

Resilient Networking

Module 4: Name Resolution Security



Thorsten Strufe – *This module prepared in cooperation with Mathias Fischer, Michael Roßberg, and Günter Schäfer*

Winter Term 2021 – KIT/TUD

KASTEL Security Research Labs

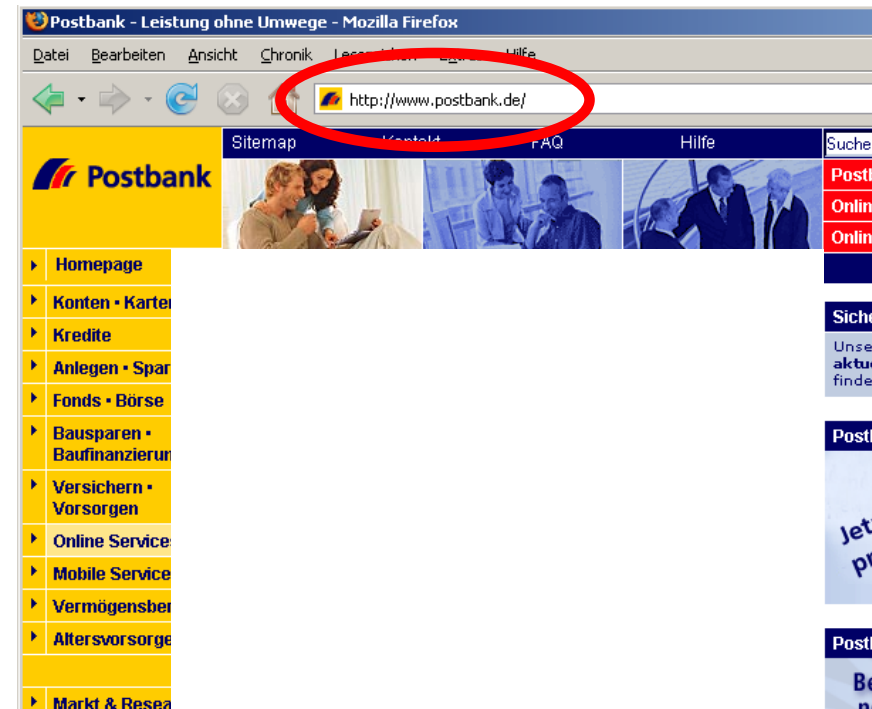


Module Outline

- Overview of DNS
- Known attacks on DNS
 - Denial-of-Service
 - Cache Poisoning
- Securing DNS
 - Split-horizon DNS
 - DNS Cookies (RFC 7873)
 - DNSSEC
 - DNSCurve
 - PNRP
 - GNS

DNS – The Domain Name System

- What is DNS?
- Naming Service for (almost all) Internet traffic
- Lookup of (resolve)
 - Host-Addresses
 - Mail-Servers
 - Alias Names
 - Alternative Name Servers
 - ...
- Distributed Database consisting of multitude of servers



DNS – Names

- **People:** many identifiers:
 - SSN, name, passport #
- **Internet hosts, routers:**
 - IP address (32 bit) - used for addressing datagrams
 - “Name”, e.g., www.yahoo.com - used by humans
- **Q:** Map between IP addresses and name ?
- **Domain Name System:**
 - *Distributed database* implemented in hierarchy of many *name servers*
 - *Application-layer protocol:* hosts, routers, name servers communicate to *resolve* names (address/name translation)
 - Note: core Internet function, implemented as application-layer protocol
 - Complexity at network’s “edge”

DNS – what does it do?

■ ***DNS services***

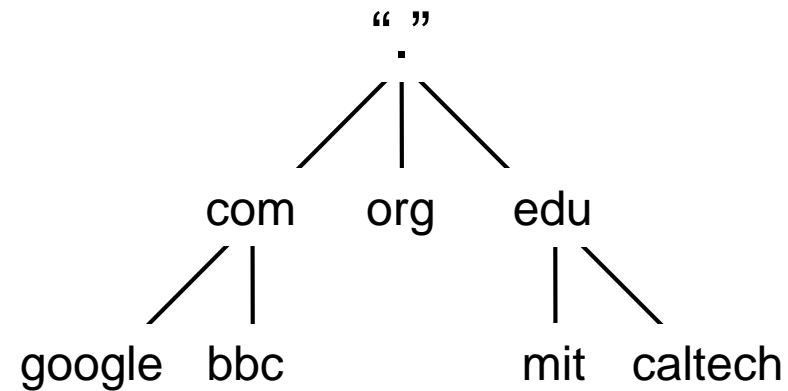
- Hostname to IP address translation
- Host aliasing
 - Canonical and alias names
- Mail server aliasing
- Load distribution
 - Replicated Web servers: set of IP addresses for one canonical name

■ ***Why not centralize DNS?***

- Single point of failure
- Traffic volume
- Distant centralized database
- Maintenance
- *does not scale!*

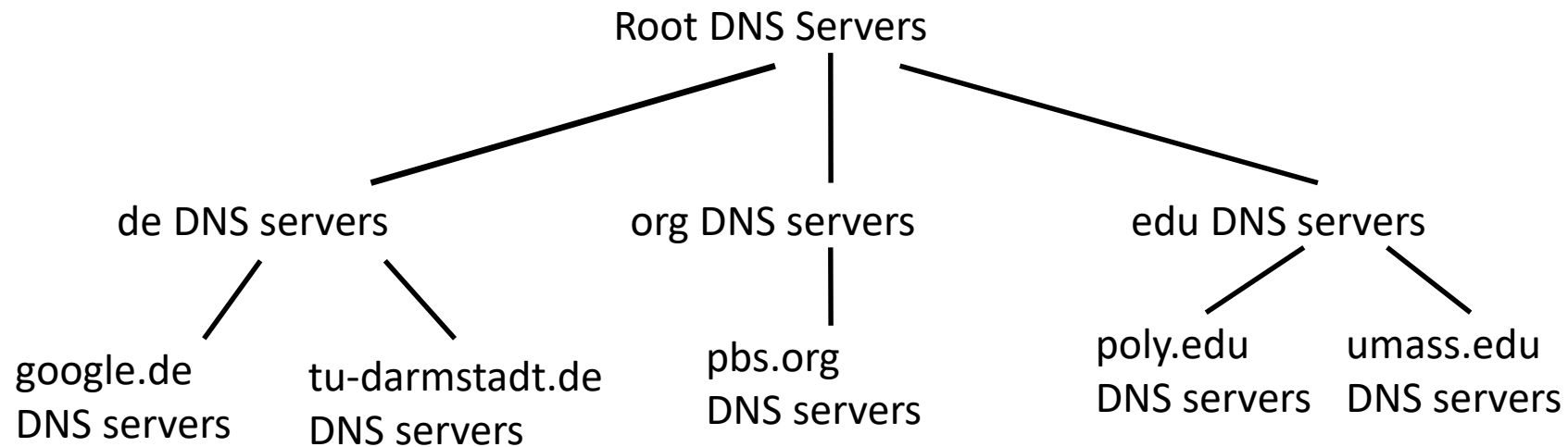
What does this „it scales“ mean anyways!?

DNS – Data Organization: Domains / Zones

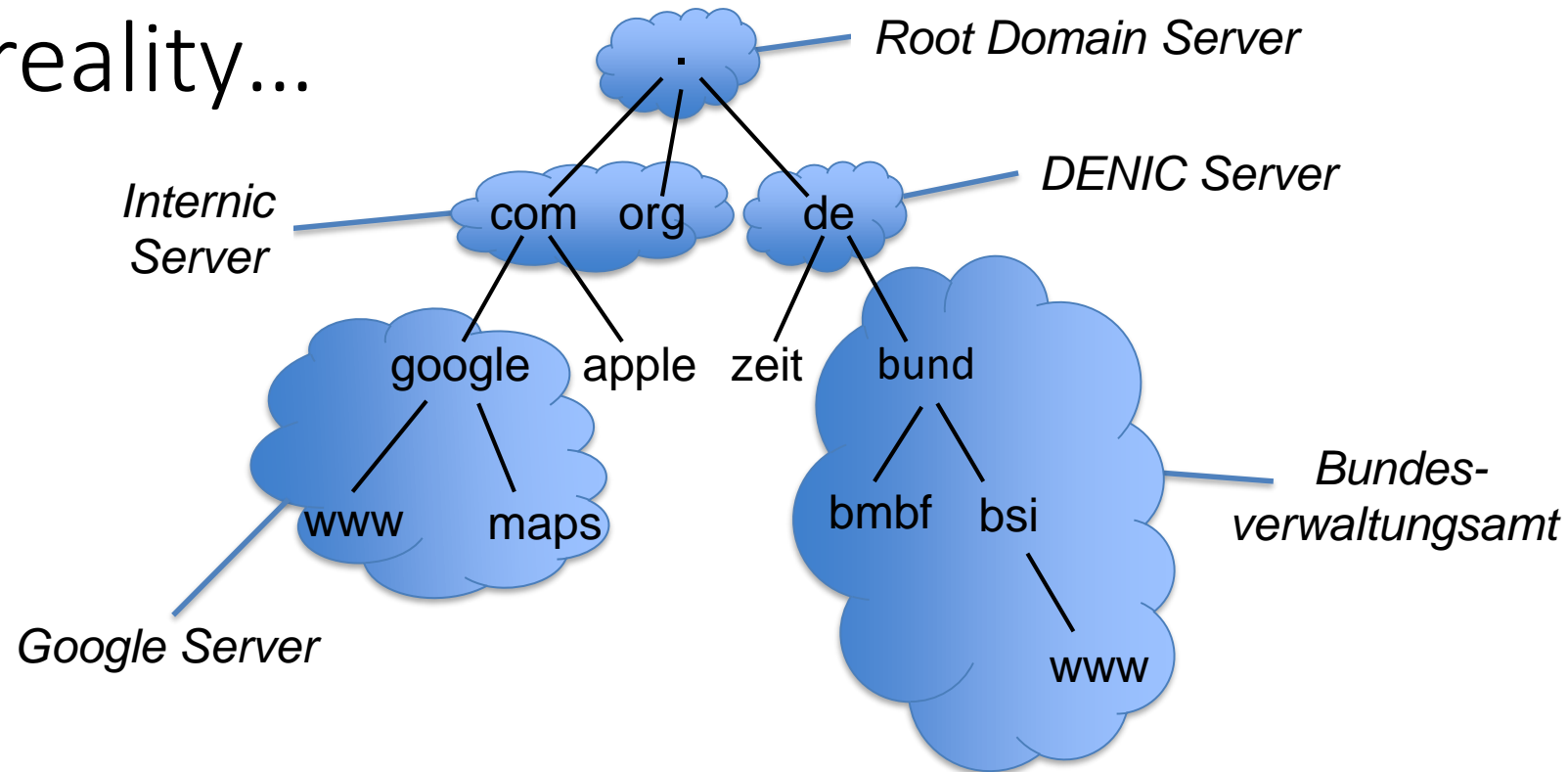


- Structured Namespace
- Hierarchical organization in sub domains/zones
- Sourced at “root zone” (“.”)
- Parent zones maintain pointers to child zones (“*zone cuts*”)
- Zone data is stored as “Resource Records” (RR)

Distributed, Hierarchical Database



...and in reality...



