University of Rome Tor Vergata
ICT and Internet Engineering

Network and System Defense

Marco Bonola, Alessandro Pellegrini, Angelo Tulumello

A.A. 2023/2024

Lecture 11: DNS Security

Angelo Tulumello

sources: blog.cloudflare.com + extra material (tutorials, blogs, RFCs, etc...)

Intro

- The Domain Name System (DNS) is one of the oldest and most fundamental components of the modern Internet
- DNS maps domain names to Internet Protocol (IP) addresses
- Guess what? In the early 1980s, when DNS was designed, there was no consideration for strong security mechanisms in the protocol
- As the network grew and evolved, DNS remained unchanged as an insecure and unauthenticated protocol
- In 1993, the IETF started a public discussion around how DNS could be made more trustworthy. Eventually, a set of extensions to DNS, called Domain Name System Security Extensions (***DNSSEC***), were settled on, and formally published in 2005

A brief recap

DNS: A Distributed Key Value Store Before It Was Cool

- If a client wants to connect to an address such as ‘www.example.com’ and needs to know which IP address corresponds to this address, they can ask DNS. Typically, all DNS messages are sent over UDP.
- There are several types of ***Resource Records*** (***RR***) that DNS can answer questions about
  - One of the most important RRs is called an ***“A record”***, which stores the IPv4 address associated with the domain
- To find out the value of any record for a given domain, you can ask a DNS resolver. For example, it you wanted the IP address for www.example.com, the question you could ask is:
  - “What is the A record for the domain www.example.com?”

```
Transaction ID: 0x5ad4
▽ Flags: 0x0100 Standard query
    0... .... .... .... = Response: Message is a query
    .000 0... .... .... = Opcode: Standard query (0)
    .... ..0. .... .... = Truncated: Message is not truncated
    .... ...1 .... .... = Recursion desired: Do query recursively
    .... .... .0.. .... = Z: reserved (0)
    .... .... ...0 .... = Non-authenticated data: Unacceptable
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
▽ Queries
  ▽ example.com: type A, class IN
      Name: example.com
      [Name Length: 11]
      [Label Count: 2]
      Type: A (Host Address) (1)
      Class: IN (0x0001)
```

DNS Request

DNS Response

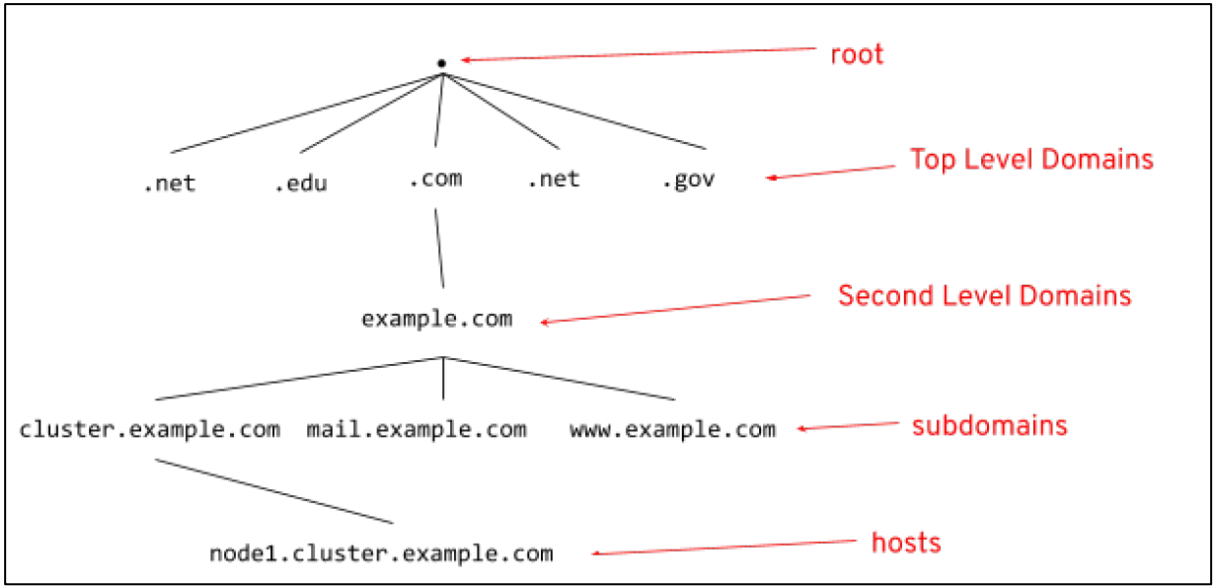
si possono avere più query e più risposte

```
Transaction ID: 0x5ad4
▽ Flags: 0x8180 Standard query response, No error
    1... .... .... .... = Response: Message is a response
    .000 0... .... .... = Opcode: Standard query (0)
    .... .0.. .... .... = Authoritative: Server is not an authority for domain
    .... ..0. .... .... = Truncated: Message is not truncated
    .... ...1 .... .... = Recursion desired: Do query recursively
    .... .... 1... .... = Recursion available: Server can do recursive queries
    .... .... .0.. .... = Z: reserved (0)
    .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated by the server
    .... .... ...0 .... = Non-authenticated data: Unacceptable
    .... .... .... 0000 = Reply code: No error (0)
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
▽ Queries
  ▽ example.com: type A, class IN
      Name: example.com
      [Name Length: 11]
      [Label Count: 2]
      Type: A (Host Address) (1)
      Class: IN (0x0001)
▽ Answers
  ▽ example.com: type A, class IN, addr 93.184.216.119
      Name: example.com
      Type: A (Host Address) (1)
      Class: IN (0x0001)
      Time to live: 19071
      Data length: 4
      Address: 93.184.216.119 (93.184.216.119)
```

How DNS works (brief recap)

- ***DNS is a distributed key/value database***
  - The ***values*** returned can in theory be anything but in practice need to fit into well known types, such as addresses, mail exchanges, sever lists, free format text records etc.
  - The ***keys*** consist of a name, type, and class
- The name space is ***hierarchical*** (see below) and DNS information is “published” by ***Authoritative Servers***
  generalmente per ogni componente di questa gerarchia si ha un authoritative server che mantiene le informazioni di questo dominio. Il root inoltra la richiesta ad un top level domain ad esempio '.com'. quando si fa una query si parte dal root per poi scendere nella gerarchia.
- DNS Resolvers will locate the information requested by asking the appropriate authoritative servers by ***following the naming hierarchy from one Authoritative DNS server to the next one***
- The ISP typically provides a ***Recursive Resolver*** that will perform resolution on behalf of customers
  la risoluzione della query è ricorsiva (offerta dall'internet service provider). si può avere anche nel pc il recursive resolver, ma non è questo il caso perché la rete ointernet è molto vasta.
- Web-enabled applications like browsers use something called a ***Stub Resolver*** to interact with the DNS

The DNS namespace



The DNS namespace

- DNS names consist of labels that are separated by dots.
- Thus “www.example.com.” consists of 4 labels
  - “www” (***leaf***, or ***subdomain***)
  - “example” (***domain***)
  - “com” (***TLD***)
  - “.” i.e. the ***root***.
- Resolvers search by starting at the longest match of the labels,
  - i.e., if it only knows about the root then, it starts the search there. If they know about .com they start there

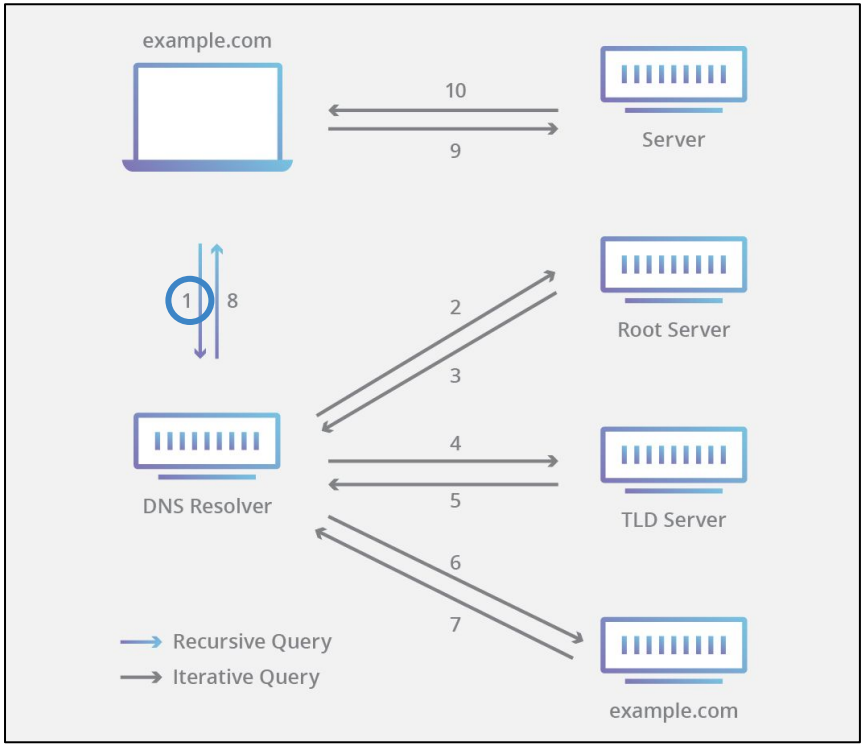
The DNS namespace

- There are ***13 rootservers*** maintained by 12 different organizations.
- The root zone is maintained by ***IANA***, which is a part of ICANN, and is published by Verisign which operates two of the root servers
- The contents of the root zone is ***a list of the authoritative servers for each TLD***
- The answer from the root server contains a set of records called ***NS (Name Server)***. These records list the nameservers which should have more information about the requested zone, i.e., the ".com" servers.
- Each one of the TLDs authoritative nameservers knows about the ***authoritative nameservers for the domains under them*** (cloudflare.com, google.com, etc.).
- Following this downward, you will eventually get to the server that can answer queries about records for the name you are looking for.

Recap: the 4 DNS servers involved in a query

- ***DNS recursor*** - The DNS recursor is a server designed to receive queries from client machines through applications such as web browsers. Typically the recursor is then responsible for making additional requests in order to satisfy the client's DNS query
- ***Root nameserver*** - The root server is the first step in translating (resolving) human readable host names into IP addresses
- ***TLD nameserver*** - The top level domain server (TLD) can be thought of as a specific rack of books in a library. This nameserver is the next step in the search for a specific IP address, and it hosts the last portion of a hostname (In example.com, the TLD server is "com")
- ***Authoritative nameserver*** - This final nameserver can be thought of as a dictionary on a rack of books, in which a specific name can be translated into its definition. The authoritative nameserver is the last stop in the nameserver query. If the authoritative name server has access to the requested record, it will return the IP address for the requested hostname back to the DNS Recursor that made the initial request

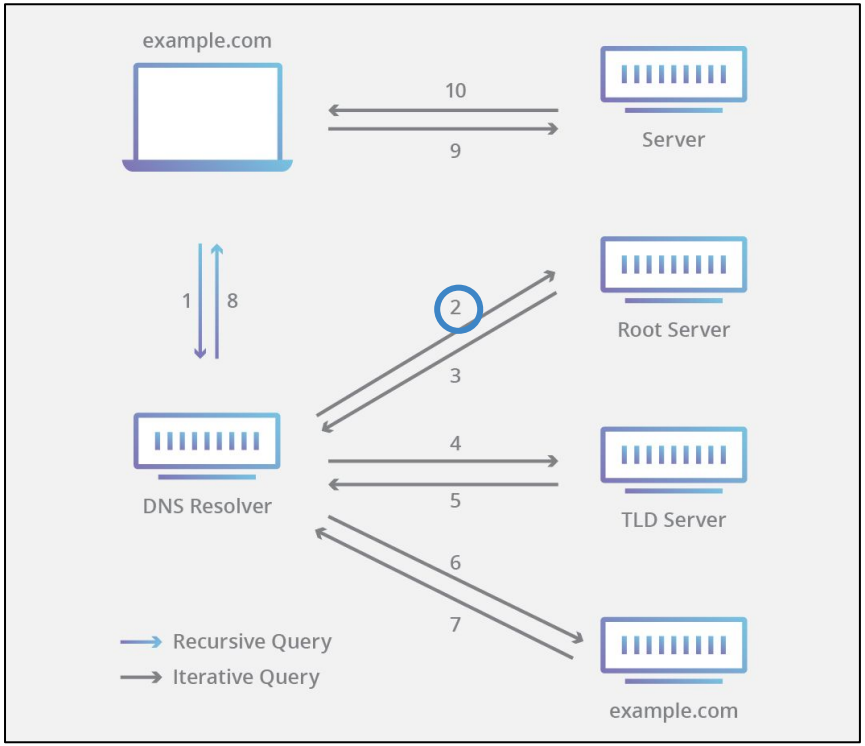
The resolution process



il pc chiede direttamente il nome di dominio a cui è interessato.
Questa query viene presa dal DNS resolver (programm in PC, in access gateway o DNS resolver in ISP).
DNS resolver lo immaginiamo 'blank' (senza cache).
DNS resolver contatta un root server che risponde con l'IP di un authority DNS server per il dominio cercato, in questo caso '.com'. l'IP del TLD da contattare.
il DNS resolver invia la richiesta al TLD server di cui ha ricevuto l'IP. Questo risponde con l'IP dell'authoritative nameserver.

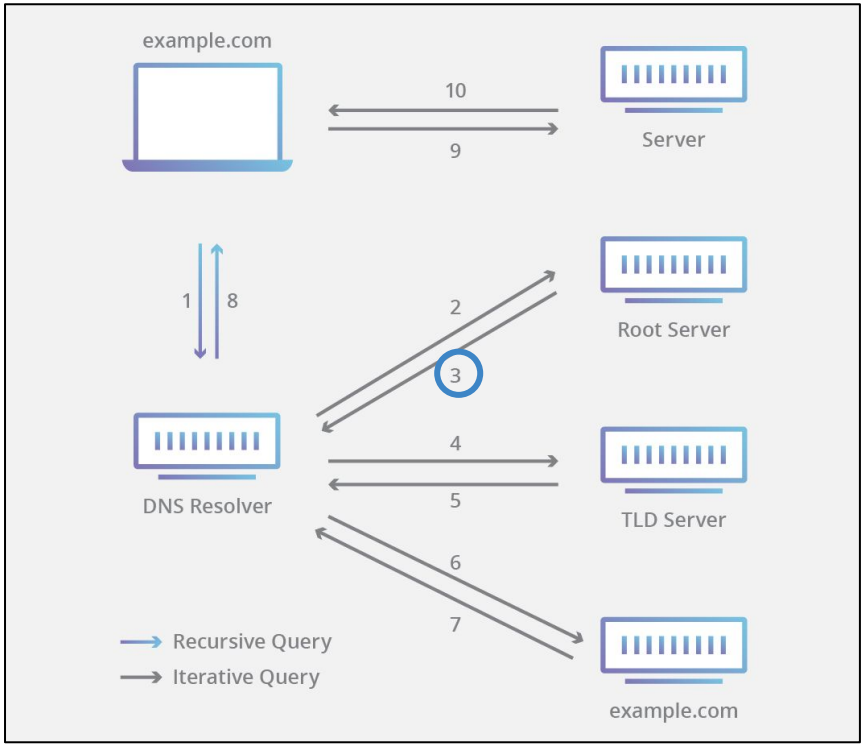
A user types ‘example.com’ into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.

The resolution process



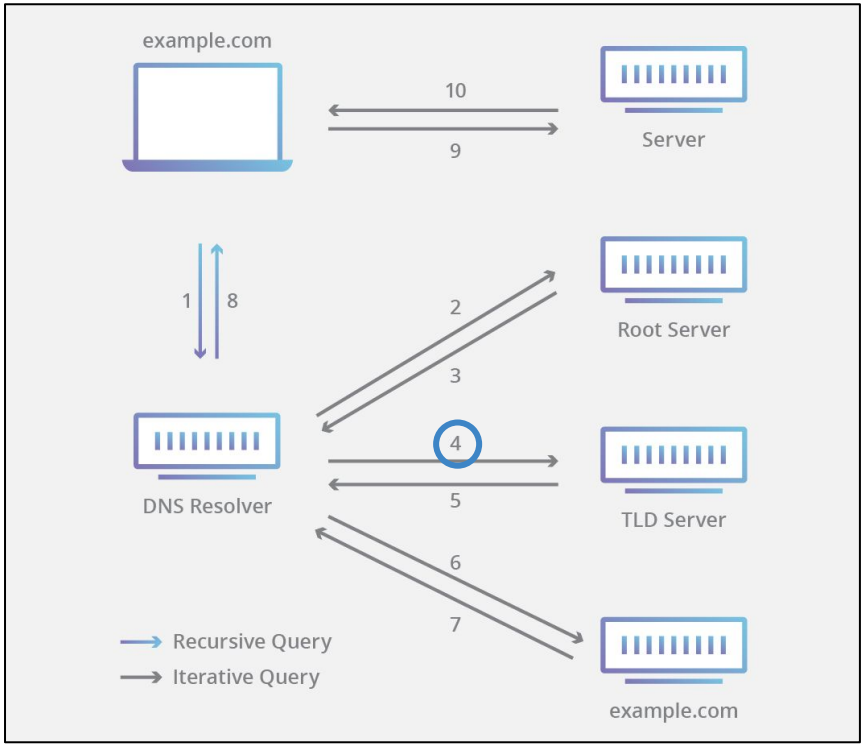
The resolver then queries a DNS root nameserver (.)

The resolution process



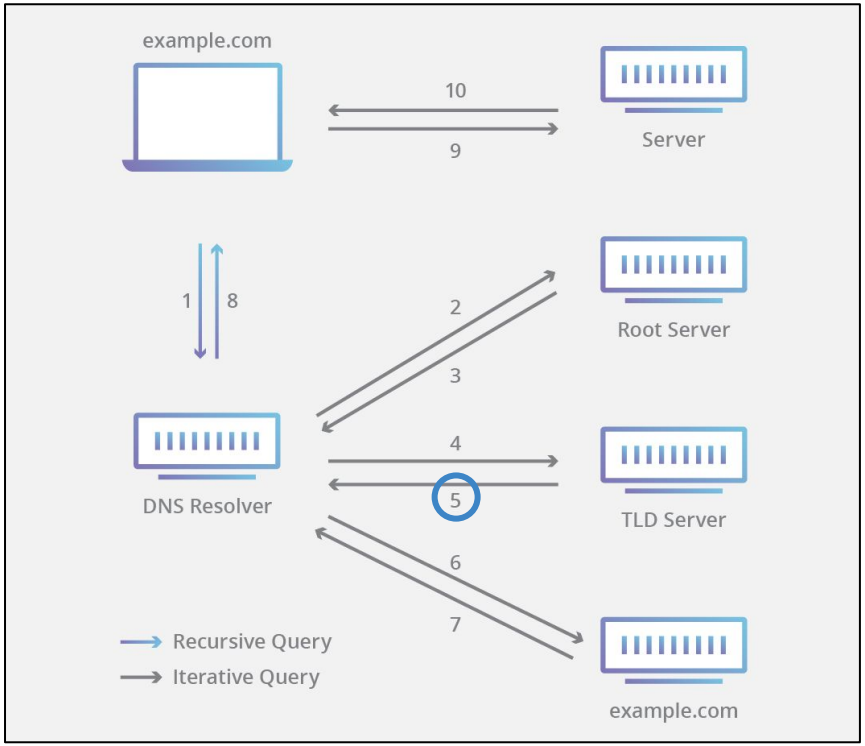
The root server then responds to the resolver with the address of a Top Level Domain (TLD) DNS server (such as .com or .net), which stores the information for its domains. When searching for example.com, our request is pointed toward the .com TLD

The resolution process



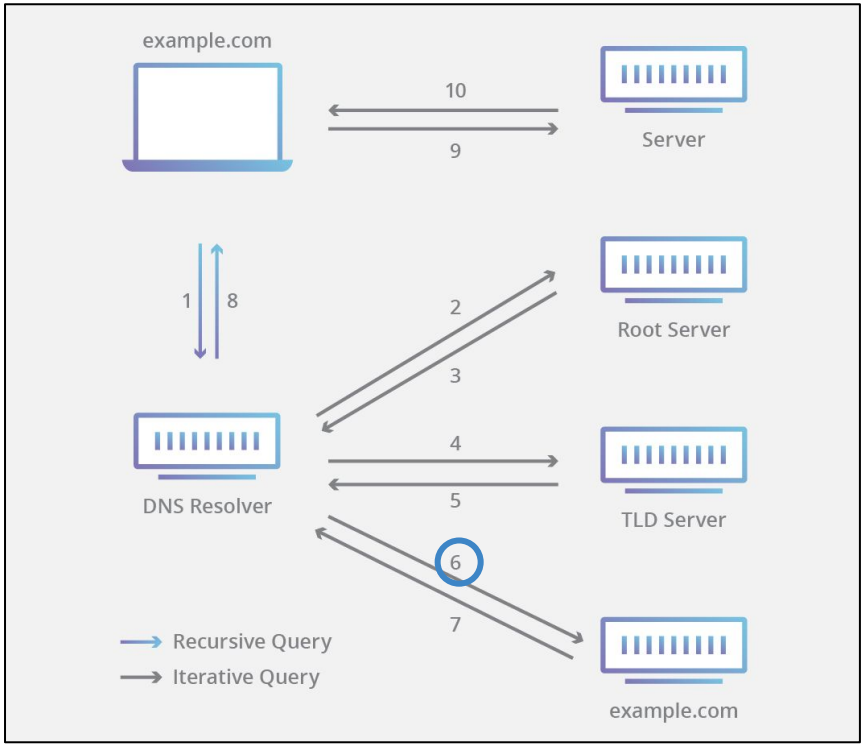
The resolver then makes a request to the .com TLD

