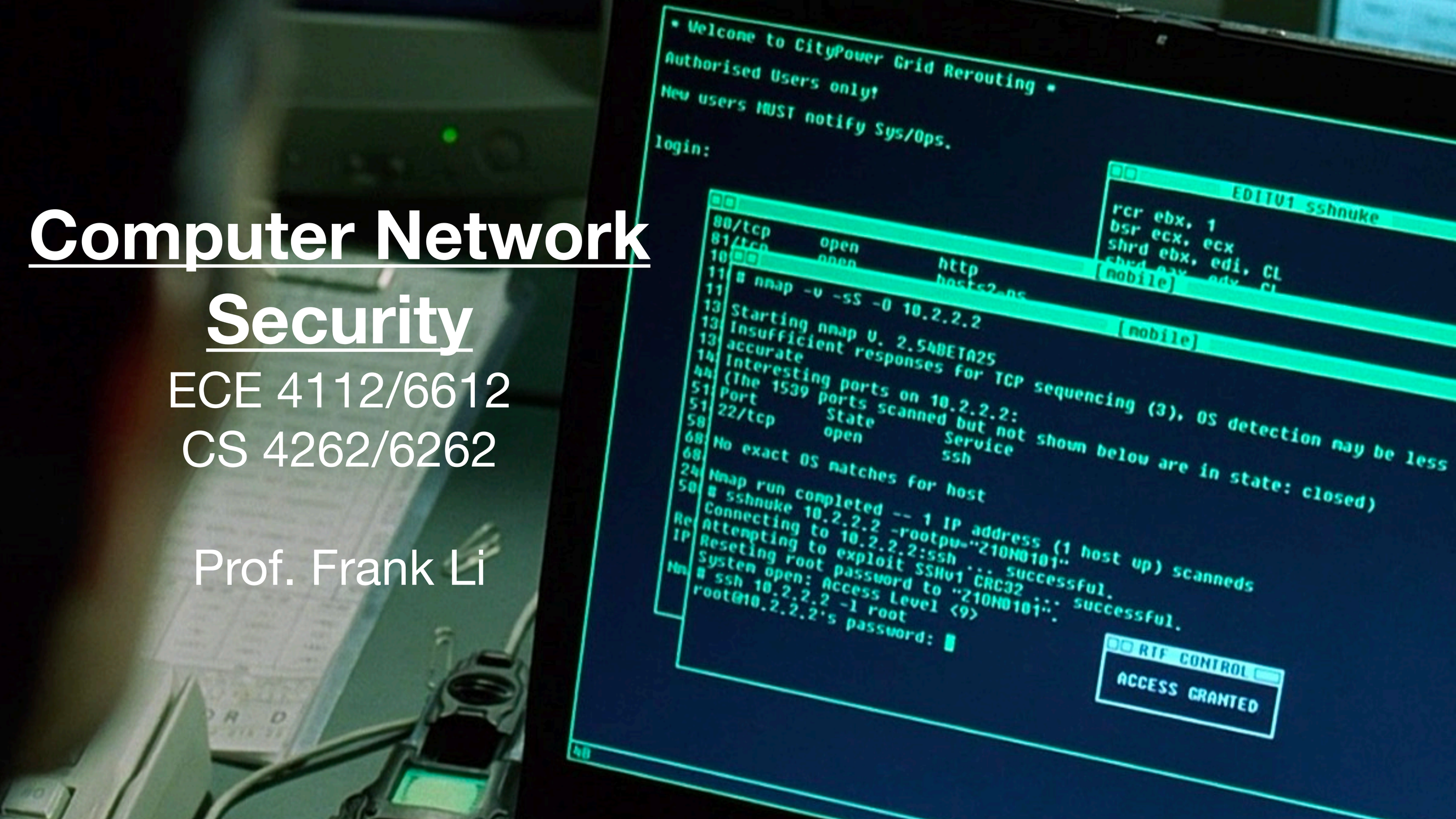
Computer Network Security

ECE 4112/6612
CS 4262/6262

Prof. Frank Li



Logistics

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

Quiz 1 regrades due tonight (on Piazza)

