# Computer Network

# Security

ECE 4112/6612 CS 4262/6262

Prof. Frank Li

```
* Welcome to CityPower Grid Rerouting *
           Authorised Users only!
          Hew users Must notify sys/ops.
          login:
                                                                                                EDITUT SShnuke
                                                                               rer ebx, 1
                                                                              bsr ecx, ecx
                                                                              shrd ebx, edi, CL
                                   open
                                                    http://www.
                 11 B DDAP -V -55 -0 10.2.2.2
                                                                              nobile
                13 Starting nmap U. 2.54BETN25
                   Starting map U. 2.540EIR25
Insufficient responses for TCP sequencing (3), OS detection may be less
                   Interesting ports on 10.2.2.2:

(The 1539 Ports Scanned but not shown below are in state: closed)
              68 No exact OS matches for host
                Mnap run completed -- 1 IP address (1 host up) scanneds
Access Level (9)

Amap run completed -- 1 IP address (1 host up) scanner

Be schnuke 10.2.2.2 -rootpu-"Z10N0101". Successful.

Be attempting to 10.2.2.2:55h

Connecting to 10.2.2.2:55h

Connecting to 10.2.2.2:55h

Reseting root exploit SSHv1 :- Successful.

System open: Access Level (9)

Be sch 10.2.2.2 -1 root

Foot010.2.2.2.5 password:
                                                                                 RIF CONTROL
                                                                              ACCESS CRANTED
```

# Logistics

HW2 to be released shortly, due Tuesday, Oct 17 midnight

Quiz 1 regrades due tonight (on Piazza)

Project proposal comments provided

# Going back to DNSSEC

## Securing DNS Lookups

- How can we ensure when clients look up names with DNS, they can trust answers they receive?
- Idea: make DNS results like certs
  - I.e., a verifiable signature that guarantees who generated a piece of data; signing happens off-line

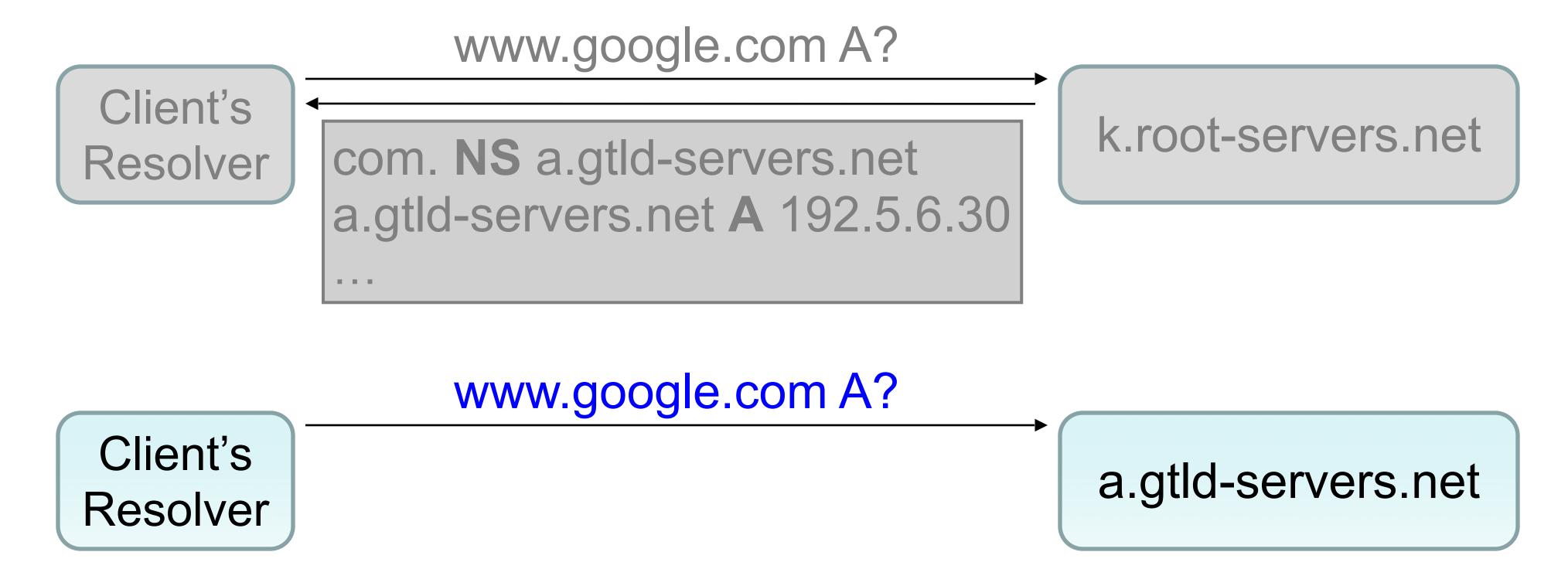
#### Another PKI! **PKI** root of trust! Known public key. Root Name Servers Public key signed by root server .org Name Servers .edu Name Servers .com Name Servers umass.edu pbs.org yahoo.com amazon.com Name Server Name Server Name Server Name Servers Name Servers Public key signed by .com name server