The resolution process



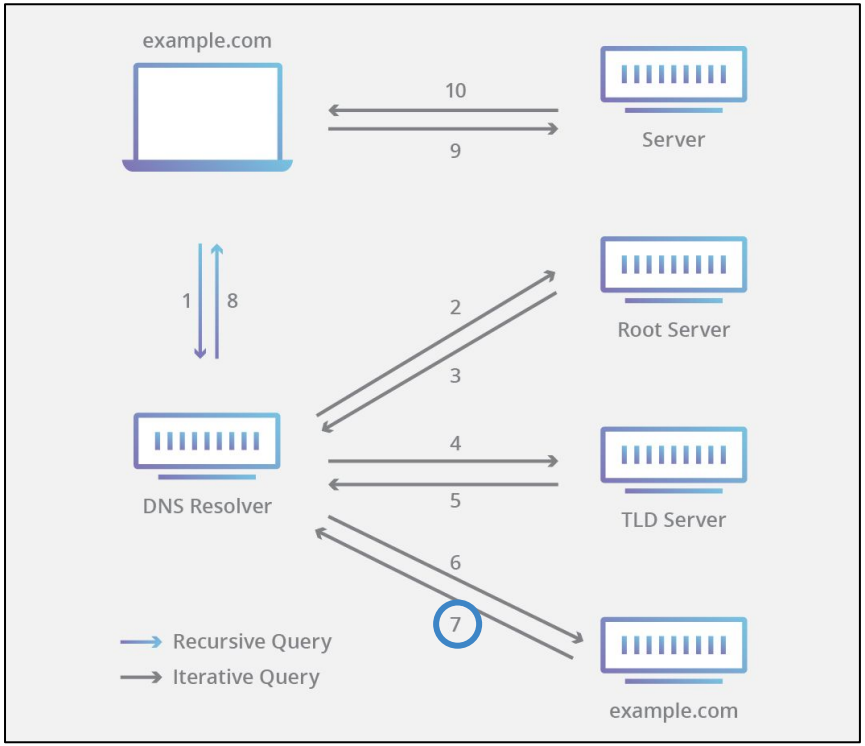
The TLD server then responds with the IP address of the domain's nameserver, example.com

The resolution process



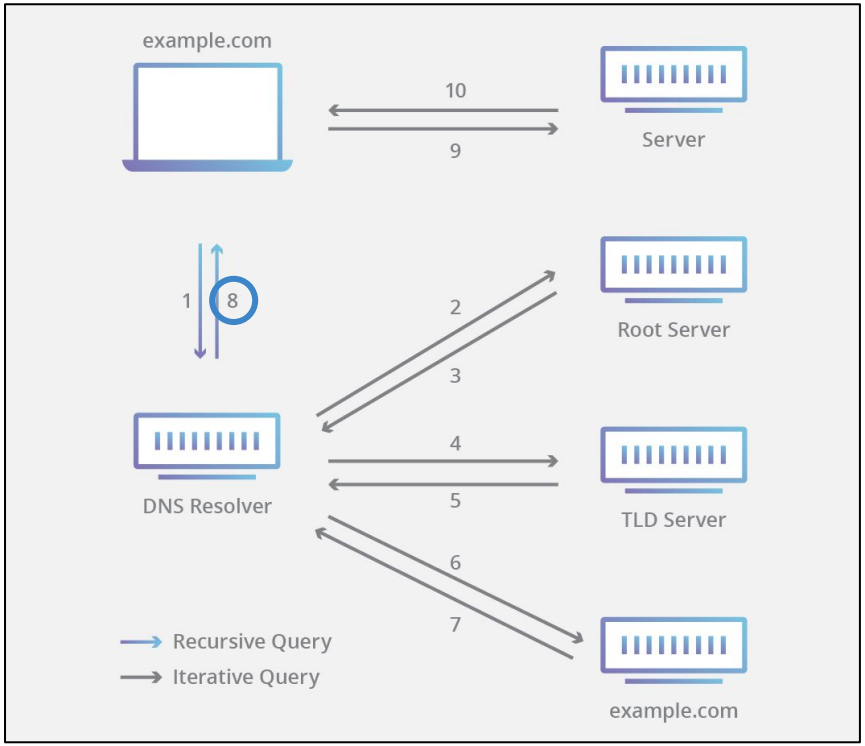
Lastly, the recursive resolver sends a query to the domain's nameserver

The resolution process



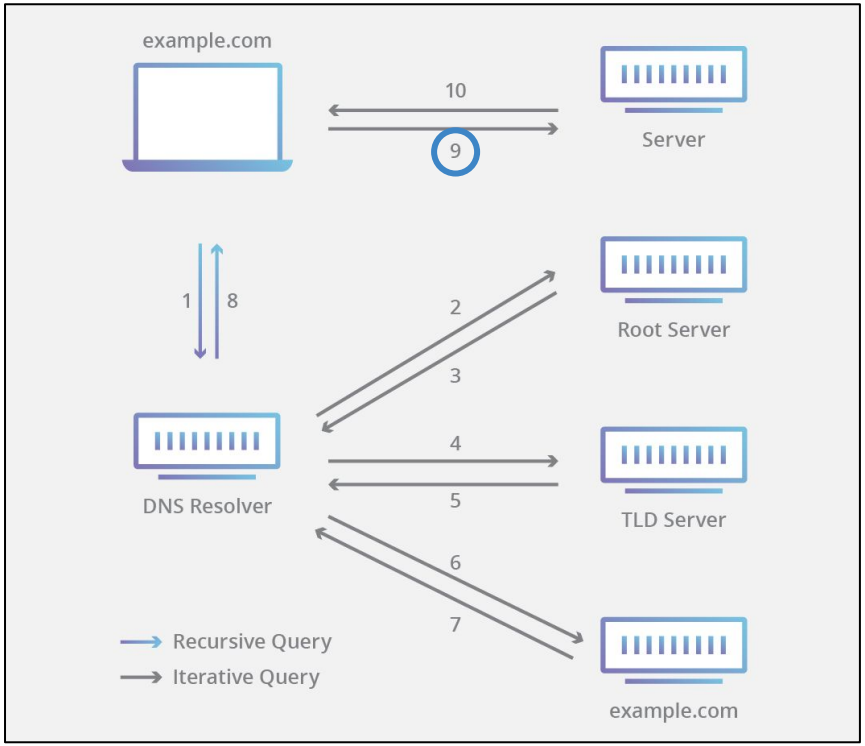
The IP address for example.com is then returned to the resolver from the nameserver

The resolution process



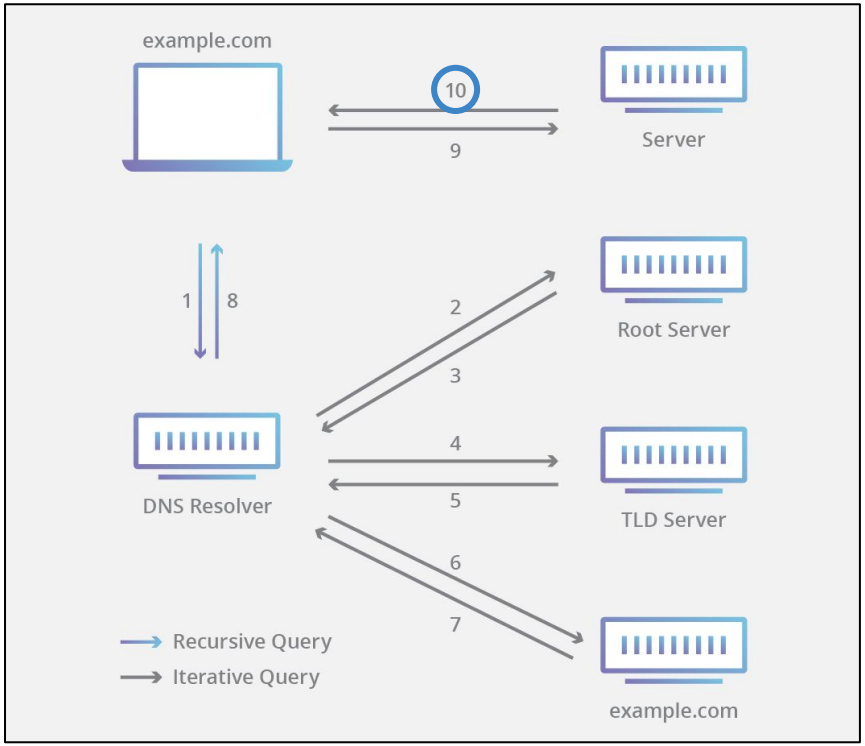
The DNS resolver then responds to the web browser with the IP address of the domain requested initially

The resolution process



Once the 8 steps of the DNS lookup have returned the IP address for example.com, the browser is able to make the request for the web page. The browser makes a HTTP request to the IP address

The resolution process



The server at that IP returns the webpage to be rendered in the browser

DNS Caching

- DNS caching involves storing data closer to the requesting client so that the DNS query can be resolved earlier and additional queries further down the DNS lookup chain can be avoided,
  - thereby improving load times and reducing bandwidth/CPU consumption.
  - DNS data can be cached in a variety of locations, each of which will store DNS records for a set amount of time determined by a time-to-live (TTL).
- ***Browser DNS caching***
  - Modern web browsers are designed by default to cache DNS records for a set amount of time.
- ***Operating system (OS) level DNS caching***
  - The operating system level DNS resolver is the second and last local stop before a DNS query leaves your machine.
  - When the recursive resolver inside the ISP receives a DNS query, like all previous steps, it will also check to see if the requested host-to-IP-address translation is already stored inside its local persistence layer
    - if the A record is not in cache the NS record might be in the cache. In this case the recursive resolver query those name servers directly,
    - If the resolver does not have the NS records, it will send a query to the TLD servers (.com in our case), skipping the root server
    - In the unlikely event that the resolver does not have records pointing to the TLD servers, it will then query the root servers.

DNS vulnerabilities and well known attacks

Vulnerability 1: DNS spoofing/cache poisoning

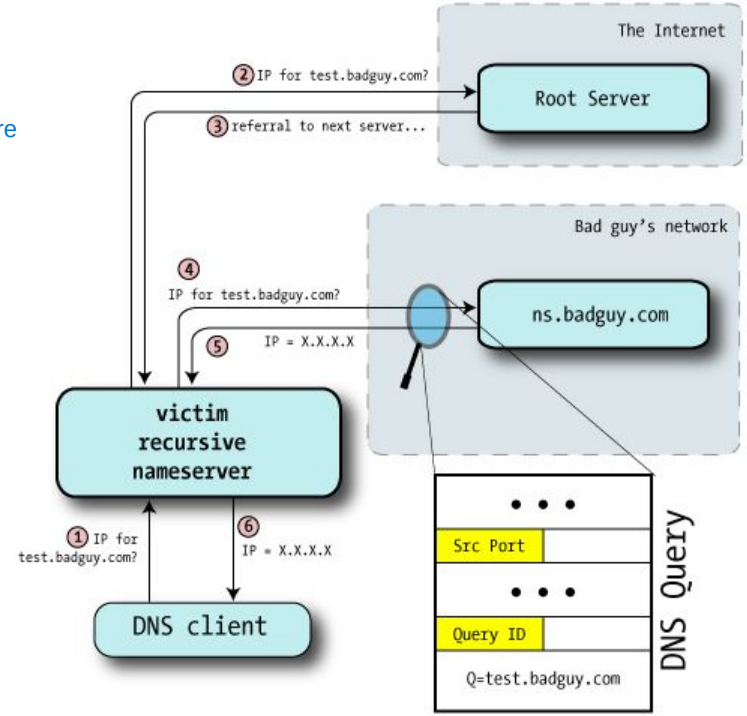
- This is an attack where forged ***DNS data is introduced into a DNS resolver's cache***, resulting in the resolver returning an incorrect IP address for a domain
  - ***NOTE***: the DNS spoofing attack shown in the first class is a special case of this attack
  - Clearly being able to connect to the same network as the victim's is unrealistic
  - Moreover, a bad guy would like to trick all ISP's customers...
  - The root of the problem is the same, but the more general attack is more complex (next slide)
- Instead of going to the correct website, traffic can be diverted to a malicious machine or anywhere else the attacker desires
- Often this will be a replica of the original site used for malicious purposes such as distributing malware or collecting login information
- Extra material: http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

Poisoning the cache

- It's not so simple as just sending random DNS packets to a nameserver, as ***DNS only accepts responses to pending queries***
  - unexpected responses are simply ignored
- How does a nameserver know that any response packet is "expected"?
  - The response arrives on the ***same UDP port*** we sent it from
  - The ***Question section*** (duplicated in the reply) matches the Question in the pending query.
  - The ***Query ID*** matches the pending query
  - The ***Authority*** and ***Additional sections*** represent names that are within the same domain as the question
- ***If all of these conditions are satisfied, a nameserver will accept a packet as a genuine response to a query, and use the results found inside***

Guessing the Query ID and source port

- In old nameservers the Query ID simply increments by one on each outgoing request nelle vecchie implementazioni non era protetto era solo un contatore
- We can't directly ask the nameserver for its query ID, but we can provoke it into telling us: per ottenere il query ID
  - Bad guy asks the victim nameserver to look up a name in a zone for a nameserver he controls
  - Victim nameserver receives the request and makes the usual rounds to resolve the name
  - Eventually, the victim nameserver will be directed to the bad guy's nameserver
  - Bad guy monitors this lookup of test.badguy.com and discovers the source port and Query ID used
- At this point he knows the last query ID and source port used by the victim nameserver.



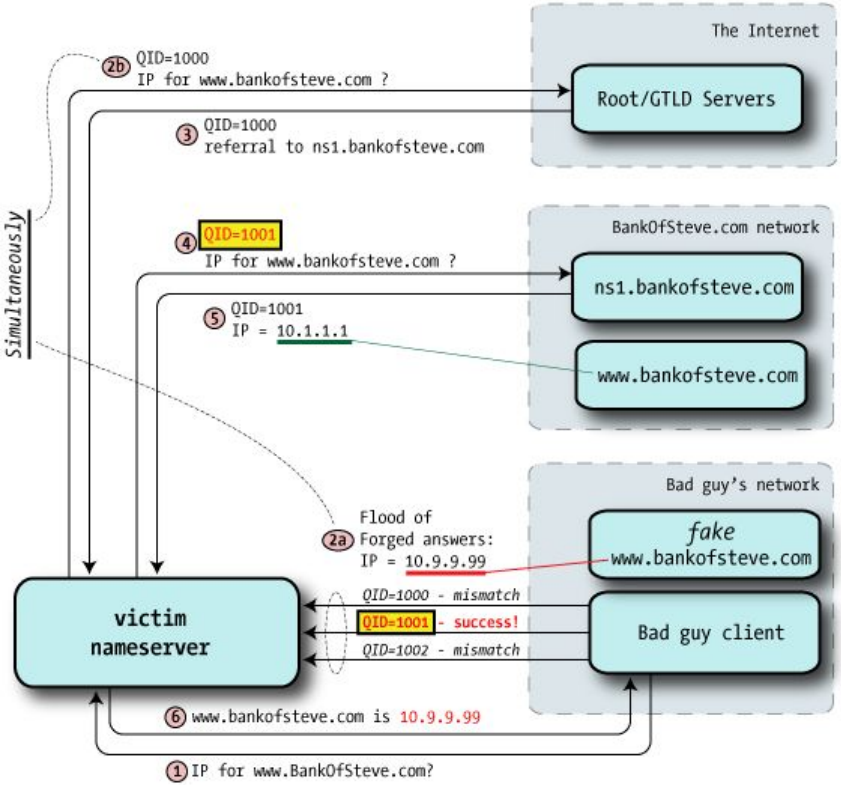
Cache poisoning attack: single domain

- With the ability to easily predict a query ID, and since our victim nameserver always sends queries from the same UDP port, it should be easy enough to cause some trouble
- We'll attempt to poison a particular nameserver with a fraudulent IP for a legitimate banking website, ***www.BankOfSteve.com***
- The bad guy's intention is to get all of the ISP's customers to visit his own malicious site instead of the real one operated by the Bank.

questo attacco non è nella stessa local area network ma è in internet.

Cache poisoning attack: single domain

- **Step 1** — Bad guy sends a DNS query to the victim nameserver for the hostname it wishes to hijack. This example assumes that the victim nameserver allows recursive queries from the outside world.
- **Step 2a** — Knowing that the victim will shortly be asking **ns1.bankofsteve.com** (as directed from the root/GTLD servers) for an IP address, the bad guy starts flooding the victim with forged DNS reply packets. All purport to be from ns1.bankofsteve.com, but include the answer with the IP of badguy's fraudulent webserver.
- **Steps 2b & 3** — Root/GTLD servers provide referral to **ns1.bankofsteve.com**. This may be multiple queries, but we're showing just one for simplicity.
- **Step 4** — victim nameserver asks **ns1.bankofsteve.com** for the IP address of **www.bankofsteve.com**, and it uses query ID 1001 (one higher than the previous query).
- **Step 5** — the real nameserver provides a legitimate response to this query, with QID=1001. But if the bad guy has successfully matched the query ID in the step **2a** flood, this legal reply arrives too late and is ignored. Oops.
- **Step 6** — With the bogus IP address (of the bad guy's webserver) in cache it provides this poisoned answer to the requesting DNS client.
- **Step 7** (not illustrated) — future DNS clients asking for **www.bankofsteve.com** will receive the same fraudulent answer.



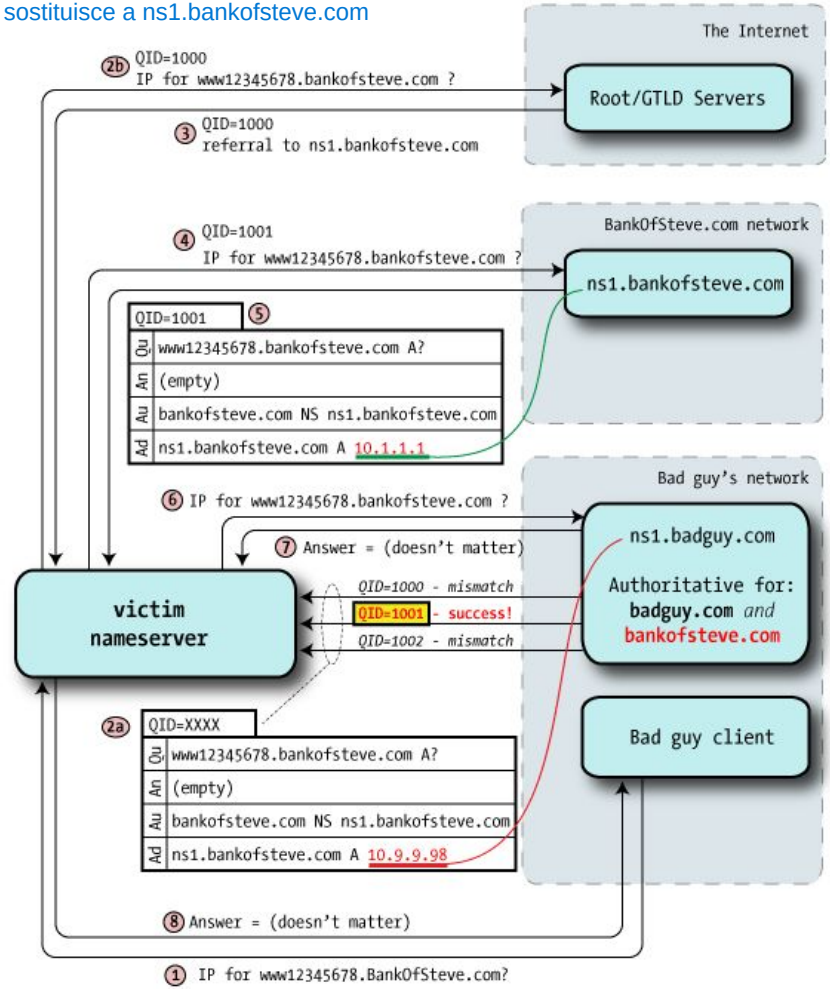
Does it work?