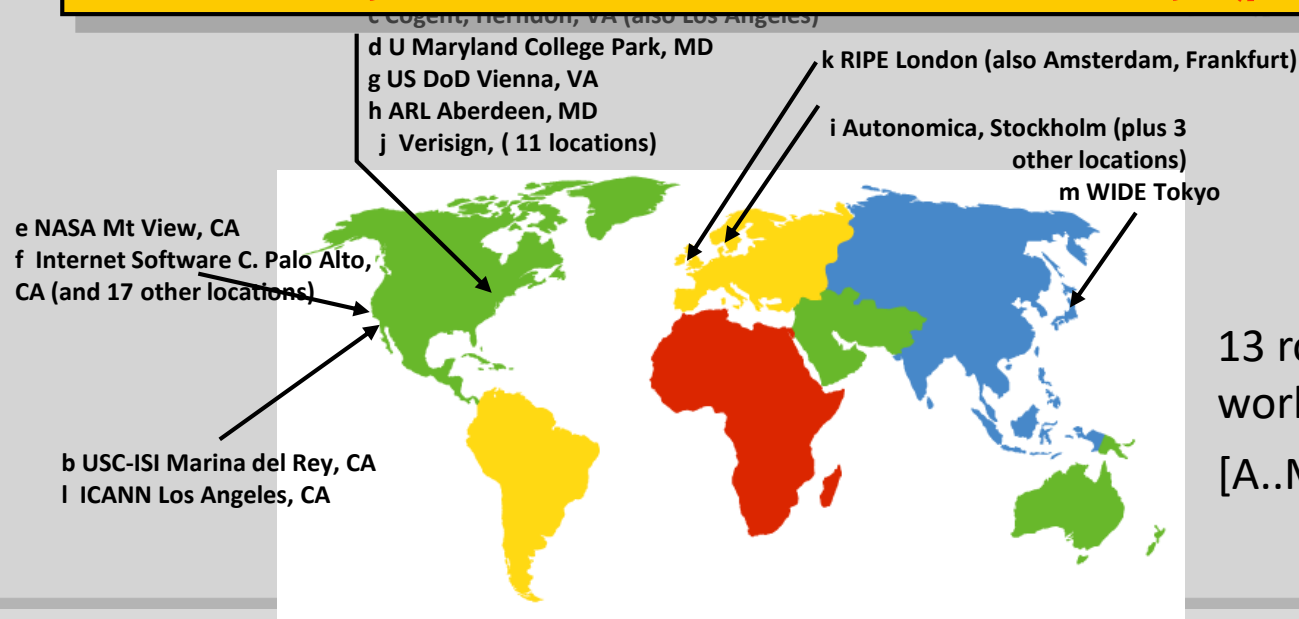
Client wants IP for `www.dud.inf.tu-dresden.de`; 1st approx:

- Client queries a root server to find **de** DNS server
- Client queries de DNS server to get **tu-dresden.de** DNS server
- Client queries tu-dresden.de DNS server to get IP address for `www.dud.inf.tu-dresden.de`

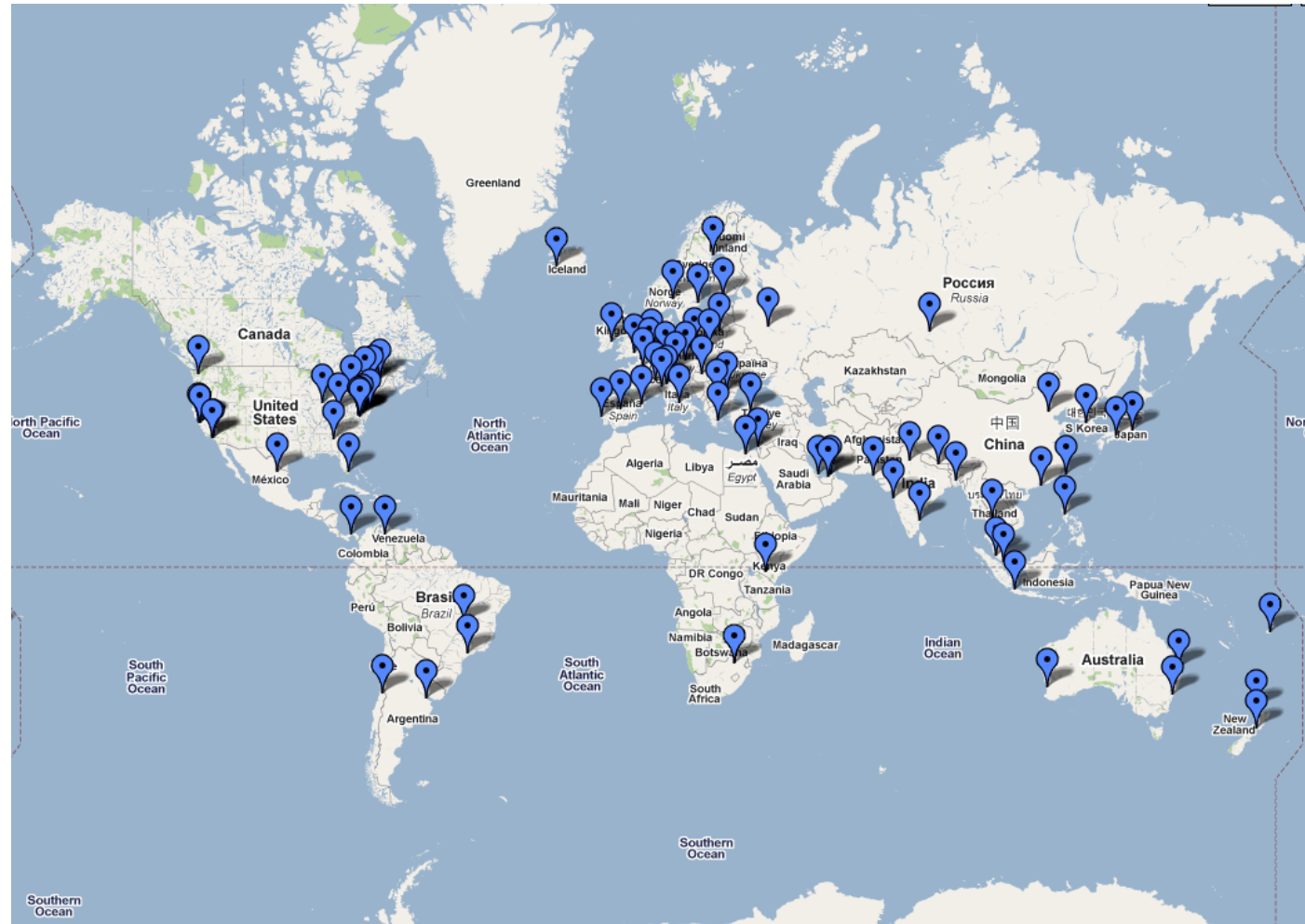
DNS: Root Name Servers

- Contacted by local name server that can not resolve name
- Root name server:
 - Contacts authoritative name server if name mapping not known
 - Gets mapping
 - Returns mapping to local name server

So, how many root nameservers are there actually? (physically)



DNS: Root Name Servers



<http://www.root-servers.org/>

DNS – Components

■ Authoritative Server

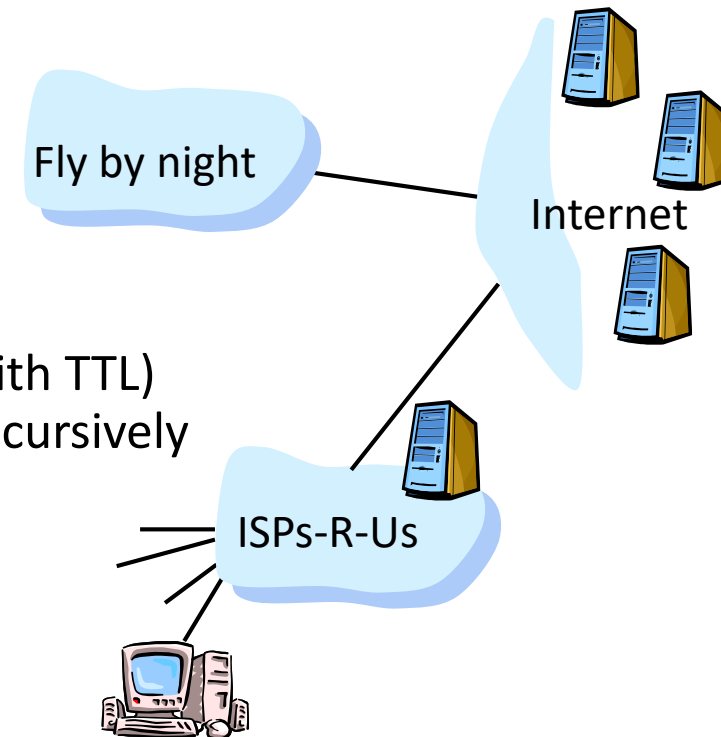
- Server maintaining authoritative content of a complete DNS zone
- Top-Level-Domain (TLD) servers & auth servers of organization's domains
- Pointed to in parent zone as authoritative
- Possible load balancing: master/slaves

■ Recursive (Caching) Server

- Local proxy for DNS requests
- Caches content for specified period of time (soft-state with TTL)
- If data not available in the cache, request is processed recursively

■ Resolver

- Software on client's machines (part of the OS)
- Windows-* and *nix: Stub resolvers
 - Delegate request to local server
 - Recursive requests only, no support for iterative requests



DNS – Resource Record Type

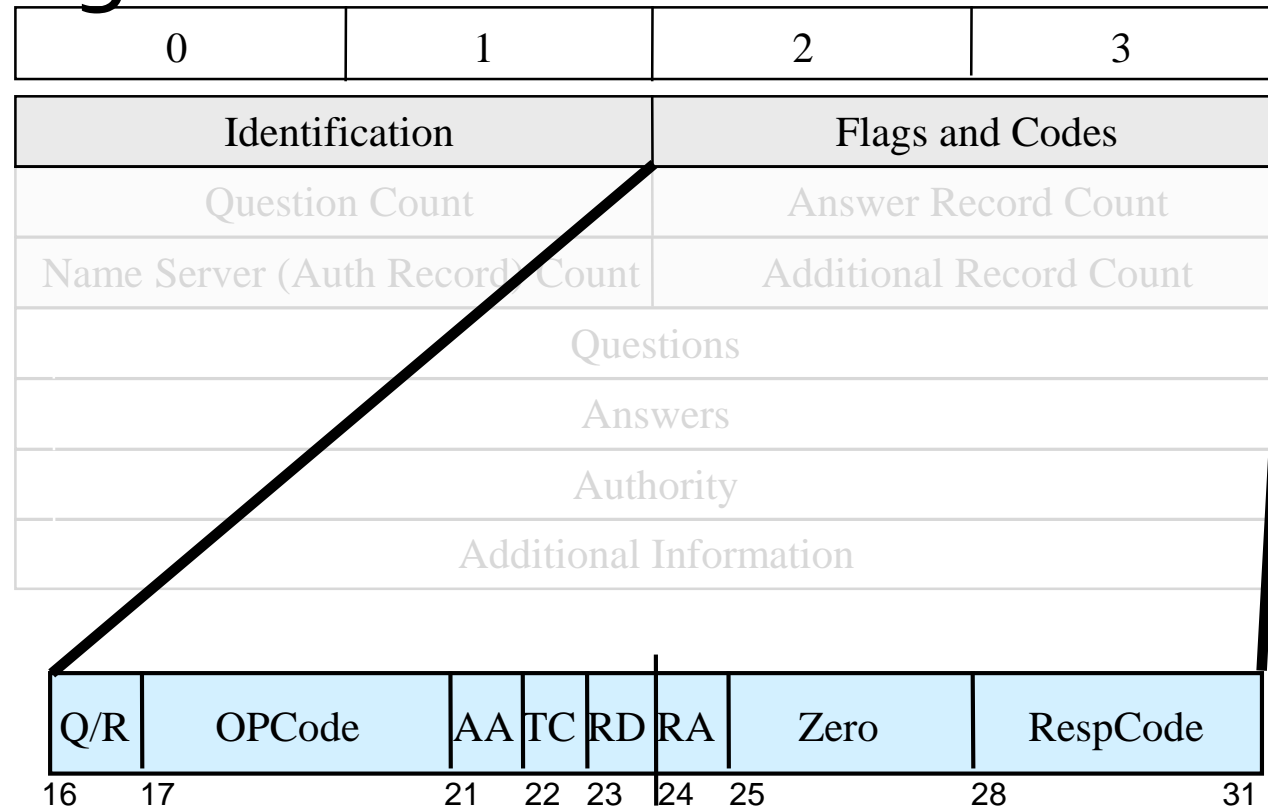
- Atomic entries in DNS are called “Resource Records” (RR)
- Format: <name> [<ttl>] [<class>] <type> <rdata>
 - name (domain name of resource)
 - ttl (Time-to-live)
 - class (used protocol): IN (Internet), CH (Chaosnet)...
 - type (record type): A (Host-Address), NS (Name Server), MX (Mail Exchange), CNAME (Canonical Name), AAAA (IPv6-Host-Address), DNAME (CNAME, IPv6)
 - rdata (resource data): **Content!** (*What did we want to look up?*)