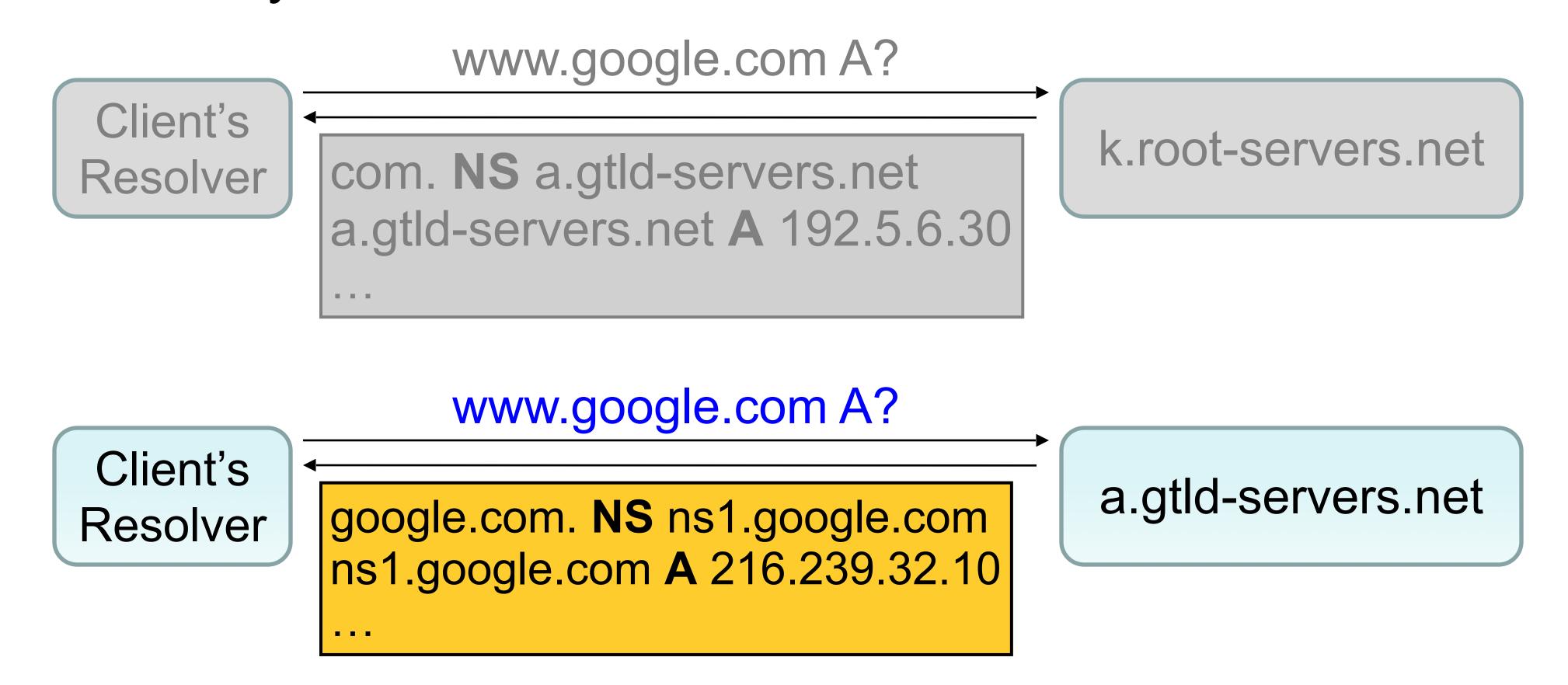
## Operation of DNSSEC

- DNSSEC = standardized DNS security extensions currently being deployed
- As resolver queries root NS down to authoritative NS, at each level it gets signed statements regarding the key(s) used by the next level
  - Builds a chain of trusted keys
  - Resolver has root's key wired into it
- Final answer from authoritative NS is signed by that level's key
  - Resolver can trust it's the right key because of chain of support from higher levels
- All keys as well as signed results are cacheable

Client's Resolver k.root-servers.net











#### www.google.com A?

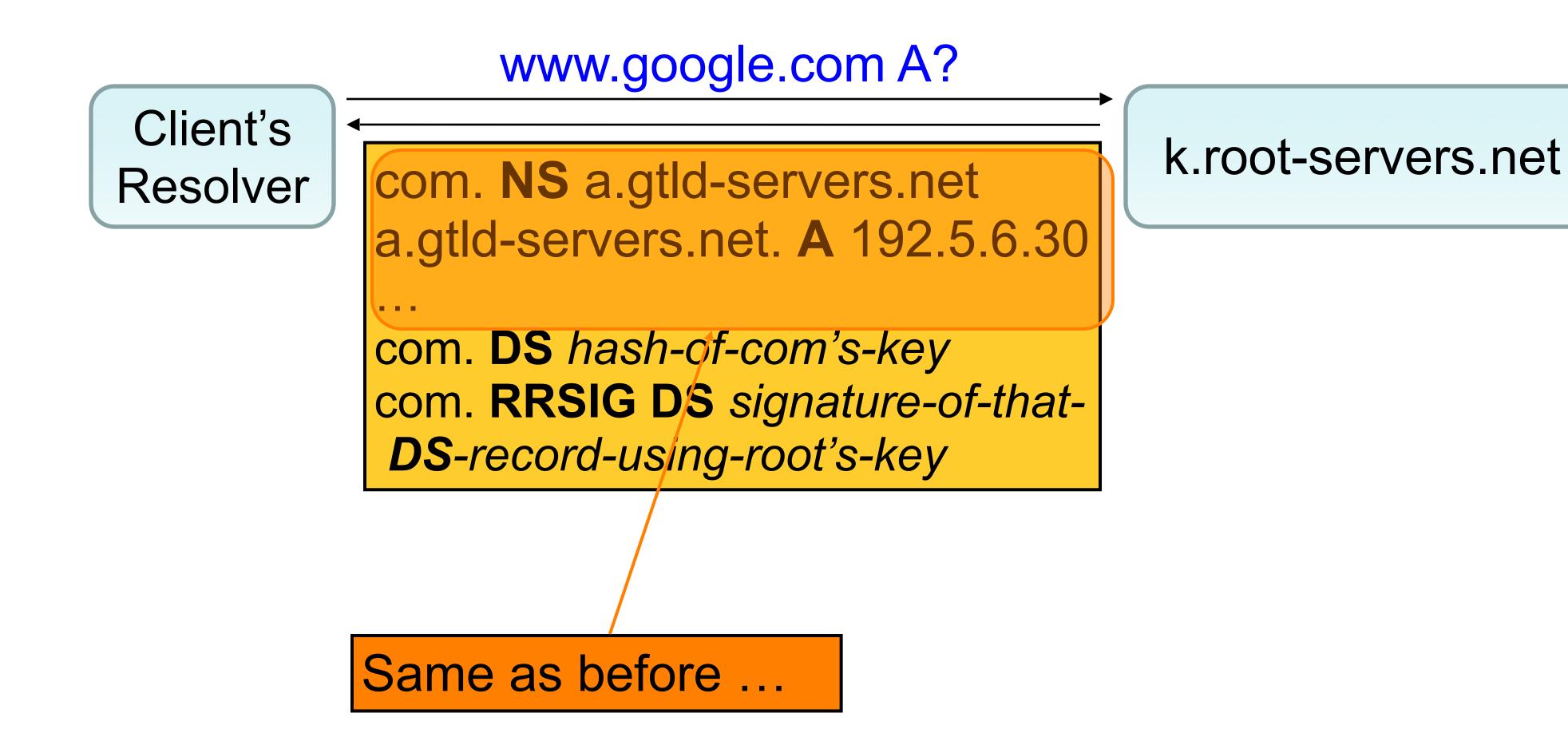
Client's Resolver

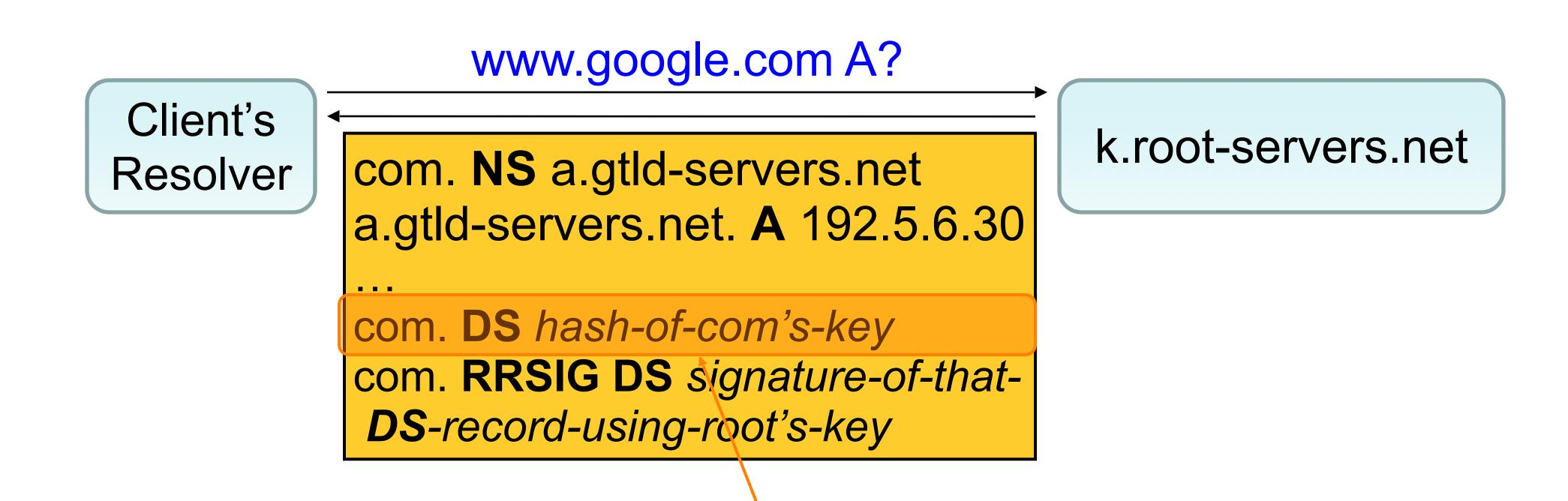
com. **NS** a.gtld-servers.net a.gtld-servers.net. **A** 192.5.6.30