- The rule is: ***first good answer wins***. Most of the forged answers are dropped but if just one gets it right, the nameserver will accept the answer as genuine
  - The real answer that arrives later is dropped, because the query is no longer pending
- Requirements to succeed with this attack
  - ***The name can't already be in the cache***
    - If www.bankofsteve.com is already in the victim nameserver's cache, all of the external queries are avoided, and there's simply no way to poison it in this manner
    - If the bad guy still wants to poison that particular hostname, he has to wait for it to expire from cache (as determined by the TTL)
  - ***The bad guy has to guess the query ID***
    - This is made easy with (now-obsolete) nameservers that increment the Query ID by one each time — even a busy nameserver has a fairly small range to guess from
  - ***The bad guy has to be faster than the real nameserver***
    - If the victim nameserver and the real nameserver are topologically close (network wise), then the steps 2/3 and 4/5 may complete so quickly that the bad guy has a too-small window to exploit
- Obvious mitigation: ***randomization of Query IDs***

GOAL: spoof l'intera zone di bankofsteve, invece di ritornare l'IP di www.bankofsteve.com si sostituisce a ns1.bankofsteve.com

Kaminsky's attack

- ***Black Hat USA 2008***
- ***Step 1*** — bad guy client requests a random name within the target domain (www12345678.bankofsteve.com), something unlikely to be in cache even if other lookups for this domain have been done recently.
- ***Step 2a*** — As before, the bad guy sends a stream of forged packets to the victim, but instead of A records as part of an Answer, it instead delegates to another nameserver via Authority records. "I don't know the answer, but you can ask over there".
  - The authority data may well contain the "real" bankofsteve.com nameserver hostnames, but the glue points those nameservers at badguy IPs.
  - This is the crucial poisoning, because a Query ID match means that the victim believes that badguy's nameservers are authoritative for bankofsteve.com.
- ***The bad guy now owns the entire zone***



But does it work with Query ID randomization?

- As shown in the previous slide ***this is unlikely to be successful***: because of Query ID randomization, it's not likely that that the bad guy will manage to get a hit in the short time required to match on 64k IDs
- Instead, the bad guy can issue a flurry of queries, each for a different random name under the main domain
- The first request caused the nameserver to perform the usual root-first resolution, but it eventually caches the valid ns1.bankofsteve.com values
- Subsequent queries within this domain go directly to that nameserver, skipping the root steps
- The bad guy then throws a flood of forged data at the victim about that second random name, though the odds here are still pretty long
  - but when repeated over and over — and run from automated tools — success by the bad guy is likely
  - ***It's been reported that success can commonly be achieved in 10 seconds.***

What's the fix?

- An additional source of randomness is required nevertheless, and that's been done by randomizing the source port
- Rather than use just a single UDP port, which is trivial to discover, a much larger range of ports is allocated by the nameserver and then used randomly when making outbound queries
- Say we can preallocate 2,500 UDP ports to use for these random queries, and for discussion we'll round this down to an even power of two: $2^{11} = 2{,}048$.
- This yields this much larger transaction space: $\boldsymbol{2^{16} \times 2^{11} = 2^{27} = 134\ M}$

Vulnerability 2: DNS tunneling

- This attack uses DNS to tunnel other protocols through DNS queries and responses
- The DNS Tunneling client encodes the payload data within DNS Query packet by using base64 encoding scheme then transmits the payload data as DNS Query to the server.
- Payload data is prepended as the hostname of a DNS Query.
- The server responds the query with its base64 encoded payload data in DNS Response packet by using RDATA field of various DNS Resource Record (RR) types.
  - TXT, NULL and CNAME records are the most commonly used in DNS tunneling.

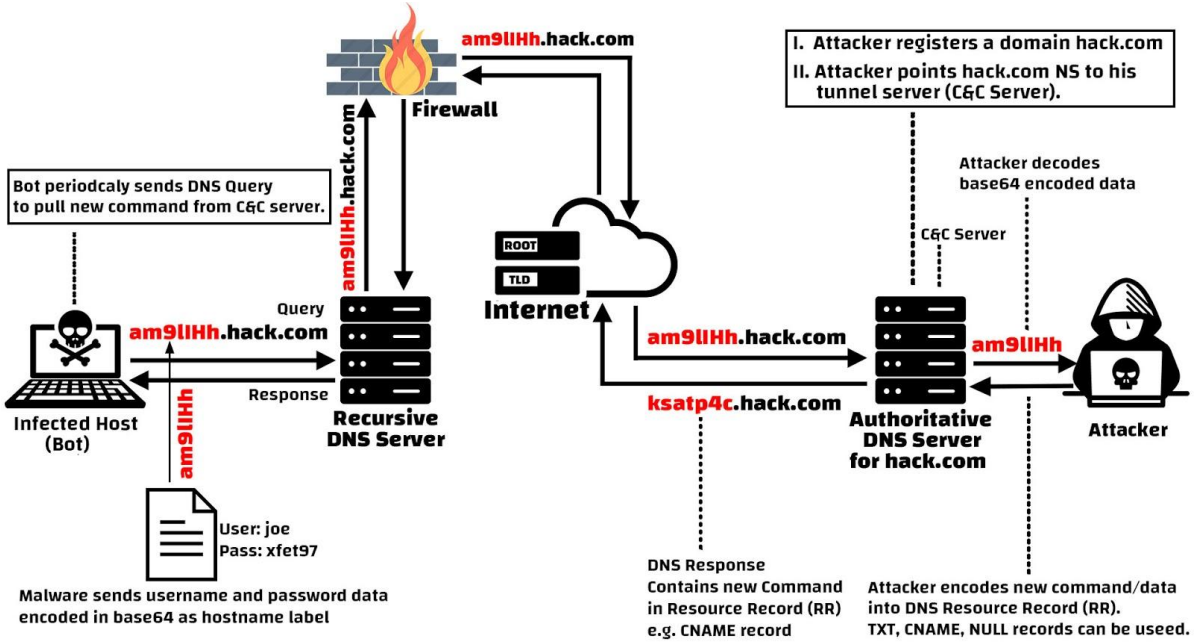
Vulnerability 2: DNS tunneling

EXAMPLE : TUNNELING BOTNET C&C DATA AND COMMANDS OVER DNS

- The attacker infects a user computer of an organization with a malicious malware.
- The organization has a Firewall to monitor and block malicious traffic.
- Web browsing and most other communication from local computers to the Internet relies on the DNS service.
- For that reason restricting the DNS communication can cause disruption of legitimate connectivity.
- Therefore DNS protocol is always allowed to outbound/inbound in Firewall.
- And attacker takes this advantage to employ DNS tunneling as covert communication channel for C&C server.
- It’s very hard to differentiate the benign and malicious usage of DNS protocol for the traditional Firewall or IDS.

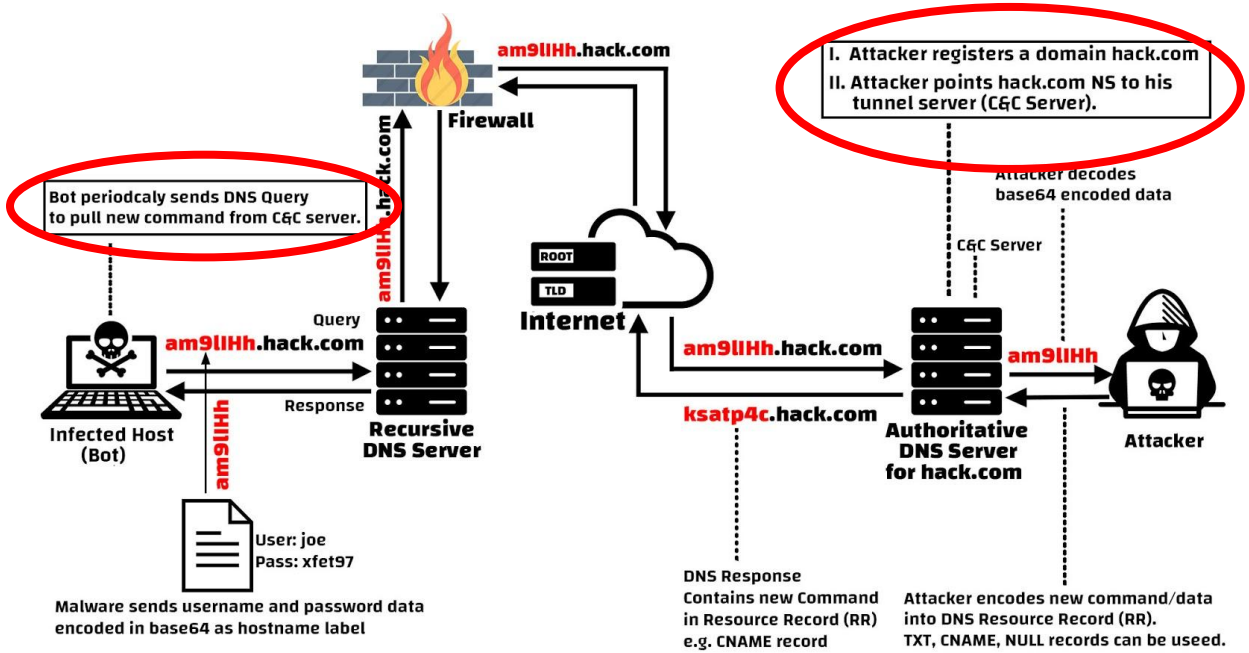
Vulnerability 2: DNS tunneling

EXAMPLE : TUNNELING BOTNET C&C DATA AND COMMANDS OVER DNS



Vulnerability 2: DNS tunneling

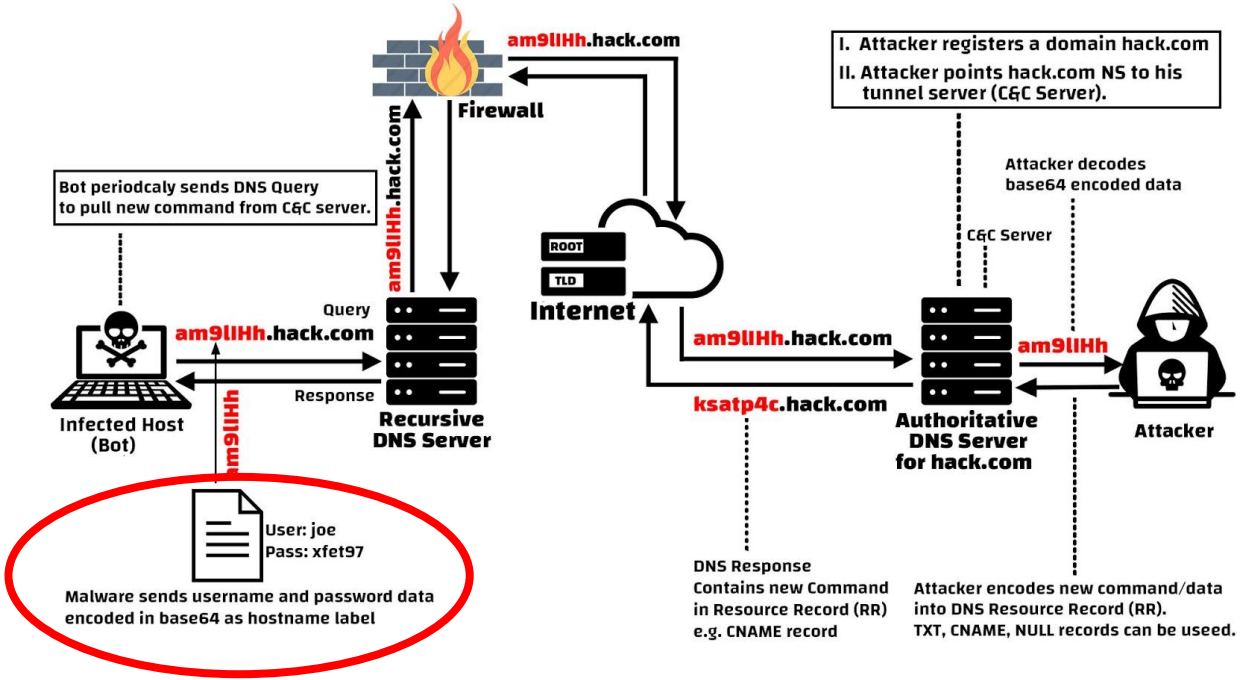
EXAMPLE : TUNNELING BOTNET C&C DATA AND COMMANDS OVER DNS



Attacker registered a domain name hack.com and its Name Server has pointed to the attacker's C&C server where DNS Tunneling server program is installed. Since the Malware binary is hard coded with the domain name hack.com so all the DNS Query made by the malware will be forwarded to the attacker's C&C server.

Vulnerability 2: DNS tunneling

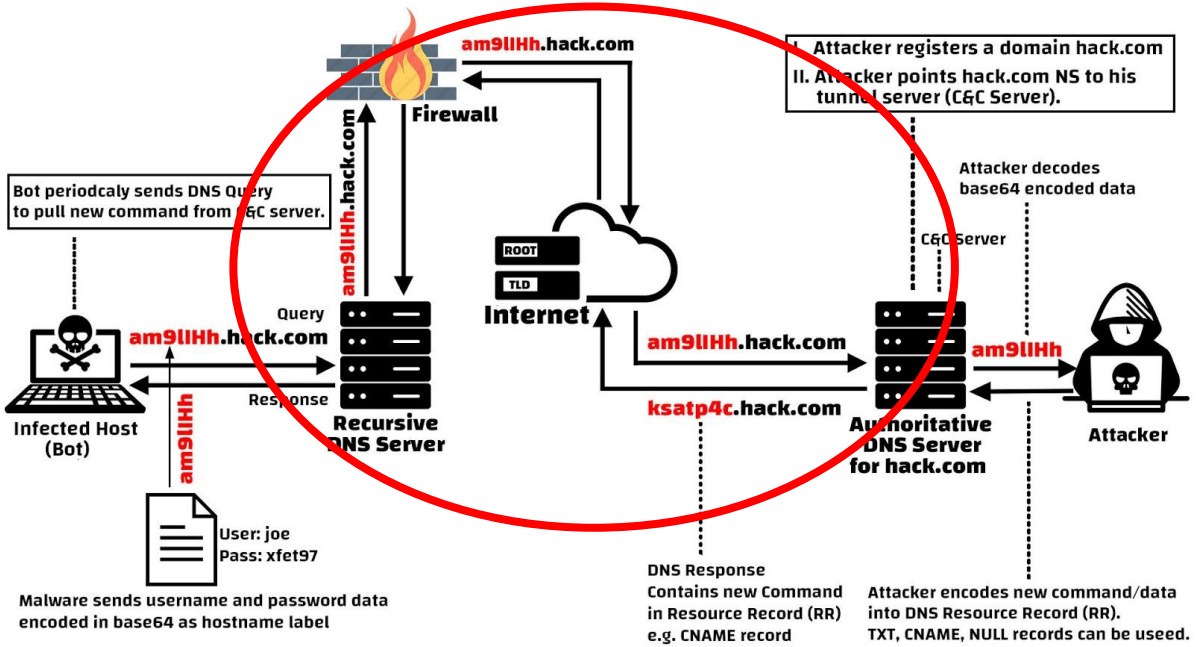
EXAMPLE : TUNNELING BOTNET C&C DATA AND COMMANDS OVER DNS



The malware stole sensitive username and password data then sends those data as a DNS query to the Recursive DNS Server

Vulnerability 2: DNS tunneling

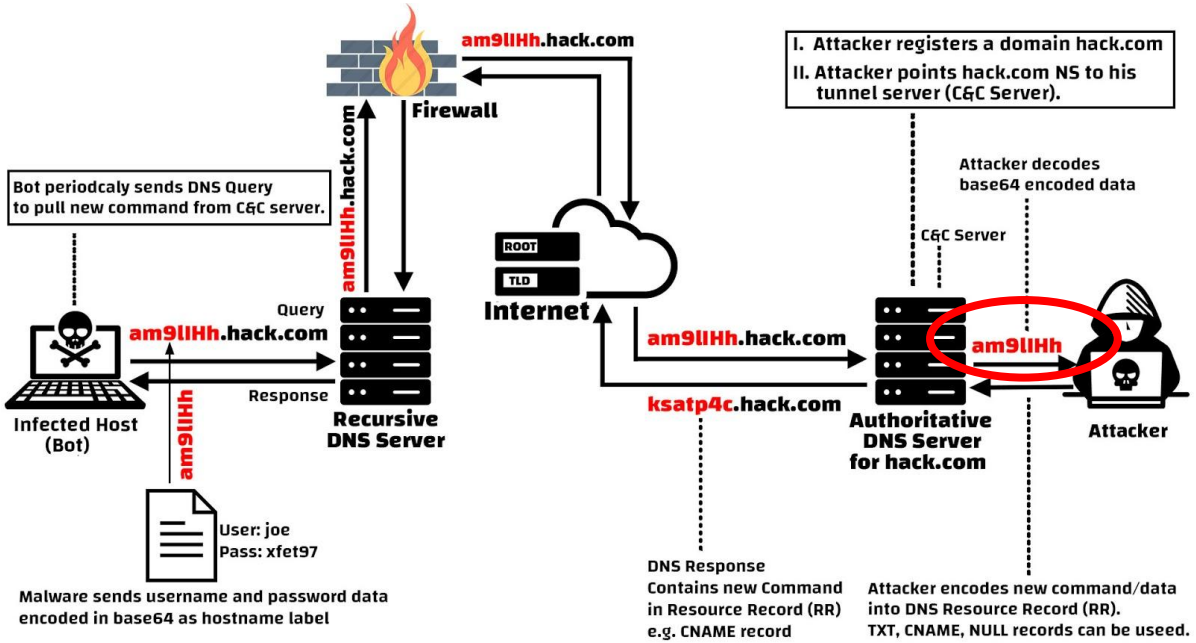
EXAMPLE : TUNNELING BOTNET C&C DATA AND COMMANDS OVER DNS



The Recursive DNS Server cannot resolve the domain from its cache so it recursively forward the DNS query packet through the Firewall to the Root Server, TLD Server, finally the DNS query is routed to the attacker's C&C Server (Name Server) where the DNS tunneling server program is running.

Vulnerability 2: DNS tunneling

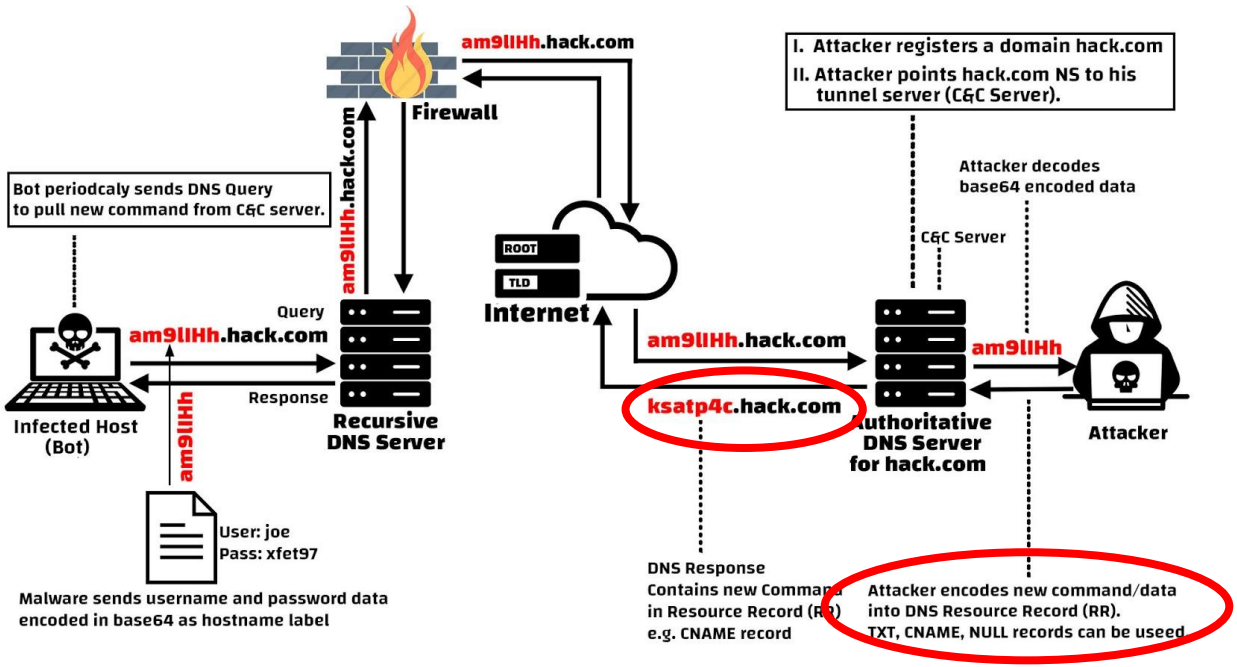
EXAMPLE : TUNNELING BOTNET C&C DATA AND COMMANDS OVER DNS



Attacker track down DNS Queries with stolen data from server query log then decrypts the DNS Query and gets the username and password.

Vulnerability 2: DNS tunneling

EXAMPLE : TUNNELING BOTNET C&C DATA AND COMMANDS OVER DNS



Attacker's Nameserver (C&C Server) sends back DNS Response with new command encoded into Resource Record (RR) back to the infected host. That's how botmaster can establish a C&C bidirectional channel

Vulnerability 3: DNS hijacking

- DNS queries are incorrectly resolved in order to unexpectedly redirect users to malicious sites.
- To perform the attack, perpetrators either install malware on user computers, take over routers, or intercept or hack DNS communication.
- For example:
  - The attacker creates a dummy site that looks and feels just like the site they are targeting.
  - The attacker uses a targeted attack (such as spear phishing) to obtain login credentials to the Admin panel of the DNS provider for the target site.
  - The attacker then goes into the DNS admin panel and changes the DNS records for the site they are targeting (this is known as DNS Hijacking), so that users trying to access the site will instead be sent to the dummy site.
  - The attacker forges a TLS encryption certificate that will convince a user's browser that the dummy site is legitimate.
  - Unsuspecting users go to the URL of the compromised site and get redirected to the dummy site.
  - The users then attempt to log in on the dummy site, and their login credentials are harvested by the attacker.

