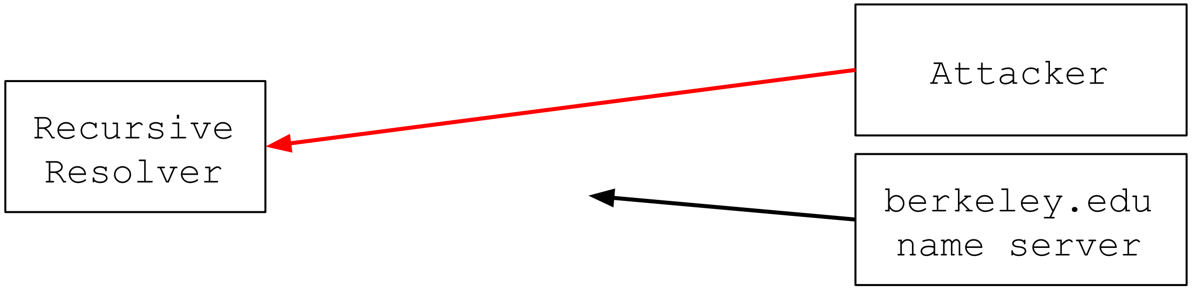
DNS (continued) and DNSSEC

CS 161 Spring 2022 - Lecture 20

Security Risk: On-Path Attackers

- DNS is not secure against on-path attackers
- On-path attackers can poison the cache by sending a spoofed response
  - If the spoofed response arrives before the legitimate response, the victim will cache the attacker's malicious records
  - The on-path attacker can see every field in the unencrypted DNS request. Nothing to guess!



Security Risk: Off-Path Attackers

- The off-path attacker needs to guess the ID field to spoof a response
  - If the ID in the response doesn't match the ID in the request, the resolver won't accept the response
- If the ID number is randomly generated:
  - Probability of guessing correctly = $1/2^{16}$
  - Recall: The ID number is 16 bits long
  - Requires approximately 65,000 tries to successfully send a spoofed packet
  - This is too small!

Source Port	Destination Port	UDP Header
Checksum	Length	
ID number	Flags	DNS Header
Question count	Answer count	
Authority count	Additional count	
Question Records		DNS Payload
Answer Records		
Authority Records		
Additional Records		

Security Risk: Off-Path Attackers

- What if the ID field is incremented by 1 for every request?
- Off-path attacker can spoof a packet as follows:
  - Trick the victim into visiting the attacker’s website
  - Include this HTML on the attacker’s website: `<img src="http://www.attacker.com">`
  - The victim’s browser will make a DNS query for `www.attacker.com`
  - If the attacker controls the `attacker.com` DNS name server, they can see the request and learn the ID field
  - Include this HTML on the attacker’s website: `<img src="http://www.google.com">`
  - The victim’s browser will make a DNS query for `www.google.com`
  - The attacker knows the ID is 1 more than the previous ID, so they can spoof a response!
- **ID numbers need to be random in DNS requests**

Kaminsky Attack

- Notice: If the attacker places `<img src="http://www.google.com">` multiple times on their website, the browser will only make 1 DNS query
    - The browser caches address of `www.google.com`
    - The attacker only gets one try
- Dan Kaminsky, security researcher, noticed that DNS clients would cache additional glue records as if they were authoritative answers, even though they aren’t

Kaminsky Attack

- Now, the attacker can gain more tries at once:
  - The attacker includes
    - `<img src="http://fake1.google.com">`
    - `<img src="http://fake2.google.com">`
    - `<img src="http://fake3.google.com">`
    - `<img src="http://fake4.google.com">`
  - For each, the client makes a request for the domain name
  - The attacker’s spoofed response contains:
    - Authority: `fake1.google.com. 172800 IN NS www.google.com.`
    - Additional: `www.google.com. 172800 IN A 6.6.6.6`
  - The client now caches the record for **www.google.com**, and the cache is poisoned!
  - Changes from “Race once per TTL” to “Race until win”: Same number of packets, but can now try continuously!

Defense: Source Port Randomization

- Randomize the source port of the DNS query
  - The attacker must guess the destination port of the response in addition to the query ID
  - This adds 16 bits to guess, to total 2^{32} possibilities
- Other ways to increase entropy:
  - Randomly caPitAliZe the domain, since the question is copied in the response
    - Works because 99% of DNS authority implementations simply copy the question to start the answer

Source Port	Destination Port	UDP Header
Checksum	Length	
ID number	Flags	DNS Header
Question count	Answer count	
Authority count	Additional count	
Question Records		DNS Payload
Answer Records		
Authority Records		
Additional Records		

Defense: Glue Validation

- **Don't cache glue records as part of DNS lookups**
  - They are necessary, since NS records are given in terms of domain names, not IP addresses
  - If you want to cache, you can perform a separate recursive DNS lookup to validate the glue record authoritatively
- Issue: This was not implemented by all DNS software
  - Unbound, a major DNS implementation, implemented validation
  - BIND, the oldest and most common implementation, did not
    - Mainly for political reasons: They supported DNSSEC, which uses cryptography to validate DNS records

Profiting from DNS Attacks

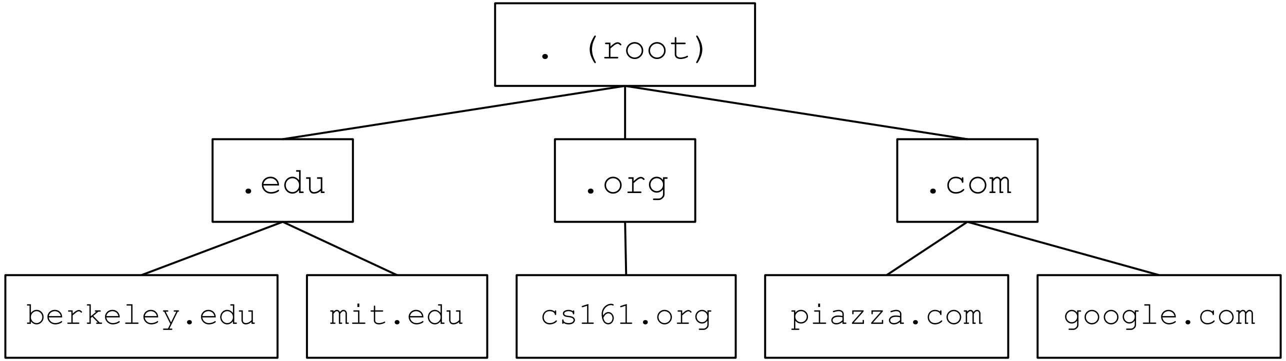
- Suppose you take over a lot of home routers… How do you make money from your attack?
- Change the DNS server settings
  - Each router is programmed with the IP address of the recursive resolver
  - Replace the IP address of the recursive resolver with the attacker's IP address
  - Cache poisoning attacks are now possible!
- Redirect all DNS requests for ads to an attacker-controlled domain and serve attacker-chosen ads to the victim
  - The attacker can now sell this advertising space!
- TLS can defend against this (recall: end-to-end security)

Oh, and that recursive resolver...

- Malcode changing DNS resolver settings…
  - In order to change advertising by returning different IP for ad server names
- ISPs having DNS recursive resolvers lie about nonexistent domains
  - In order to redirect the web browser to a "helpful" page that just happens to include lots of ads
- ISPs having DNS recursive resolvers lie about the IP for Amazon and other stores…
  - In order to stealthily introduce affiliate links so they get a share of purchases
- ISPs having DNS recursive resolvers lie about the IP for Google, Yahoo, and Bing
  - In order to replace search results with advertisements
- **Takeaway**: The recursive resolver needs to be considered one of the most significant adversaries in DNS lookups!