. . .

com. **DS** hash-of-com's-key com. **RRSIG DS** signature-of-that-**DS**-record-using-root's-key

k.root-servers.net



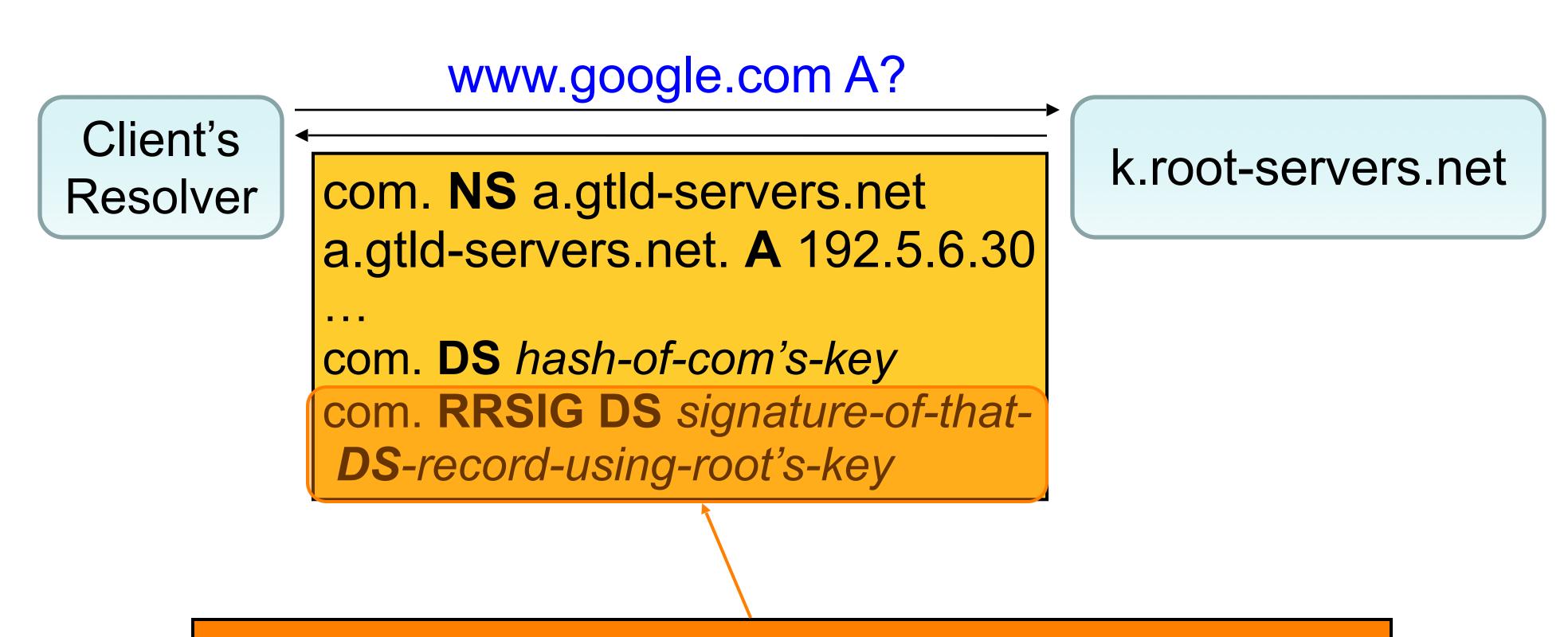


New **RR** ("Delegation Signer") tells us if we have correct copy of .com's public key (by comparing hash values)