Vulnerability 3: DNS hijacking

- There are four basic types of DNS redirection:
  - ***Local DNS hijack*** — attackers install Trojan malware on a user's computer, and change the local DNS settings to redirect the user to malicious sites.
  - ***Router DNS hijack*** — many routers have default passwords or firmware vulnerabilities. Attackers can take over a router and overwrite DNS settings, affecting all users connected to that router.
  - ***Man in the middle DNS attacks*** — attackers intercept communication between a user and a DNS server, and provide different destination IP addresses pointing to malicious sites.
  - ***Rogue DNS Server*** — attackers can hack a DNS server, and change DNS records to redirect DNS requests to malicious sites.

Vulnerability 4: NXDOMAIN attack

- This is a type of DNS flood attack where an attacker floods a DNS server with requests, asking for records that do not exist, in an attempt to cause a denial-of-service for legitimate traffic.
- This can be accomplished using sophisticated attack tools that can auto-generate unique subdomains for each request.
- NXDOMAIN attacks can also target a recursive resolver with the goal of filling the resolver's cache with junk requests.

Vulnerability 5: Phantom domain attack

- A phantom domain attack has a similar result to an NXDOMAIN attack on a DNS resolver.
- The attacker sets up a bunch of ***'phantom'*** domain servers that either respond to requests very slowly or not at all.
- The resolver is then hit with a flood of requests to these domains and the resolver gets tied up waiting for responses, leading to slow performance and denial-of-service.

Vulnerability 6: Random subdomain attack

- In this case, the attacker sends DNS queries for several random, nonexistent subdomains of one legitimate site.
- The goal is to create a denial-of-service for the domain’s authoritative nameserver, making it impossible to lookup the website from the nameserver.
- As a side effect, the ISP serving the attacker may also be impacted, as their recursive resolver's cache will be loaded with bad requests.

Vulnerability 7: Domain lock-up attack

- ❏ Attackers orchestrate this form of attack by setting up special domains and resolvers to create TCP connections with other legitimate resolvers.
- ❏ When the targeted resolvers send requests, these domains send back slow streams of random packets, tying up the resolver’s resources.

Vulnerability 8: Botnet-based CPE attack

- These attacks are carried out using CPE devices (Customer Premise Equipment; this is hardware given out by service providers for use by their customers, such as modems, routers, cable boxes, etc.).
- The attackers compromise the CPEs and the devices become part of a botnet, used to perform random subdomain attacks against one site or domain.

Vulnerability 9: DNS query confidentiality

- Another important DNS security issue is user privacy. DNS queries are not encrypted.
- Even if users use a DNS resolver like 1.1.1.1 that does not track their activities, DNS queries travel over the Internet in plaintext.
- This means anyone who intercepts the query can see which websites the user is visiting
- This lack of privacy has an impact on security and, in some cases, human rights;
- if DNS queries are not private, then it becomes easier for governments to censor the Internet and for attackers to stalk users' online behavior.

Main measures to secure DNS

fare il bind tra record e proprietario di quel server. Non risolve tutte le vulnerabilità, ad esempio non garantisce confidentiality

- ***DNSSEC***: The security extensions to DNS add protection for DNS records, and allow the resolvers and applications to authenticate the data received
- ***DNS over TLS and DNS over HTTPS*** are two standards for encrypting DNS queries in order to prevent external parties from being able to read them
- ***DNS firewall***, a tool that can provide a number of security and performance services for DNS servers

DNSSEC main features

Intro

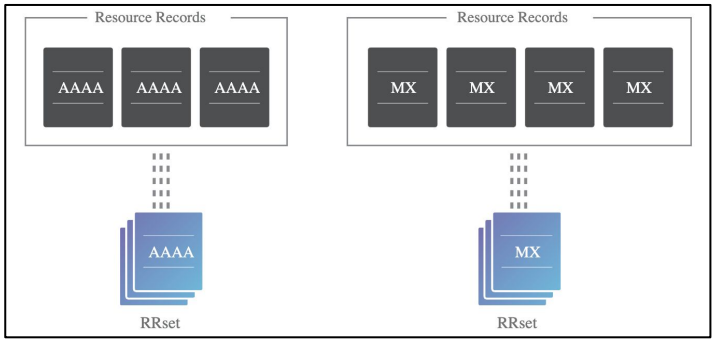
- DNSSEC creates a secure domain name system by ***adding cryptographic signatures to existing DNS records***. servono ad autenticare la risposta ricevuta a seguito di una query
- These digital ***signatures are stored in DNS name servers alongside common record types*** like A, AAAA, MX, CNAME, etc.
- By checking its associated signature, you can verify that a requested DNS record comes from its authoritative name server and wasn't altered en-route, opposed to a fake record injected in a man-in-the-middle attack.
- DNSSEC was originally specified in RFCs 4033, 4034, 4035
- Subsequently, additional RFCs have been issued: RFCs 4470, 4641, 5155, 6014

DNSSEC record types

- To facilitate signature validation, DNSSEC adds a few new DNS record types:
  - ***RRSIG*** - Contains a cryptographic signature
  - ***DNSKEY*** - Contains a public signing key
  - ***DS*** - Contains the hash of a DNSKEY record
  - ***NSEC*** and ***NSEC3*** - For explicit denial-of-existence of a DNS record
  - ***CDNSKEY*** and ***CDS*** - For a child zone requesting updates to DS record(s) in the parent zone

Resource Record sets

- The first step towards securing a zone with DNSSEC is to group all the records with the same type into a resource record set (RRset)
- It's actually ***this full RRset that gets digitally signed***, opposed to individual DNS records.
  - Of course, this also means that you must request and validate all of the A and AAAA records from a zone with the same label instead of validating only one of them.



INTERNET DRAFT DNS Resource Record Sets 2015-05-22

1 Introduction

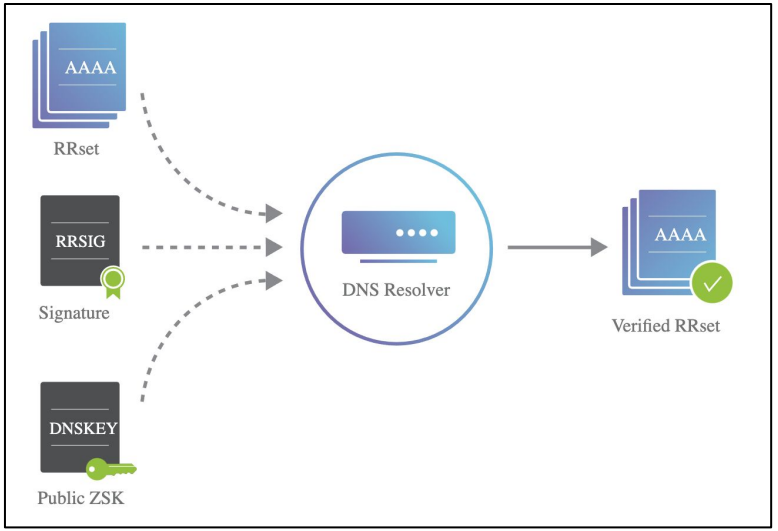
Each DNS Resource Record (RR) has a label, class, type, and data. It is meaningless for two records to ever have label, class, type and data all equal - servers should suppress such duplicates if encountered. It is however possible for most record types to exist with the same label, class and type, but with different data. Such a group of records is hereby defined to be a Resource Record Set (RRSet).

Zone-Signing Keys

- Each zone in DNSSEC has a ***zone-signing key pair (ZSK)***:
  - the private key digitally signs each RRset in the zone
  - the public key verifies the signature
- To enable DNSSEC, a zone operator creates digital signatures for each RRset using the private ZSK and stores them in their name server as RRSIG records
- ***The zone operator also needs to make their public ZSK available*** by adding it to their name server in a DNSKEY record

Zone-Signing Keys

- When a DNSSEC resolver requests a particular record type, the name server also returns the corresponding RRSIG
- The resolver can then pull the DNSKEY record containing the public ZSK from the name server
- ***Together, the RRset, RRSIG, and public ZSK can validate the response.***
- If we trust the zone-signing key in the DNSKEY record, we can trust all the records in the zone
- ***We need a way to validate the public ZSK***

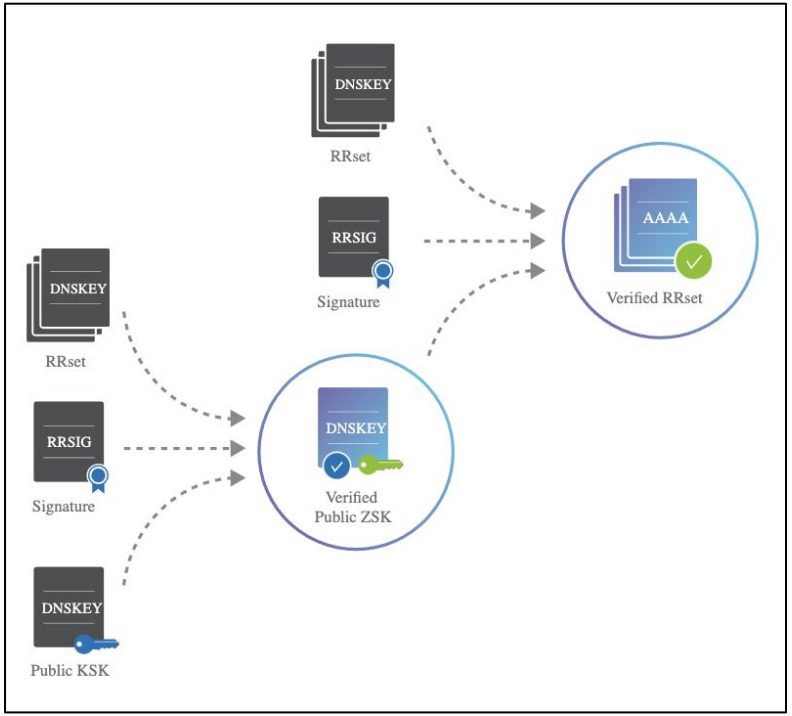


Key-Signing Keys

- In addition to a zone-signing key, DNSSEC name servers also have a ***key-signing key (KSK)***
- The ***KSK validates the DNSKEY record*** in exactly the same way as our ZSK secured the rest of our RRsets in the previous slides
- It signs the public ZSK (which is stored in a DNSKEY record), ***creating an RRSIG for the DNSKEY***
- The name server publishes the public KSK in another DNSKEY record
- ***Both the public KSK and public ZSK are signed by the private KSK***
- Resolvers can then use ***the public KSK to validate the public ZSK***

The validation process

1. Request the desired RRset, which also returns the corresponding RRSIG record.
2. Request the DNSKEY records containing the public ZSK and public KSK, which also returns the RRSIG for the DNSKEY RRset.
3. Verify the RRSIG of the requested RRset with the public ZSK.
4. Verify the RRSIG of the DNSKEY RRset with the public KSK
5. (the DNSKEY RRset and corresponding RRSIG records can be cached)



Why separate KSK and ZSK?

- it’s difficult to swap out an old or compromised KSK.
- Changing the ZSK, on the other hand, is much easier.
- This allows us to use a smaller ZSK without compromising the security of the server, minimizing the amount of data that the server has to send with each response.

But things get more complicated...

- ❏ We've now established trust ***within our zone***
- ❏ but DNS is a hierarchical system, and zones rarely operate independently
- ❏ Complicating things further, the key-signing key is signed by itself, which doesn't provide any additional trust
- ❏ ***We need a way to connect the trust in our zone with its parent zone***

Delegation Signer Records

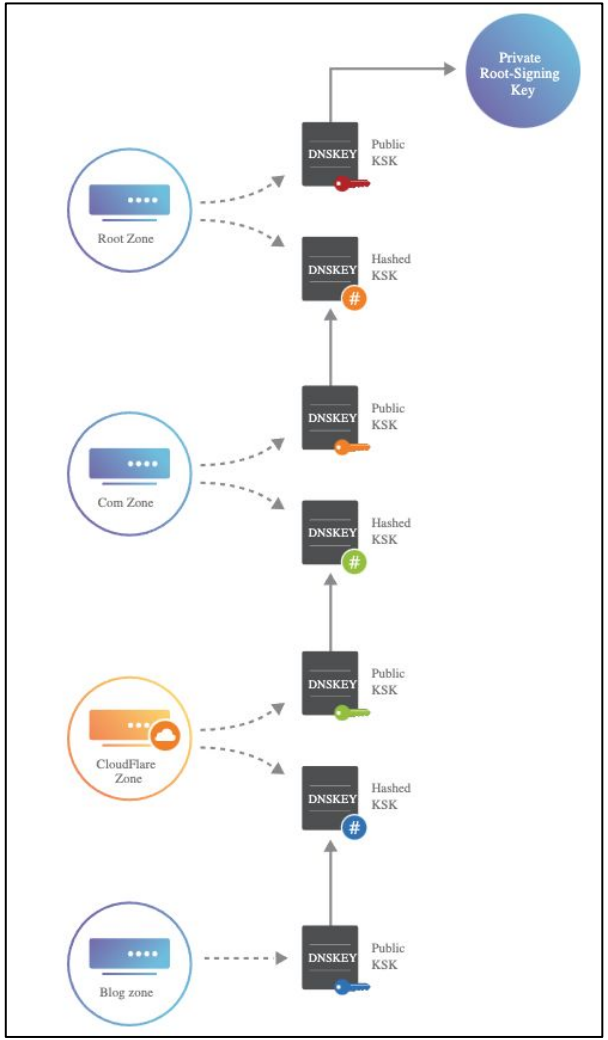
- DNSSEC introduces a ***delegation signer (DS)*** record to allow the transfer of trust from a parent zone to a child zone.
- A zone operator hashes the DNSKEY record containing the public KSK and gives it to the parent zone to publish as a DS record.
- Every time a resolver is referred to a child zone, the parent zone also provides a DS record.
- This DS record is how resolvers know that the child zone is DNSSEC-enabled.
- To check the validity of the child zone's public KSK:
  - the resolver hashes it and compares it to the DS record from the parent.
  - If they match, the resolver can assume that the public KSK hasn't been tampered with, which means it can trust all of the records in the child zone.
  - This is how a chain of trust is established in DNSSEC.

NOTE

- Any change in the KSK also requires a change in the parent zone's DS record.
- Changing the DS record is a multi-step process that can end up breaking the zone if it's performed incorrectly.
- First, the parent needs to add the new DS record, then they need to wait until the TTL for the original DS record to expire before removing it.
- ***This is why it's much easier to swap out zone-signing keys than key-signing keys.***

DNSSEC chain of trust

- the DS record is signed just like any other RRset, which means it has a corresponding RRSIG in the parent.
- The whole validation process repeats until we get to the parent's public KSK.
- To verify that, we need to go to that parent's DS record, and on and on we go up the chain of trust.
- However, when we finally get to ***the root DNS zone***, we have a problem: there's no parent DS record to validate against



The Root Signing Ceremony

- Several selected individuals from around the world come together and sign the root DNSKEY RRset in a very public and highly audited way.
- The ceremony produces an RRSIG record that can be used to verify the root name server's public KSK and ZSK.
- Instead of trusting the public KSK because of the parent's DS record, we assume that it's valid because we trust the security procedures around accessing the private KSK
- The private signing key used in this process is quite literally the key to the entire DNSSEC-protected Internet
- More details: https://www.cloudflare.com/dns/dnssec/root-signing-ceremony/

Explicit Denial of Existence

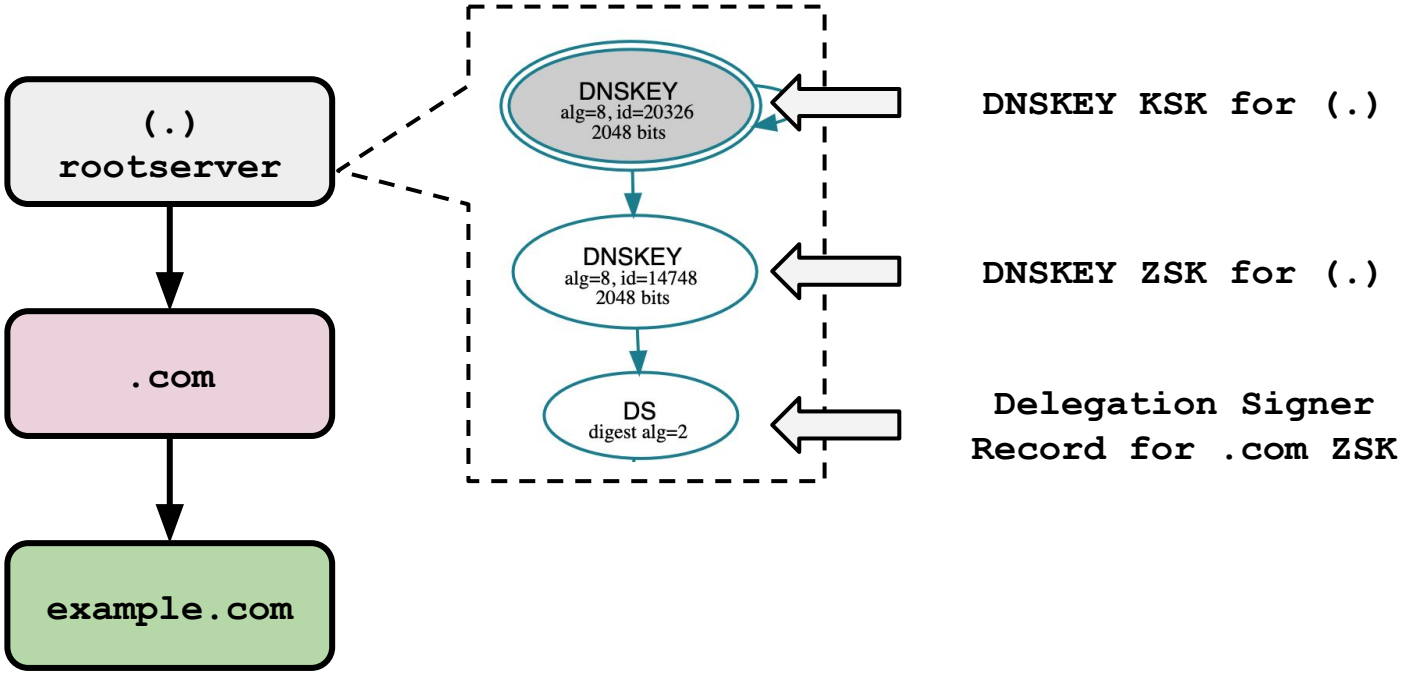
- If you ask DNS for the IP address of a domain that doesn't exist, it returns an empty answer
  - there's no way to explicitly say, "sorry, the zone you requested doesn't exist."
  - This is a problem if you want to authenticate the response, since there's no message to sign.
- DNSSEC fixes this by adding the NSEC and NSEC3 record types
  - ***They both allow for an authenticated denial of existence***
- NSEC works by returning the "next secure" record (in alphabetic order). For example, consider a name server that defines AAAA records for api, blog, and www.
- This effectively tells you that store doesn't exist.
  - ***And, since the NSEC record is signed, you can validate its corresponding RRSIG just like any RRset.***
- Unfortunately, ***this solution allows anybody to walk through the zone and gather every single record without knowing which ones they're looking for***.
  - This can be a potential security threat if the zone administrator was counting on the contents of the zone being private
  - we have the solution to this problem (in a few slides..)