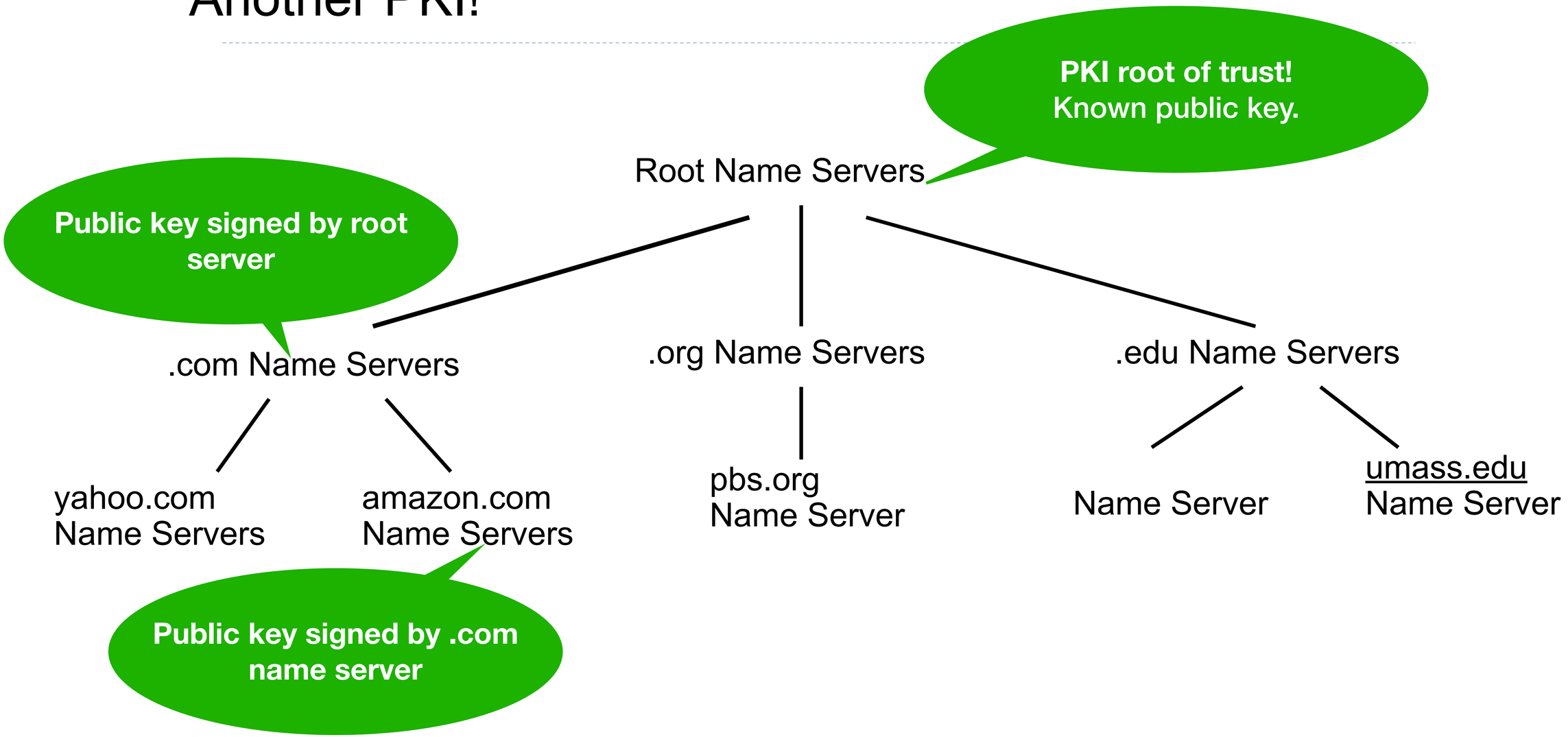
Project proposal comments provided

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!