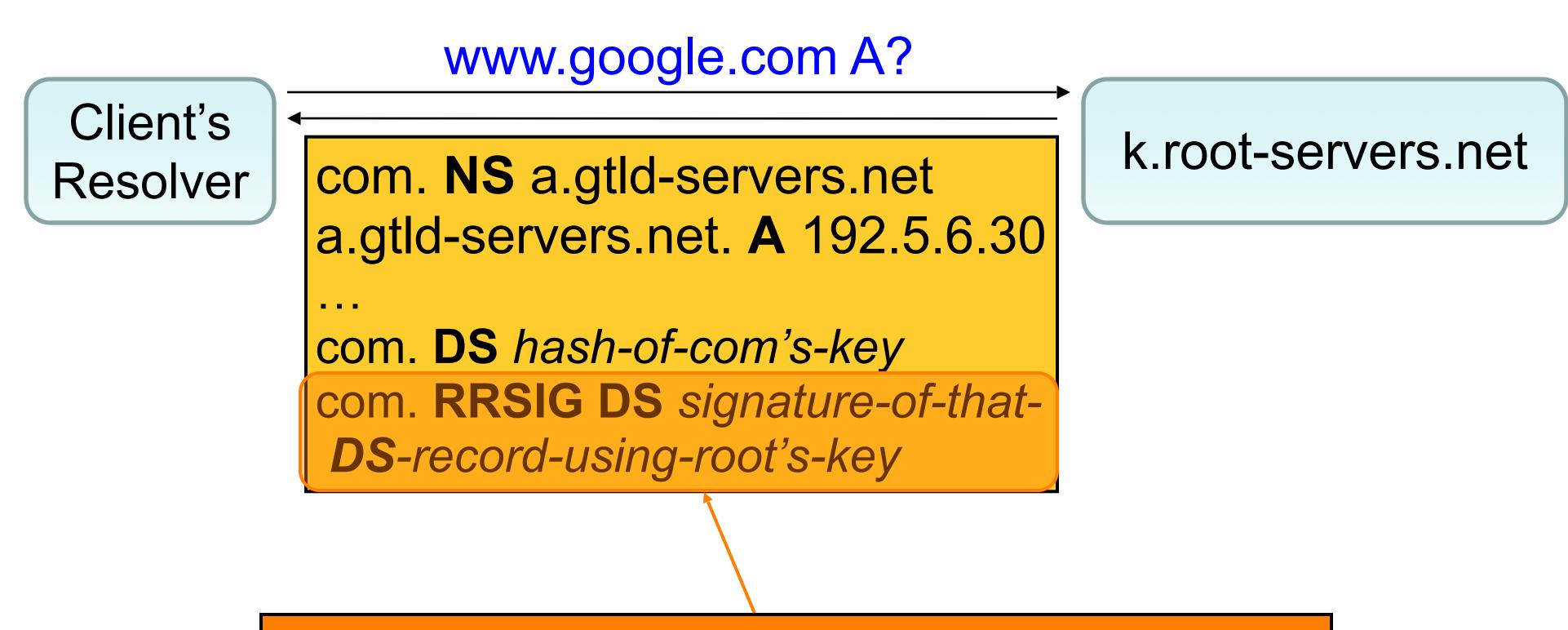


k.root-servers.net

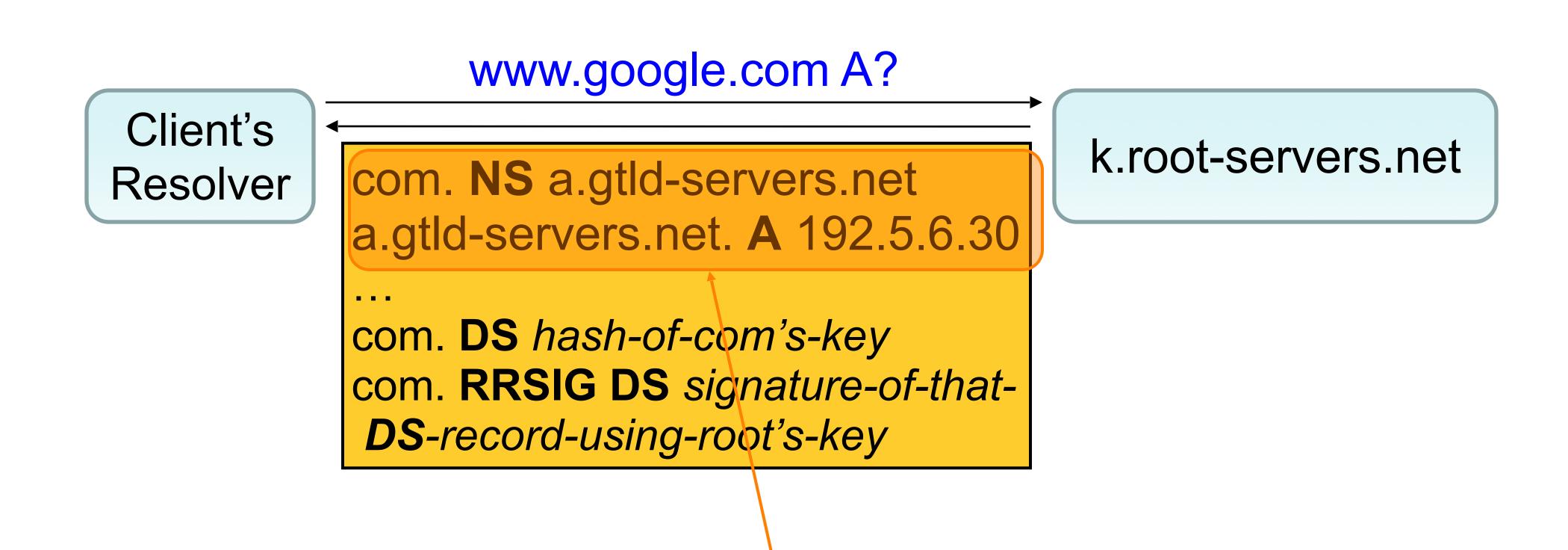
Getting .com NS's key is a bit complicated...we'll talk about in a bit. Assume we can get it for now...



Another new **RR** = <u>signature</u> of *another* **RR**. Here, this signature is of a **DS** record, signed by root's private key



Root's public key **hardwired** into resolvers. Client only proceeds with DNSSEC if it can validate the signature.



Note: there's no signature over the **NS** or **A** information! If DNS data tampered with, will find out later.



#### www.google.com A?

Client's Resolver

google.com. **NS** ns1.google.com ns1.google.com. **A** 216.239.32.10

. . .

google.com. **DS** hash-ofgoogle.com's-key google.com. **RRSIG DS** signatureof-that-**DS**-record-using-com's-key a.gtld-servers.net