A real example

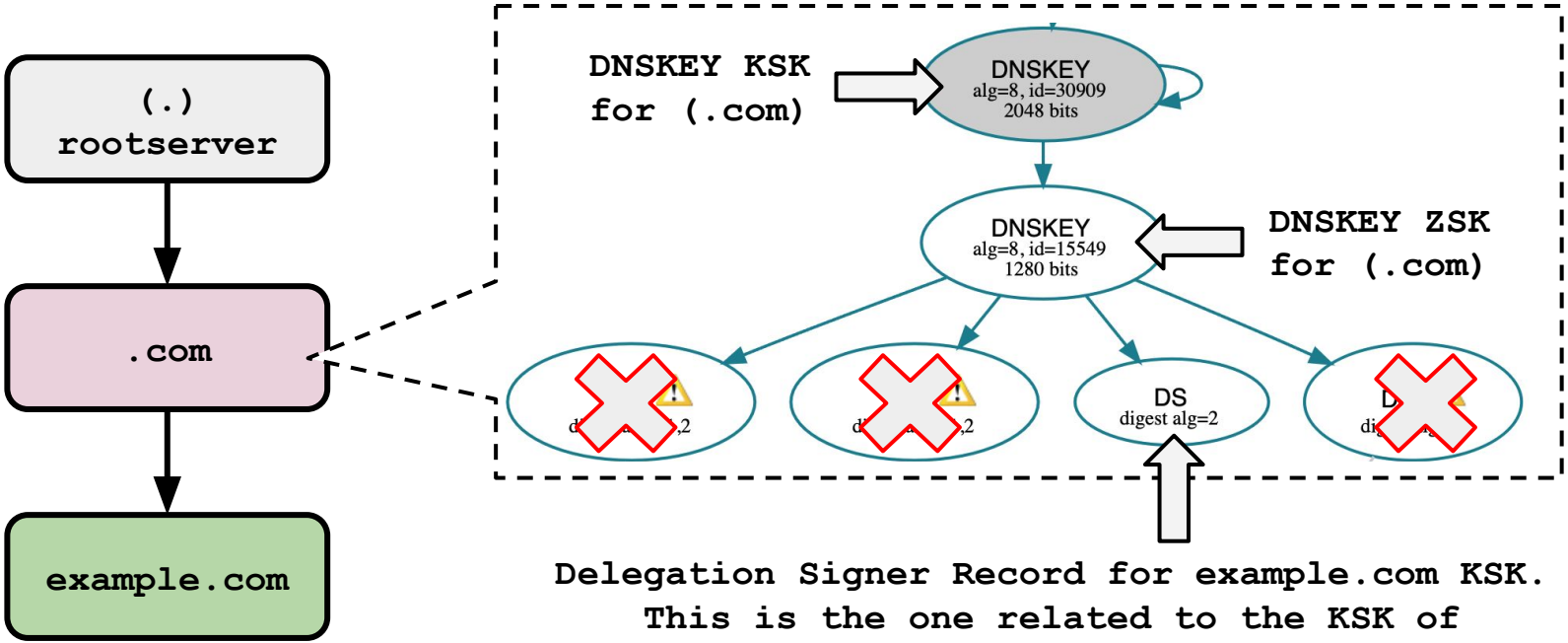
Goal of this section

- Better understanding of the DNSSEC details with a real example
- Let's try to understand the validation chain and some protocol details by trying to resolve the name **`example.com`**
- We are using 2 tools
  - ***DNSViz*** (https://dnsviz.net/) is a tool for visualizing the status of a DNS zone. It was designed as a resource for understanding and troubleshooting deployment of the DNS Security Extensions (DNSSEC). It provides a visual analysis of the DNSSEC authentication chain for a domain name and its resolution path in the DNS namespace, and it lists configuration errors detected by the tool
  - **`dig`** is a network administration command-line tool for querying the Domain Name System
- Nice reading that inspired the second part of this section: https://metebalci.com/blog/a-minimum-complete-tutorial-of-dnssec/

The high level picture

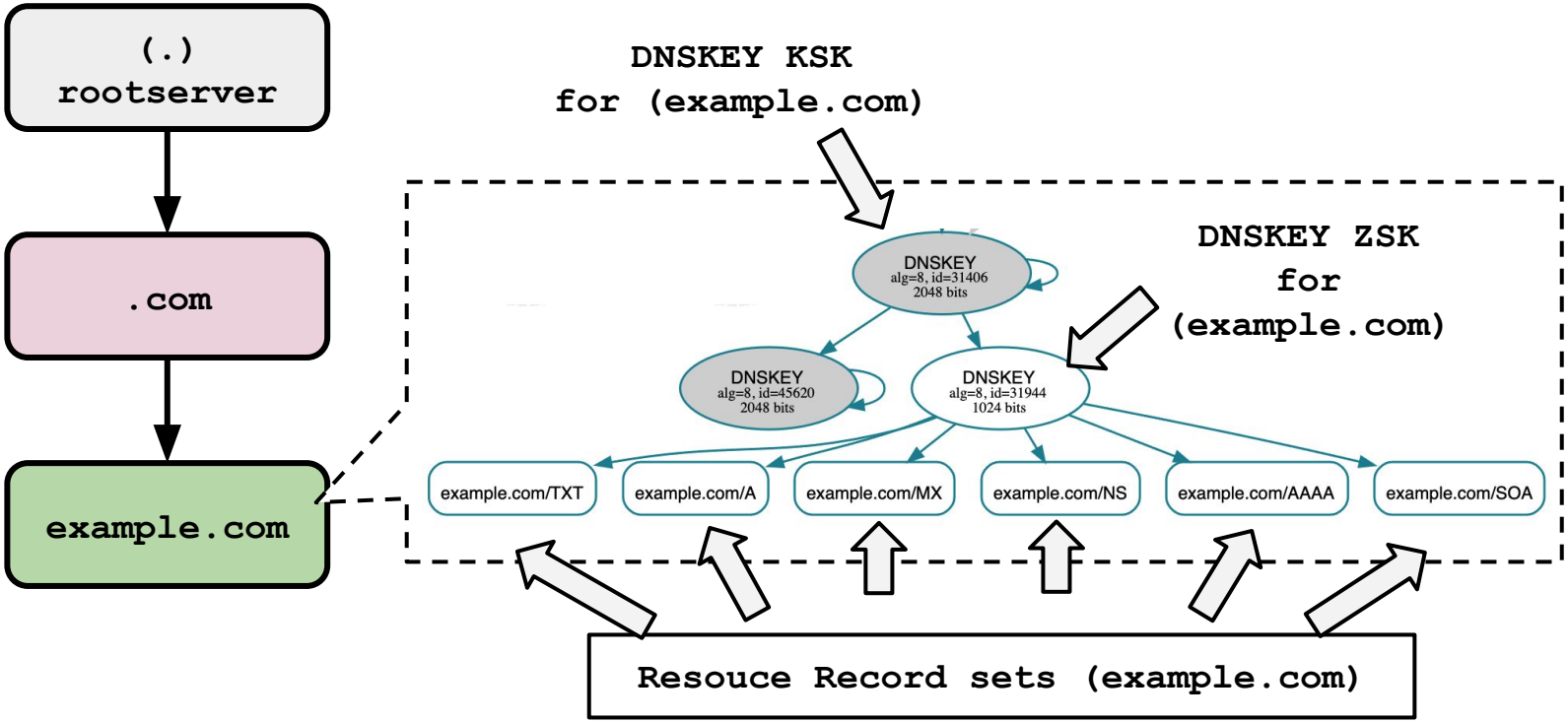


The high level picture



Delegation Signer Record for example.com KSK. This is the one related to the KSK of example.com. The other 3 DS records were actually returned by the server but they are not used to sign the delegated zones

The high level picture



A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.         86377     IN        A         93.184.216.34

example.com.         86377     IN        RRSIG     A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

"legacy" A record: example.com --> 93.184.216.34

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

RRSIG record. This record contains the signature made by the authoritative name server for example.com

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

Type Covered: record type this RRset covers, above it is A.

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

"original" time to leave

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

Algorithm: 8, RSA/SHA-256

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

Labels: number of labels in the owner name, meaning number of labels in example.com, label is each non wildcard part, so here it is 2 (example and com)

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

signature expiration and inception (i.e. the time the signature was generated) fields.

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

Key Tag value, 31944 above. Key Tag indicates which DNSKEY record is used for this signature. There is a specific algorithm to calculate Key Tag number based on DNSKEY data.

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

Signer's Name is example.com., which is the name of the zone of the covered RRset.

A query for the A record of example.com with dig

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com A

;; ANSWER SECTION:

example.com.          86377      IN         A          93.184.216.34

example.com.          86377      IN         RRSIG      A 8 2 86400
20211129175238 20211108194821 31944 example.com.

PbWyo6MlRSQq0mmbtC4aev0Cx2QVyGiBTHeGCMCelegHYxktI7eSck95
ZeRrWcAX78A/ew+qn01x490Q0pm2+gpbXTXg8unj0SBw+UCjlKGV+5C9
Bj80jXfti5SthCqpi0waf5o/608IlIft1UXhDqpQVplUljeL0mZDZC2l AbA=
```

Signature is Base64 encoded RSA/SHA-256 signature of concatenation of this RRSIG data (without signature) and the records in the RRset.

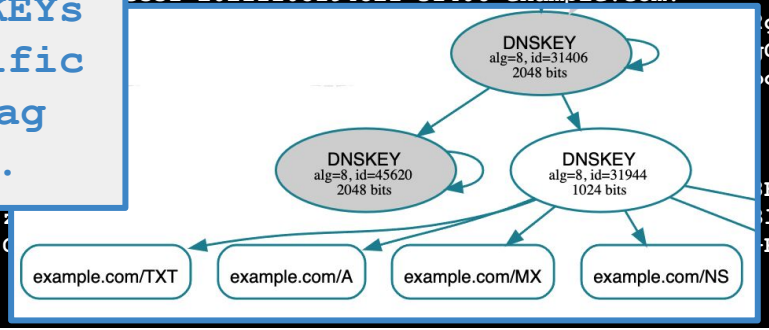
Let's resolve the DNSKEY records

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com DNSKEY
;; ANSWER SECTION:
example.com.        29     IN     DNSKEY 256 3 8 AwEAAc+fwtUCsQQx9BAoSNkZbEYB+N/Ncy6S4h8EtM9GGQFPBKlsO2YC
qO02DTVxUtP0KdyXlJKvy7JQjQqho+lCCBYxpnzg2DY76ANVpQ5mbZuN q5j0FjaCL5i/RZqqDCYxqcC5uuICoDFf/Bxf0OyTbmkTBqBiQrxi55ib
oc8woxzN

example.com.        29     IN     DNSKEY 257 3 8 AwEAAZ0aqu1rJ6orJynrRfNpPmayJZoAx9Ic2/Rl9VQWLMHyjxxem3VU
SoNUIFXERQbj0A9Ogp0zDM9YIccKLRd6LmWiDCt7UJQxVdD+heb5Ec4q lqGmyX9MDabkvX2NvMwsUecbYBq8oXeTT9LRmCUt9KUt/WOi6DKECxoG
/bWTykrXyBR8elD+SQY43OAVjlWrVltHxgp4/rhBCvRbmdflunaPIgu2 7eE2U4myDSLT8a4A0rB5uHG4PkOa9dIRs9y00M2mWf4lyPee7vi5few2
dbayHXmieGcaAHrx76NGAABeY393xjlmDNcUkF1gpNWUla4fWZbbaYQz A93mLdrng+M=

example.com.        29     IN     DNSKEY 257 3 8 AwEAAbOFAxl+Lkt0UMglZizKEC1AxUu8zlj65KYatR5wBWMrh18TYzK/
ig6Y1t5YTWCO68bynorpNu9fqNFALX7bVl9/gybA0v0EhF+dgXmoUfRX 7ksMGgBvtfa2/Y9a3klXNLqkTszIQ4PEMVCjtryl19Be9/PkFeC9ITjg
MRQsQhmB39eyMYnal+f3bUxKk4fq7cuEU0dbRpue4H/N6jPucXWOwiMA kTJhghqgy+o9FfIp+tR/emKao94/wpVXDcPf5B18j7xz2SvTTxiuqCzC
MtsxnikZHcoh1j4g+Y1B8zIMIvrEM+pZGhh/Yuf4RwCBgaYCi9hpiMWV vS4WBzx0/lU=

example.com.        29     IN     RRSIG  DNSKEY 8 2 3600 20211129123551 20211108194821 31406 example.com.
hX6gfy126r3qliEIn38FY2FFrZ8b+fOc5+BtDYAnMffleNb0W83FqPgg IFBByR5+iFP6qpaxTCBc8WG3Inn0jTkMn85Y58rx4sui5niDNCl2gThd
nngAqF1l7MGKK0E6rol2KPZI0XlsZhblsHEHQ5A87P9Gvvq8cURwN+g7 CxepLWA3vQI4g54KZtQHQcXuRdQ/xbTz6jpYHCjw8r5vAc5bwaagGltH
m8cvr6zUDMumYSUjMsPM+3VGQ5dczzO3l5Os3B1EUDX5chttnBI+8uF+ K9ak06+ZP1Uil2Q+iMGc8ObFiAAiPx9BUvYpBmWp5DJQm2xKU5Hpqzpu
H0Lsvw==

example.com.        29     IN     RRSIG  DNSKEY 8 2 3600 20211129123551 20211108194821 45620 example.com.
bnDf+A8JNzDU3MCmPv0m0qY8gTgYn9EerWDz03bf+mmKr64rMVQXVKh6 TJeD8o3OPngpH81kUTUI0+fJpBa5cfWEfD8C4v2Vd9sXr11oglh8E43Q
usraRpiTSar1eZv5VxMLTpKEN1toK9fsqGuT9oMXYynmukoGjcx+sVNN z19QsYddXGkxh1dXKD/83jxK79hodOWgMAgowU8gztBJSOkuYB43l9aK
L3NC+GPkyN01+tjjYO9dmOFGbCLnqWM51quX+PkDFGaYagK4KPGZWiIW Gli++cU3z7Ou7gypw7I04PXNsOlwsBG3IANo2O/niWwSlBB6IzP+R2MH
DhZmcA==
```

Let's resolve the DNSKEY records

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com DNSKEY
;; ANSWER SECTION:
example.com.        29     IN     DNSKEY 256 3 8 AwEAAc+fwtUCsQQx9BAoSNkZbEYB+N/Ncy6S4h8EtM9GGQFPBKlsO2YC
qO02DTVxUtP0KdyXlJKvy7JQjQqho+lCCBYxpnzg2DY76ANVpQ5mbZuN q5j0FjaCL5i/RZqqDCYxqcC5uuICoDFf/Bxf0OyTbmkTBqBiQrxi55ib
oc8woxzN

example.com.        29     IN     DNSKEY 257 3 8 AwEAAZ0aqu1rJ6orJynrRfNpPmayJZoAx9Ic2/Rl9VQWLMHyjxxem3VU
SoNUIFXERQbj0A9Ogp0zDM9YIccKLRd6LmWiDCt7UJQxVdD+heb5Ec4q lqGmyX9MDabkvX2NvMwsUecbYBq8oXeTT9LRmCUt9KUt/WOi6DKECxoG
/bWTykrXyBR8elD+SQY43OAVj1WrVltHxgp4/rhBCvRbmdflunaPIgu2 7eE2U4myDSLT8a4A0rB5uHG4PkOa9dIRs9y00M2mWf4lyPee7vi5few2
dbayHXmieGcaAHrx76NGAABeY393xjlmDNcUkF1gpNWUla4fWZbbaYQz A93mLdrng+M=

example.com.        29     IN     DNSKEY 257 3 8 AwEAAbOFAxl+Lkt0UMglZizKEC1AxUu8zlj65KYatR5wBWMrh18TYzK/
ig6Y1t5YTWCO68bynorpNu9fqNFALX7bVl9/gybA0v0EhF+dgXmoUfRX 7ksMGgBvtfa2/Y9a3klXNLqkTszIQ4PEMVCjtryl19Be9/PkFeC9ITjg
MRQsQhmB39eyMYnal+f3bUxKk4fq7cuEU0dbRpue4H/N6jPucXWOwiMA kTJhghqgy+o9FfIp+tR/emKao94/wpVXDcPf5B18j7xz2SvTTxiuqCzC
MtsxnikZHcoh1j4g+Y1B8zIMIvrEM+pZGhh/Yuf4RwCBgaYCi9hpiMWV vS4WBzx0/lU=

ex                                                    3551 20211108194821 31406 example.com.
hX                                                                                          gThd
nn                                                                                          GltH
m8                                                                                          qzpu
H0

ex
bn                                                                                          E43Q
usraRp1TSar1eZv5VxMLTpKEN1toK9fsqGuT9oMXYynmukoGjcx+sVNN                                    l9aK
L3NC+GPkyN01+tjjYO9dmOFGbCLnqWM51quX+PkDFGaYagK4KPGZWiIW                                    R2MH
DhZmcA==
```

records 1, 2, 3 are the 3 DNSKEYs in the graph. There is a specific algorithm to calculate Key Tag number based on DNSKEY data.



Let's resolve the DNSKEY records

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com DNSKEY
;; ANSWER SECTION:
example.com.        29     IN     DNSKEY 256 3 8 AwEAAc+fwtUCsQQx9BAoSNkZbEYB+N/Ncy6S4h8EtM9GGQFPBKlsO2YC
qO02DTVxUtP0KdyXlJKvy7JQjQqho+lCCBYxpnzg2DY76ANVpQ5mbZuN q5j0FjaCL5i/RZqqDCYxqcC5uuICoDFf/Bxf0OyTbmkTBqBiQrxi55ib
oc8woxzN  ###KEYID 31944

example.com.        29     IN     DNSKEY 257 3 8 AwEAAZ0aqu1rJ6orJynrRfNpPmayJZoAx9Ic2/Rl9VQWLMHyjxxem3VU
SoNUIFXERQbj0A9Ogp0zDM9YIccKLRd6LmWiDCt7UJQxVdD+heb5Ec4q lqGmyX9MDabkvX2NvMwsUecbYBq8oXeTT9LRmCUt9KUt/WOi6DKECxoG
/bWTykrXyBR8elD+SQY43OAVjlWrVltHxgp4/rhBCvRbmdflunaPIgu2 7eE2U4myDSLT8a4A0rB5uHG4PkOa9dIRs9y00M2mWf4lyPee7vi5few2
dbayHXmieGcaAHrx76NGAABeY393xjlmDNcUkF1gpNWUla4fWZbbaYQz A93mLdrng+M=  ###KEYID 45620

example.com.        29     IN     DNSKEY 257 3 8 AwEAAbOFAxl+Lkt0UMglZizKEC1AxUu8zlj65KYatR5wBWMrh18TYzK/
ig6Y1t5YTWCO68bynorpNu9fqNFALX7bVl9/gybA0v0EhF+dgXmoUfRX 7ksMGgBvtfa2/Y9a3klXNLqkTszIQ4PEMVCjtryl19Be9/PkFeC9ITjg
MRQsQhmB39eyMYnal+f3bUxKk4fq7cuEU0dbRpue4H/N6jPucXWOwiMA kTJhghqgy+o9FfIp+tR/emKao94/wpVXDcPf5B18j7xz2SvTTxiuqCzC
MtsxnikZHcoh1j4g+Y1B8zIMIvrEM+pZGhh/Yuf4RwCBgaYCi9hpiMWV vS4WBzx0/lU=  ###KEYID 31406

example.com.        29     IN     RRSIG  DNSKEY 8 2 3600 20211129123551 20211108194821 31406 example.com.
```

example.com	3600	DNSKEY	256 3 8 AwEAAc+fwtUCsQQx9BAoSNkZbEYB+N/N cy6S4h8EtM9GGQFPBKlsO2YCqO02DTVx UtP0KdyXlJKvy7JQjQqho+lCCBYxpnzg 2DY76ANVpQ5mbZuNq5j0FjaCL5i/RZqq DCYxqcC5uuICoDFf/Bxf0OyTbmkTBqBi Qrxi55iboc8woxzN ; **key tag = 31944** 257 3 8 AwEAAZ0aqu1rJ6orJynrRfNpPmayJZoA x9Ic2/Rl9VQWLMHyjxxem3VUSoNUIFXE RQbj0A9Ogp0zDM9YIccKLRd6LmWiDCt7 UJQxVdD+heb5Ec4qlqGmyX9MDabkvX2N vMwsUecbYBq8oXeTT9LRmCUt9KUt/WOi 6DKECxoG/bWTykrXyBR8elD+SQY43OAV jlWrVltHxgp4/rhBCvRbmdflunaPIgu2 7eE2U4myDSLT8a4A0rB5uHG4PkOa9dIR s9y00M2mWf4lyPee7vi5few2dbayHXmi eGcaAHrx76NGAABeY393xjlmDNcUkF1g pNWUla4fWZbbaYQzA93mLdrng+M= ; **key tag = 45620** 257 3 8 AwEAAbOFAxl+Lkt0UMglZizKEC1AxUu8 zlj65KYatR5wBWMrh18TYzK/ig6Y1t5Y TWCO68bynorpNu9fqNFALX7bVl9/gybA 0v0EhF+dgXmoUfRX7ksMGgBvtfa2/Y9a 3klXNLqkTszIQ4PEMVCjtryl19Be9/Pk FeC9ITjgMRQsQhmB39eyMYnal+f3bUxK k4fq7cuEU0dbRpue4H/N6jPucXWOwiMA kTJhghqgy+o9FfIp+tR/emKao94/wpVX DcPf5B18j7xz2SvTTxiuqCzCMtsxnikZ Hcoh1j4g+Y1B8zIMIvrEM+pZGhh/Yuf4 RwCBgaYCi9hpiMWVvS4WBzx0/lU= ; **key tag = 31406**

```
DhZmcA==
```

Let's resolve the DNSKEY records

records 4 and 5 are the 2 signatures for the DNSKEY record sets.

1 Record 4 is computed with the private key in 31406. Since this key is the KSK of example.com (see the diagram), this RRSIG is the signature made by the zone authoritative server for the zone the DNSKEY records in example.com.