DNS: Summary

- DNS (Domain Name System): An Internet protocol for translating human-readable domain names to IP addresses
  - DNS name servers on the Internet provide answers to DNS queries
  - Name servers are arranged in a domain hierarchy tree
  - Lookups proceed down the domain tree: name servers will direct you down the tree until you receive an answer
  - The stub resolver tells the recursive resolver to perform the lookup



DNS: Summary

- DNS message structure
    - DNS uses UDP for efficiency
    - DNS packets include a random 16-bit ID field to match requests to responses
    - Data is encoded in records, which are name-value pairs with a type
        - **A (answer) type records**: Maps a domain name to an IPv4 address
        - **NS (name server) type records**: Designates another DNS server to handle a domain
    - Records are separated into four sections
        - Question: Contains query
        - Answer: Contains direct answer to query
        - Authority: Directs the resolver to the next name server
        - Additional: Provides extra information (e.g. the location of the next name server)
    - Resolvers cache as many records as possible (until their time-to-live expires)

DNS Security: Summary

- Cache poisoning attack: Send a malicious record to the resolver, which caches the record
  - Causes packets to be sent to the wrong place (e.g. to the attacker, who becomes a MITM)
- Risk: Malicious name servers
  - Defense: Bailiwick checking: Resolver only accepts records in the name server's zone
- Risk: Network attackers
  - MITM attackers can poison the cache without detection
  - On-path attackers can race the legitimate response to poison the cache
  - Off-path attackers must guess the ID field (Defense: Make the ID field random)
    - Kaminsky attack: Query non-existent domains and put the poisoned record in the additional section (which will still be cached). Lets the off-path attacker try repeatedly until succeeding
    - Defense: Source port randomization (more bits for the off-path attacker to guess)

Next: DNSSEC

- DNS over TLS
  - Issues
- DNSSEC
  - High-level design
  - Design details
  - Implementation details
  - Key-signing keys and zone-signing keys
  - NSEC: Signing non-existent domains
  - In practice

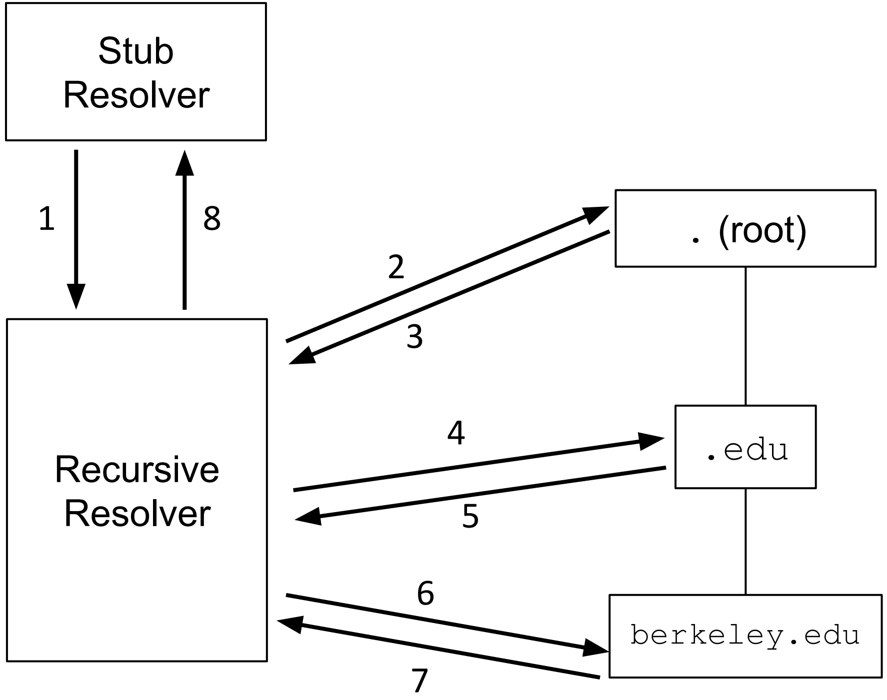
DNS over TLS

Securing DNS Lookups

- Recall: DNS is not secure against several threats
  - Malicious name servers
  - Network attackers (MITM, on-path, off-path)
- We want **integrity** on the response
  - Recall: Integrity means an attacker can’t tamper with the results
  - Prevents cache poisoning attacks
- We do not need **confidentiality** on the response
  - DNS results are public: The attacker can always look up the results themselves!
  - Even if the attacker couldn’t see the DNS response, they can still see which IP you connect to later

DNS over TLS

- Idea: TLS is end-to-end secure, so let's send all DNS requests and responses over TLS



DNS over TLS: Issues

- Performance: DNS needs to be lightweight and fast. TLS is slow.
  - Recall: TLS requires a long cryptographic handshake before any messages can be sent
- Caching: DNS records are cached. TLS doesn't help us with caching.
  - What if someone changes the record while it's stored in the cache?
- Security: DNS over TLS doesn't defend against malicious name servers.
  - A malicious name server can still poison the cache
- Security: DNS over TLS doesn't defend against malicious recursive resolvers.
  - The recursive resolver is a full MITM: a malicious recursive resolver can poison the cache before returning the result to the user
  - The recursive resolver is the most common MITM adversary in DNS

Object Security and Channel Security

- Main problem: DNS over TLS secures the communication channel, but doesn't help you trust who you're talking to
  - Example: TLS secures your communication with the recursive resolver, but you still need to implicitly trust the recursive resolver. What if the recursive resolver is malicious?
- **Channel security**: Securing the communication channel between two end hosts
- **Object security**: Securing a piece of data (in transit or in storage)
- TLS provides channel security, but to secure DNS, we need object security

DNS over TLS in Practice

- Recently introduced by Firefox
  - Enabled by default in the United States
- Benefits
  - A local network adversary can’t see or manipulate the names you are looking up
  - Performance hit is very low: setting up the TLS connection is expensive in latency...But that only happens at the start
- Drawbacks
  - Only defends against network attackers, not malicious name servers
  - Network attackers can perform a **downgrade attack**: Block the TLS connection, forcing the browser to fall back on ordinary DNS
  - Network attackers can still target the final communications
- DNS over TLS traffic is routed through Cloudflare
  - Cloudflare is a full MITM
  - The only protection is contractual: Cloudflare promises not to misuse your data
- **Takeaway**: DNS over TLS is not enough to fully secure DNS

DNSSEC: High-Level Design

DNSSEC

- **DNSSEC** (**DNS Security Extensions**): An extension of the DNS protocol that ensures integrity on the results
  - Designed to cryptographically prove that returned answers are correct
  - Uses a hierarchical, distributed trust system to validate records
- DNSSEC is backwards-compatible
  - Some, but not all name servers support DNSSEC
  - DNSSEC is built on top of ordinary DNS

Warning: Unfiltered DNSSEC Ahead

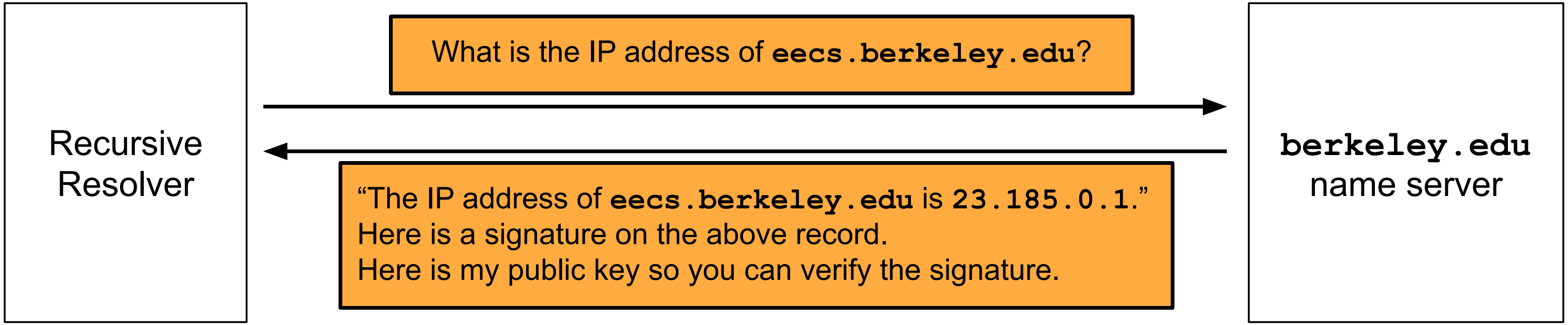
- What you're about to see is the full DNSSEC protocol used in practice, with few simplifications
- Why show complete DNSSEC?
    - DNSSEC is a well-thought-out cryptographic protocol designed to solve a real-world problem
    - DNSSEC is an example of a real-world PKI (public-key infrastructure) that delegates trust using real-world business relationships
    - DNSSEC lets you appreciate what it's like to build real-world security