#### Similar as before:

- Identify google.com's public key
- DS signed by .com NS's key



#### www.google.com A?

Client's Resolver

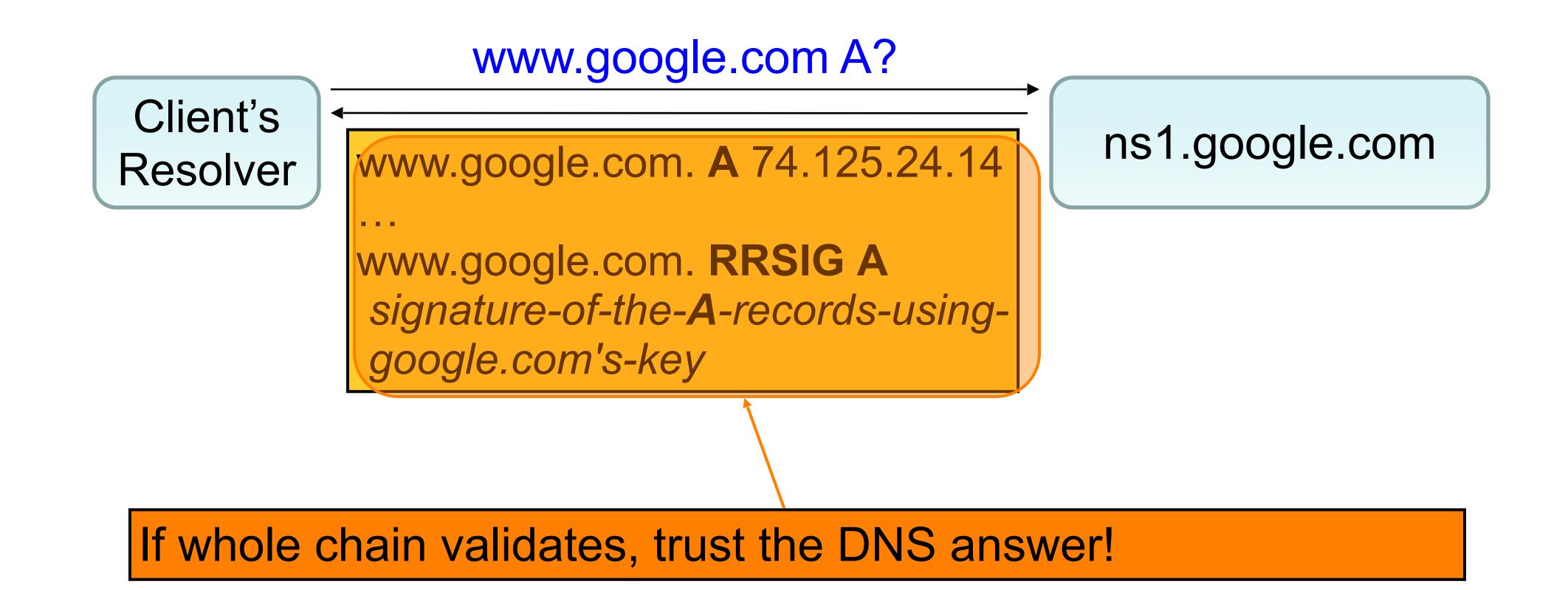
www.google.com. A 74.125.24.14

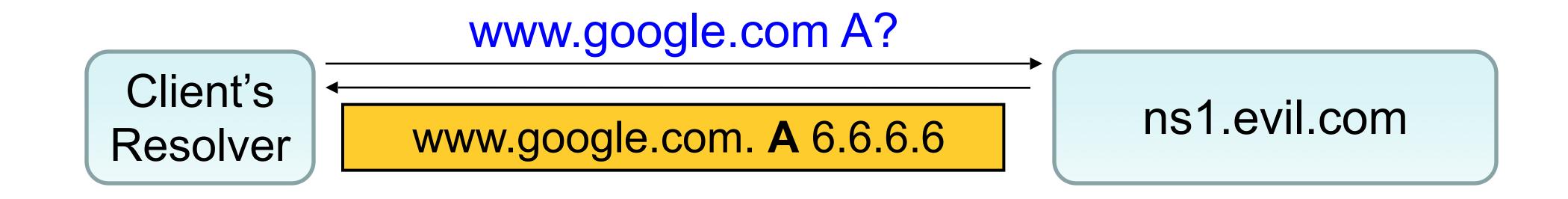
. . .

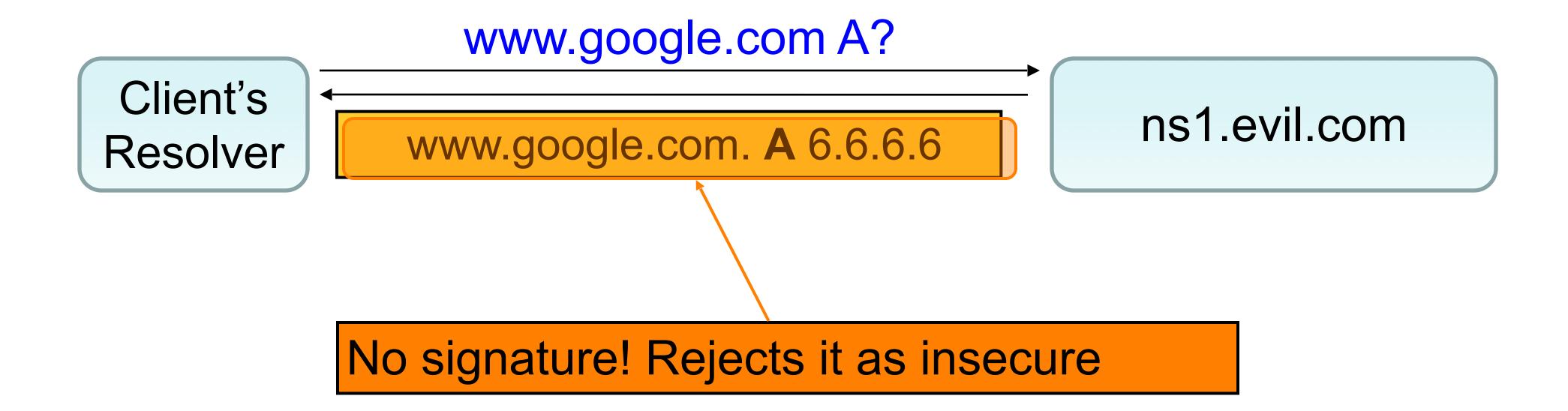
www.google.com. **RRSIG A**signature-of-the-**A**-records-usinggoogle.com's-key