2 Record 5 is the signature made with key 45620, which is not used to authenticate the RRs (see the diagram from dnsviz.net). According to the graph this key is a ZSK signed by ZSK 31406 but which is not used to sign the RRs (to be investigated... nice homework :))

3

4
```
example.com.        29     IN     RRSIG  DNSKEY 8 2 3600 20211129123551 20211108194821 31406 example.com.
hX6gfy126r3qliEIn38FY2FFrZ8b+fOc5+BtDYAnMffleNb0W83FqPgg IFBByR5+iFP6qpaxTCBc8WG3Inn0jTkMn85Y58rx4sui5niDNCl2gThd
nngAqF1l7MGKK0E6rol2KPZI0XlsZhblsHEHQ5A87P9Gvvq8cURwN+g7 CxepLWA3vQI4g54KZtQHQcXuRdQ/xbTz6jpYHCjw8r5vAc5bwaagGltH
m8cvr6zUDMumYSUjMsPM+3VGQ5dczzO3l5Os3B1EUDX5chttnBI+8uF+ K9ak06+ZP1Uil2Q+iMGc8ObFiAAiPx9BUvYpBmWp5DJQm2xKU5Hpqzpu
H0Lsvw==
```

5
```
example.com.        29     IN     RRSIG  DNSKEY 8 2 3600 20211129123551 20211108194821 45620 example.com.
bnDf+A8JNzDU3MCmPv0m0qY8gTgYn9EerWDz03bf+mmKr64rMVQXVKh6 TJeD8o3OPngpH81kUTUI0+fJpBa5cfWEfD8C4v2Vd9sXr11oglh8E43Q
usraRpiTSar1eZv5VxMLTpKEN1toK9fsqGuT9oMXYynmukoGjcx+sVNN z19QsYddXGkxh1dXKD/83jxK79hodOWgMAgowU8gztBJSOkuYB43l9aK
L3NC+GPkyN01+tjjYO9dmOFGbCLnqWM51quX+PkDFGaYagK4KPGZWiIW Gli++cU3z7Ou7gypw7I04PXNsOlwsBG3IANo2O/niWwSlBB6IzP+R2MH
DhZmcA==
```

DNSKEY record details

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com DNSKEY
;; ANSWER SECTION:
example.com.        29     IN     DNSKEY 256 3 8 AwEAAc+fwtUCsQQx9BAoSNkZbEYB+N/Ncy6S4h8EtM9GGQFPBKlsO2YC
qO02DTVxUtP0KdyXlJKvy7JQjQqho+lCCBYxpnzg2DY76ANVpQ5mbZuN q5j0FjaCL5i/RZqqDCYxqcC5uuICoDFf/Bxf0OyTbmkTBqBiQrxi55ib
oc8woxzN

example.com.        29     IN     DNSKEY 257 3 8 AwEAAZ0aqu1rJ6orJynrRfNpPmayJZoAx9Ic2/Rl9VQWLMHyjxxem3VU
SoNUIFXERQbj0A9Ogp0zDM9YIccKLRd6LmWiDCt7UJQxVdD+heb5Ec4q lqGmyX9MDabkvX2NvMwsUecbYBq8oXeTT9LRmCUt9KUt/WOi6DKECxoG
/bWTykrXyBR8elD+SQY43OAVjlWrVltHxgp4/rhBCvRbmdflunaPIgu2 7eE2U4myDSLT8a4A0rB5uHG4PkOa9dIRs9y00M2mWf4lyPee7vi5few2
dbayHXmieGcaAHrx76NGAABeY393xjlmDNcUkF1gpNWUla4fWZbbaYQz A93mLdrng+M=

example.com.        29     IN     DNSKEY 257 3 8 AwEAAbOFAxl+Lkt0UMglZizKEC1AxUu8zlj65KYatR5wBWMrh18TYzK/
ig6Y1t5YTWCO68bynorpNu9fqNFALX7bVl9/gybA0v0EhF+dgXmoUfRX 7ksMGgBvtfa2/Y9a3klXNLqkTszIQ4PEMVCjtryl19Be9/PkFeC9ITjg
MRQsQhmB39eyMYnal+f3bUxKk4fq7cuEU0dbRpue4H/N6jPucXWOwiMA kTJhghqgy+o9FfIp+tR/emKao94/wpVXDcPf5B18j7xz2SvTTxiuqCzC
MtsxnikZHcoh1j4g+Y1B8zIMIvrEM+pZGhh/Yuf4RwCBgaYCi9hpiMWV vS4WBzx0/lU=
```

Flags, 16-bits, and its values are 256 and 257 above. Bits 0–6 and 8–14 are reserved and should be zero. So only bit 7 and 15 can be one, and possible values are only 0, 256 (bit-7 is set) and 257 (both bit-7 and bit-15 is set). If the bit 7 is 1 (value 256 or 257), DNSKEY holds a DNS zone key; if it is 0 (value 0), it holds another type of key. So both of these hold a DNS zone key. Bit 15 (value 257) indicates a Secure Entry Point. Only DNSKEYs marked as zone keys can sign the DNS records.

DNSKEY record details

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com DNSKEY
;; ANSWER SECTION:
example.com.        29     IN     DNSKEY 256 3 8 AwEAAc+fwtUCsQQx9BAoSNkZbEYB+N/Ncy6S4h8EtM9GGQFPBKlsO2YC
qO02DTVxUtP0KdyXlJKvy7JQjQqho+lCCBYxpnzg2DY76ANVpQ5mbZuN q5j0FjaCL5i/RZqqDCYxqcC5uuICoDFf/Bxf0OyTbmkTBqBiQrxi55ib
oc8woxzN

example.com.        29     IN     DNSKEY 257 3 8 AwEAAZ0aqu1rJ6orJynrRfNpPmayJZoAx9Ic2/Rl9VQWLMHyjxxem3VU
SoNUIFXERQbj0A9Ogp0zDM9YIccKLRd6LmWiDCt7UJQxVdD+heb5Ec4q lqGmyX9MDabkvX2NvMwsUecbYBq8oXeTT9LRmCUt9KUt/WOi6DKECxoG
/bWTykrXyBR8elD+SQY43OAVjlWrVltHxgp4/rhBCvRbmdflunaPIgu2 7eE2U4myDSLT8a4A0rB5uHG4PkOa9dIRs9y00M2mWf4lyPee7vi5few2
dbayHXmieGcaAHrx76NGAABeY393xjlmDNcUkF1gpNWUla4fWZbbaYQz A93mLdrng+M=

example.com.        29     IN     DNSKEY 257 3 8 AwEAAbOFAxl+Lkt0UMglZizKEC1AxUu8zlj65KYatR5wBWMrh18TYzK/
ig6Y1t5YTWCO68bynorpNu9fqNFALX7bVl9/gybA0v0EhF+dgXmoUfRX 7ksMGgBvtfa2/Y9a3klXNLqkTszIQ4PEMVCjtryl19Be9/PkFeC9ITjg
MRQsQhmB39eyMYnal+f3bUxKk4fq7cuEU0dbRpue4H/N6jPucXWOwiMA kTJhghqgy+o9FfIp+tR/emKao94/wpVXDcPf5B18j7xz2SvTTxiuqCzC
MtsxnikZHcoh1j4g+Y1B8zIMIvrEM+pZGhh/Yuf4RwCBgaYCi9hpiMWV vS4WBzx0/lU=
```

Algorithm, 8-bits, the public key's cryptographic algorithm, it is 8 above. 8 means RSA/SHA-256. For this algorithm, the key size must be between 512-bits and 4096-bits (according to RFC 5702).

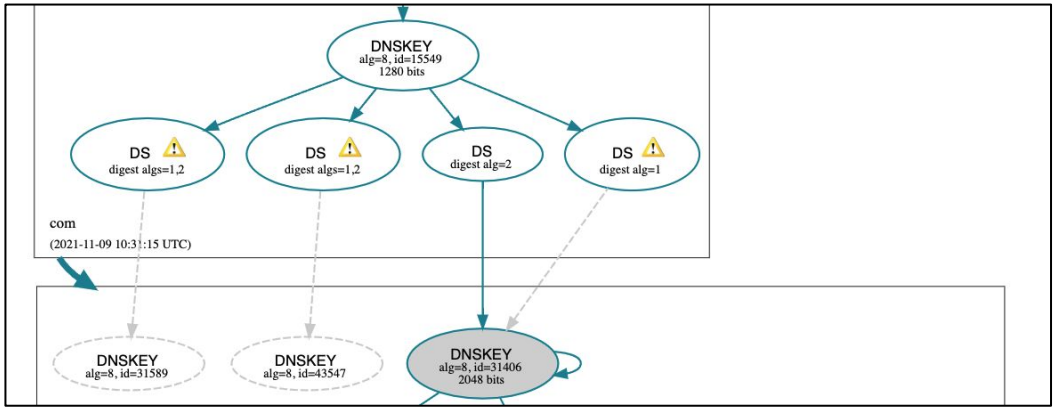
DNSKEY record details

```
marlon@marlonsMBP ~ % dig +qr +dnssec @1.1.1.1 example.com DNSKEY
;; ANSWER SECTION:
example.com.        29     IN     DNSKEY 256 3 8 AwEAAc+fwtUCsQQx9BAoSNkZbEYB+N/Ncy6S4h8EtM9GGQFPBKlsO2YC
qO02DTVxUtP0KdyXlJKvy7JQjQqho+lCCBYxpnzg2DY76ANVpQ5mbZuN q5j0FjaCL5i/RZqqDCYxqcC5uuICoDFf/Bxf0OyTbmkTBqBiQrxi55ib
oc8woxzN

example.com.        29     IN     DNSKEY 257 3 8 AwEAAZ0aqu1rJ6orJynrRfNpPmayJZoAx9Ic2/Rl9VQWLMHyjxxem3VU
SoNUIFXERQbj0A9Ogp0zDM9YIccKLRd6LmWiDCt7UJQxVdD+heb5Ec4q lqGmyX9MDabkvX2NvMwsUecbYBq8oXeTT9LRmCUt9KUt/WOi6DKECxoG
/bWTykrXyBR8elD+SQY43OAVjlWrVltHxgp4/rhBCvRbmdflunaPIgu2 7eE2U4myDSLT8a4A0rB5uHG4PkOa9dIRs9y00M2mWf4lyPee7vi5few2
dbayHXmieGcaAHrx76NGAABeY393xjlmDNcUkF1gpNWUla4fWZbbaYQz A93mLdrng+M=

example.com.        29     IN     DNSKEY 257 3 8 AwEAAbOFAxl+Lkt0UMglZizKEC1AxUu8zlj65KYatR5wBWMrh18TYzK/
ig6Y1t5YTWCO68bynorpNu9fqNFALX7bVl9/gybA0v0EhF+dgXmoUfRX 7ksMGgBvtfa2/Y9a3klXNLqkTszIQ4PEMVCjtryl19Be9/PkFeC9ITjg
MRQsQhmB39eyMYnal+f3bUxKk4fq7cuEU0dbRpue4H/N6jPucXWOwiMA kTJhghqgy+o9FfIp+tR/emKao94/wpVXDcPf5B18j7xz2SvTTxiuqCzC
MtsxnikZHcoh1j4g+Y1B8zIMIvrEM+pZGhh/Yuf4RwCBgaYCi9hpiMWV vS4WBzx0/lU=
```

Last field is DNSKEY RDATA in Base64 encoding, the format of public key in the record depends on the algorithm, in this case it contains the size of the exponent, the public exponent and the modulus (according to RFC 5702 and RFC 3110).

Let's see the DS record (remember this is in the .com zone)

```
marlon@marlonsMBP nsd % dig +qr +dnssec @1.1.1.1 example.com DS
;; ANSWER SECTION:
example.com.        86400  IN     DS     31406 8 1 189968811E6EBA862DD6C209F75623D8D9ED9142
example.com.        86400  IN     DS     31406 8 2 F78CF3344F72137235098ECBBD08947C2C9001C7F6A085A17F518B5D 8F6B916D
example.com.        86400  IN     DS     31589 8 1 3490A6806D47F17A34C29E2CE80E8A999FFBE4BE
example.com.        86400  IN     DS     31589 8 2 CDE0D742D6998AA554A92D890F8184C698CFAC8A26FA59875A990C03 E576343C
example.com.        86400  IN     DS     43547 8 1 B6225AB2CC613E0DCA7962BDC2342EA4F1B56083
example.com.        86400  IN     DS     43547 8 2 615A64233543F66F44D68933625B17497C89A70E858ED76A2145997E DF96A918
example.com.        86400  IN     RRSIG  DS 8 2 86400 20211115051557 20211108040557 15549 com.
rHi1CS2yKrG5cqZwcSP1TWtP/gWzs/DHMKULLb5qGfiMVythE5egWqup o6dS/JrfdIRzLMPo7+BLHm8psKAdOf4OGvViyT9tvnKSYDXEZItH6puh
HqOh3WkZ1jjmZHJmUGM/XlQi5wkhrnfp19htU1PjieRjD3jld7qyraco QlHimTu/QWkBw2wn3kXw3UE5hUS61UF3IgQNA0FbO/+Mjg==
```



as you can see the situation is strange. In the end we only care about the lines in blue…

NOTE: all DS records of example.com are signed together with .com ZSK 15549

Let's see the DS record (remember this is in the .com zone)

```
example.com.            86400 IN    DS    31406 8 2
F78CF3344F72137235098ECBBD08947C2C9001C7F6A085A17F518B5D 8F6B916D

example.com.            86400 IN    RRSIG DS 8 2 86400 20211115051557 20211108040557 15549 com.
rHi1CS2yKrG5cqZwcSP1TWtP/gWzs/DHMKULLb5qGfiMVythE5egWqup
o6dS/JrfdIRzLMPo7+BLHm8psKAdOf4OGvViyT9tvnKSYDXEZItH6puh
HqOh3WkZ1jjmZHJmUGM/XlQi5wkhrnfp19htU1PjieRjD3jld7qyraco
QlHimTu/QWkBw2wn3kXw3UE5hUS61UF3IgQNA0FbO/+Mjg==
```

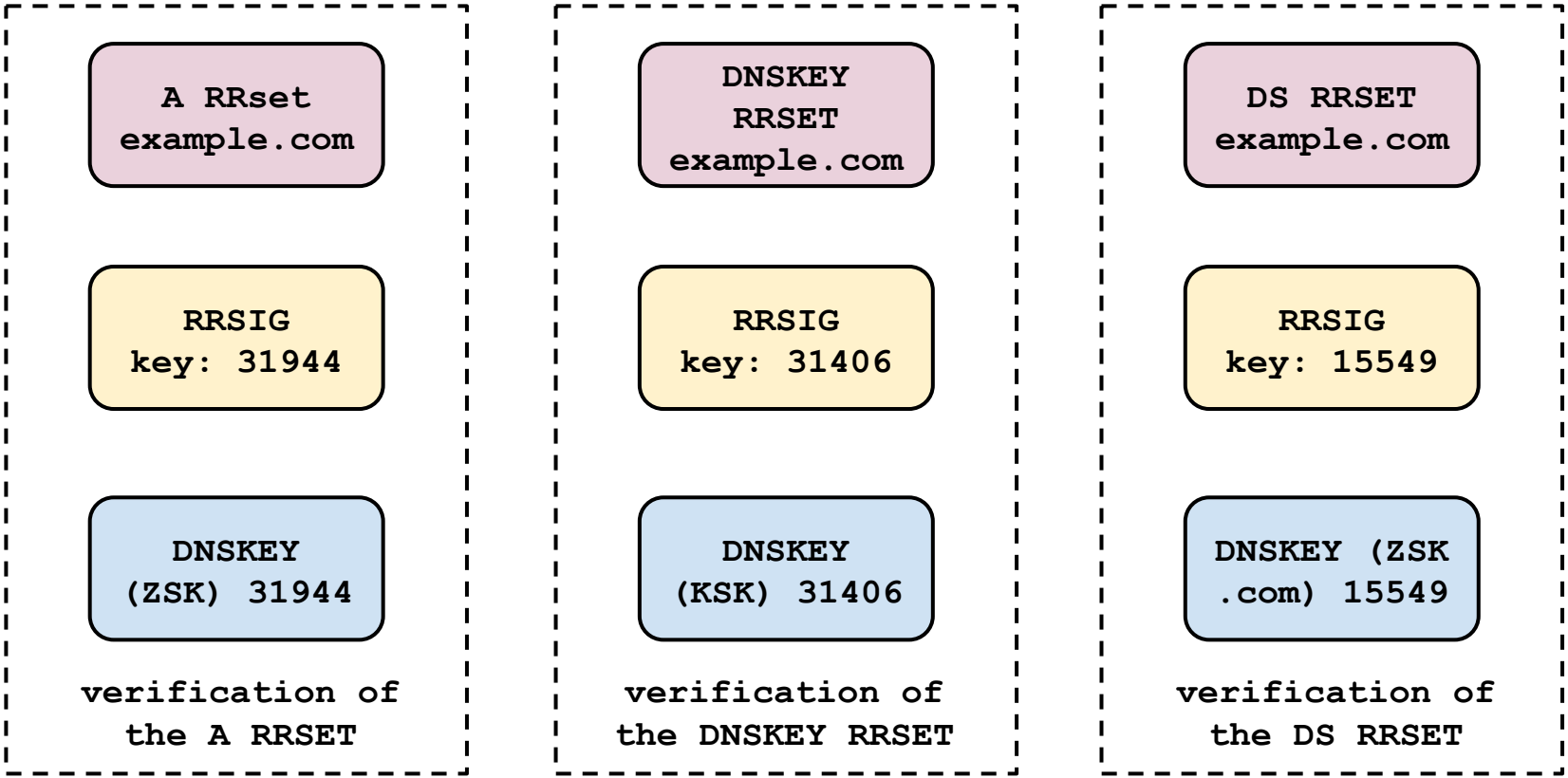
The DS record refers to a DNSKEY RR (31406, i.e. the KSK of example.com), and holds a (hash) digest of the DNSKEY RR. It is used to authenticate the DNSKEY RR.
8 is the RSASHA256 (the algorithm used to create the RRSIG for this record - this must match with the algorithm in the RRSIG record) and 2 is the digest type SHA256 (this is used to compute the digest of the DNSKEY referred by the DS record i.e. key 31406)

Let's see the DS record (remember this is in the .com zone)

```
example.com.            86400 IN    DS    31406 8 2
F78CF3344F72137235098ECBBD08947C2C9001C7F6A085A17F518B5D 8F6B916D

example.com.            86400 IN    RRSIG DS 8 2 86400 20211115051557 20211108040557 15549 com.
rHi1CS2yKrG5cqZwcSP1TWtP/gWzs/DHMKULLb5qGfiMVythE5egWqup
o6dS/JrfdIRzLMPo7+BLHm8psKAdOf4OGvViyT9tvnKSYDXEZItH6puh
HqOh3WkZ1jjmZHJmUGM/XlQi5wkhrnfp19htU1PjieRjD3jld7qyraco
QlHimTu/QWkBw2wn3kXw3UE5hUS61UF3IgQNA0FbO/+Mjg==
```

this is the RRSIG for all DS records related to example.com.
The signature is made with the .com ZSK (id 15549)

Validation


A RRset
example.com
RRSIG
key: 31944
DNSKEY
(ZSK) 31944
verification of
the A RRSET
DNSKEY
RRSET
example.com
RRSIG
key: 31406
DNSKEY
(KSK) 31406
verification of
the DNSKEY RRSET
DS RRSET
example.com
RRSIG
key: 15549
DNSKEY (ZSK
.com) 15549
verification of
the DS RRSET

How to obtain the key_id from the DNSKEY record

- While trying to automate DNS zone generation we need to calculate some of the values programmatically. Two of the auto-generated values had to do with DNSSEC entries: ***The key tag (or keyid)*** and the ***DS record's signatures***.
- The required details on how these are calculated are found in the following places:
  - Key tag: RFC4034 – Appendix B
  - DNSKEY RDATA wire format: RFC4034 – Section 2.1
  - DS' digest field: RFC4034 – Section 5.1.4
  - Wire format of names: RFC1035 – Section 3.1
- See this post for the python code: https://v13.gr/2012/08/17/dnssec-key-tag-keyid-and-ds-signature-calculation-in-python/

Zone Content Exposure

Zone Content Exposure

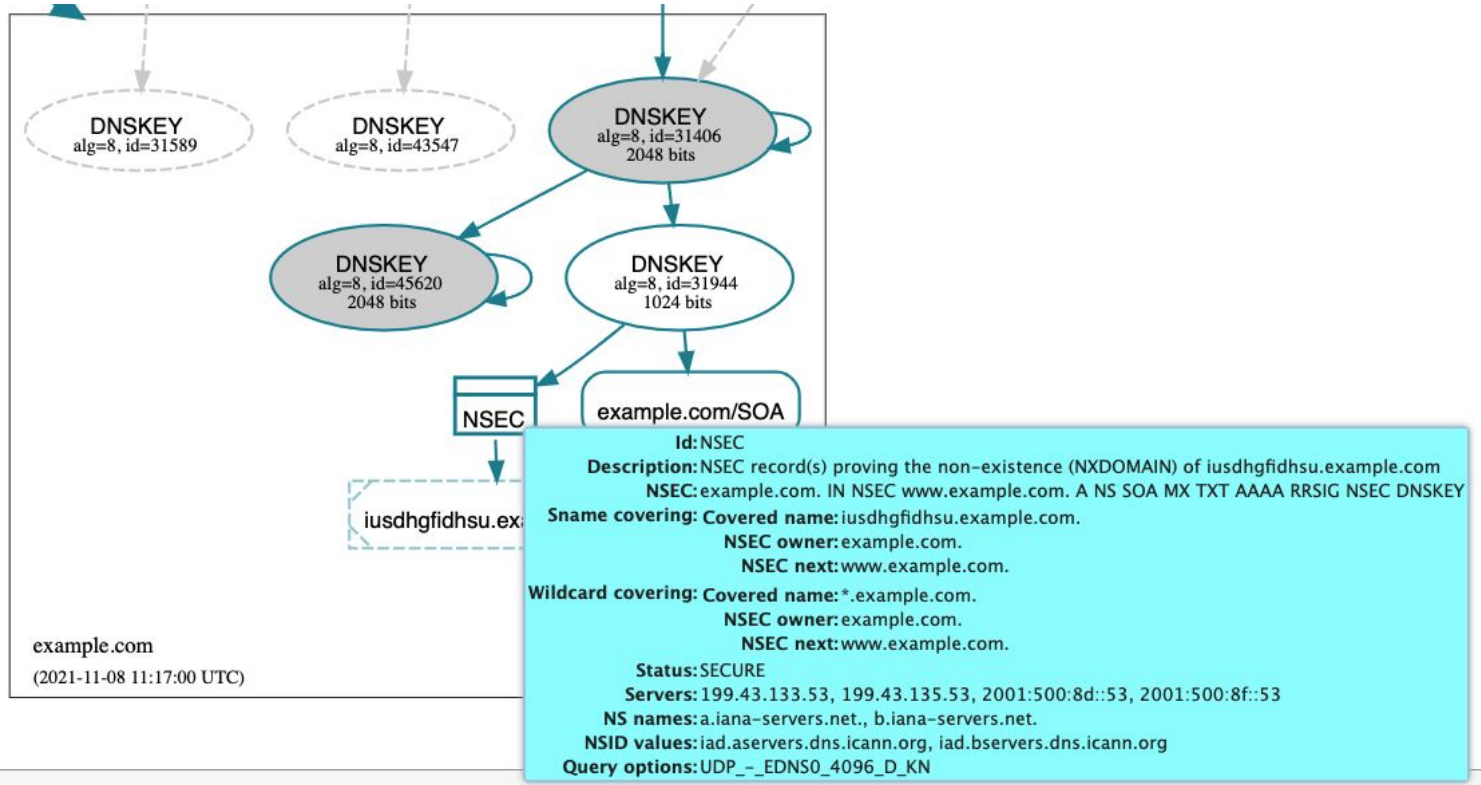
- DNSSEC can expose subdomains
- Historically, DNSSEC is used to sign static zones
  - A static zone is a complete set of records for a given domain
- The DNSSEC signature records are created using the KSK and ZSK in a central location and sent to the authoritative server to be published.
- This set of records allows an authoritative server to answer any question it is asked, including questions about subdomains that don’t exist
- Unlike standard DNS, where the server returns an unsigned NXDOMAIN (Non-Existent Domain) response when a subdomain does not exist, DNSSEC guarantees that every answer is signed

non può essere firmata on the fly un nuovo record

NextSECure record

- DNSSEC guarantees that every answer is signed. This is done with a special record that serves as a proof of non-existence called the ***NextSECure (NSEC) record***
- An NSEC record can be used to say: “there are no subdomains between subdomains X and subdomain Y”
- For statically signed zones, there are, by definition, a fixed number of records
- Since each NSEC record points to the next, this results in a finite ‘ring’ of NSEC records that covers all the subdomains non possono essere mantenuti segreti sottodomini in una zona
- Anyone can ‘walk’ a zone by following one NSEC record to the next until they know all subdomains. This method can be used to reveal all of the names in that zone---possibly exposing internal information.