Scratchpad: Let's Design It Together

- Question 1: What kind of cryptographic primitive should we use to ensure integrity on the records?
  - We should use a scheme that provides integrity: either MACs (symmetric-key) or digital signatures (public-key)
  - Digital signatures are the best solution here: We want everyone to be able to verify integrity (not just the people with the symmetric key)
- Question 2: How do we ensure the returned record is correct and has not been tampered?
  - Recall digital signatures: Only the owner of the private key can sign records, and everyone with the public key can verify
  - The name server should sign the record with their private key
  - We should verify the record with their public key

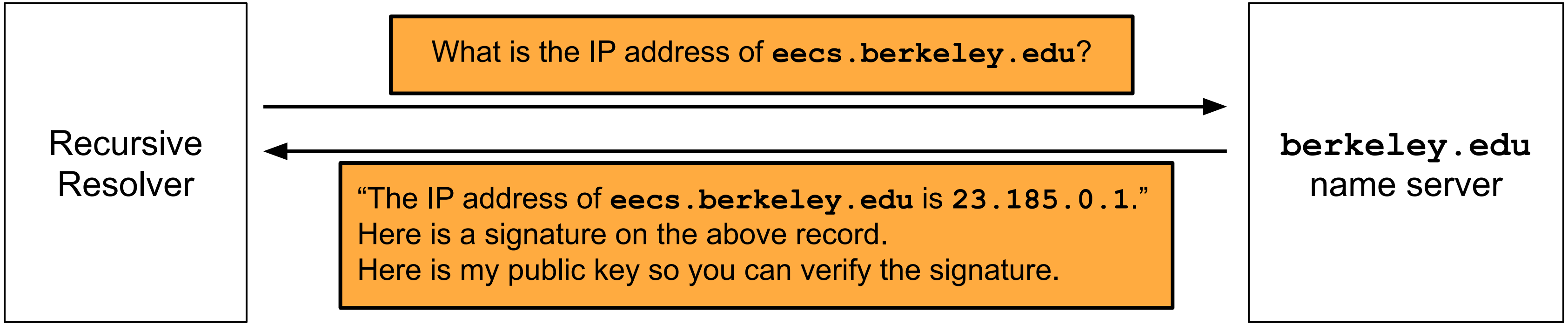
Scratchpad: Let's Design It Together

- Question 3: What does the name server need to send in order to ensure integrity on a record?
  - The record
  - A signature over the record, signed with the private key
  - The public key



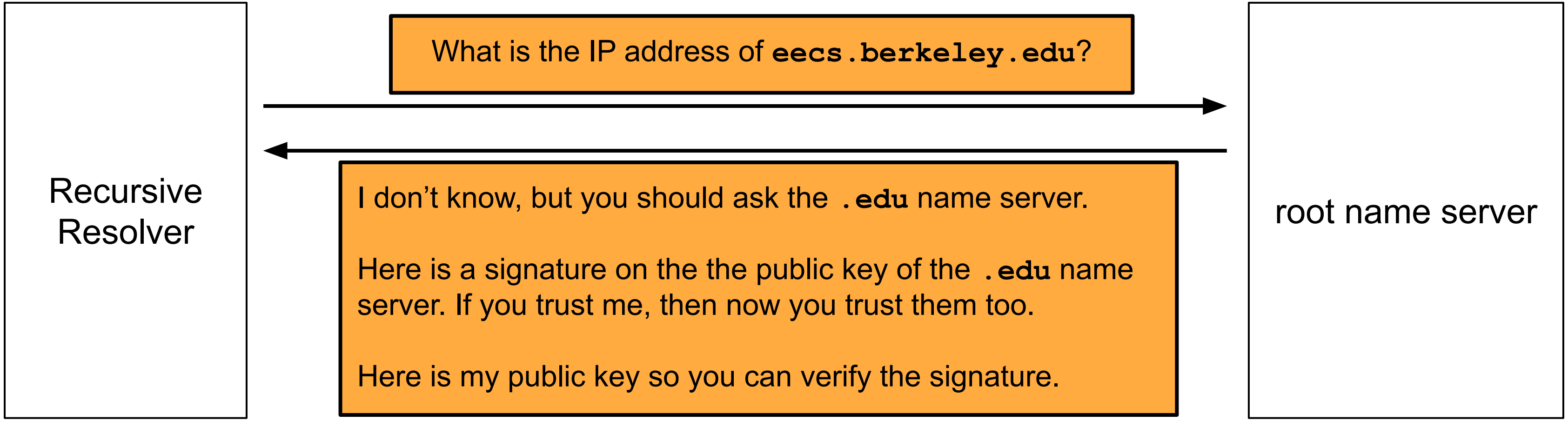
Scratchpad: Let’s Design It Together

- What are some issues with this design?
  - What if the name server is malicious? They could still return malicious records and sign them.
  - How do we make sure nobody tampered with the public key?
  - Do these sound like problems that we’ve solved before in this class? Yes: certificates!



Scratchpad: Let’s Design It Together

- Question 4: How does a name server delegate trust to a child name server?
  - Just like in a certificate chain, the parent must sign the child’s public key.
- Question 5: PKIs need a trust anchor. Who do we implicitly trust in DNSSEC?
  - We implicitly trust the top of the certificate hierarchy, which is the root name server.



DNSSEC: Design Details

Idea #1: Sign Records

- Digital signatures provide integrity
  - Only the name server with the private key can generate signatures
  - Everybody can verify signatures with the public key
- Digital signatures defeat network attackers
  - An off-path, on-path, or MITM attacker can no longer tamper with records
  - The recursive resolver can no longer tamper with records
- Signatures can be cached with the records for object security
  - Any time we fetch a record from the cache, we can verify its integrity

Idea #2: Public-Key Infrastructure (PKI)

- Name servers are arranged in a hierarchy, as in ordinary DNS
- Parents can delegate trust to children
  - The parent signs the child's public key to delegate trust to the child
  - If you trust the parent name server, then now you trust the child name server
- Trust anchor: We implicitly trust the root name server
  - The root name server's public key is hard-coded into resolvers
- PKI defeats malicious name servers
  - A malicious name server (assuming they don't have access to the private key, only the signatures) won't have a valid chain of trust back to the root

Idea #3: Constrained Path of Trust

- In the Web PKI by default we have a gazillion trust anchors
  - And each of these can create certificates that say "this certificate is trusted for everything"
- In the DNS PKI we have a constrained path of trust
  - `.` is trusted for everything
  - **`.edu`** is only trusted for names ending in **`.edu`**
- And those trust relationships are already along established business relationships

Idea #4: DNSSEC is mostly useless for name records!

- What are DNS records used for? Establishing communication with a server…
- Against a MITM or on-path attacker, standard DNS is trivially vulnerable
  - But with good randomization & other protections it is robust against off-path attackers
- But if the attacker is a MITM or on-path for DNS…
  - They are likely to be a MITM or on-path attacker for the actual communication!
  - So if the protocol is E2E secure…
    - Who cares about DNS being correct?
  - And if the protocol was not…
    - The attacker can always attack the final protocol instead
- Robust use: DNSSEC for secure distribution of other keys
  - E.G. DKIM (Domain Keys Identified Email):
    Keys are distributed through DNS…
    And used by mail servers to sign the emails themselves

Steps of a DNSSEC Lookup (Attempt #1)

Recursive Resolver

What is the IP address of **`eecs.berkeley.edu`**?