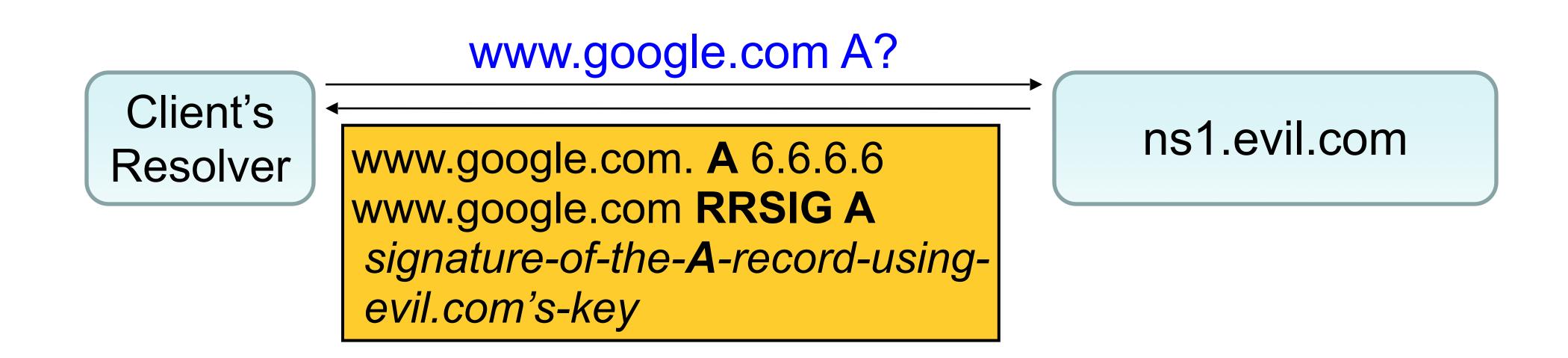
ns1.google.com

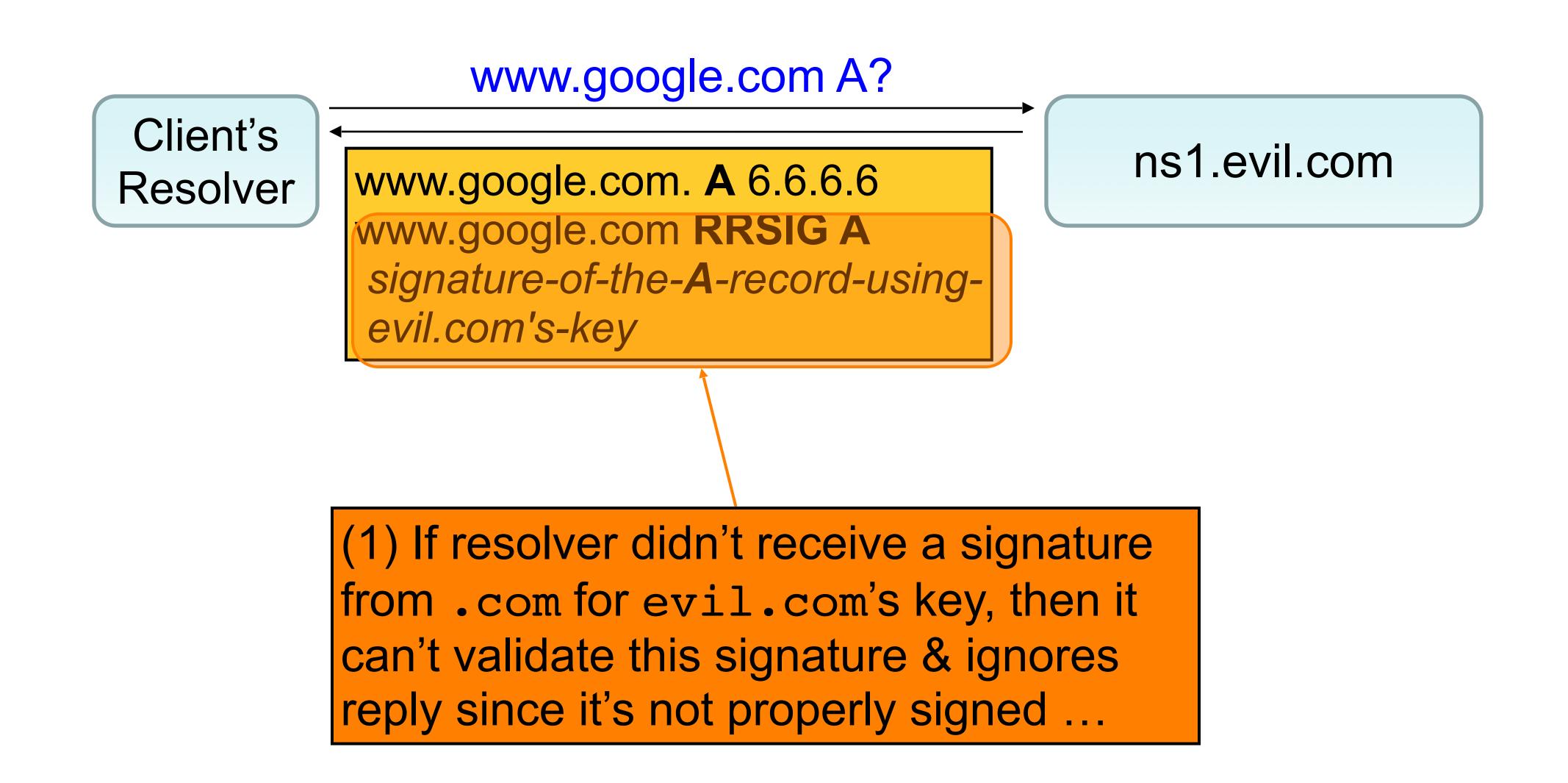


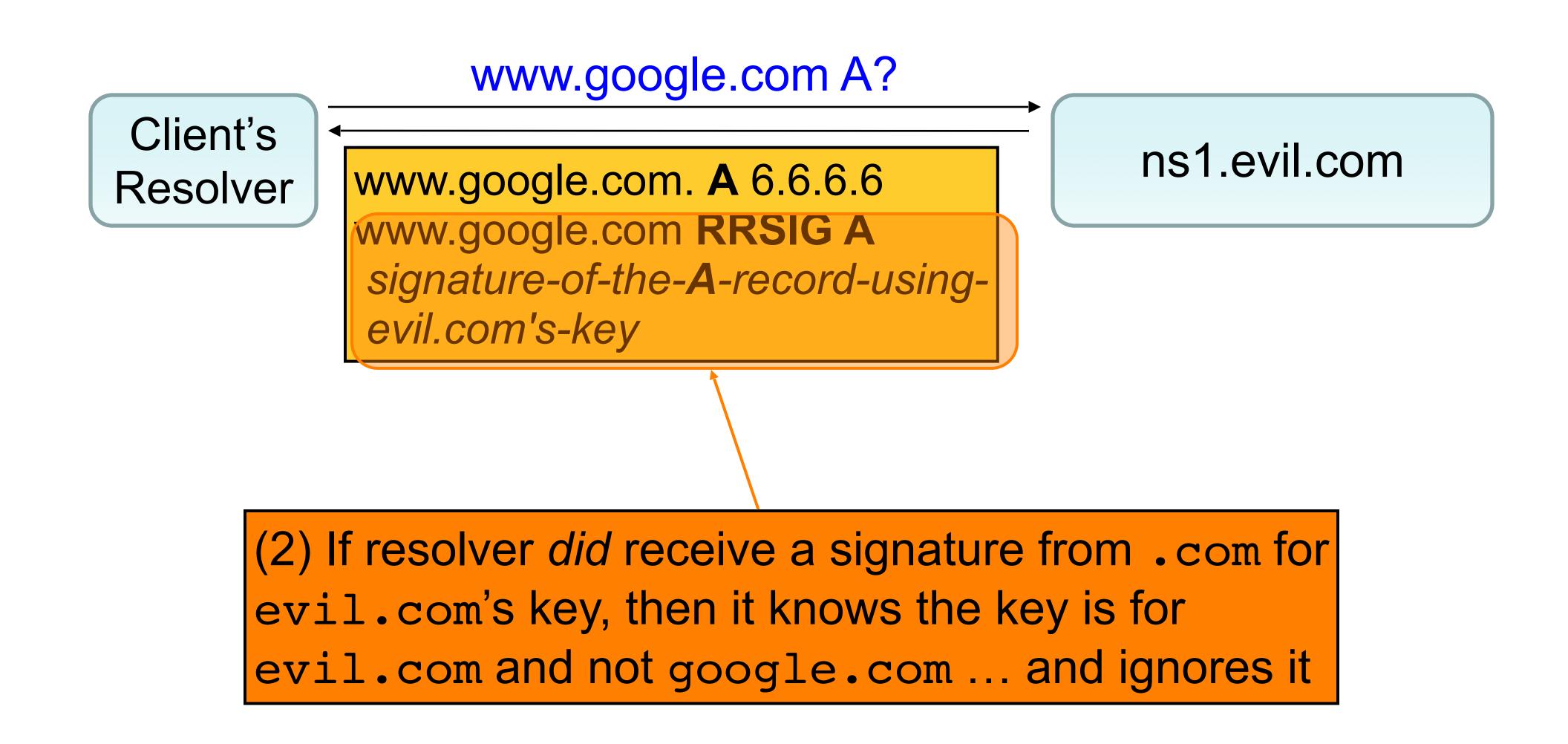


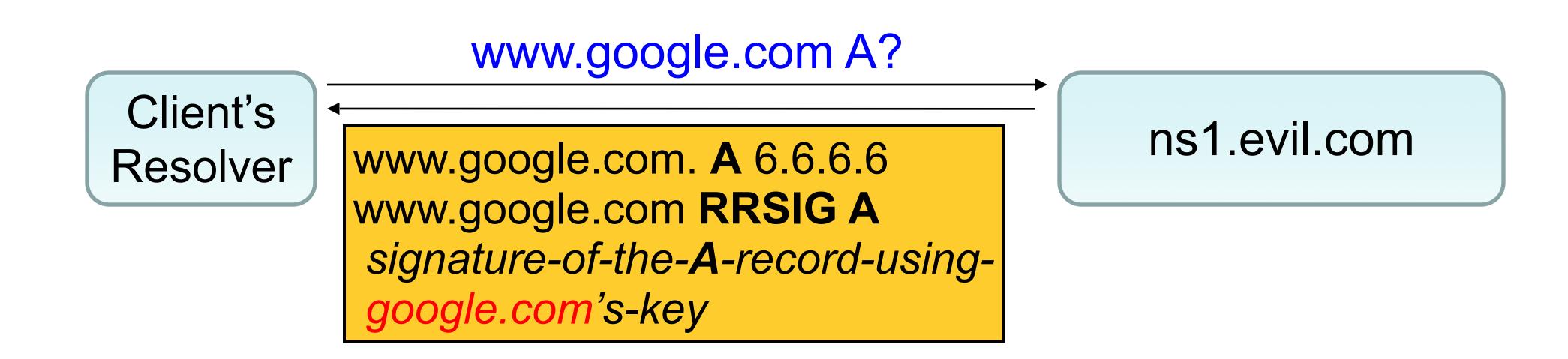


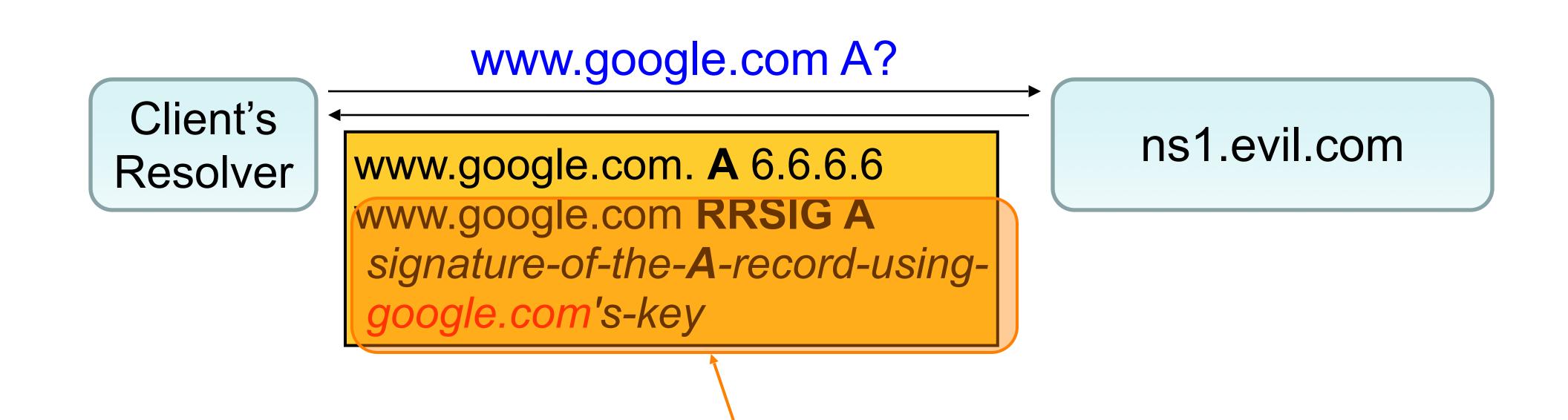












If signature **actually** comes from google.com's key, resolver will believe it ...

- ... but no such signature should exist unless either:
- (1) google.com intended to sign the RR, or
- (2) google.com's private key was compromised

## DNSSEC: Accessing keys

Client's Resolver k.root-servers.net

To get keys needed for validation, our client contacts each NS during the query asking for its keys.

Here we ask the root for its keys (one of which we already know as our **trust anchor**).

## DNSSEC: Accessing keys

#### . DNSKEY?

Client's Resolver

- . **DNSKEY** cryptogoop for root's key-signing key (KSK)
- . **DNSKEY** cryptogoop for root's zone-signing key (ZSK)
- . **DNSKEY** cryptogoop for possibly other keys

. . .

. RRSIG DNSKEY signature-ofthose-DNSKEY-records-usingroot's-KSK k.root-servers.net

## DNSSEC: Accessing keys

. DNSKEY?

Client's Resolver

. DNSKEY cryptogoop for root's key-signing key (KSK)

DNSKEY cryptogoop for root's zone-signing key (ZSK)

. **DNSKEY** cryptogoop for possibly other keys

. . .

. RRSIG DNSKEY signature-ofthose-DNSKEY-records-usingroot's-KSK k.root-servers.net

Each **DNSKEY** is a public key plus a description of the algorithms it's associated with (e.g., RSA+SHA256)

. DNSKEY?

Client's Resolver

. DNSKEY cryptogoop for root's key-signing key (KSK)