Specific DNS Records

RR Format: (name, ttl, class, type, value)

- Type=A
 - **name** is hostname
 - **value** is IP address
- Type=NS
 - **name** is domain (e.g. foo.com)
 - **value** is IP address of authoritative name server for this domain
- Type=MX
 - **value** is name of mailserver associated with **name**
- Type=CNAME
 - **name** is alias name for some “canonical” (the real) name
`www.ibm.com` is really `servereast.backup2.ibm.com`
 - **value** is canonical name

DNS – Message Format



- Q/R *Query/Response Flag*
- *Operation Code*
- AA *Auth. Answer Flag*
- TC *Truncation Flag*
- RD *Recursion Desired Flag*
- RA *Recursion Available Flag*
- Zero (three resv. bits)
- *Response Code*

DNS – Header Fields

- Identifier: a 16-bit identification field generated by the device that creates the DNS query. It is copied by the server into the response, so it can be used by that device to match that query to the corresponding reply
- Query/Response Flag: differentiates between queries and responses (0 ~ Query, 1 ~ Response)
- Operation Code: specifies the type of message (Query, Status, Notify, Update)
- Authoritative Answer Flag (AA): set to 1 if the answer is authoritative
- Truncation Flag: When set to 1, indicates that the message was truncated due to its length (might happen with UDP, requestor can then decide to ask again with TCP as transport service)
- Recursion Desired: set to 1 if requester desired recursive processing
- Recursion Available: set to 1 if server supports recursive queries

DNS: Caching and Updating Records

- Once (any) name server learns mapping, it caches mapping
 - Stored as “soft state”: Cache entries timeout (disappear) after some time
 - TLD servers typically cached in local name servers
 - Thus, root name servers not often visited

- Updating records, independent of TTL
 - RFC 2136 defines dynamic updates
 - BIND (>8) implements nsupdate (upon TSIG, see below)

Inserting Records Into DNS

- Example: just created startup “Fireblog”
- Register name `fireblog.de` at a registrar (e.g., denic)
 - Need to provide registrar with names and IP addresses of your authoritative name server (**primary** and **secondary**)
 - Registrar inserts two RRs into the de TLD server:
 - (`fireblog.de`, `dns1.fireblog.de`, NS)
 - (`dns1.fireblog.de`, `212.212.212.1`, A)
- Add authoritative server Type A record for `www.fireblog.de` and Type MX record for `fireblog.de`

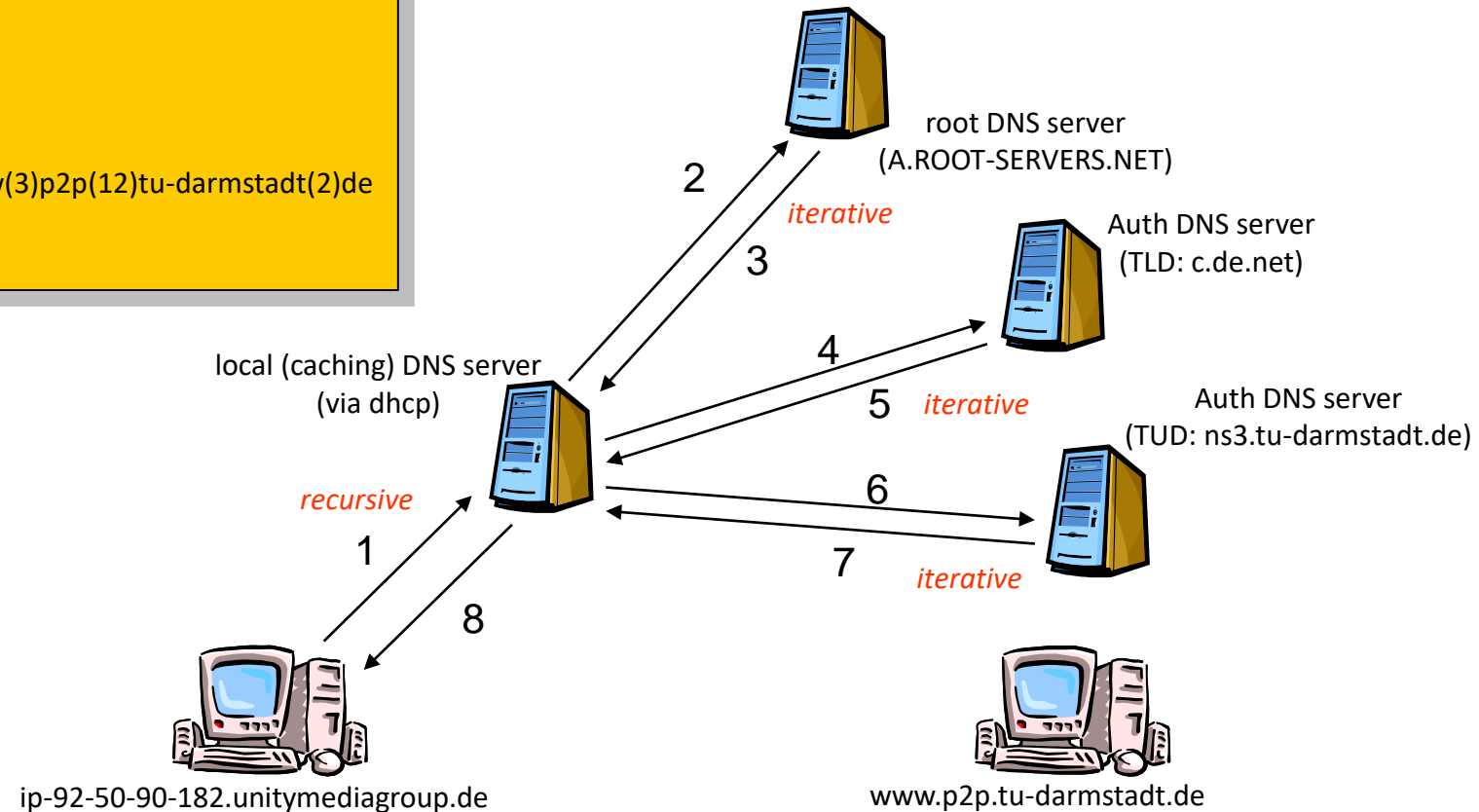
DNS – Recursive and Iterative Queries

DNS HEADER (send)

- Identifier: 0x3116
- Flags: 0x00 (Q)
- Opcode: 0 (Standard query)
- Return code: 0 (No error)
- Number questions: 1
- Number answer RR: 0
- Number authority RR: 0
- Number additional RR: 0

QUESTIONS (send)

- Queryname: (3)www(3)p2p(12)tu-darmstadt(2)de
- Type: 1 (A)
- Class: 1 (Internet)



A Quick Example...

```
strufe@eris:~$ dnstracer -v www.p2p.tu-darmstadt.de
```

```
Tracing to informatik.tu-darmstadt.de[a] via 130.83.163.141, maximum of 3 retries
```

```
130.83.163.141 (130.83.163.141) IP HEADER
```

```
-Destination address: 130.83.163.141
```

```
-DNS HEADER (send)
```

```
-- Identifier: 0x3116
```

```
-- Flags: 0x00 (Q)
```

```
-- Opcode: 0 (Standard query)
```

```
-- Return code: 0 (No error)
```

```
-- Number questions: 1
```

```
-- Number answer RR: 0
```

```
-- Number authority RR: 0
```

```
-- Number additional RR: 0
```

```
-QUESTIONS (send)
```

```
-- Queryname: (3)www(3)p2p(12)tu-darmstadt
```

```
-- Type: 1 (A)
```

```
-- Class: 1 (Internet)
```

```
-DNS HEADER (recv)
```

```
-- Identifier: 0x3116
```

```
-- Flags: 0x8080 (R RA)
```

```
-- Opcode: 0 (Standard query)
```

```
-- Return code: 0 (No error)
```

```
-- Number questions: 1
```

```
-- Number answer RR: 2
```

```
-- Number authority RR: 0
```

```
-- Number additional RR: 0
```

```
-.....
```

```
QUESTIONS (recv)
```

```
- Queryname: (3)www(3)p2p(12)tu-darmstadt(2)de
```

```
- Type: 1 (A)
```

```
- Class: 1 (Internet)
```

```
ANSWER RR
```

```
- Domainname: (6)charon(7)dekanat(10)informatik(12)tu-darmstadt(2)de
```

```
- Type: 1 (A)
```

```
- Class: 1 (Internet)
```

```
- TTL: 1592 (26m32s)
```

```
- Resource length: 4
```

```
- Resource data: 130.83.162.6
```

```
ANSWER RR
```

```
- Domainname: (3)www(3)p2p(12)tu-darmstadt(2)de
```

```
- Type: 5 (CNAME)
```

```
- Class: 1 (Internet)
```

```
- TTL: 49817 (13h50m17s)
```

```
- Resource length: 28
```

```
- Resource data: (6)charon(7)dekanat(10)informatik(12)tu-darmstadt(2)de
```

```
Got answer [received type is cname]
```

