
- ➔ such domains often use Opt-Out=1
  - Less NSEC3 records ➔ less hashing and less **signing**
  - In fact, a common (main?) motivation to use NSEC3, not NSEC!

DNSSEC: Status

- DNSSEC signed domains
  - Root + (most) TLDs are signed
  - Much less signing by `regular' domains



DNSSEC: Status

- DNSSEC signed domains
  - Root + (most) TLDs are signed
  - Much less signing by `regular' domains
- DNSSEC validation
  - Concern: DNSSEC interoperability failures
    - And DNS response modification (mainly NX)
  - Invalid: resolvers respond with SERVFAIL
    - Causes client to try any alternate resolver known
  - Limited incentive: not visible to customer
    - Few/no clients makes validation mandatory
    - Some clients use trusted, secure resolver
    - Some applications use trusted secure resolvers (DoH, DoT)

Domain Name System (DNS) Security

- DNS: quick recap
- DNS poisoning
  - Motivation: the hacker's swiss knife
  - Method 1 (historical): by Gratuitous `glue' RR
  - Method 2: send from corrupt NS
  - Method 3: send spoofed DNS response
  - Method 4: dangling DNS records
- DNSSEC: Cryptographic security for DNS
- **DNS Privacy issues and defenses**

DNS Privacy Concerns [RFC7626,9076]

- Privacy? Isn't DNS data public?
- DNS leaks data about activities on Internet
- Most Internet activities involve one/few DNS queries
- Queries can expose private information:
  - What website user visits
  - And more: email sender/recipient domains, sw user runs…
  - Fingerprinting → identify user / device
    - Allow even identification of device behind NAT
  - Yet… DNS queries are sent in clear text
- By default, queries sent to DHCP-defined resolver
- ISPs & CDNs often add end-user data to queries
  - Goals: parental filtering and geo-optimized responses

DNS Privacy Concerns [RFC7626,9076]

- Privacy? Isn't DNS data public?
- DNS leaks data about activities on Internet
- Most Internet activities involve one/few DNS queries
- Queries can expose private information
- DNS poisoning can expose privacy, even if using https!
  - Allows off-path attacker to be MitM to https traffic
  - Launch attacks against TLS that require MitM capabilities
    - Most TLS attacks assume MitM capabilities!
    - E.g., downgrade attacks, cryptanalytical attacks, …
  - Allows TLS-traffic analysis and disconnection attacks
  - And: DNS has important non-web applications! (like what?)

DNS Privacy Defenses

- Focus on stub-to-resolver communication, query data
  - Resolver-to-Authoritative is harder, less critical
  - Harder: prevent exposure from traffic analysis, side-channels
- DoT: DNS over TLS (RFC 7858,8310)
  - Resolver listens on port 853
  - TCP handshake, TLS handshake…
  - Use same connection for all request; pipeline for efficiency
  - Server authentication (public key validation):
    - Opportunistic: no server authentication
    - Out-of-band key-pinned
    - Several other options in RFC 8310 [deployment?]
- DoH: DNS over HTTPS (RFC 8484)

DNS Privacy Defenses

- Focus on stub-to-resolver communication, query data
  - Resolver-to-Authoritative is harder, less critical
  - Harder: prevent exposure from traffic analysis, side-channels
- DoT: DNS over TLS (RFC 7858)
- DoH: DNS over HTTPS (RFC 8484)
  - Performs HTTPS request to predefined URI template
    - Domain, partial path, and extension format [RFC6570]
    - E.g.: https://doh.opendns.com/dns-query{?dns} (for FF)
  - A single DNS query per request, using GET or POST
    - With GET, {?dns} is replaced by the formatted DNS query
  - All requests use DNS-ID of 0 (zero), to allow caching
    - Freshness lifetime (Max-Age) ≤ min TTL in response RRs
    - Client can still match request to response, since… ?

DNS Security: Summary

- DNS is a key part of Internet infrastructure
- Often abused and attacked
- 'Plain' DNS is vulnerable
  - Trivial MitM attack, clever off-path attacks
- DNSSEC improves authentication
  - Deployed – but (too?) slowly
  - Security challenges remain (key agility allows downgrade?)
- Do we need privacy defenses, too?
  - DoH! (DNS over HTTP)
  - Or DoT, or (few other designs)
  - Limited: only client-to-resolver and only content
  - No protection against traffic analysis, side-channels