Example: iusdhgfidhsu.example.com



Example: iusdhgfidhsu.example.com

```
marlon@marlonsMBP ~ % dig iusdhgfidhsu.example.com +dnssec
[...truncated...]
;; QUESTION SECTION:
;iusdhgfidhsu.example.com.    IN    A

;; AUTHORITY SECTION:
example.com.            135   IN    SOA   ns.icann.org. noc.dns.icann.org. 2021120710 7200 3600 1209600
3600
example.com.            135   IN    RRSIG SOA 8 2 3600 20220109092501 20211218211841 31944 example.com.
pD5I3leuwVHCuAbMDYjgvt5GmiCiYwTd23nMouzey29gZ4jT4V1rIU9T
mkaIMz6N4Ie0itspvVDXdTGE6JhVNbHrOBq9UsP90ZNOzNA1iuc+09Ic
K0zh4qbYoYsvDkp8BH6uSTNz/gJz1nGsKGDkVXEM6LY3vHpZGob/JlNc tfE=
example.com.            1935  IN    NSEC  www.example.com. A NS SOA MX TXT AAAA RRSIG NSEC DNSKEY
example.com.            1935  IN    RRSIG NSEC 8 2 3600 20220108121020 20211218211841 31944 example.com.
eYrJpIfblCW23E96kfrvT6KbFZidpTufZdrhAajU55OB7eQc2676bGnc
WckuVcFeLNL4SYk5qz47IC1fLRsy9MAguwjbC1pdDQkiIkXlJRcNvkjb
GBveIBbquyv1rfqWjzjPLa5S4hULlcKERhPtDrBJgu/f3BnSLIY1GoQT ido=
```

this record basically says that there is no record between "example.com" and "www.example.com"

Example: iusdhgfidhsu.example.com

```
marlon@marlonsMBP ~ % dig iusdhgfidhsu.example.com +dnssec
[...truncated...]
;; QUESTION SECTION:
;iusdhgfidhsu.example.com.    IN    A

;; AUTHORITY SECTION:
example.com.            135   IN    SOA   ns.icann.org. noc.dns.icann.org. 2021120710 7200 3600 1209600
3600
example.com.            135   IN    RRSIG SOA 8 2 3600 20220109092501 20211218211841 31944 example.com.
pD5I3leuwVHCuAbMDYjgvt5GmiCiYwTd23nMouzey29gZ4jT4V1rIU9T
mkaIMz6N4Ie0itspvVDXdTGE6JhVNbHrOBq9UsP90ZNOzNA1iuc+09Ic
K0zh4qbYoYsvDkp8BH6uSTNz/gJz1nGsKGDkVXEM6LY3vHpZGob/JlNc tfE=
example.com.            1935  IN    NSEC  www.example.com. A NS SOA MX TXT AAAA RRSIG NSEC DNSKEY
example.com.            1935  IN    RRSIG NSEC 8 2 3600 20220108121020 20211218211841 31944 example.com.
eYrJpIfblCW23E96kfrvT6KbFZidpTufZdrhAajU55OB7eQc2676bGnc
WckuVcFeLNL4SYk5qz47IC1fLRsy9MAguwjbC1pdDQkiIkXlJRcNvkjb
GBveIBbquyv1rfqWjzjPLa5S4hULlcKERhPtDrBJgu/f3BnSLIY1GoQT ido=
```

and these are the existing records for example.com: A NS SOA MX TXT AAAA RRSIG NSEC DNSKEY

Should DNS records be secret?

- DNS records are not supposed to be secret, but in practice they are sometimes considered so
- Subdomains have been used to keep things (such as a corporate login page) private for a while, and suddenly revealing the contents of the zone file may be unexpected and unappreciated
- Before DNSSEC the only way to discover the contents of names in a zone was to either query for them, or attempt to perform a transfer of the zone from one of the authoritative servers
- Zone Transfers (AXFR) are frequently blocked.

NSEC3 record

- NSEC3 was introduced to fight zone enumeration concerns
  - but even NSEC3 can be used to reveal the existence of subdomains
- The NSEC3 record is like an NSEC record, but, rather than a signed gap of domain names for which there are no answers to the question, NSEC3 provides a signed gap of hashes of domain names
- NSEC3 chain for a zone containing “example.com” and “www.example.com” could be

```
231SPNAMH63428R68U7BV359PFPJI2FC.example.com. NSEC3
1 0 3 ABCDEF NKDO8UKT2STOL6EJRD1EKVD1BQ2688DM A NS
SOA TXT AAAA RRSIG DNSKEY NSEC3PARAM
```

salted hash of example.com

```
NKDO8UKT2STOL6EJRD1EKVD1BQ2688DM.example.com. NSEC3
1 0 3 ABCDEF 231SPNAMH63428R68U7BV359PFPJI2FC A TXT
AAAA RRSIG
```

NSEC3 record

- NSEC3 was introduced to fight zone enumeration concerns
  - but even NSEC3 can be used to reveal the existence of subdomains
- The NSEC3 record is like an NSEC record, but, rather than a signed gap of domain names for which there are no answers to the question, NSEC3 provides a signed gap of hashes of domain names
- NSEC3 chain for a zone containing “example.com” and “www.example.com” could be

```
231SPNAMH63428R68U7BV359PFPJI2FC.example.com. NSEC3
1 0 3 ABCDEF NKDO8UKT2STOL6EJRD1EKVD1BQ2688DM A NS
SOA TXT AAAA RRSIG DNSKEY NSEC3PARAM
```

salted hash of www.example.com

```
NKDO8UKT2STOL6EJRD1EKVD1BQ2688DM.example.com. NSEC3
1 0 3 ABCDEF 231SPNAMH63428R68U7BV359PFPJI2FC A TXT
AAAA RRSIG
```

NSEC3 record

- NSEC3 was introduced to fight zone enumeration concerns
  - but even NSEC3 can be used to reveal the existence of subdomains
- The NSEC3 record is like an NSEC record, but, rather than a signed gap of domain names for which there are no answers to the question, NSEC3 provides a signed gap of hashes of domain names
- NSEC3 chain for a zone containing “example.com” and “www.example.com” could be

```
231SPNAMH63428R68U7BV359PFPJI2FC.example.com. NSEC3
1 0 3 ABCDEF NKDO8UKT2STOL6EJRD1EKVD1BQ2688DM A NS
SOA TXT AAAA RRSIG DNSKEY NSEC3PARAM
```

next DNSSEC record

```
NKDO8UKT2STOL6EJRD1EKVD1BQ2688DM.example.com. NSEC3
1 0 3 ABCDEF 231SPNAMH63428R68U7BV359PFPJI2FC A TXT
AAAA RRSIG
```

NSEC3 record

- With NSEC3, when you request a record that does not exist, a signed NSEC3 record is returned with the next zone present ordered alphabetically by hash
- With the hashes that correspond to all the valid names in the zone, a dictionary attack can be used to figure out the real names
- Short names are easily guessed, and by using a dictionary, longer names can be revealed as existing without having to flood the authoritative nameservers with guesses

DNSSEC white lies

- NSEC is like revealing plaintext passwords, and NSEC3 is like revealing a Unix-style passwords file. Neither technique is very secure. With NSEC3 a subdomain is only as private as it is hard to guess.
- This vulnerability can be mitigated by a techniques introduced in RFCs 4470 and 4471 called "***DNSSEC white lies***" viene generato un record falso sul momento
  - from RFC 4470 << *This mechanism involves changes to NSEC records for instantiated names, which can still be generated and signed in advance, as well as the on-demand generation and signing of new NSEC records whenever a name must be proven not to exist* >>. *In the "next name" field of instantiated names' NSEC records, rather than list the next instantiated name in the zone,* ***list any name that falls lexically after the NSEC's owner name and before the next instantiated name in the zone****. This relaxes the requirement that the "next name" field contains the next owner name in the zone* >>
  - << *Whenever an NSEC record is needed to prove the non-existence of a name, a new NSEC record is dynamically produced and signed. The new NSEC record has an owner name lexically before the QNAME but lexically following any existing name and a "next name" lexically following the QNAME but before any existing name* >>
- DNSSEC implementations may leverage ***ECDSA's efficient signature generation to sign DNSSEC records on-the-fly***

Security considerations about white lies

1. On-demand signing requires that a zone's authoritative servers have access to its private keys. Storing private keys on well-knows Internet-accessible servers may make them more vulnerable to unintended disclosure.
2. Since generation of digital signatures tends to be computationally demanding, the requirement for on-demand signing makes authoritative servers vulnerable to a denial of service attack.
3. If the epsilon functions are predictable, on-demand signing may enable a chosen-plaintext attack on a zone's private keys

Reflection/Amplification Threat

faccio tante richieste ma le risposte le faccio arrivare a un IP scelto da me / invio 10 megabyte di richiesta, la risposta e di molti GB ad esempio con le grandi aziende (google.com) avrebbe una risposta molto vasta

Reflection/Amplification Threat

- DNSSEC (like DNS) is vulnerable to reflection attacks
- DNSSEC also works over UDP, and the answers to DNS queries can be very long, containing multiple DNSKEY and RRSIG records.
- This is an attractive target for attackers since it allows them to ‘amplify’ their reflection attacks
- If a small volume of spoofed UDP DNSSEC requests is sent to nameservers, the victim will receive a large volume of reflected traffic.
- Sometimes this is enough to overwhelm the victim’s server, and cause a denial of service.
- More about reflection/amplification attacks and DDoS is coming soon...

How much do we amplificate?

- Asking for a TLD that does not exist from a root server returns an answer that is around 100 bytes, the signed answer for the same question is about 650 bytes or an amplification factor of 6.5.
- The root is signed using a 1,024 bit RSA key and uses NSEC for negative answers.
- Asking for a domain that does not exist in a TLD using NSEC3 signed with 1,024 bit key will yield an amplification factor of around 10.
- There are other queries that can yield even higher amplification factors, the most effective being the “ANY” query.

ECDSA & DNSSEC

DDoS Amplification

- Every time you request a record from a DNSSEC server, it also returns the signature associated with that record, as well as the public key used to verify that signature. That's potentially a lot of information.
- Making the response size for DNSSEC queries as small as possible is an important requirement to prevent abuse of our DNS infrastructure by would-be attackers.
- The small size of ECDSA keys and signatures goes a long way towards that end.

Motivations

- Algorithm 13 is a variant of the Elliptic Curve Digital Signing Algorithm (ECDSA).
- While currently used by less than 0.01% of domains, ECDSA helped different operators to eliminate the final two barriers for widespread DNSSEC adoption:
  - ***Zone enumeration***
  - ***DDoS amplification***
- Zone enumeration is prevented by live signing, and this is only computationally efficient with the fast signature generation of ECDSA.
- Elliptic curves also produce significantly smaller keys and signatures than their RSA counterparts, which means responses to DNS queries are smaller.
- This greatly reduces the amplification factor of DNS-based DDoS attacks

ECDSA & DNSSEC - Zone Enumeration

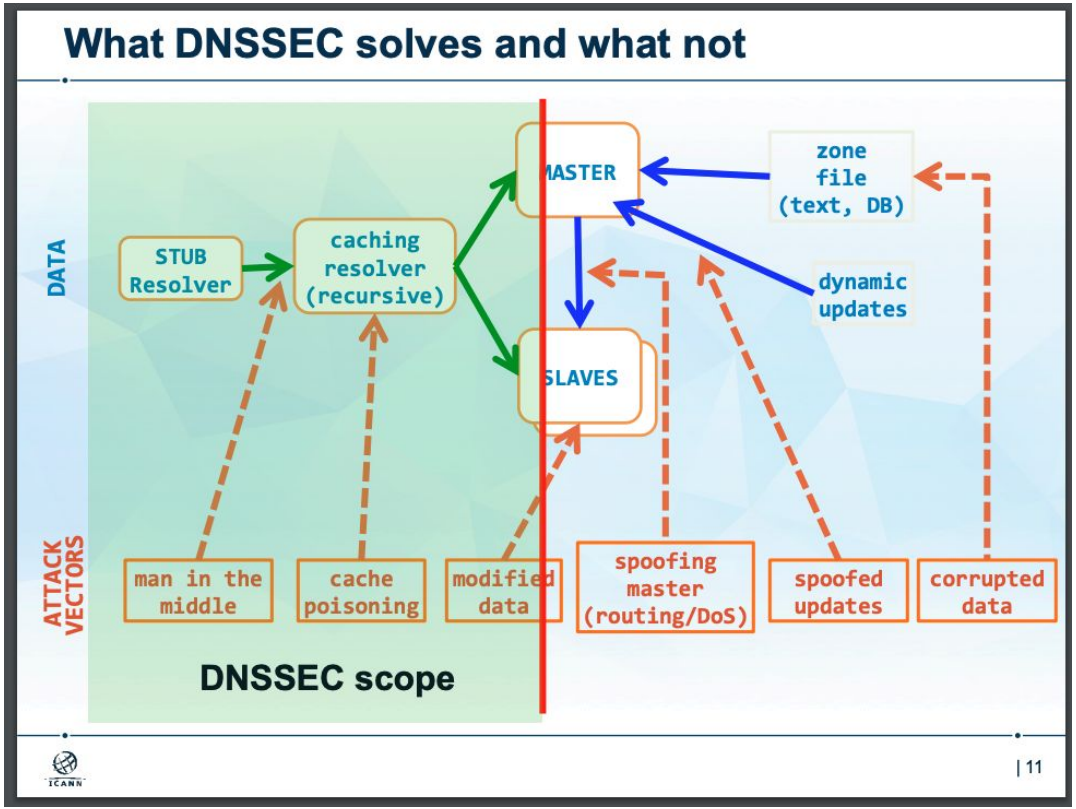
- DNSSEC introduces authenticated denial-of-existence via NSEC and NSEC3 records.
- However, as already discussed both NSEC and NSEC3 allow attackers to walk the zone.
- The solution is a clever technique called “DNSSEC white lies” but it can only be implemented if DNSSEC records are signed on-the-fly.
- RSA is the most widespread signing algorithm is DNSSEC, partly because it’s the only required algorithm defined by the protocol.
- Unfortunately, live signing with RSA is prohibitively expensive.
- The performance gains of ECDSA are dramatic. It’s 10x less computationally expensive to generate an ECDSA signature than a comparable RSA signature.
- This makes live signing (and DNSSEC white lies) feasible, even at scale.

ECDSA vs. RSA Response Size

- Achieving 128-bit security with ECDSA requires a 256-bit key, while a comparable RSA key would be 3072 bits.
  - ***That's a 12x amplification factor just from the keys.***
  - But, most RSA keys are not 3072 bits, so a 12x amplification factor may not be the most realistic figure.
- Let's take a look at a worst-case real-world scenario for DDoS amplification, which is a negative response (NSEC record).
  - For a domain behind Cloudflare (which uses ECDSA signatures and DNSSEC white lies), a typical DNSSEC response is 377 bytes.
  - Compare this to 1075 bytes for a domain not using ECDSA or DNSSEC white lies.
- When you consider the fact that every other large-scale DNSSEC implementation relies on RSA signatures, it's unappealing for an attacker to leverage a DNSSEC infrastructure based on ECDSA as a DDoS vector.

DNSSEC scope and adoption

DNSSEC scope



Current Status DNSSEC

- As of today, not all TLDs support DNSSEC, and registries are legitimized to do so.
- The DNSSEC is still a draft and not a recognized single-standard model TLDs are required strictly to adopt.
- DNSSEC is currently enabled for more than 90% of the TLDs in the DNS, with .se, the Swedish ccTLD, being the first TLD globally to do so in September 2005.
- Two parties must enable DNSSEC: on one side, the registrars responsible for publishing DNS information must ensure that DNSSEC signs their DNS data.
- On the other side, network operators must enable DNSSEC validation on their resolvers that handle DNS lookups for users
- Unfortunately, it is still not widely deployed or used today, and when deployed, it is not done in the right way. ***The rate of DNSSEC validation in June 2021 is estimated at 26,53% worldwide.***

Code	Region	DNSSEC Validates	Partial Validates	Samples	Weight	Weighted Samples
XA	World	26.53%	10.26%	8,842,927	1	8,842,927
XF	Oceania	36.59%	9.15%	44,013	1.44	63,173
XE	Europe	35.31%	7.21%	845,735	1.53	1,296,549
XC	Americas	30.71%	6.17%	1,612,685	0.97	1,570,335
XD	Asia	23.63%	10.99%	4,977,324	1.01	5,040,945
XB	Africa	22.00%	17.97%	1,363,168	0.64	871,733
XG	Unclassified	0.00%	0.00%	124	1.54	190

DNS over TLS or HTTPS

Why do we need an additional security mechanism?

1. ***For Privacy***
   - ❏ DNSSEC is a set of security extensions for verifying the identity of DNS root servers and authoritative nameservers in communications with DNS resolvers.
   - ❏ It is designed to prevent DNS cache poisoning, among other attacks. It does not encrypt communications.
   - ❏ DNS over TLS or HTTPS, on the other hand, does encrypt DNS queries
2. ***But also for security is some cases..***
   - ❏ Stub resolvers are minimal DNS resolvers that use recursive query mode to offload most of the work of DNS resolution to a recursive name server.
     - ❏ A validating stub resolver can also potentially perform its own signature validation by setting the Checking Disabled (CD) bit in its query messages
     - ❏ Non-validating stub resolvers must rely on external DNSSEC validation services
   - ❏ ***Non-validating stub resolvers MUST use a secure channel with the DNS name server***

DNS over TLS

- ***RFC 7858*** defines the use of Transport Layer Security (TLS) to provide privacy and authentication for DNS.
- Encryption/authentication/message-integrity provided by TLS eliminates opportunities for eavesdropping, on-path tampering with DNS queries in the network and DNS name server impersonification

DNS over TLS

- A DNS server that supports DNS over TLS MUST listen for and accept TCP connections on port 853
- A DNS client desiring privacy from DNS over TLS from a particular server MUST establish a TCP connection to port 853 on the server
- Once the DNS client succeeds in connecting via TCP on the well-known port for DNS over TLS, it proceeds with the TLS handshake
- All messages (requests and responses) in the established TLS session MUST use the two-octet length field as defined in RFC 1035
  - *<< Messages sent over TCP connections use server port 53 (decimal). The message is prefixed with a two byte length field which gives the message length, excluding the two byte length field [...] >>*
- In order to minimize latency, clients SHOULD pipeline multiple queries over a TLS session
- In order to amortize TCP and TLS connection setup costs, clients and servers SHOULD NOT immediately close a connection after each response. Instead, clients and servers SHOULD reuse existing connections for subsequent queries as long as they have sufficient resources.