I don't know, but you should ask the **`.edu`** name server.

Here is a signature on the public key of the **`.edu`** name server. If you trust me, then now you trust them too.

Here is my public key so you can verify the signature.

root name server

Steps of a DNSSEC Lookup (Attempt #1)

Recursive Resolver

What is the IP address of **`eecs.berkeley.edu`**?

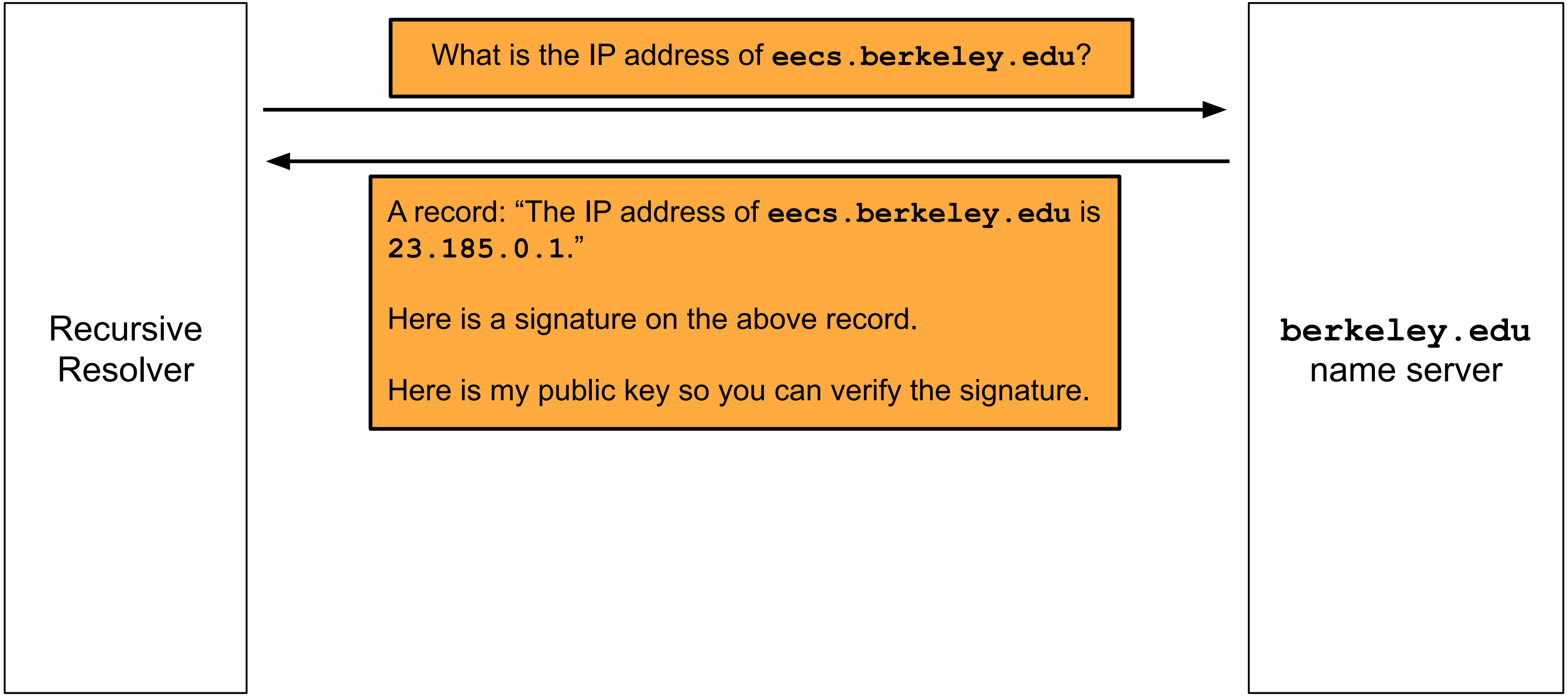
I don’t know, but you should ask the **`berkeley.edu`** name server.

Here is a signature on the public key of the **`berkeley.edu`** name server. If you trust me, then now you trust them too.

Here is my public key so you can verify the signature.

`.edu` name server

Steps of a DNSSEC Lookup (Attempt #1)



DNSSEC: Implementation

Warning: Unfiltered DNSSEC Ahead

- We’re now going to show you the entire DNSSEC protocol, with all its implementation details and edge cases.
- Some parts are less important for the intuition of DNSSEC and won’t be tested on exams. We’re going to highlight these parts in blue.

Review: DNS Packet Format

- The DNS header contains metadata about the query (e.g. ID number, flags)
- There are 8 bits for flags

Source Port	Destination Port	UDP Header
Checksum	Length	UDP Header
ID number	Flags	DNS Header
Question count	Answer count	DNS Header
Authority count	Additional count	DNS Header
Question Records		DNS Payload
Answer Records		DNS Payload
Authority Records		DNS Payload
Additional Records		DNS Payload

OPT Pseudosection

- Ordinary DNS has size limits
  - 8 bits for flags
  - Messages are limited to 512 bytes
- DNSSEC messages exceed these limits
  - Additional flags needed in DNSSEC
    - **DO** flag indicates we support DNSSEC and want DNSSEC records
    - **CD** flag indicates we support DNSSEC, but we don't want to verify the DNSSEC signatures for now
  - Messages are larger than 512 bytes
- Remember: We want DNSSEC to be backwards-compatible
  - We can't modify the existing DNS limits! What should we do?

OPT Pseudosection

- Solution: Encode extra flags in a record called the **OPT Pseudosection**
  - This record has type OPT
  - This record is sent in the additional section
- **EDNS0** (**Extension Mechanisms for DNS**): The protocol that adds the OPT pseudosection
  - If DNSSEC is enabled, the resolver sends the OPT record in the request, and the name server sends the OPT record in the reply
  - The OPT pseudosection can be used to specify the size of larger UDP replies
- **Takeaway**: We found a way to add extra functionality to DNSSEC while supporting ordinary DNSSEC (backwards compatibility)

Resource Record Sets (RRSETs)

- Recall: A DNS record has a name, type, and value
- A group of DNS records with the same name and type form a **resource record set** (**RRSET**)
  - Example: All the AAAA records for a given domain
- RRSETs will be useful for simplifying signatures
  - Instead of signing every record separately, we can sign an entire RRSET at once

New DNSSEC Record Types

- We need new record types to send cryptographic information in DNSSEC packets
  - RRSIG (resource record signature): encode signatures on records
  - DNSKEY: encode public keys
  - DS (delegated signer): encode the child's public key (used to delegate trust)

New DNSSEC Record Types: RRSIG

- RRSIG type records encode a signature on records
  - One RRSIG record (with one signature) can sign an entire RRSET
- RRSIG type records contain some additional metadata
  - Type: What type of DNS record we're signing
  - Algorithm: What algorithm we're using to create the signature
  - Label: Number of segments in the DNS name
  - Original TTL: The TTL for the records in the RRSET
  - Signature expiration time (in Unix time: seconds since January 1, 1970)
  - Signature inception time: When the signature was created (in Unix time)
  - Key tag: What key was used (roughly, a checksum on key bits)
  - The name of the signer

RRSIG in Action (Blue slide)

- Type (CNAME)
- Algorithm (10 -> RSA/SHA512)
- # of labels (3)
- Original TTL
- Valid to (2021-11-11-01:27:12z)
- Valid from (2021-11-07-01:04:00z)
- Key tag (15743)
- Signer (berkeley.edu)
- Signature itself
- **Takeaway**: This is not actually secure! Because amazon is not signing the record that the alias points to!

```
> dig +dnssec www.berkeley.edu

...
www.berkeley.edu.      300     IN      CNAME   www-production-1113102805.us-west-2.elb.amazonaws.com.
www.berkeley.edu.      300     IN      RRSIG   CNAME 10 3 300 20211111012712 20211107010400 15743 berkeley.edu. {cryptogoop}
www-production-1113102805.us-west-2.elb.amazonaws.com. 60 IN A 52.38.34.157
www-production-1113102805.us-west-2.elb.amazonaws.com. 60 IN A 52.26.98.57
```