So where is the Info?

```
strufe@eris:~$ dnstracer -v -qns tu-darmstadt.de
```

```
Tracing to tu-darmstadt.de[ns] via 130.83.163.130
```

```
130.83.163.130 (130.83.163.130) IP HEADER
```

```
- Destination address: 130.83.163.130
```

```
DNS HEADER (send)
```

```
- Identifier: 0x4C45
```

```
- Flags: 0x00 (Q)
```

```
- Opcode: 0 (Standard query)
```

```
- Return code: 0 (No error)
```

```
- Number questions: 1
```

```
- Number answer RR: 0
```

```
- Number authority RR: 0
```

```
- Number additional RR: 0
```

```
QUESTIONS (send)
```

```
- Queryname: (12)tu-darmstadt(2)de
```

```
- Type: 2 (NS)
```

```
- Class: 1 (Internet)
```

```
DNS HEADER (recv)
```

```
- Identifier: 0x4C45
```

```
- Flags: 0x8080 (R RA)
```

```
- Opcode: 0 (Standard query)
```

```
- Return code: 0 (No error)
```

```
- Number questions: 1
```

```
- Number answer RR: 5
```

```
- Number authority RR: 0
```

```
- Number additional RR: 9
```

```
.....
```

```
QUESTIONS (recv)
```

```
- Queryname: (12)tu-darmstadt(2)de
```

```
- Type: 2 (NS)
```

```
- Class: 1 (Internet)
```

```
ANSWER RR
```

```
- Domainname: (12)tu-darmstadt(2)de
```

```
- Type: 2 (NS)
```

```
- Class: 1 (Internet)
```

```
- TTL: 70523 (19h35m23s)
```

```
- Resource length: 6
```

```
- Resource data: (3)ns1(3)hrz(12)tu-darmstadt(2)de
```

```
ANSWER RR
```

```
- Domainname: (12)tu-darmstadt(2)de
```

```
- Type: 2 (NS)
```

```
- Class: 1 (Internet)
```

```
- TTL: 70523 (19h35m23s)
```

```
- Resource length: 5
```

```
- Resource data: (2)ns(6)man-da(2)de
```

```
ANSWER RR
```

```
- Domainname: (12)tu-darmstadt(2)de
```

```
- Type: 2 (NS)
```

```
- Class: 1 (Internet)
```

```
- TTL: 70523 (19h35m23s)
```

```
- Resource length: 6
```

```
- Resource data: (3)ns2(3)hrz(12)tu-darmstadt(2)de
```

```
.....
```

Answer ctd...

```
.....  
ADDITIONAL RR  
- Domainname:      (3)ns1(3)hrz(12)tu-darmstadt(2)de  
- Type:            1 (A)  
- Class:           1 (Internet)  
- TTL:             17335 (4h48m55s)  
- Resource length: 4  
- Resource data:   130.83.22.63  
ADDITIONAL RR  
- Domainname:      (2)ns(6)man-da(2)de  
- Type:            28 (unknown)  
- Class:           1 (Internet)  
- TTL:             38386 (10h39m46s)  
- Resource length: 16  
- Resource data:   2001:41b8:0000:0001:0000:0000:0000:0053  
ADDITIONAL RR  
- Domainname:      (2)ns(6)man-da(2)de  
- Type:            1 (A)  
- Class:           1 (Internet)  
- TTL:             38386 (10h39m46s)  
- Resource length: 4  
- Resource data:   82.195.66.249  
ADDITIONAL RR  
- Domainname:      (3)ns2(3)hrz(12)tu-darmstadt(2)de  
- Type:            28 (unknown)  
- Class:           1 (Internet)  
- TTL:             17335 (4h48m55s)  
- Resource length: 16  
- Resource data:   2001:41b8:083f:0022:0000:0000:0000:0063  
.....
```

```
.....  
ADDITIONAL RR  
- Domainname:      (3)ns2(3)hrz(12)tu-darmstadt(2)de  
- Type:            1 (A)  
- Class:           1 (Internet)  
- TTL:             17335 (4h48m55s)  
- Resource length: 4  
- Resource data:   130.83.22.60  
ADDITIONAL RR  
- Domainname:      (3)ns2(6)man-da(2)de  
- Type:            1 (A)  
- Class:           1 (Internet)  
- TTL:             38386 (10h39m46s)  
- Resource length: 4  
- Resource data:   217.198.242.225  
ADDITIONAL RR  
- Domainname:      (3)ns3(3)hrz(12)tu-darmstadt(2)de  
- Type:            28 (unknown)  
- Class:           1 (Internet)  
- TTL:             17335 (4h48m55s)  
- Resource length: 16  
- Resource data:   2001:41b8:083f:0056:0000:0000:0000:0060  
ADDITIONAL RR  
- Domainname:      (3)ns3(3)hrz(12)tu-darmstadt(2)de  
- Type:            1 (A)  
- Class:           1 (Internet)  
- TTL:             17335 (4h48m55s)  
- Resource length: 4  
- Resource data:   130.83.56.60  
Got answer
```

DNS – Lessons Learned

- Structure name space (divide et impera)
- Simple „routing“ b/c of structured (hierarchical) namespace

- Store information at multiple locations
- Maintain multiple connections

- → Be redundant! (Replicate...)
 - primary and secondary server, multiple TLD servers

- Delegation using iterative or recursive forwarding
 - (Btw: what are the pros and cons of each?)

Security of the Domain Name System

- Vital service for the Internet

- “Do you know the IP-Address of your mail server?”
- “You know you shouldn’t follow the link

`http://malicio.us/phishing/yourbank.html`

but what about

`http://www.yourbank.de ??`”

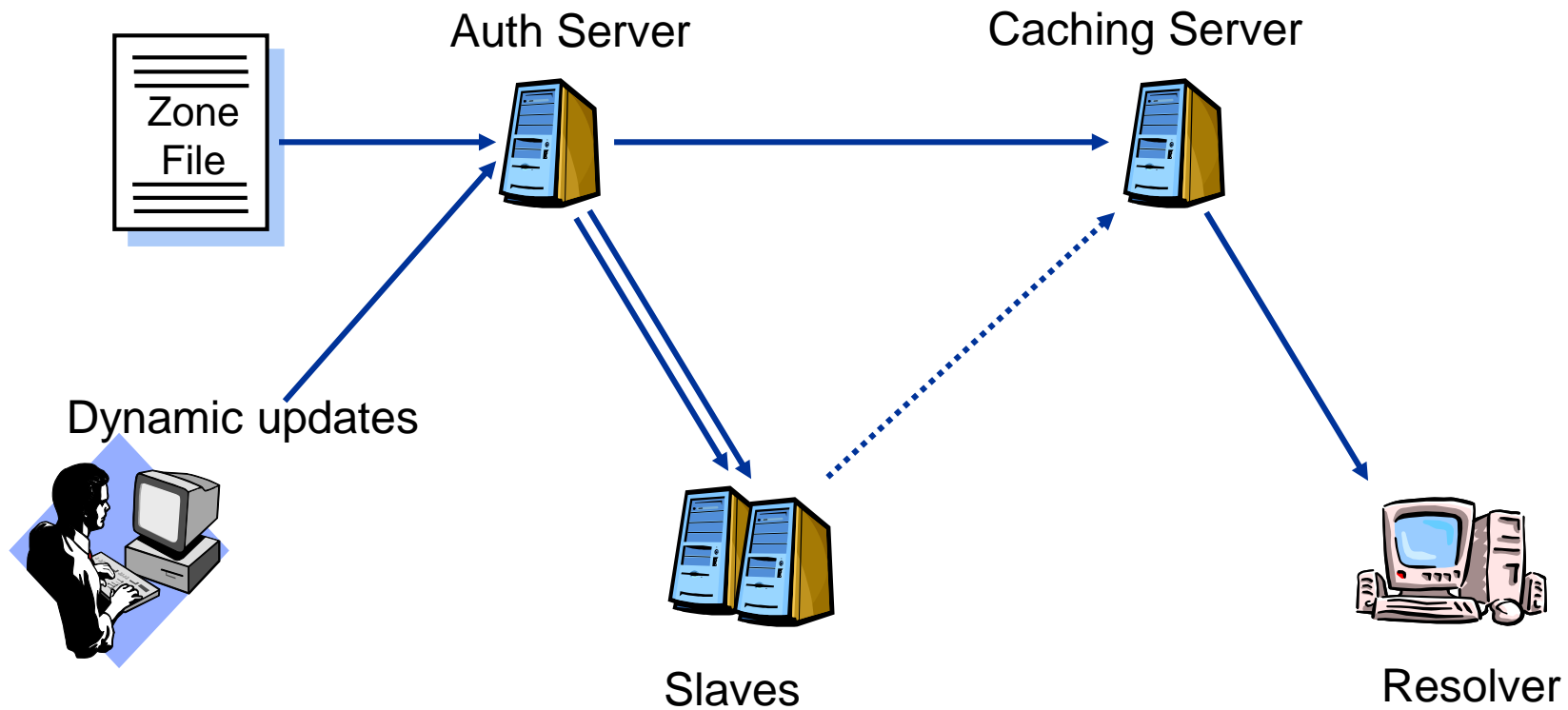
- But: DNS does not support

- Data integrity
- Data origin authentication

- Threats:

- Denial of Service
- Data Authenticity/Integrity

DNS – Data Flow



[<http://www.ripe.net/training/dnssec/>]

DNS Security Issues Outlined

- Robustness towards DDoS
 - General issues
 - Redundancy
- Robustness towards data corruption
 - Cache Poisoning and simple countermeasures
 - More complex countermeasures:
 - Split-Split DNS / Split-horizon DNS
 - Cryptographic countermeasures
 - DNS Cookies
 - DNSSEC
 - DNSCurve

How does this relate to security of routing?