DNS over HTTPS

- ***RFC 8484*** defines a specific protocol, DNS over HTTPS (DoH), for sending DNS queries and getting DNS responses over HTTP using https URIs (and therefore TLS security for integrity and confidentiality).
- Each DNS query-response pair is mapped into an HTTP exchange
- A DoH client encodes a single DNS query into an HTTP request using either the HTTP GET or POST method
  - When the HTTP method is GET the single variable "dns" is defined as the content of the DNS request encoded with base64url
  - When using the POST method, the DNS query is included as the message body of the HTTP request, and the Content-Type request header field indicates the media type of the message. POSTed requests are generally smaller than their GET equivalents.

DNS over HTTPS example (from rfc8484)

- These examples use a DoH service with a URI Template of **`"https://dnsserver.example.net/dns-query{?dns}"`** to resolve IN A records

```
:method = GET
:scheme = https
:authority = dnsserver.example.net
:path =
/dns-query?dns=AAABAAABAAAAAAAAA3d3
dwdleGFtcGxlA2NvbQAAAQAB
accept = application/dns-message
```

```
:method = POST
:scheme = https
:authority = dnsserver.example.net
:path = /dns-query
accept = application/dns-message
content-type = application/dns-message
content-length = 33

<33 bytes in hex encoding>
00 00 01 00 00 01 00 00 00 00 00 00 03
77 77 77 07 65 78 61 6d 70 6c 65  03 63
6f 6d 00 00 01 00 01
```

DOT vs DOH

- From a network security standpoint, DoT is arguably better.
  - It gives network administrators the ability to monitor and block DNS queries, which is important for identifying and stopping malicious traffic.
  - DoH queries, meanwhile, are hidden in regular HTTPS traffic, meaning they cannot easily be blocked without blocking all other HTTPS traffic as well.
- However, from a privacy perspective, DoH is arguably preferable.
  - With DoH, DNS queries are hidden within the larger flow of HTTPS traffic.
  - This gives network administrators less visibility but provides users with more privacy.

Lab: DNSSEC with BIND

Goal

- ❏ protect our zone **`angelotulumello.xyz`** with DNSSEC
- ❏ **`angelotulumello.xyz`** was registered @ namecheap.com
- ❏ the domain was configured to use personal DNS nameservers

Bind9 configuration

per configurare DNS server

```
$ cat named.conf.local

zone "angelotulumello.xyz" {
type master;
file "/etc/bind/db.angelotulumello.xyz";
allow-update { none; };
};
```

```
$ cat named.conf.options
options {
     directory "/var/cache/bind";
     auth-nxdomain no;    # conform to RFC1035
     listen-on-v6 { any; };
     dnssec-validation auto;
};
```

enable dns validation

Initial zone configuration

```
# cat db.angelotulumello.xyz
$TTL 3h
@ IN SOA ns1.angelotulumello.xyz. admin.angelotulumello.xyz. (
1 ; Serial
3h ; Refresh after 3 hours
1h ; Retry after 1 hour
1w ; Expire after 1 week
1h ) ; Negative caching TTL of 1 day
;
@ IN NS ns1.angelotulumello.xyz.
@ IN NS ns2.angelotulumello.xyz.
angelotulumello.xyz. IN MX 10 mail.angelotulumello.xyz.
angelotulumello.xyz. IN A A.A.A.A
ns1 IN A A.A.A.A
ns2 IN A A.A.A.A
www IN A A.A.A.A
mail IN A A.A.A.A
ftp IN CNAME angelotulumello.xyz.
```

Creating a Zone Signing Key (ZSK)

algoritmo crittografico utilizzato

nome della zona

```
$ dnssec-keygen -a ECDSAP384SHA384 -n ZONE angelotulumello.xyz
Generating key pair.
Kangelotulumello.xyz.+014+14196
$ ls Kangelotulumello.xyz.+014+14196.*
Kangelotulumello.xyz.+014+14196.key  Kangelotulumello.xyz.+014+14196.private
```

DNSSEC DNSKEY record

private key associated to the DNSKEY record

Dumping the ZSK DNSKEY record

tutte le informazioni riguardo la chiave

```
# cat Kangelotulumello.xyz.+014+44989.key
; This is a zone-signing key, keyid 44989, for angelotulumello.xyz.
; Created: 20230402080509 (Sun Apr  2 10:05:09 2023)
; Publish: 20230402080509 (Sun Apr  2 10:05:09 2023)
; Activate: 20230402080509 (Sun Apr  2 10:05:09 2023)
angelotulumello.xyz. IN DNSKEY 256 3 14 da0E0UgFl0KjHiyJNdMcKm/TiQYD8lvI8GJXrClDY9UBmSW2cotn633P
Hc8EqKHgoTXG/xbqiiBq3vZhH3Z6u+LVjicSzl+w98S/+ufOFSINhIpA AunaOsAJPBdwbkBE
```

Dumping the ZSK private key

```
cat Kangelotulumello.xyz.+014+44989.private
Private-key-format: v1.3
Algorithm: 14 (ECDSAP384SHA384)
PrivateKey: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Creating a Key Signing Key (KSK)

opzione per generare KSK

```
# dnssec-keygen -f KSK -a ECDSAP384SHA384 -n ZONE angelotulumello.xyz
Generating key pair.
Kangelotulumello.xyz.+014+09686



# cat Kangelotulumello.xyz.+014+09686.key
; This is a key-signing key, keyid 9686, for angelotulumello.xyz.
; Created: 20230402080549 (Sun Apr  2 10:05:49 2023)
; Publish: 20230402080549 (Sun Apr  2 10:05:49 2023)
; Activate: 20230402080549 (Sun Apr  2 10:05:49 2023)
angelotulumello.xyz. IN DNSKEY 257 3 14 ENaKTxl7GMvRUv0em9ETghpKKWyvO2U5j/NXMbw6LWpWPU5GYL1Yimi1
RzZ+fXvG+D+tKpGeoU8dR7XLi+ui4s/sRo/IuCmL6sQyr5uJQrwAkZMr 7hrLPZROIXDW0+1J
```

Adding the keys to the zone configuration file

now have 4 keys - private/public pairs of ZSK and KSK.
We have to add the public keys which contain the DNSKEY record to the zone file.

```
# cat db.angelotulumello.xyz
$TTL 3h
@ IN SOA ns1.angelotulumello.xyz. admin.angelotulumello.xyz. (
1 ; Serial
3h ; Refresh after 3 hours
1h ; Retry after 1 hour
1w ; Expire after 1 week
1h ) ; Negative caching TTL of 1 day
;
@ IN NS ns1.angelotulumello.xyz.
@ IN NS ns2.angelotulumello.xyz.
angelotulumello.xyz. IN MX 10 mail.angelotulumello.xyz.
angelotulumello.xyz. IN A A.A.A.A
ns1 IN A A.A.A.A
ns2 IN A A.A.A.A
www IN A A.A.A.A
mail IN A A.A.A.A
ftp IN CNAME angelotulumello.xyz.

$INCLUDE Kangelotulumello.xyz.+014+09686.key
$INCLUDE Kangelotulumello.xyz.+014+44989.key
```

Signing the zone

The angelotulumello.xyz zone is signed with the dnssec-signzone command.

dnssec-signzone -3 <salt> -A -N INCREMENT -o <zonename> -t <zonefilename>

```
$ dnssec-signzone -A -3 $(head -c 1000 /dev/random | sha1sum | cut -b 1-16) -N INCREMENT -o
angelotulumello.xyz -t db.angelotulumello.xyz
Verifying the zone using the following algorithms: ECDSAP384SHA384.
Zone fully signed:
Algorithm: ECDSAP384SHA384: KSKs: 1 active, 0 stand-by, 0 revoked
                            ZSKs: 1 active, 0 stand-by, 0 revoked
db.angelotulumello.xyz.signed
Signatures generated:                       18
Signatures retained:                         0
Signatures dropped:                          0
Signatures successfully verified:            0
Signatures unsuccessfully verified:          0
Signing time in seconds:                 0.028
Signatures per second:                 642.329
Runtime in seconds:                      0.223
```

checking the output files

The previous command generates two files.

1. ***`db.angelotulumello.xyz.signed` contains RRSIG records for each DNS record set***
2. ***`dsset-angelotulumello.xyz.` is the DS record***

RRSIG record for the www A record set

```
www.angelotulumello.xyz. 10800  IN A     A.A.A.A
                         10800  RRSIG   A 14 3 10800 (
                                        20230502070901 20230402070901 44989 angelotulumello.xyz.
                                        Wwk7urvxX4KeN++4BbT+yC/4gRjScmVLkeyO
                                        zTZQTEcDaoLDVVJLZZRGLcLyus1we1pcjiep
                                        t/9cxruyCanlApyOJuP0Zxdp8IfgmpPRX0jI
                                        +74RxMnl8tj3GDrj5ql0 )
```

DS record

```
# cat dsset-angelotulumello.xyz.
angelotulumello.xyz.    IN DS 9686 14 2 02EB9345109FD0B4BA6415481A51BD85939CA1BE2111F1178DEA4BB2 96711C77
```

changing the zone file in named.conf and restart bind9

```
# cat named.conf.local
zone "angelotulumello.xyz" {
type master;
file "/etc/bind/db.angelotulumello.xyz.signed";
};


# systemctl restart bind9
```

checking the DNSKEY record with dig (in local...)

```
# dig DNSKEY angelotulumello.xyz @localhost

; <<>> DiG 9.16.1-Ubuntu <<>> DNSKEY angelotulumello.xyz @localhost
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12663
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: dcac8cb92d3c3e19010000006429513c8a81bd26202ecffb (good)
;; QUESTION SECTION:
;angelotulumello.xyz.           IN      DNSKEY

;; ANSWER SECTION:
angelotulumello.xyz.    10800   IN      DNSKEY  256 3 14 da0E0UgFl0KjHiyJNdMcKm/TiQYD8lvI8GJXrClDY9UBmSW2cotn633P
Hc8EqKHgoTXG/xbqiiBq3vZhH3Z6u+LVjicSzl+w98S/+ufOFSINhIpA AunaOsAJPBdwbkBE
angelotulumello.xyz.    10800   IN      DNSKEY  257 3 14 ENaKTxl7GMvRUv0em9ETghpKKWyvO2U5j/NXMbw6LWpWPU5GYL1Yimi1
RzZ+fXvG+D+tKpGeoU8dR7XLi+ui4s/sRo/IuCmL6sQyr5uJQrwAkZMr 7hrLPZROIXDW0+1J

;; Query time: 0 msec
;; SERVER: ::1#53(::1)
;; WHEN: dom apr 02 11:56:12 CEST 2023
;; MSG SIZE  rcvd: 300
```

and if we try to validate angelotulumello.xyz from an “external” resolver?

and if we try to validate angelotulumello.xyz from an "external" resolver?

- Everything would be OK except the fact that the signer delegation for angelotulumello.xyz is not certified by the parent zone
  - ***in other words angelotulumello.xyz is not really authenticated!!!***
- We need to complete the certification chain: ***. → .xyz → angelotulumello.xyz***
- The .xyz zone must certify the delegation of the angelotulumello.xyz zone
- In other words, .xyz must sign the delegation signer (DS) record
- How to realize this step depends on how the DNS registrar implements the signer delegation certification procedure
- In this case, *namecheap.com* allows us to upload the DS record(s) for angelotulumello.xyz from the domain dashboard (see the next slide)

Configuring the DS records with the registrar

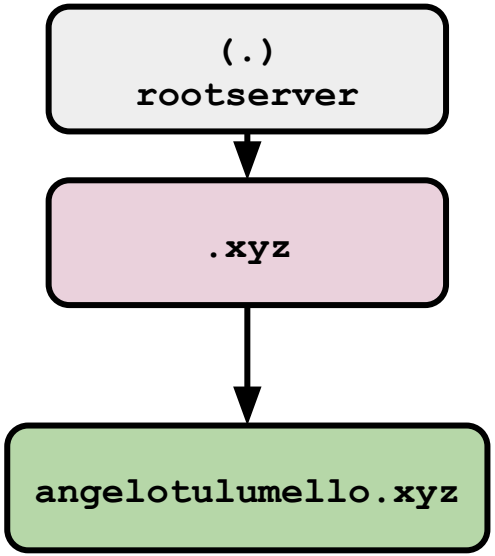
When we ran the dnssec-signzone command, apart from the .signed zone file, a file named **`dsset-angelotulumello.xyz`** was also created.

This contains the DS records that should be entered in our domain registrar's control panel.

aggiunta da namecheap.com

angelotulumello.xyz DNSSEC validation

https://dnsviz.net/d/angelotulumello.xyz/dnssec/

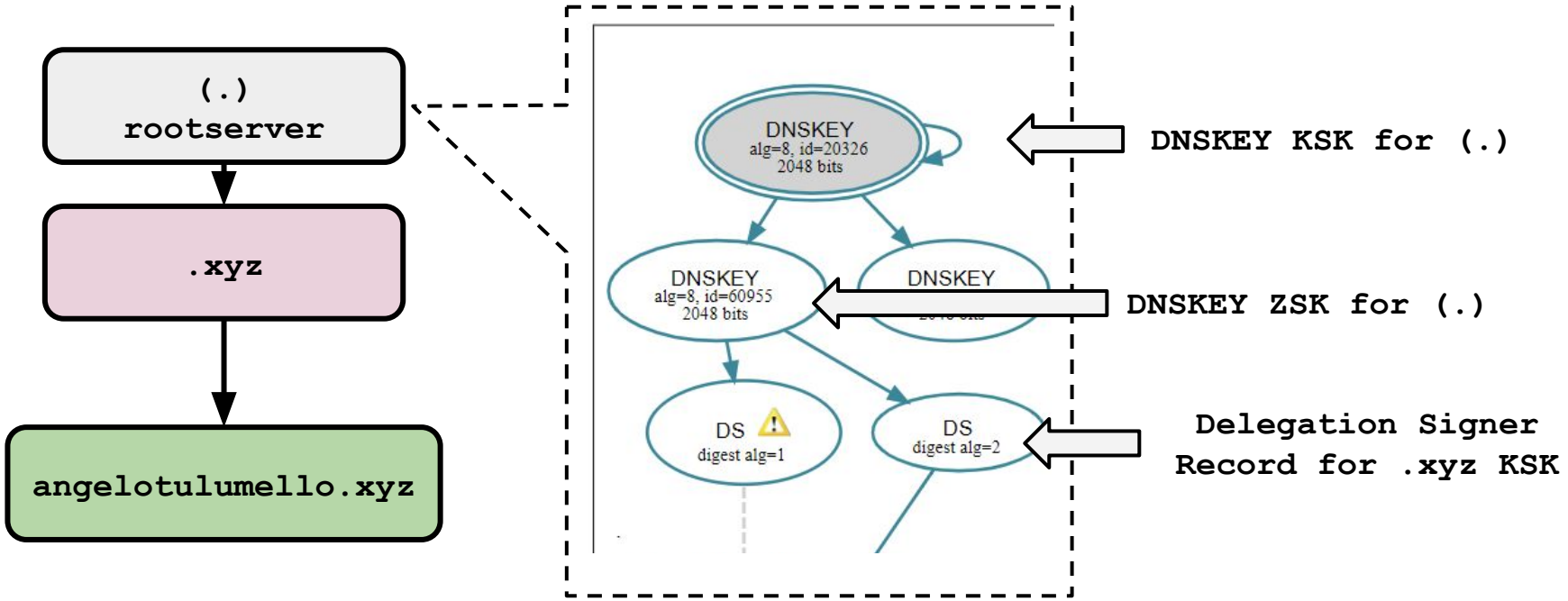


Analyzing DNSSEC problems for angelotulumello.xyz

.	Found 3 DNSKEY records for . DS=20326/SHA-256 verifies DNSKEY=20326/SEP Found 1 RRSIGs over DNSKEY RRset RRSIG=20326 and DNSKEY=20326/SEP verifies the DNSKEY RRset
xyz	Found 2 DS records for xyz in the . zone DS=3599/SHA-256 has algorithm RSASHA256 DS=3599/SHA-1 has algorithm RSASHA256 Found 1 RRSIGs over DS RRset RRSIG=60955 and DNSKEY=60955 verifies the DS RRset Found 2 DNSKEY records for xyz DS=3599/SHA-256 verifies DNSKEY=3599/SEP Found 1 RRSIGs over DNSKEY RRset RRSIG=3599 and DNSKEY=3599/SEP verifies the DNSKEY RRset
angelotulumello.xyz	Found 1 DS records for angelotulumello.xyz in the xyz zone DS=9686/SHA-256 has algorithm ECDSAP384SHA384 Found 1 RRSIGs over DS RRset RRSIG=53358 and DNSKEY=53358 verifies the DS RRset Found 2 DNSKEY records for angelotulumello.xyz DS=9686/SHA-256 verifies DNSKEY=9686/SEP Found 2 RRSIGs over DNSKEY RRset RRSIG=9686 and DNSKEY=9686/SEP verifies the DNSKEY RRset ns1.angelotulumello.xyz is authoritative for angelotulumello.xyz angelotulumello.xyz A RR has value 93.146.67.147 Found 1 RRSIGs over A RRset RRSIG=44989 and DNSKEY=44989 verifies the A RRset
angelotulumello.xyz	ns2.angelotulumello.xyz is authoritative for angelotulumello.xyz angelotulumello.xyz A RR has value 93.146.67.147 Found 1 RRSIGs over A RRset RRSIG=44989 and DNSKEY=44989 verifies the A RRset

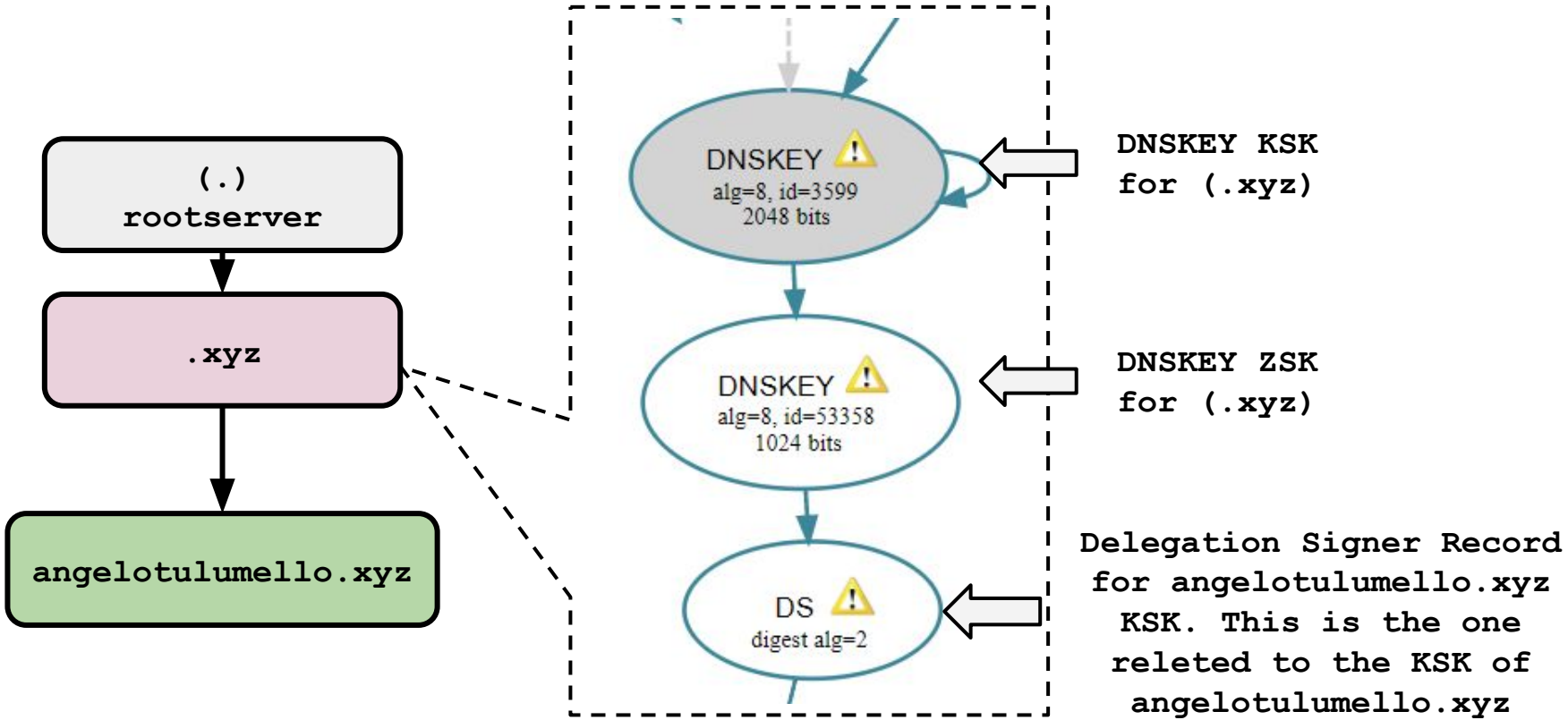
angelotulumello.xyz DNSSEC validation

https://dnsviz.net/d/angelotulumello.xyz/dnssec/



angelotulumello.xyz DNSSEC validation

https://dnsviz.net/d/angelotulumello.xyz/dnssec/



angelotulumello.xyz DNSSEC validation

https://dnsviz.net/d/angelotulumello.xyz/dnssec/