New DNSSEC Record Types: DNSKEY

- DNSKEY type records encode the name server's own public keys
- DNSKEY type records contain some additional metadata too
  - 16 bits of flags
  - Protocol identifier (currently not in use, so always set to 3)
  - Algorithm identifier

New DNSSEC Record Types: DS

- DS type records encode the hash of the child's public keys
  - Used to delegate trust
- DS type records contain some additional metadata too
  - The key tag
  - The algorithm identifier
  - The hash function used (we'll see this next)
- **Takeaway**: Real-world protocols like DNSSEC require a lot of metadata to function correctly!
  - It's usually pretty uninteresting, though, which is why we abstract it away for you

New DNSSEC Record Types: DS

- Recall delegating trust: The parent signs the child's public key to delegate trust to the child
- DNSSEC delegates trust with two records:
    - A DS type record with the hash of the signer's name and the child's public key
    - An RRSIG type record with a signature on the DS record

Steps of a DNSSEC Lookup (Attempt #2)

Recursive Resolver

What is the IP address of **`eecs.berkeley.edu`**?

I don't know, but you should ask the **`.edu`** name server.

- NS record: Domain of the **`.edu`** name server
- A record: IP address of the **`.edu`** name server

Here is a signature on the public key of the **`.edu`** name server. If you trust me, then now you trust them too.

- DS record: Hash of the **`.edu`** name server's public key
- RRSIG DS record: Signature on the DS record

Here is my public key so you can verify the signature.

- DNSKEY record: The root name server's public key

root name server

Steps of a DNSSEC Lookup (Attempt #2)

Recursive Resolver

What is the IP address of **`eecs.berkeley.edu`**?

I don't know, but you should ask the **`berkeley.edu`** name server.

- NS record: Domain of the **`berkeley.edu`** name server
- A record: IP address of the **`berkeley.edu`** name server

Here is a signature on the public key of the **`berkeley.edu`** name server. If you trust me, then now you trust them too.

- DS record: Hash of the **`berkeley.edu`** name server's public key
- RRSIG DS record: Signature on the DS record

Here is my public key so you can verify the signature.

- DNSKEY record: The **`.edu`** name server's public key

`.edu` name server

Steps of a DNSSEC Lookup (Attempt #2)

Recursive Resolver

What is the IP address of **`eecs.berkeley.edu`**?

- A record: “The IP address of **`eecs.berkeley.edu`** is **`23.185.0.1`**.”

Here is a signature on the above record.

- RRSIG A record: Signature on the A record

Here is my public key so you can verify the signature.

- DNSKEY record: The **`berkeley.edu`** name server’s public key

`berkeley.edu` name server

Key-Signing Keys and Zone-Signing Keys

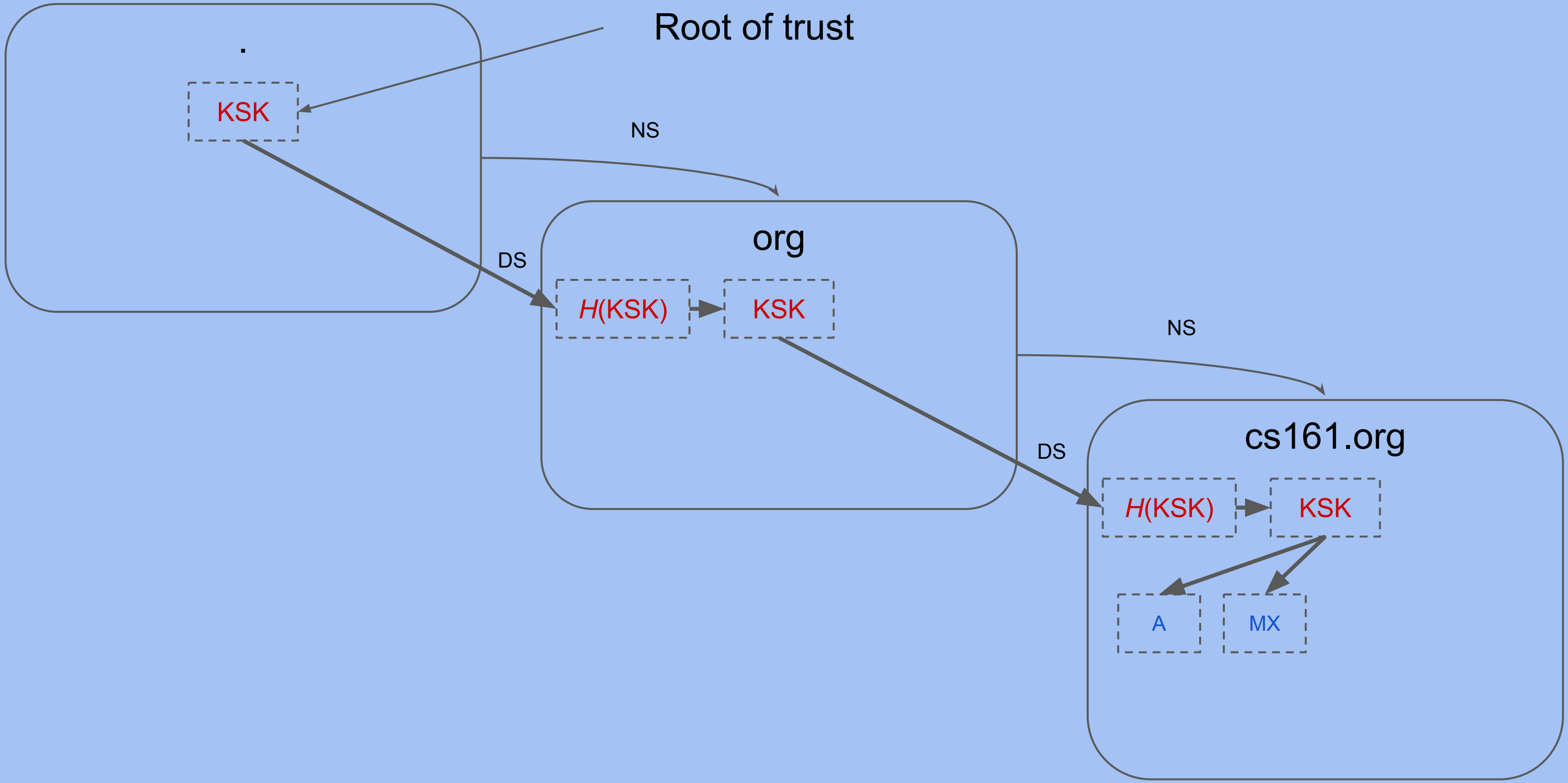
Motivation: Recovering from Key Compromise

- What if a name server wants to change the keys it uses to sign records?
  - Example: This is necessary if the attacker compromises a private key
- The name server needs to inform its parent, since the parent must change its DS record too!
  - This process is complicated and can go wrong in many ways
  - We want to avoid this process whenever possible
- Solution: Divide each name server into an *upper half* and *lower half*
  - If we need to change the keys in the lower half, we don't need to contact another name server: the parent is the upper half of the *same* name server!