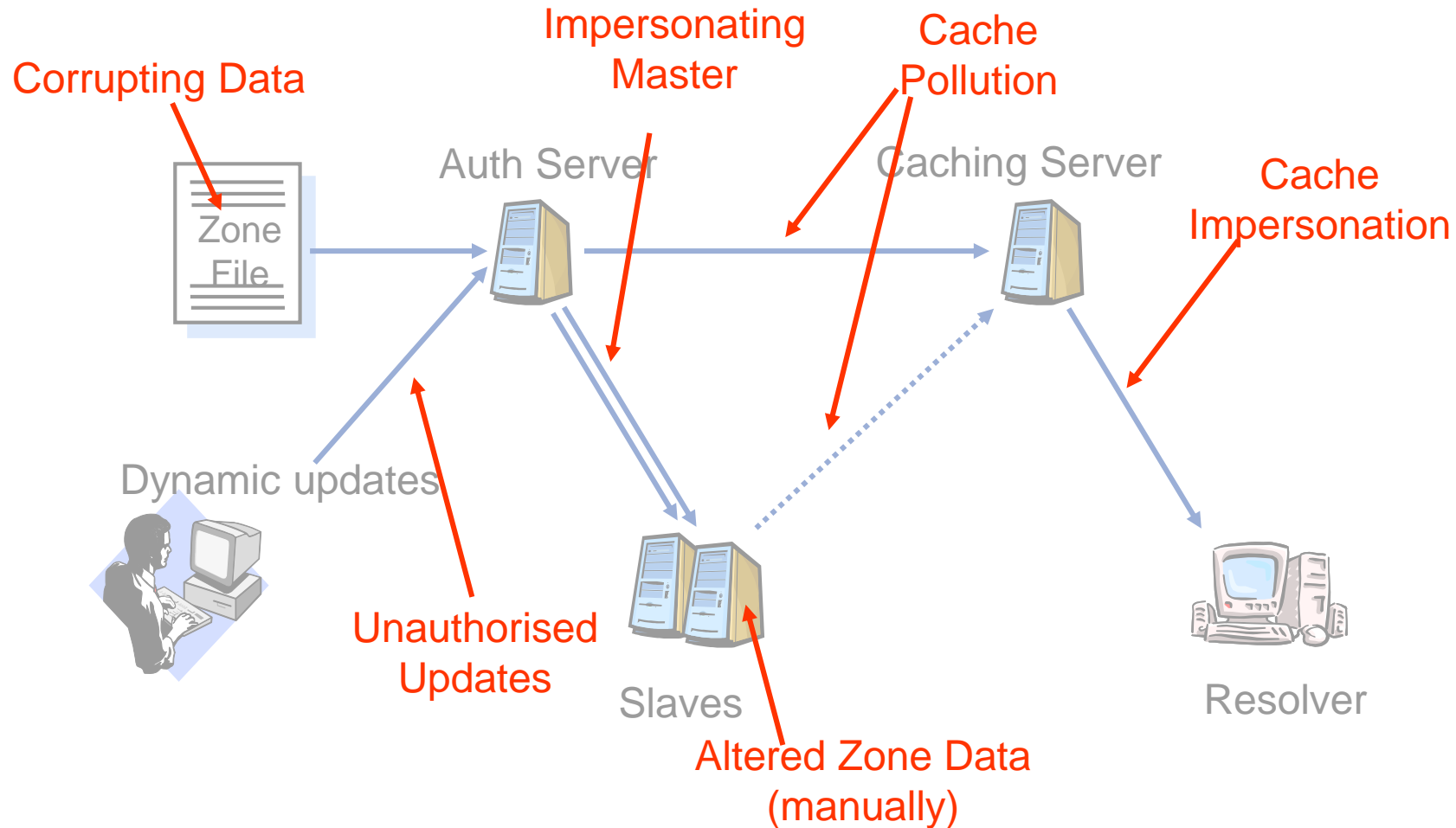
Threats to DNS: Denial of Service

- DNS as vital service a “worthy” target
 - Without DNS most Internet services will not work (usage of names rather than IP-Addresses for numerous reasons!)
- DDoS Attacks on root servers: via notorious “typos” in TLDs
- DNS Amplification Attack (15.02.2006)
 - Spoofed queries (60 Bytes) may generate potentially large responses (4KBytes)
 - Exploit open recursive servers to generate load on other DNS servers
 - Exploit open servers as reflectors when flooding a victim with traffic (via source IP Address spoofing in request)

Robustness towards DDoS

- General issues
 - Secure DNS server
 - OS selection and updates
 - Firewalls
 - Server software selection and updates
 - Redundancy and over-provisioning
 - Root “.”: 13 name server “names” ({a..m}.root-servers.net)
 - “com”, “net”, “de”: several name servers each
 - Anycast
 - Announcement of an IP prefix from multiple locations
 - Requests from different parts of the internet are routed to different machines with the same IP address
 - Done with several DNS servers

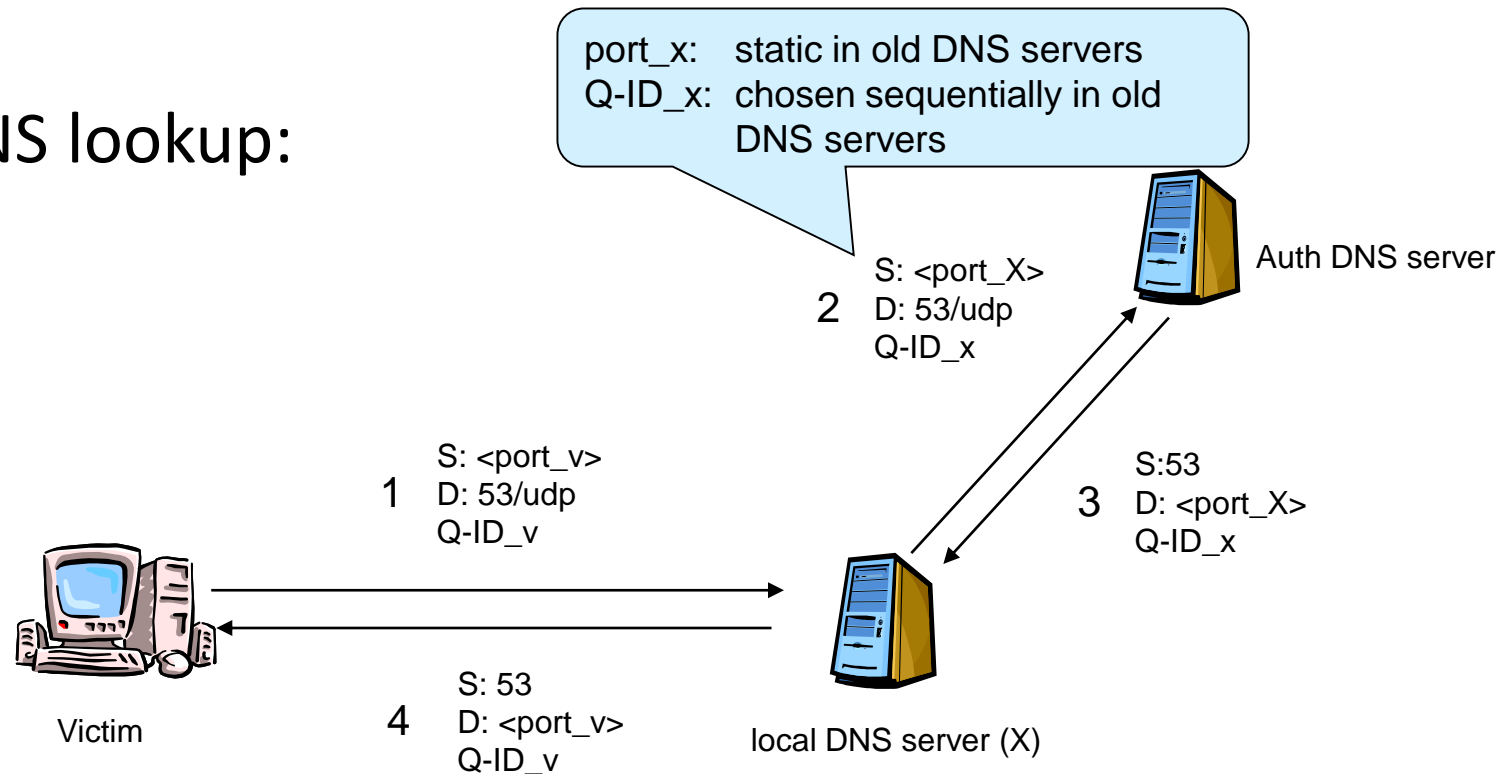
DNS – Threats to Data Integrity



Threats to DNS: Data Corruption / Cache Poisoning

- All resolved RRs are cached at local DNS servers
- DNS slave servers replicate zone data from master

- Normal DNS lookup:

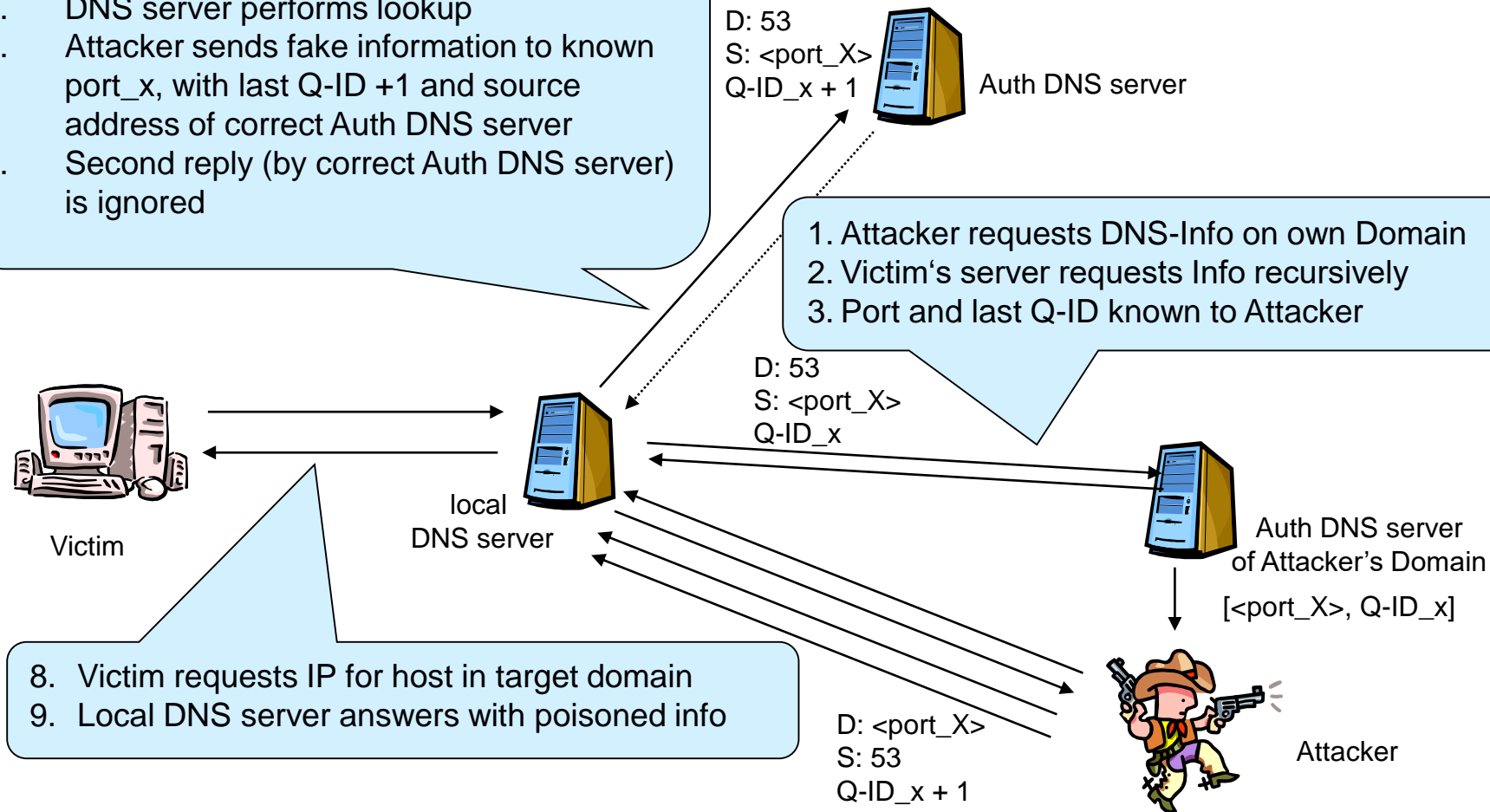


DNS Threats – Cache Poisoning: Simple Poisoning(1)

- **Attack:** plant fake data in slaves / caching servers
(and nobody will realize the redirection from
`www.yourbank.de` to `malicio.us/phishing/yourbank.html ...`)
- DNS via UDP/IP, no handshakes for connection establishment
- Transactions in DNS only identified by tuple of
<auth server(ip-address), auth server(port), transaction id>
- With knowledge about transactions: distribute malicious data
- **IP Address** of authoritative name servers are well known
- In many implementations same **port** for all transactions
- **Q-ID** unknown, but: BIND used to choose them sequentially...