- . **DNSKEY** cryptogoop for root's zone-signing key (ZSK)
- . **DNSKEY** cryptogoop for possibly other keys

. . .

. RRSIG DNSKEY signature-ofthose-DNSKEY-records-usingroot's-KSK k.root-servers.net

The KSK is used to sign all of the DNSKEY entries in the zone.

. DNSKEY?

Client's Resolver

DNSKEY cryptogoop for root's key-signing key (KSK)

- . **DNSKEY** cryptogoop for root's zone-signing key (ZSK)
- . **DNSKEY** cryptogoop for possibly other keys

. . .

. RRSIG DNSKEY signature-ofthose-DNSKEY-records-usingroot's-KSK k.root-servers.net

The client has a hash of the root's **KSK** hardwired into its config as a trust anchor.

. DNSKEY?

Client's Resolver

. DNSKEY cryptogoop for root's key-signing key (KSK)

- . **DNSKEY** cryptogoop for root's zone-signing key (ZSK)
- . **DNSKEY** cryptogoop for possibly other keys

. . .

RRSIG DNSKEY signature-ofthose-DNSKEY-records-usingroot's-KSK k.root-servers.net

For everything below the root (e.g., .com and google.com) we get a hash of the KSK via a **DS** record, as shown earlier, so we can tell if we get the right KSK in a **DNSKEY** entry.

#### . DNSKEY?

Client's Resolver

. DNSKEY cryptogoop for root's key-signing key (KSK)

. DNSKEY cryptogoop for root's zone-signing key (ZSK)

. **DNSKEY** cryptogoop for possibly other keys

. . .

. RRSIG DNSKEY signature-ofthose-DNSKEY-records-usingroot's-KSK k.root-servers.net

The **ZSK** is used for signing all of the other RRSIG entries in the zone, including DS records for subzones.

(E.g., .com signs its **DS** record for google.com using .com's **ZSK**)