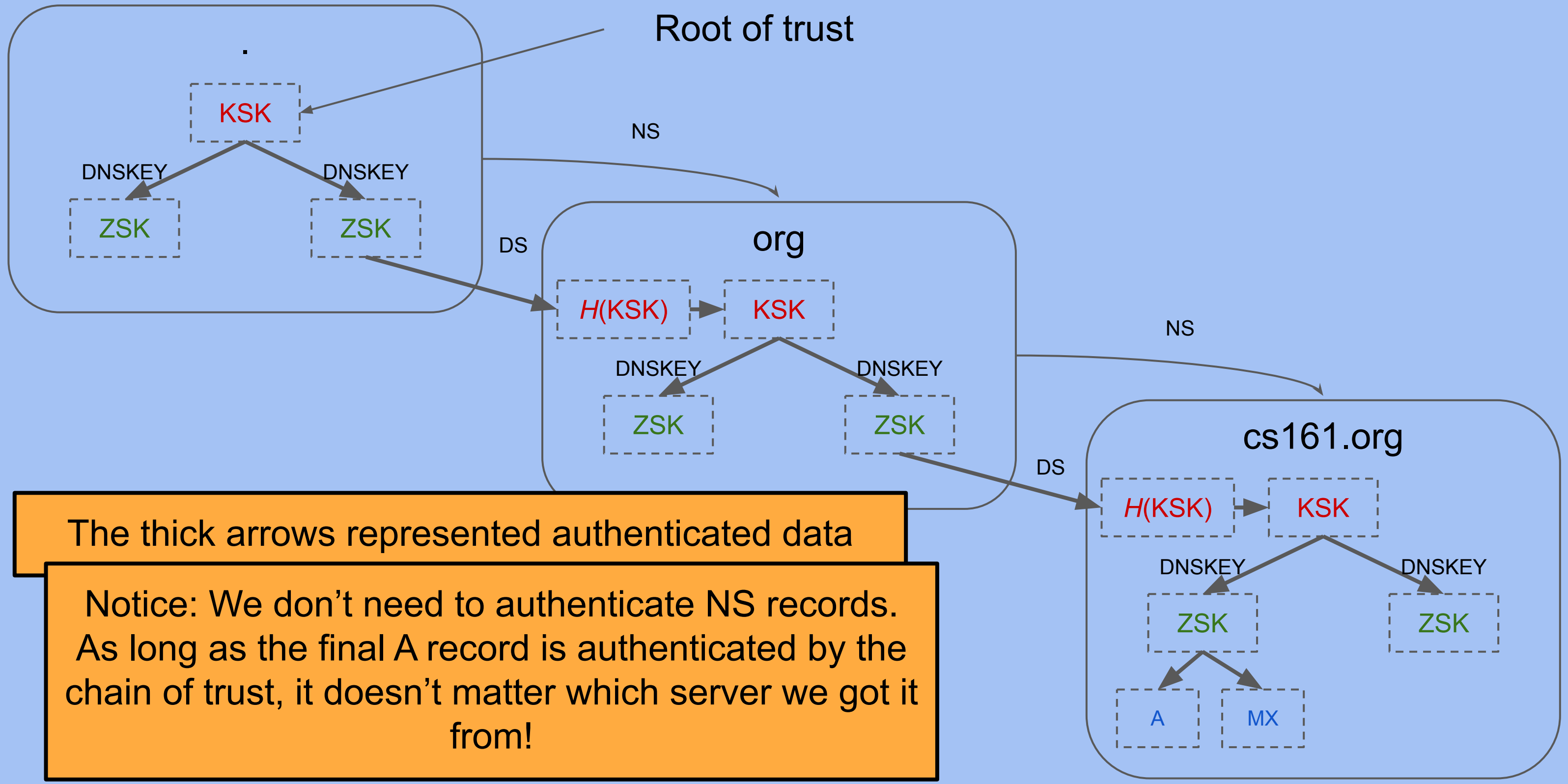
Key-Signing Keys and Zone-Signing Keys

- Each name server has *two* kinds of public-private key pairs
- The **key-signing key** (**KSK**) is used to sign only the zone-signing key
  - Intuition: The KSK is the “upper half” of the name server.
  - The “upper half” endorses the “lower half”
- The **zone-signing key** (**ZSK**) is used to sign all other records
  - Intuition: The ZSK is the “lower half” of the name server
  - The “lower half” endorses the “upper half” of the next name server (or the final answer)
- Example
  - Now, the **`berkeley.edu`** name server has two key pairs (KSK and ZSK)
  - The private KSK is used to sign the public ZSK
  - The private ZSK is used to sign the final A record

Path of Trust (without KSKs and ZSKs)



Path of Trust (with KSKs and ZSKs)



The thick arrows represented authenticated data

Notice: We don't need to authenticate NS records. As long as the final A record is authenticated by the chain of trust, it doesn't matter which server we got it from!

Steps of a DNSSEC Lookup (Attempt #3)

Recursive Resolver

What are your public keys?

Here are my public keys.

- DNSKEY record: The root name server's public KSK
- DNSKEY record: The root name server's public ZSK

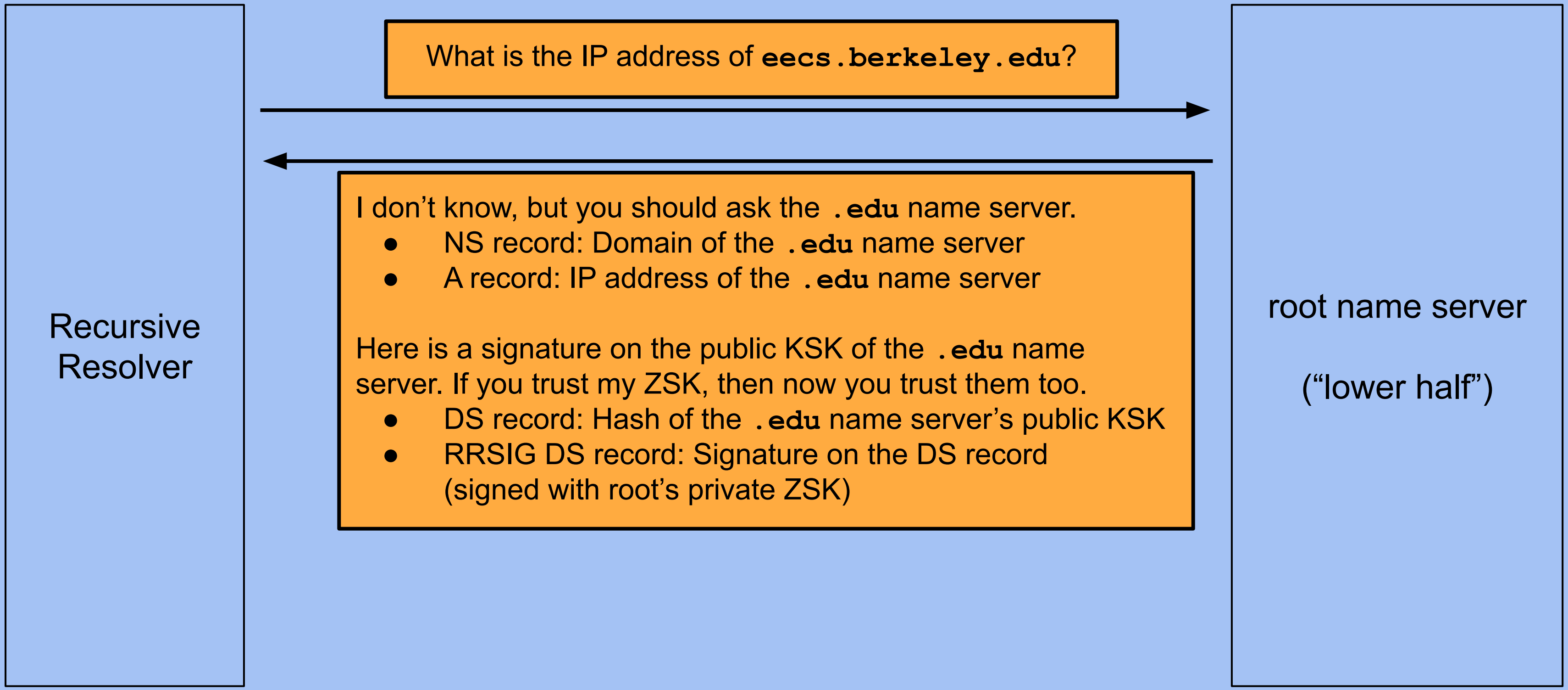
Here is a signature on my ZSK. If you trust my KSK, then now you trust my ZSK.