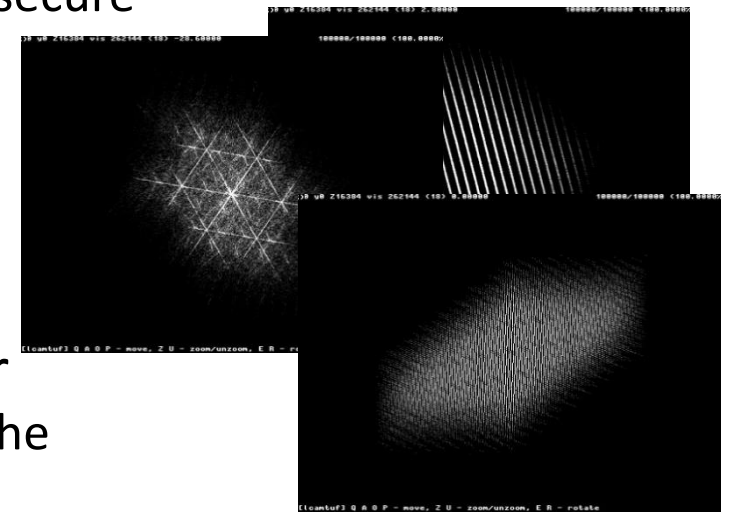
DNS Threats – Cache Poisoning: Simple Poisoning(2)

4. Attacker sends request for target domain
5. DNS server performs lookup
6. Attacker sends fake information to known port_x, with last Q-ID +1 and source address of correct Auth DNS server
7. Second reply (by correct Auth DNS server) is ignored



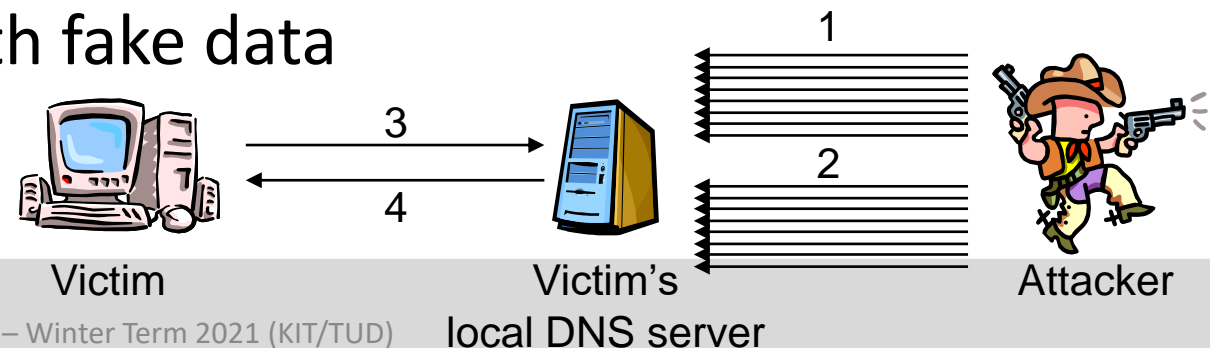
Mitigation of Cache Poisoning

- Random ports for each transaction (BIND8)
 - Since Version 8 BIND uses PRNG for port number and query id selection
 - However PRNG == **Pseudo** Random Number Generator, with knowledge about previous port numbers future port numbers can be guessed if PRNG not cryptographically secure
- More random ports for each transaction (BIND9)
 - New and better PRNG since BIND9, random numbers are harder to guess
- Cache Poisoning only after aging of entry in local DNS server
 - Only if attacker attacks at the right moment, he can poison the cache
 - Typical TTL:
 - 172800 (2d) for most name servers
 - Seconds to hours for A-Entries of organizations (tu-ilmenau.de 24h, deutschebank.de:60mins, commerzbank.de 30mins, postbank.de 30s, microsoft.com:60mins (*where do you get your sec-updates today?*))
- Nevertheless: cache poisoning is still not solved...



Cache Poisoning with “Brute Force”

1. Attacker sends **multitude of requests** for targeted domain to local DNS server of victim and
2. Attacker sends **multitude of fake replies** with IP and port from auth server of targeted domain, guessing transaction id for one of the recursive requests from local caching server to auth server ($2^{16} \times 2^{16} = 2^{32} \approx 4$ Billion possible combinations)
3. Victim requests data about targeted domain
4. Local caching server responds with fake data



More Sophisticated Cache Poisoning

- Usually not a high number of chances **when TTL high**, e.g., 1 day
- Imagine the attacker M:
 - M → Cache: Give me `kslkskdf.bank.com` (w/ random “kslkskdf”)
 - The cache server must now ask the Authoritative Server at `bank.com`
 - M → Cache: Not responsible for `kslkskdf.bank.com`, but `www.bank.com` is. You may reach `www.bank.com` at `A.B.C.D` (`A.B.C.D` being the address of the attacking host)
 - The cache will then ask `A.B.C.D` for `kslkskdf.bank.com` and will also remember the “name server” `www.bank.com`

■

More Sophisticated Cache Poisoning - Defense

- How can we increase the entropy of DNS queries?
- **Idea:** DNS does not distinguish between upper and lower case, encode more bits in the name
- Now the same attack:
 - M → Cache: Give me `kslkskdf.bank.com`
 - The Cache Server must now ask the Authoritative Server at `bank.com`
Cache → Auth Server: Give me `kSLkSkdF.bAnK.COM`
 - M → Cache: Not responsible for `kslkskdf.bank.com`, but `www.bank.com` is. You may reach `www.bank.com` at `141.24.212.114`. (Ignored as `kslkskdf.bank.com` does not match the case of the query)
 - Auth Server → Cache: `kSLkSkdF.bAnK.COM` is unknown
- Entropy is increased to 2^{32+n} for n being the letters in a domain name
- Helps for `www.tu-dresden.de` but not much for `tud.de`

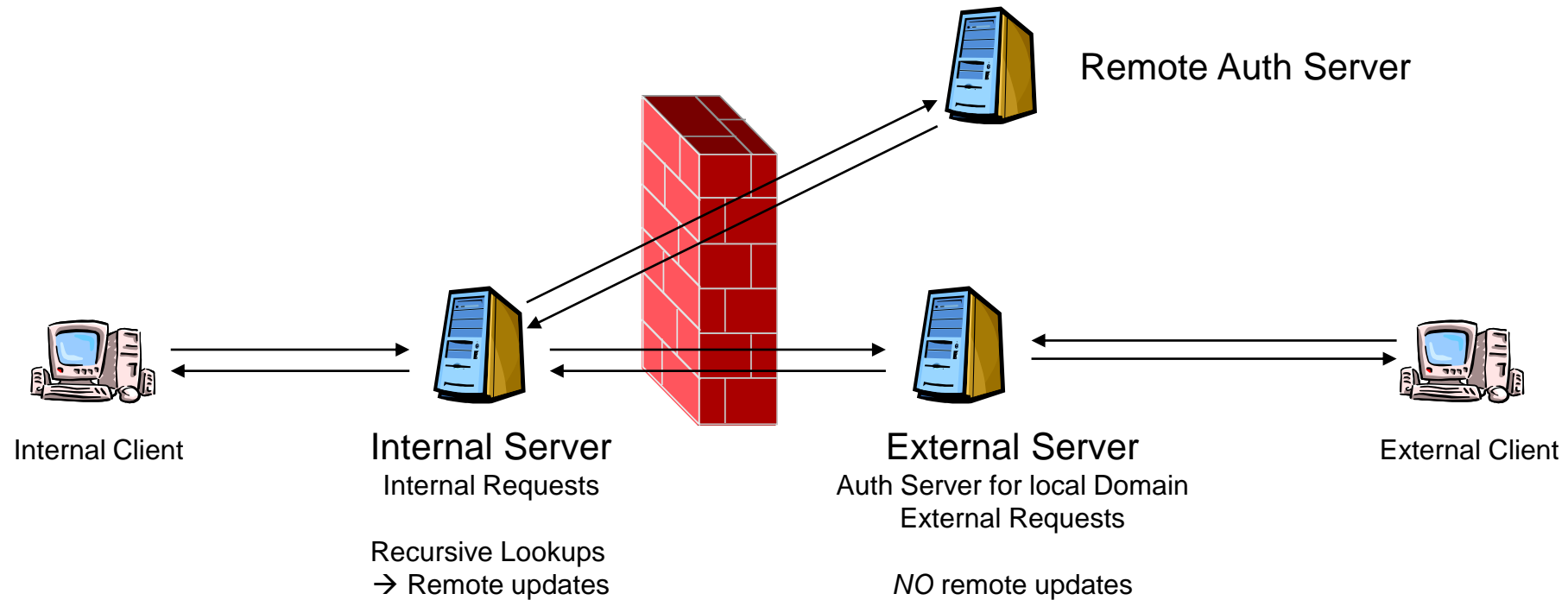
Most Sophisticated Cache Poisoning

- DNS is usually transported over UDP, which may fragment
- What happens when a DNS reply gets fragmented?
 - Random port numbers, Query ID and perhaps the original question (e.g. `kSLkSkdF.bAnK.COM`) are in the first fragment
 - Depending on the query and the MTU the actual answer may be in the second fragment
 - Fragments are matched by a 16 bit identification field...
- Attackers thus can try to
 - Find queries leading to large answers
 - Spoof PMTU related ICMP messages to set the fragmentation boundary
 - Send a “second” fragment with different identifications to the cache
 - Send the query to the cache
 - Wait for the cache to reassemble the reply and the crafted second fragment...
- DNS server should avoid large answers and PMTU discovery...

Prevent Data Corruption: Split-Split DNS

- **Goal:** Avoid cache poisoning from external machines
- **Idea:** Split the name service functions
 - Name resolution (look up of DNS info)
 - Domain information (Auth service of local DNS info)
- Internal server
 - Implements name resolution
 - Performs recursive look-ups at remote DNS servers
 - Located behind firewall and only accepts requests from local LAN
- External server
 - Authoritative server of domain
 - Accepts remote requests but **never accepts** remote updates
- Zone transfer from external to internal server allowed

Split-Split DNS/ Split-Horizon DNS



DNS Cookies

- Goals
 - DNS transaction security
 - Prevent off-path attacks (poisoning)
 - Limit spoofing, DoS

- Core ideas:
 - „Authenticate“ query in DNS response
 - Establish semi-state between clients and servers

- Approach:
 - EDNS option
 - Include client/server cookies (extending query ID)

[Eastlake, Andrews: „RFC 7873: Domain Name System Cookies “]

DNS Cookies – States and Behavior

- States of communication
 1. „Unauthenticated“: Client contacts server for the first time
 2. „Authenticated“: Client and server share some state

- Server behavior:
 1. Provide service at low priority, offer server cookie
 2. Provide normal DNS service (return server cookie)

- Client behavior:
 1.
 - a. Send DNS query, client cookie and ask for server cookie, OR
 - b. Query for server cookie, then (2)
 2. Send server cookie, DNS query, client cookie

DNS Cookies – Cookies and OPT RR

- Generating DNS cookies:

- ```
cookie ← PRNG(Client IP, Server IP, k)
```
- with temporary secret k at server/client*

- Message format:

| 0                          | 1 | 2             | 3 |
|----------------------------|---|---------------|---|
| Option Code = 10           |   | Option Length |   |
| Client Cookie ([0-3])      |   |               |   |
| Client Cookie ctd. ([4-7]) |   |               |   |
| Server Cookie              |   |               |   |
| Server Cookie ctd.         |   |               |   |

- Option length: [8 (only client cookie), {>=16, <=40} (client/server cookies)]
- Client cookie: fixed size 8 octets
- Server cookie: variable size, 8-32 bytes

# DNS Cookies - Protection

- So what do DNS cookies actually achieve?
- Protects transactions between resolvers/caches and caches/servers
- Validation of cookies:
  - Valid client cookie for server (probably no off-path poisoning)
  - Valid server cookie for client (previous transactions with same IP address, probably no IP address spoofing)
- Critical assessment:
  - Very low overhead, no protocol changes, little software adaptation
  - Currently rolled out
  - However: very weak protection (*on-path adversaries? Leaked cookies?*)
  - Unclear how servers should behave in phase 1