#### . DNSKEY?

Client's Resolver

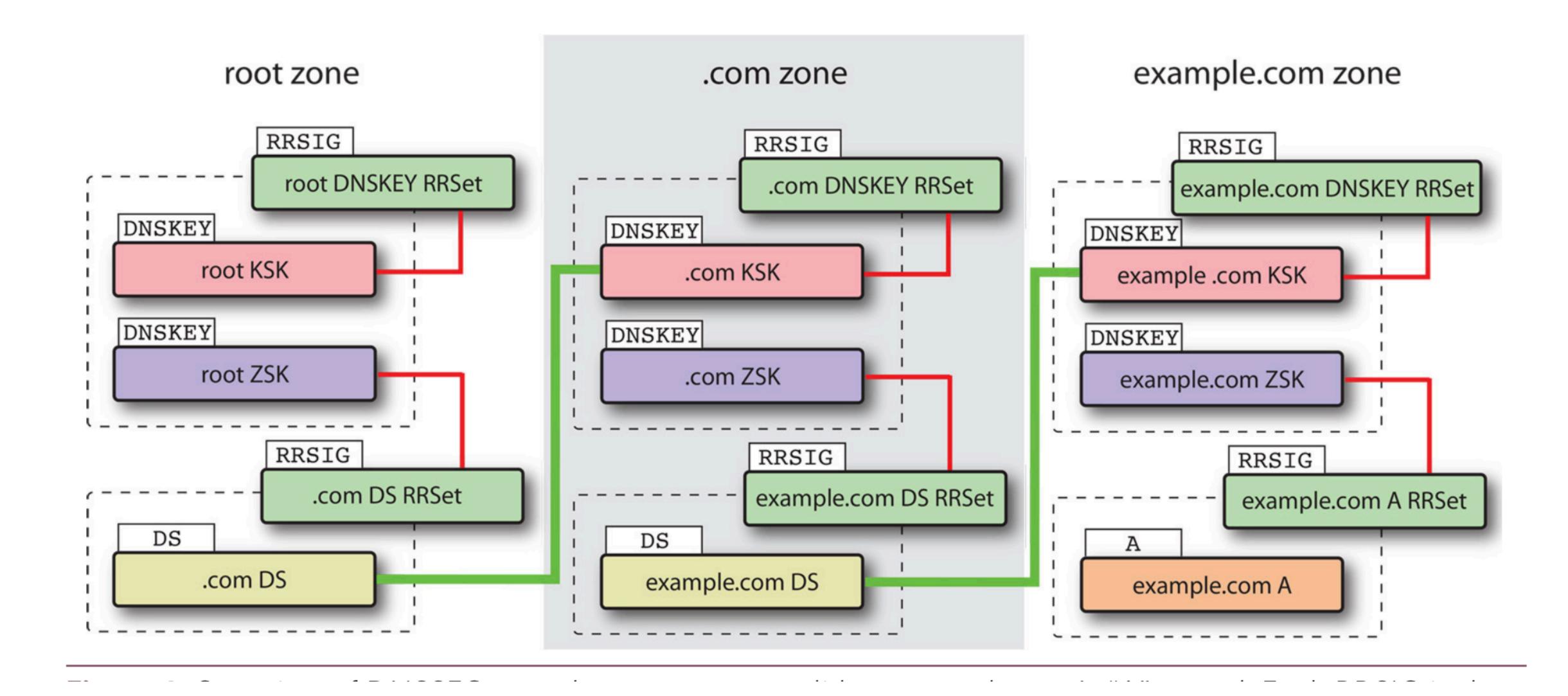
- . DNSKEY cryptogoop for root's key-signing key (KSK)
- . **DNSKEY** cryptogoop for root's zone-signing key (ZSK)
- . **DNSKEY** cryptogoop for possibly other keys

. . .

. RRSIG DNSKEY signature-ofthose-DNSKEY-records-usingroot's-KSK k.root-servers.net

Having separate key-signing-keys vs. zone-signing-keys allows a zone to change its **ZSK** without needing to get its parent to re-sign, since parent only signs the **KSK**. Enables frequent *key rollover*.

#### **End-to-end look**



## Issues With DNSSEC?

- Issue #1: Replies are Big
  - DoS amplification
  - Increased latency on low-capacity links
  - Headaches w/ older libraries that assume replies < 512B</li>

## Issues With DNSSEC?

- Issue #1: Replies are Big
  - DoS amplification
  - Increased latency on low-capacity links
  - Headaches w/ older libraries that assume replies < 512B</li>
- Issue #2: Partial deployment
  - What do you do with unsigned/unvalidated results?
  - If you trust them, weakens incentive to upgrade
  - If you don't trust them, a whole lot of things break

## Issues With DNSSEC?

- Issue #1: Replies are Big
  - DoS amplification
  - Increased latency on low-capacity links
  - Headaches w/ older libraries that assume replies < 512B</li>
- Issue #2: Partial deployment
  - What do you do with unsigned/unvalidated results?
  - If you trust them, weakens incentive to upgrade
  - If you don't trust them, a whole lot of things break
- Issue #3: Management headaches
  - What happens if when updating your site's keys you make a mistake?
  - Suddenly your Entire Site Breaks

- Issue #4: Negative results ("no such name")
  - What record does the nameserver sign?
  - Has to be a record specific to the queried domain (otherwise a generic "no such name" response could be replayed by the attacker).
  - NS could dynamically sign a record saying "gabluph.google.com doesn't exist".
    - Computationally expensive, allows DoS vulnerability w/ bogus requests

- Issue #4: Negative results ("no such name")
  - What record does the nameserver sign?
  - Instead, sign (off-line) records about alphabetical order of names (NSEC extension to DNSSEC)
    - Ex: NS signs "gabby.google.com exists, followed by gabrunk.google.com"
    - Client can see that gabluph.google.com can't exist
    - BUT: now attacker can enumerate all domains that might exist (could leak sensitive/private information)

- Issue #4: Negative results ("no such name")
  - What record does the nameserver sign?
  - NSEC3: sign alphabetical order of domain hashes
    - Ex: say gabluph.google.com hashes (under some algorithm) to d3f...ad
    - Ex: NS signs "d1a...fa exists, followed by e13...a3"
    - Client can see that gabluph.google.com can't exist b/c it's hash doesn't exist
    - Hashes, rather than domains, are revealed.
    - But...often times not hard to reverse the hash by brute-force guessing domains. There are further proposed variants on NSEC3, but involve heavier-weight crypto.

- Issue #5: Who do you really trust?
  - For your laptop (say), who does all the "grunt work" of fetching keys & validating DNSSEC signatures?
- Convenient answer: your laptop's local resolver
  - ... which you acquire via DHCP in your local coffeeshop
  - I.e., exactly the most-feared potentially untrustworthy part of the DNS resolution process!
- Alternatives?
  - ⇒ Your laptop needs to do all the validation work itself :-(

## Summary of DNSSEC

- DNSSEC: provides object security for DNS results
  - Just integrity & authentication, not confidentiality
  - No client/server setup "dialog"
  - Tailored to be caching-friendly
  - Underlying security dependent on trust in Root Name Server's key ...
  - ... plus support provided by every level of DNS hierarchy from Root to final name server... and local resolver!

## **DNS Confidentiality**

- Recently, increased interest in DNS confidentiality (beyond integrity/authenticity, as provided by DNSSEC)
- Two related solutions recently:
  - **DNS over TLS (DoT):** client and a public resolver (e.g., Google or Cloudflare DNS) establish a *long-running* TLS connection (using TCP on a standardized port) and do DNS queries/responses via that TLS connection
  - DNS over HTTPS (DoH): similar as DoT but a client and a public resolver do DNS through an HTTPS connection (then DNS traffic looks like just normal web traffic)

# **DNS Confidentiality**

- DoT/DoH Overhead: Performance overhead is non-trivial but limited, as a single TLS or HTTPS connection is established and reused for all DNS traffic (handshake cost is amortized over time, although HTTPS/TLS/TCP all still introduce some overhead)
- **DoT/DoH Privacy:** Provides confidentiality (and integrity + authentication) between client and public resolver (e.g., Google or Cloudflare DNS). Local ISP can't snoop on client's DNS.
  - But this just shifts trust from local ISP to public resolver (e.g., Google/Cloudflare)
  - In theory, DoT/DoH can be also used b/w public resolver and NSes, but not done in practice (and not expected to be done either, due to scaling issues).