- RRSIG DNSKEY record: Signature on the DNSKEY records (signed with root's private KSK)

root name server

("upper half")

Steps of a DNSSEC Lookup (Attempt #3)



Steps of a DNSSEC Lookup (Attempt #3)

Recursive Resolver

What are your public keys?

Here are my public keys.

- DNSKEY record: The **`.edu`** name server's public KSK
- DNSKEY record: The **`.edu`** name server's public ZSK

Here is a signature on my ZSK. If you trust my KSK, then now you trust my ZSK.

- RRSIG DNSKEY record: Signature on the DNSKEY records (signed with **`.edu`**'s private KSK)

`.edu` name server

("upper half")

Steps of a DNSSEC Lookup (Attempt #3)

Recursive Resolver

What is the IP address of **`eecs.berkeley.edu`**?

I don't know, but you should ask the **`berkeley.edu`** name server.

- NS record: Domain of the **`berkeley.edu`** name server
- A record: IP address of the **`berkeley.edu`** name server

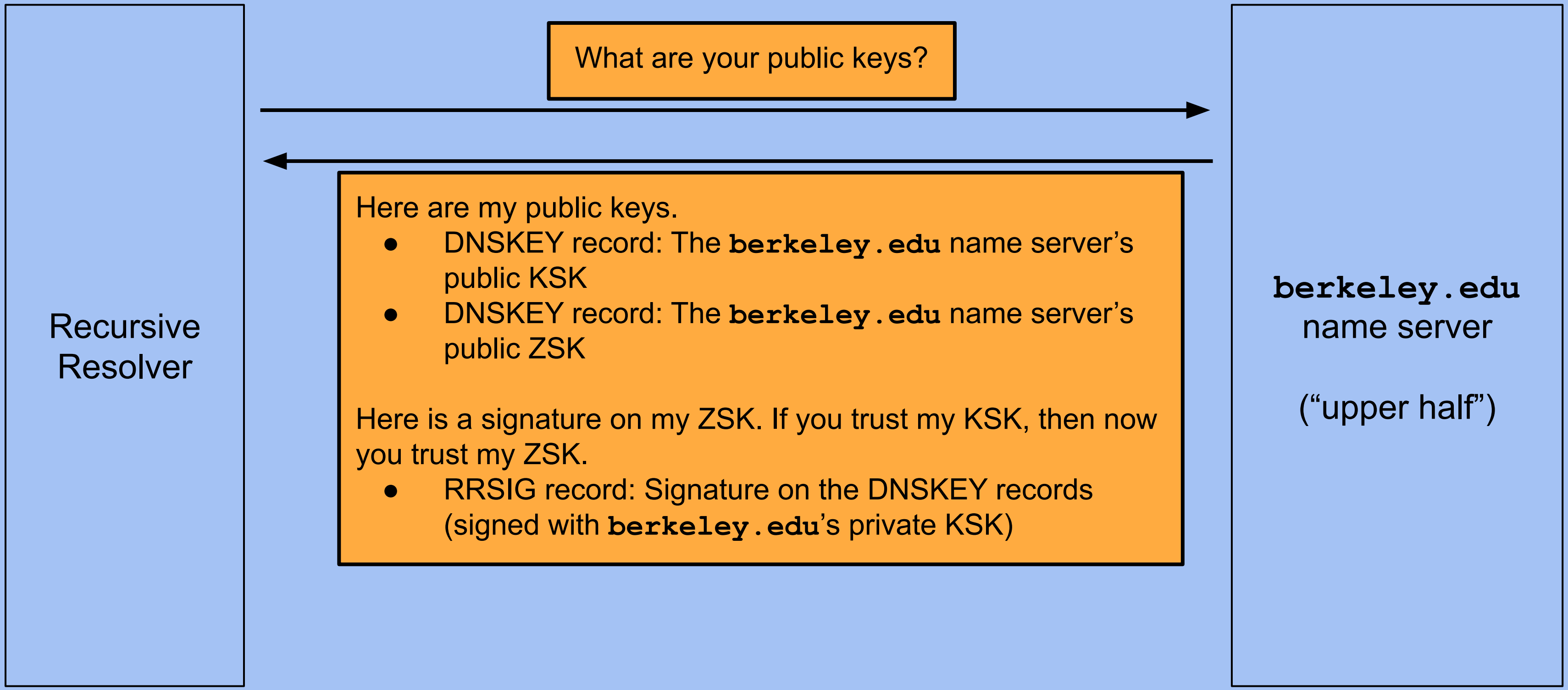
Here is a signature on the public KSK of the **`berkeley.edu`** name server. If you trust my ZSK, then now you trust them too.

- DS record: Hash of the **`berkeley.edu`** name server's public KSK
- RRSIG DS record: Signature on the DS record (signed with root's private ZSK)

`.edu` name server

("lower half")

Steps of a DNSSEC Lookup (Attempt #3)



DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec DNSKEY . @198.41.0.4
```

You can try this at home! Use the **`dig`** utility in your terminal, and remember to set the **`+norecurse`** flag so you can traverse the name server hierarchy yourself and the **`+dnssec`** flag so that you receive DNSSEC responses.

Steps of a DNSSEC Lookup (Attempt #3)

Recursive Resolver

What is the IP address of **`eecs.berkeley.edu`**?

- A record: “The IP address of **`eecs.berkeley.edu`** is **`23.185.0.1`**.”

Here is a signature on the above record.

- RRSIG record: Signature on the A record (signed with **`berkeley.edu`**’s private ZSK)

`berkeley.edu` name server

(“lower half”)

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec DNSKEY . @198.41.0.4
```

The first step is to query the root name server for its public keys.

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec DNSKEY . @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7149
;; flags: qr aa; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1472
;; QUESTION SECTION:
;.               IN    DNSKEY

;; ANSWER SECTION:
.    172800      IN    DNSKEY    256 {ZSK of root}
.    172800      IN    DNSKEY    257 {KSK of root}
.    172800      IN    RRSIG     DNSKEY {signature on DNSKEY records}
...
```

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**

The header says there's 1 record in the additional section, but the additional section is empty! What happened?

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec DNSKEY . @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7149
;; flags: qr aa; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1472
;; QUESTION SECTION:
;.              IN    DNSKEY

;; ANSWER SECTION:
.    172800     IN    DNSKEY    256 {ZSK of root}
.    172800     IN    DNSKEY    257 {KSK of root}
.    172800     IN    RRSIG     DNSKEY {signature on DNSKEY records}
...
```

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**

The additional record is actually the OPT pseudosection, which **`dig`** lists separately for us.

Note the **`do`** flag, which indicates that DNSSEC is supported.

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec DNSKEY . @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7149
;; flags: qr aa; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1472
;; QUESTION SECTION:
;.                IN    DNSKEY

;; ANSWER SECTION:
.    172800      IN    DNSKEY    256 {ZSK of root}
.    172800      IN    DNSKEY    257 {KSK of root}
.    172800      IN    RRSIG     DNSKEY {signature on DNSKEY records}
...
```

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)

The root's KSK signs the root's ZSK. If you trust the root's KSK (trust anchor), now you trust the root's ZSK.

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec eecs.berkeley.edu @198.41.0.4
```

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)

Next, we ask the root name server about the IP address of **`eecs.berkeley.edu`**.

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec eecs.b

;; Got answer:
;; ->>HEADER<<- opcode: QUERY,
;; flags: qr; QUERY: 1, ANSWER:

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do;
;; QUESTION SECTION:
;eecs.berkeley.edu.