# Securing DNS Cryptographically

- Securing DNS has different goals:
  - DNS transaction security
    - Peer/message authentication
  - DNS data security
    - Data origin authentication
    - Authenticated denial of existence

# Transaction Authentication (TSIG)

- **Idea:**

- Use signatures to secure data at zone transfer **master** → **slave**
- Pre shared symmetric key at each entity
- MD5 Hash used as signature

- **TSIG Resource Record:**

(Name, Type ("TSIG"), Class ("ANY"), TTL("0"), Length, Data(<signature>))

- Possibility to authenticate, but very complex to administrate in large domains (manual pre-sharing of keys)
  - amount of keys required:  $\frac{n(n-1)}{2}$
- Main application areas:
  - Secure communication between stub resolvers and security aware caching servers (?)
  - Zone transfers (master → slave)
  - Combined with nsupdate in data centers, to update stale information in caches

# DNS Security (DNSSEC) – Objectives

- DNS security ***objectives***:
  - End-to-end zone data *origin authentication* and integrity
  - *Detection* of data corruption and spoofing
  
- DNSSEC ***does not*** (want to) provide:
  - DoS-Protection (*in fact, it facilitates DoS Attacks on DNS servers*)
  - Data delivery guarantees (availability)
  - Guarantee for correctness of data (only that it has been signed by some authoritative entity)

[Eastlake: „RFC 2535: Domain Name System Security Extensions“ (obsolete)]

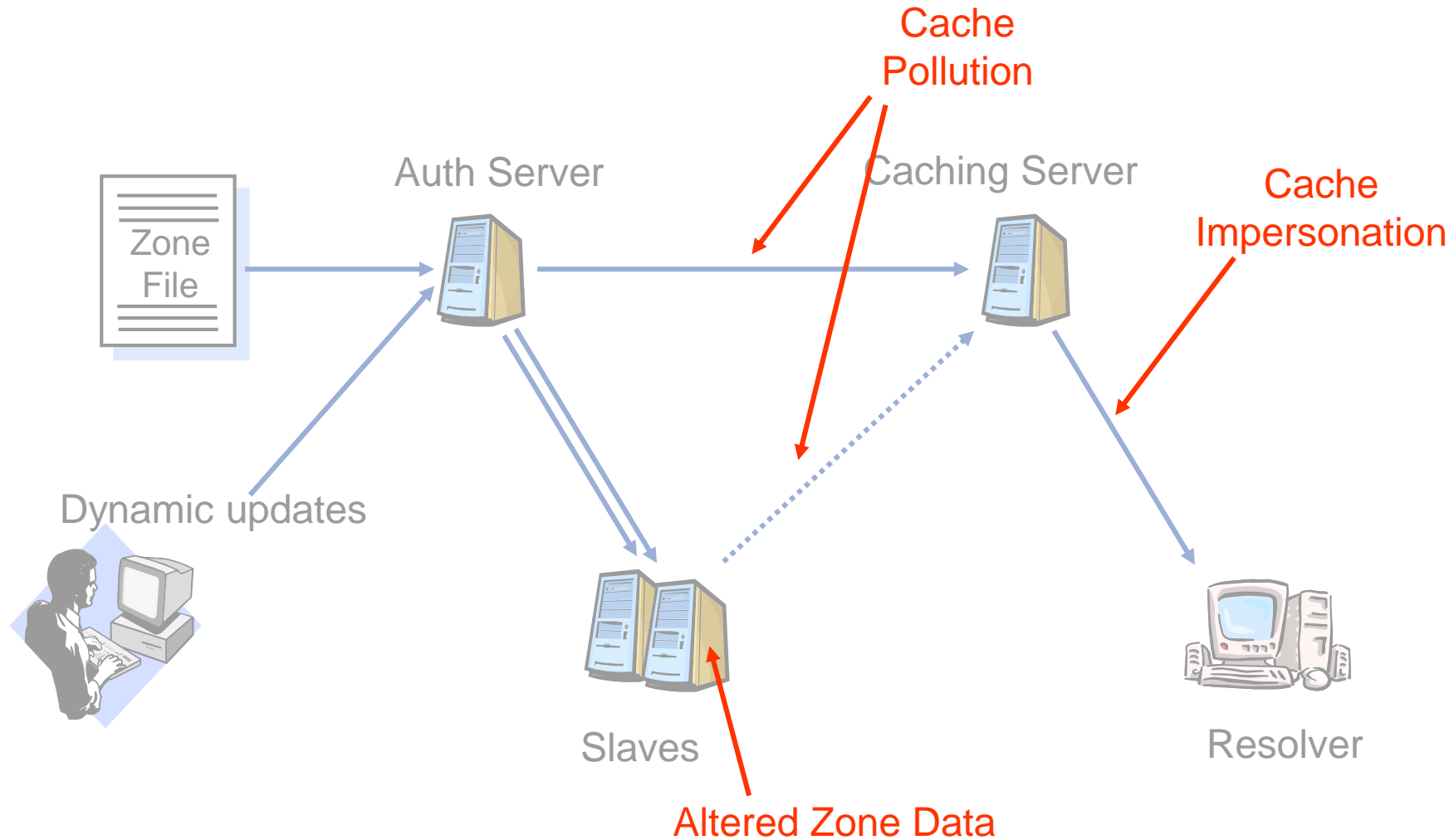
[Arends et. al: „RFC 4033: DNS Security Introduction and Requirements“]

[RFCs:4033,4034,4035,4310,4641]

# DNSSEC

- Usage of *public key cryptography* to allow for ***data origin authentication*** on a world wide scale
- **RRSets** (groups of RRs) signed with private key of authoritative entities
- **Public keys** (DNSKEYs) published using DNS
- Distinguish *zone signing key (ZSK)* and *key signing key (KSK)*
- Child zone keys are authenticated by parents (according to the zone hierarchy) and hence anchored trust chains established
- Only root zone key signing key (KSK) needed (manual distribution) to create complete trust hierarchy (in theory)
- Until then: islands of trust with manually shared anchor keys
- No key revocation → DNSSEC keys should have short expiration date (quick rollover)

# DNSSEC – Targeted Threats



# DNSSEC – Means of Securing RRsets

- Goal: authenticity and integrity of Resource Record Sets
- Means:
  - Public Key Cryptography (with Trust Chains)
  - Security integrated in DNS (new RRs)
- New Resource Record Types:
  - RRSig: RR for signatures to transmitted RRs
  - DNSKEY: RR for transmission of public keys
  - DS: RR for trust chaining (trust anchor signs key of child zone)
  - NSEC: RR for next secure zone in canonical order (authenticated denial for requested zone)



# DNSSEC – New Resource Records: RRSIG

- Resource Record for transmission of *signatures*
- RRSIG:  
(Name, Type, Algorithm, Labels, TTL, Sig Expiration, Sig Inception, Key Tag, Signer's Name, Signature)
  - Name – name of the signed RR
  - Type – RRSIG (46)
  - Algorithm – MD5(1), Diffie-Hellman(2), DSA (3)
  - Labels – number of labels in original RR (wildcard indication)
  - TTL – TTL at time of signature inception
  - Signature Expiration – End of validity period of signature
  - Signature Inception – Beginning of validity period of signature
  - Key Tag – ID of used key if signer owns multiple keys
  - Signer's Name – Name of the signer
  - Signature – Actual Signature

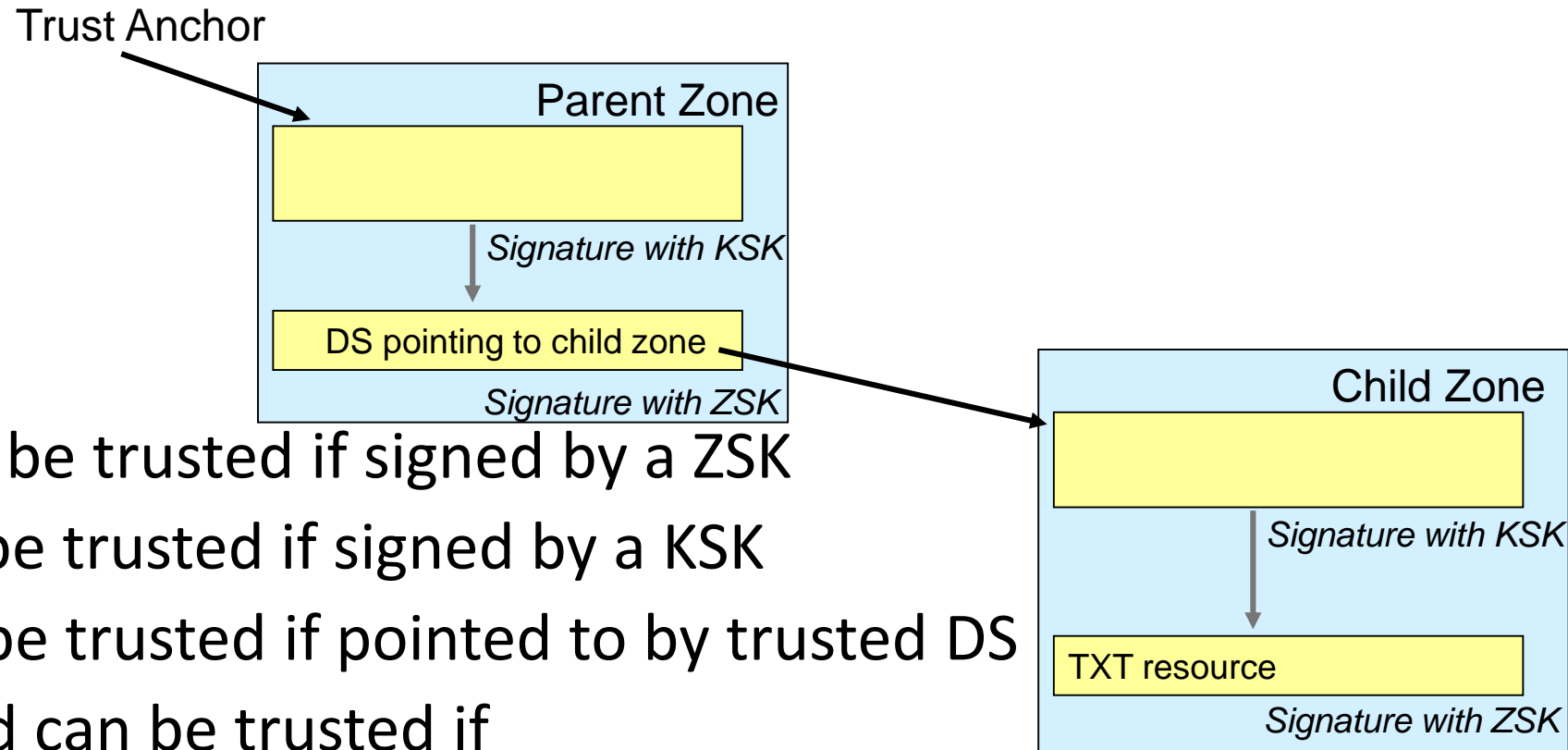
# DNSSEC – New Resource Records: DNSKEY

- Resource Record containing *public keys* for distribution
- DNSKEY : (Label, Class, Type, Flags, Protocol, Algorithm, Key)
  - Label – Name of key owner
  - Class – Always: IN (3)
  - Type – DNSKEY
  - Flags – key types: Key Signing Key (257) or Zone Signing Key (256)
  - Protocol – Always DNSSEC (3)
  - Algorithm – RSA/MD5(1), Diffie-Hellman(2), DSA/SHA-1(3), elliptic curves(4), RSA/SHA-1(5)
  - Key – Actual key

# DNSSEC – New RRs: Delegation Signer (DS)

- DS contains ***hash-value of DNSKEY*** of the name server of a sub zone
- Together with NS Resource Record, DS is used for trust chaining
- DS : (Name, Type, Key Tag, Algorithm, Digest Type, Digest)
  - Name – Name of the chained sub zone
  - Type – DS
  - Key Tag – Identification of the hashed key
  - Algorithm – RSA/MD5(1), Diffie-Hellman(2), DSA(3) (of referred DNSKEY)
  - Digest Type – SHA-1(1), SHA-256(2)
  - Digest – Actual value of hashed DNSKEY

# DNS – Authority Delegation and Trust Chaining



- Data can be trusted if signed by a ZSK
- ZSK can be trusted if signed by a KSK
- KSK can be trusted if pointed to by trusted DS
- DS record can be trusted if
  - Signed by parents ZSK
  - Signed by locally configured trusted key

# DNS – Authority Delegation and Trust Chaining (Example)

Trusted Key  
(locally configured)

| Parent Zone |       |                  |  |
|-------------|-------|------------------|--|
| child       | NS    | ns.child         |  |
|             | DS    | (...) <KSK-id>   |  |
|             | RRSIG | DS (...) parent. |  |

| Child Zone |        |                                     |  |
|------------|--------|-------------------------------------|--|
| @NS        |        |                                     |  |
| ns         | DNSKEY | (...) <KSK-id>                      |  |
|            | DNSKEY | (...) <ZSK-id>                      |  |
|            | RRSIG  | dnskey (...) <KSK-id> parent.       |  |
|            | RRSIG  | dnskey (...) <ZSK-id> child.parent. |  |
| ns         | A      | 10.5.1.2                            |  |
|            | RRSIG  | A (...) <ZSK-id> child.parent.      |  |
| www        | A      | 10.5.1.3                            |  |
|            | RRSIG  | A (...) <ZSK-id> child.parent.      |  |

# DNSSEC – New Resource Records: NSEC

- Next Secure (NSEC) gives information about the next zone / sub domain in canonical order (last entry points to first entry for the construction of a closed ring)
- Gives the ability to prove the non-existence of a DNS entry: *Authenticated Denial*
- NSEC (Name, Type, Next Domain)
  - Name – Name of the signed RR
  - Type – NSEC (47)
  - Next Domain – Name of the next domain in alphabetical order
- *Allows adversary to crawl entire name zone (“zone walking”)*

# DNSSEC – New RRs: NSEC3 (1)

- Successor to NSEC: NSEC3 and NSEC3PARAM
- Uses hashed domain names to make zone walking more difficult
- Hashing based on salt and multiple iterations to make dictionary attacks more difficult
  
- NSEC3
  - Name
    - Name of the signed RR
  - Type
    - NSEC3 (50)
      - SHA-1 (1)
  - Hash Algorithm
    - To Opt-Out unsigned names
  - Flags
    - Number of iterations of Hash Algorithm
  - Iterations
    - Length of the salt value
  - Salt Length
    - Actual salt value
  - Salt
    - Output length of hash function
  - Hash Length
    - Next Hash of domain name in alphabetical
  - Next Hashed Owner Name
    - order

# DNSSEC – New RRs: NSEC3 (2)

- Potential advantage: Salting and hashing does not allow for easily deducing hostnames from zone walks
- Problem:
  - Hostnames usually have very low entropy (to remember them)
  - Easy dictionary attacks - despite the usage of salts & iterations
  - But not used heavily anyways:
    - .: Uses NSEC
    - .com: No salt, No iterations
    - .de: Static salt BA5EBA11, 15 Iterations

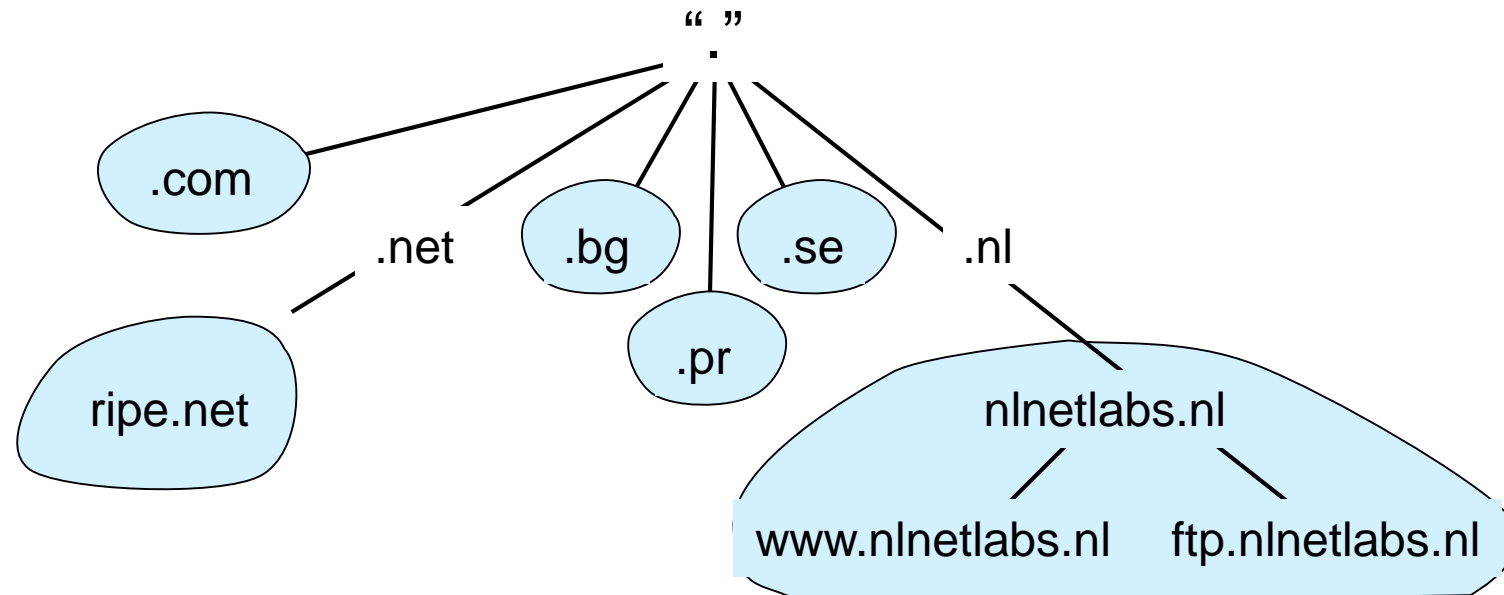


# DNSSEC: NSEC5 / Record Type Denial

- Provide authenticated denial of existence without leaking names requires online signing.
- Providers do not want to trust the DNS servers with keys...
- Cloudflare Record Type Denial
  - Send positive response but deny requested record type

# DNSSEC – Trusted Zones

- Potential anchors
- TLD: .com, .net, .nl, .se, .cz, ..., .de
- Diversity of others: 892208 DNSSEC enabled zones



<http://secspider.cs.ucla.edu/islands.html>  
<http://secspider.verisignlabs.com/>

# DNSSEC Issues

- Pro's:

- DNSSEC allows to prevent unauthorized/spoofed DNS records

- Con's:

- Added complexity (signing, checking, key distribution) **eases DoS attacks** on DNS servers
- Zones need to be signed completely (performance challenge for large companies or registries)
- Authenticated denial with NSEC gives the possibility to “walk” the chain of NSEC and to gain knowledge on the **full zone content** (all zones/ sub domains) in  $O(N)$  ==> NSEC3, ...
- Distribution of anchor keys still a **manual task** (allows for human error, social engineering)

## Deployment:

- <https://www.internetsociety.org/deploy360/dnssec/maps/>

# Alternatives to DNS(SEC)

- Some exotic approaches
  
- „Transport Layer“ Encryption
  - DNSCurve
- Distributed Name Resolution
  - Peer Name Resolution Protocol
  - GNU Name System
  
- Try different strategies to deal with Zookos triangle, informally stating that global name resolution can only offer any two of the following properties:
  - Security
  - Distributed system
  - Human-memorable Names

## DNSSCurve (I)

- Developed by Daniel Bernstein as an alternative to DNSSEC [Ber09]
- Uses online cryptographic functions, i.e., more like transport layer encryption
- Exchanges (either binary or tunneled in DNS TXT records)
  
- Online operation possible using very efficient ciphers: Curve25519 (ECC based pub key system), Poly1305 (hash function) and Salsa20 (stream cipher) all developed by djb

## DNSSCurve (II) – Distribution of public keys

- Public keys are not validated using novel records, but DNS names
- Name servers are no longer called ns.example.org but like *uz5<public\_key>.example.org*
  - uz5 means name server speaks DNSSCurve
  - Rest of the string encodes 256 bit public key
- Parent zones pointing to DNSSCurve names allow to verify sub zones
- DNSSCurve names in hyperlinks etc. allow to verify DNSSCurve servers independent of DNS hierarchy
  - → No more trust in root zone

# DNSCurve (III) – Discussion

- Slimmer design compared to DNSSEC
- Independent trust paths possible
- Support for confidentiality
- No amplification attacks possible
  
- But use of hard defined cryptographic operations is not globally acceptable
- Cipher operations are not as well examined as ciphers in DNSSEC
- *Hasn't caught on.*

# Peer Name Resolution Protocol (PNRP)

- Distributed protocol invented by Microsoft based on Pastry
- Deployed with Windows systems (disabled by default , requires IPv6)
- All computers are equal nodes in structured overlay networks (called clouds here)
  - Link cloud (for all local computers)
  - Site cloud (e.g. for all computers of a company)
  - Global cloud (for all PNRP speakers in the Internet)
- Names in the form *name.pnrp.net* can be resolved insecurely (pnrp.net being a special domain to map PNRP names in the DNS name space)
- Names in the form [p]classifier.[p]h(pkey)<authority>.pnrp.net, with authority being a SHA-1 hash of a node's public key can be resolved securely



# PNRP – Properties

- Abandoning servers might increase robustness and availability of overall system, but no wide research (yet)
  - Some mechanisms to prevent well known eclipse and sybil attacks
- Different clouds allow for increased performance and robustness within same network locations
- Secure names can be verified independently of a trust hierarchy
- Secure names cannot be remembered
- Secure names cannot be verified by a trust hierarchy
- In the insecure name space: anarchy!
  - No security guarantees at all
  - Anybody may register and reregister addresses...

## GNU Name System (GNS)

- Distributed name resolution system based on GUNet (spanning a Kademia-like structured peer-to-peer overlay)
- Offers secured names in the form *hash.zkey*
- Not memorable, but GNS offers aliases (so called *petnames*)
  - Every participant may create aliases like *alias.gnu* pointing to *something.zkey*
  - and aliases may be recursive (if users permit)... thus *bob.alice.gnu* points to a system that Alice (known by local system) calls bob
- Allows locally unambiguous names with a clear trust hierarchy
- Global names still possible by unmemorable names

# Summary

- DNS a central service of the Internet, implemented on layer 7
- Vital for secure operations
- Vulnerabilities:
  - DoS
  - Poisoning
- Countermeasures
  - Better PRNG
  - Split/Split DNS
  - TSIG
  - DNSSEC
  - DNS Cookies
  - *DNSSCurve, PNRG, GNS*