;; AUTHORITY SECTION:
edu.                  172800   IN   NS      a.edu-servers.net.
edu.                  172800   IN   NS      b.edu-servers.net.
edu.                  172800   IN   NS      c.edu-servers.net.
...
edu.                  86400    IN   DS      {hash of .edu's KSK}
edu.                  86400    IN   RRSIG   DS {signature on DS record}

;; ADDITIONAL SECTION:
a.edu-servers.net.    172800   IN   A       192.5.6.30
b.edu-servers.net.    172800   IN   A       192.33.14.30
c.edu-servers.net.    172800   IN   A       192.26.92.30
...
```

The records are all the same as ordinary DNS, except for these two extra records endorsing the `.edu` name server's public KSK.

If you trust the root's ZSK, now you trust the `.edu` name server's KSK.

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)
`edu.`	DS

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec DNSKEY edu. @192.5.6.30
```

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)
`edu.`	DS

Next, we query the **`.edu`** name server for its public keys.

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec DNSKEY edu. @192.5.6.30

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9776
;; flags: qr aa; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;edu.          IN   DNSKEY

;; ANSWER SECTION:
edu.   86400   IN   DNSKEY   256 {ZSK of .edu}
edu.   86400   IN   DNSKEY   257 {KSK of .edu}
edu.   86400   IN   RRSIG    DNSKEY {signature on DNSKEY records}
...
```

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)
`edu.`	DS
`edu.`	DNSKEY (KSK)
`edu.`	DNSKEY (ZSK)

The **`.edu`** name server's KSK signs the **`.edu`** name server's ZSK. If you trust **`.edu`**'s KSK, now you trust **`.edu`**'s ZSK.

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec eecs.berkeley.edu @192.5.6.30
```

Next, we ask the **`.edu`** name server about the IP address of **`eecs.berkeley.edu`**.

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)
`edu.`	DS
`edu.`	DNSKEY (KSK)
`edu.`	DNSKEY (ZSK)

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec eecs.berkeley.edu @192.5.6.30

;; Got answer:
;; ->>HEADER<<- opcod
;; flags: qr; QUERY:

;; OPT PSEUDOSECTION:
; EDNS: version: 0, f
;; QUESTION SECTION:
;eecs.berkeley.edu.

;; AUTHORITY SECTION:
berkeley.edu.        172800  IN  NS     adns1.berkeley.edu.
berkeley.edu.        172800  IN  NS     adns2.berkeley.edu.
berkeley.edu.        172800  IN  NS     adns3.berkeley.edu.
berkeley.edu.        86400   IN  DS     {hash of berkeley.edu's KSK}
berkeley.edu.        86400   IN  RRSIG  DS {signature on DS record}

;; ADDITIONAL SECTION:
adns1.berkeley.edu.  172800  IN  A      128.32.136.3
adns2.berkeley.edu.  172800  IN  A      128.32.136.14
adns3.berkeley.edu.  172800  IN  A      192.107.102.142
...
```

Again, the records are all the same as ordinary DNS, except for these two extra records endorsing the **`berkeley.edu`** name server's public KSK.

If you trust the **`.edu`** name server's ZSK, now you trust the **`berkeley.edu`** name server's KSK.

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)
`edu.`	DS
`edu.`	DNSKEY (KSK)
`edu.`	DNSKEY (ZSK)
`berkeley.edu.`	DS

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec DNSKEY berkeley.edu @128.32.136.3
```

The chain of trust	
Name	Type
`.`	**DNSKEY (KSK)**
`.`	DNSKEY (ZSK)
`edu.`	DS
`edu.`	DNSKEY (KSK)
`edu.`	DNSKEY (ZSK)
`berkeley.edu.`	DS

Next, we query the **`berkeley.edu`** name server for its public keys.

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec DNSKEY berkeley.edu @128.32.136.3

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4169
;; flags: qr aa; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1220
;; QUESTION SECTION:
;berkeley.edu.       IN   DNSKEY

;; ANSWER SECTION:
berkeley.edu.   172800   IN   DNSKEY   256 {ZSK of berkeley.edu}
berkeley.edu.   172800   IN   DNSKEY   257 {KSK of berkeley.edu}
berkeley.edu.   172800   IN   RRSIG    DNSKEY {signature on DNSKEY records}
...
```

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)
`edu.`	DS
`edu.`	DNSKEY (KSK)
`edu.`	DNSKEY (ZSK)
`berkeley.edu.`	DS
`berkeley.edu.`	DNSKEY (KSK)
`berkeley.edu.`	DNSKEY (ZSK)

The **`berkeley.edu`** name server's KSK signs the **`berkeley.edu`** name server's ZSK. If you trust **`berkeley.edu`**'s KSK, now you trust **`berkeley.edu`**'s ZSK.

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec eecs.berkeley.edu @128.32.136.3
```

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)
`edu.`	DS
`edu.`	DNSKEY (KSK)
`edu.`	DNSKEY (ZSK)
`berkeley.edu.`	DS
`berkeley.edu.`	DNSKEY (KSK)
`berkeley.edu.`	DNSKEY (ZSK)

Finally, we ask the **`berkeley.edu`** name server about the IP address of **`eecs.berkeley.edu`**.

DNSSEC Lookup Walkthrough


```
$ dig +norecurse +dnssec eecs.berkeley.edu @128.32.136.3

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21205
;; flags: qr aa; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1220
;; QUESTION SECTION:
;eecs.berkeley.edu.          IN   A

;; ANSWER SECTION:
eecs.berkeley.edu.   86400   IN   A       23.185.0.1
eecs.berkeley.edu.   86400   IN   RRSIG   A {signature on A record}
```

The chain of trust	
Name	Type
.	**DNSKEY (KSK)**
.	DNSKEY (ZSK)
`edu.`	DS
`edu.`	DNSKEY (KSK)
`edu.`	DNSKEY (ZSK)
`berkeley.edu.`	DS
`berkeley.edu.`	DNSKEY (KSK)
`berkeley.edu.`	DNSKEY (ZSK)
`eecs.berkeley.edu.`	A

Here's the final answer record, signed by **`berkeley.edu`**'s public ZSK. If you trust **`berkeley.edu`**'s ZSK, then now you trust the final answer.

NSEC: Signing Non-Existent Domains

Nonexistent Domains

- The DNSSEC structure works great for domains which exist
  - We have signatures over records stating that they exist
- What if the user queries for a domain that **doesn't** exist?
  - Option #1: Don't authenticate nonexistent domain (NXDOMAIN) responses
    - Issue: If NXDOMAIN responses don't have to be signed, the attacker can still spoof NXDOMAIN responses and cause denial-of-service (DoS)
  - Option #2: Keep the private key in the name server itself, so it signs NXDOMAIN responses
    - Issue: Name servers have access to the private key, which is an issue if they are malicious or hacked
    - Issue: Signing in real time is slow
  - We need a way that can prove that a domain doesn't exist ahead of time

Two Different Cases

- NOERROR without an answer record
  - The name is valid for some other type, but no record exists of that type for that name
- NXDOMAIN (Non eXistent DOMAIN)
  - No records exist for that name
- Need to handle both cases

NSEC: Authenticated Denial of Existence

- Prove nonexistence of a record type (NOERROR)
  - Sign a record stating that no record of a given type exists
  - Useful for proving that a domain doesn't support DNSSEC ("No DS records exist")
- Prove nonexistence of a domain (NXDOMAIN)
  - Provide two adjacent domains alphabetically, so that you know that no domain in the middle exists
  - Example: If I query for **`nonexistent.google.com`**, I can receive a signed NSEC response saying "No domains exist between **`maps.google.com`** and **`one.google.com`**."
  - We can sign all pairs of adjacent records ahead of time and keep them as NSEC records, along with their RRSIGs

`maps` **`one`** **`web`**

Issues with NSEC

- **Domain enumeration**: It is easy for an attacker to find every single subdomain of a domain
  - Start by querying **`a.google.com`**
  - Receive an NSEC record stating that “No domains exist between **`web.google.com`** and **`ap.google.com`**
    - Now we have learned two domain names!
  - Repeat by querying **`apa.google.com`** (alphabetically immediately after **`ap.google.com`**)
  - Receive an NSEC record stating that “No domains exist between **`ap.google.com`** and **`apps.google.com`**”
  - Repeat until you loop back around to the beginning

`web` **`ap`** **`apps`**