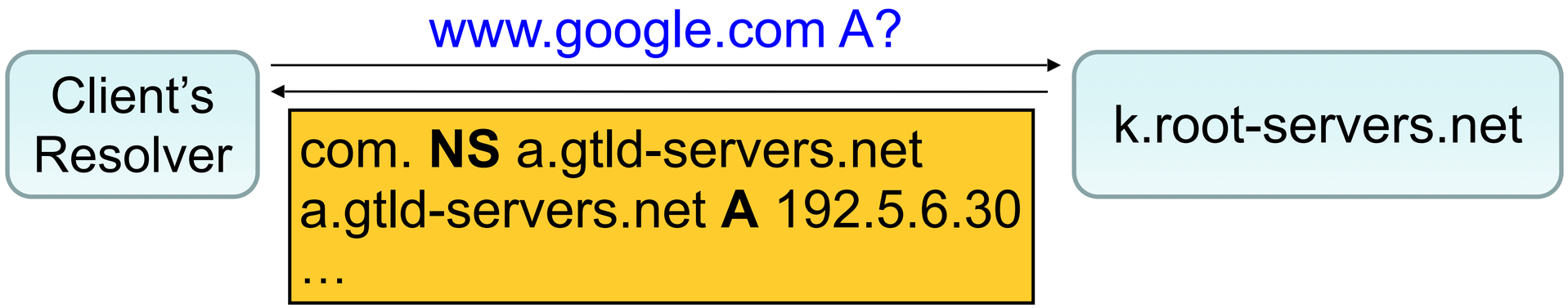

PKI root of trust!
Known public key.
Root Name Servers
Public key signed by root server
.com Name Servers
.org Name Servers
.edu Name Servers
yahoo.com
Name Servers
amazon.com
Name Servers
pbs.org
Name Server
Name Server
umass.edu
Name Server
Public key signed by .com name server

Operation of DNSSEC

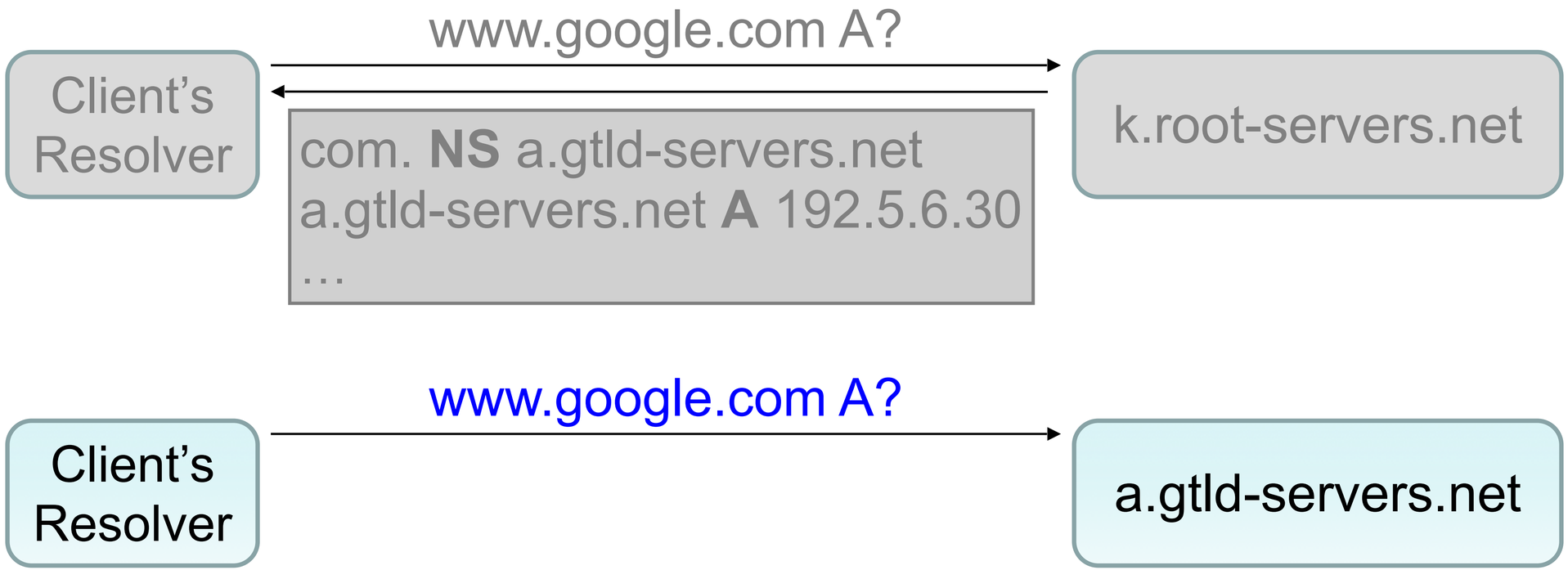
- DNSSEC = standardized DNS security extensions currently being deployed
- As resolver queries root NS down to authoritative NS, at each level it gets signed statements regarding the key(s) used by the next level
  - Builds a chain of trusted keys
  - Resolver has root's key wired into it
- Final answer 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*

Ordinary DNS:

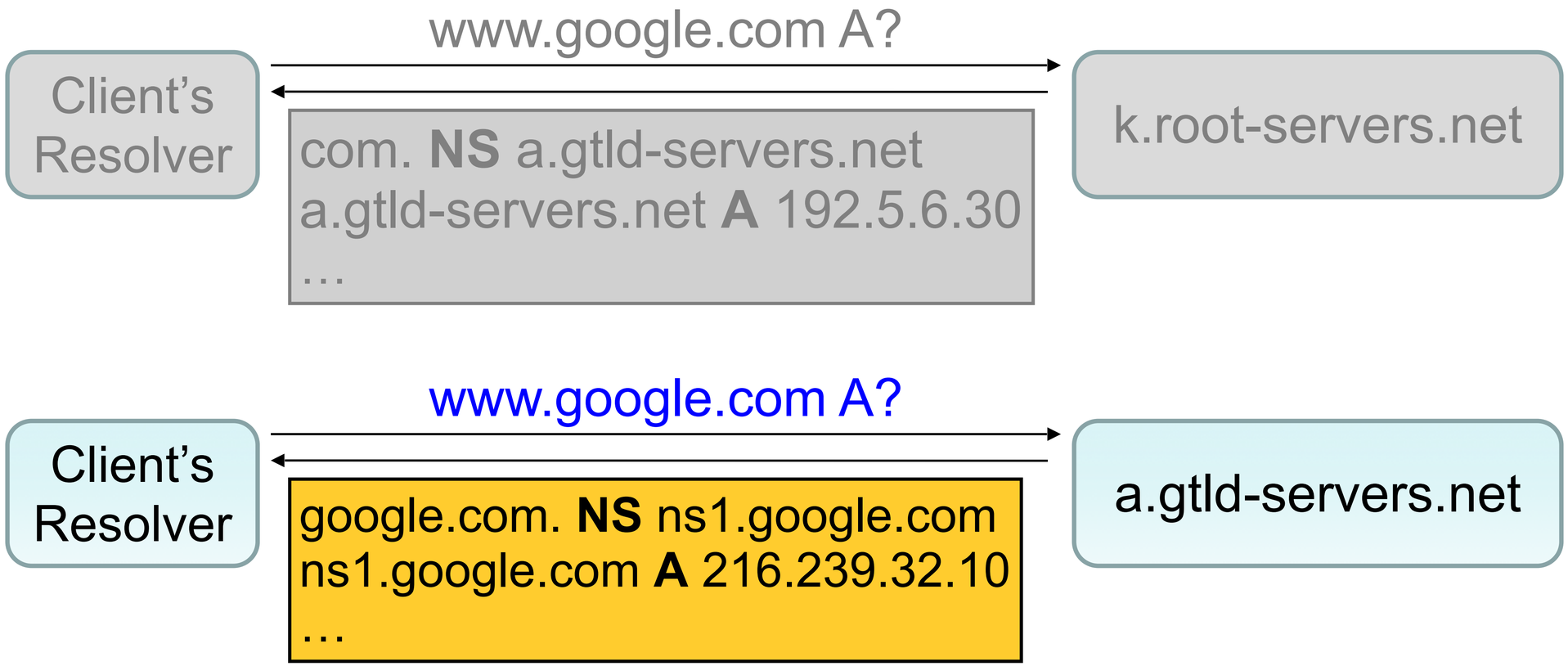
Ordinary DNS:



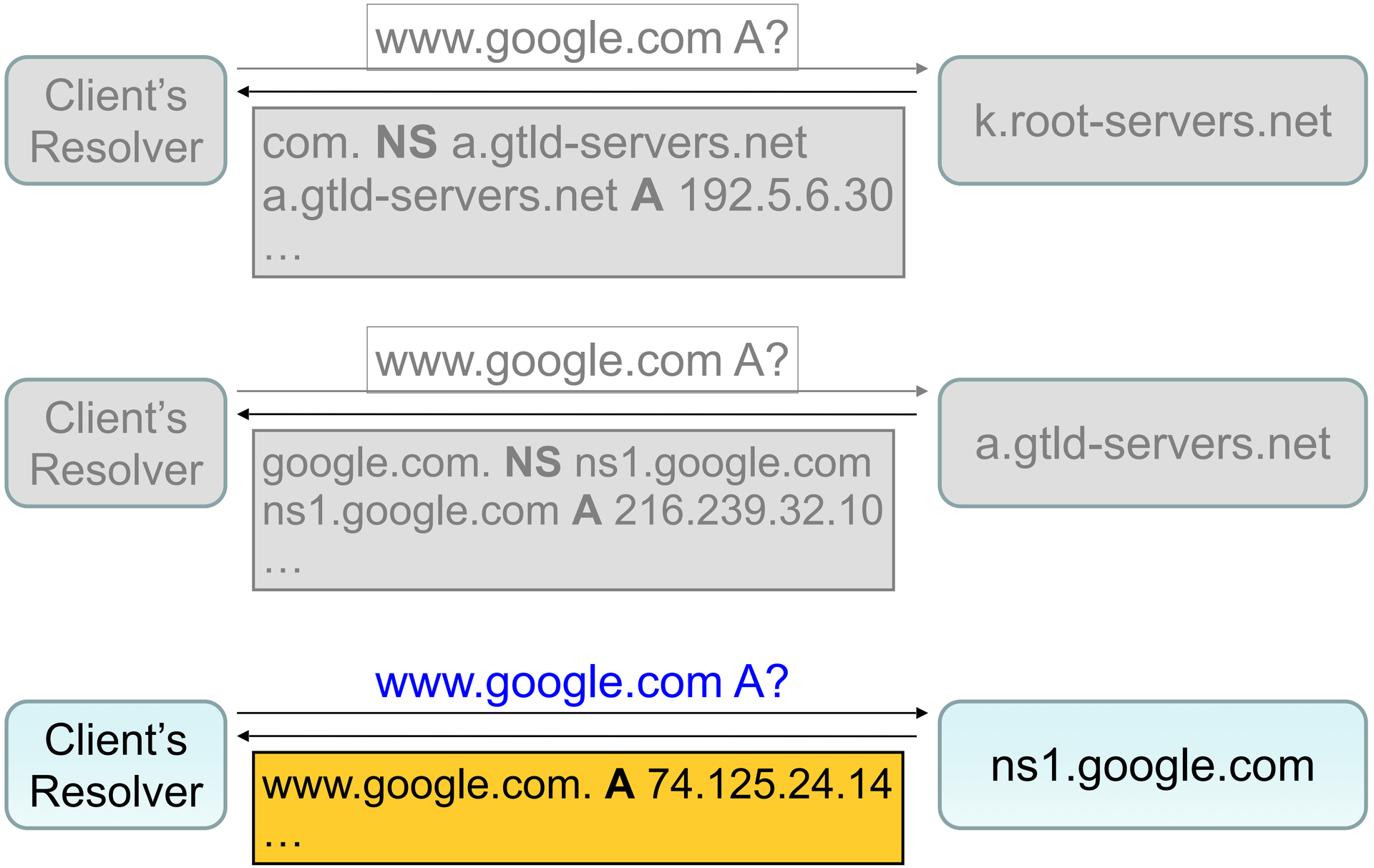
Ordinary DNS:



Ordinary DNS:

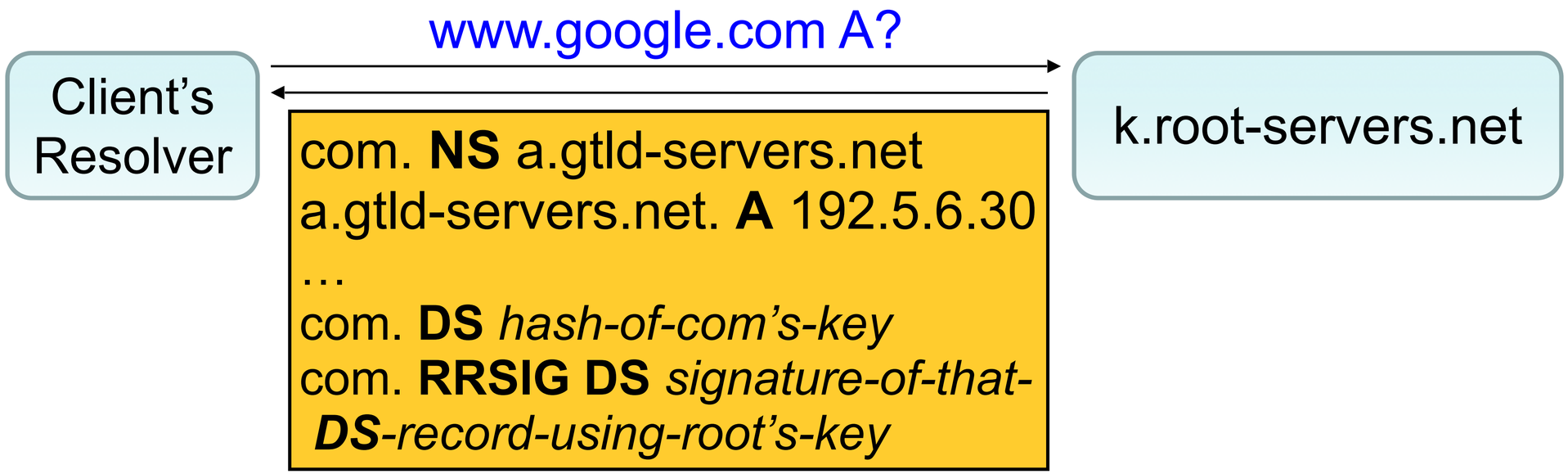


Ordinary DNS:

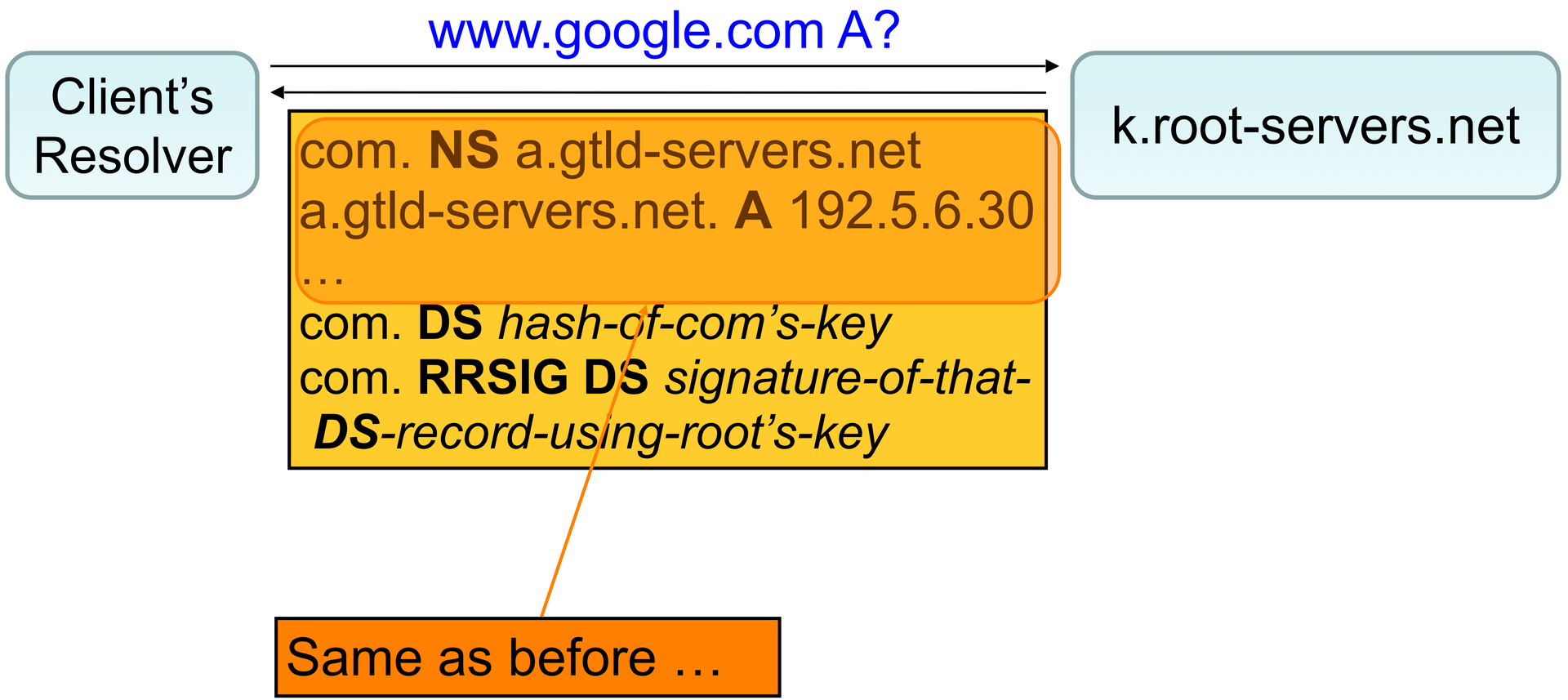


Ordinary DNS:

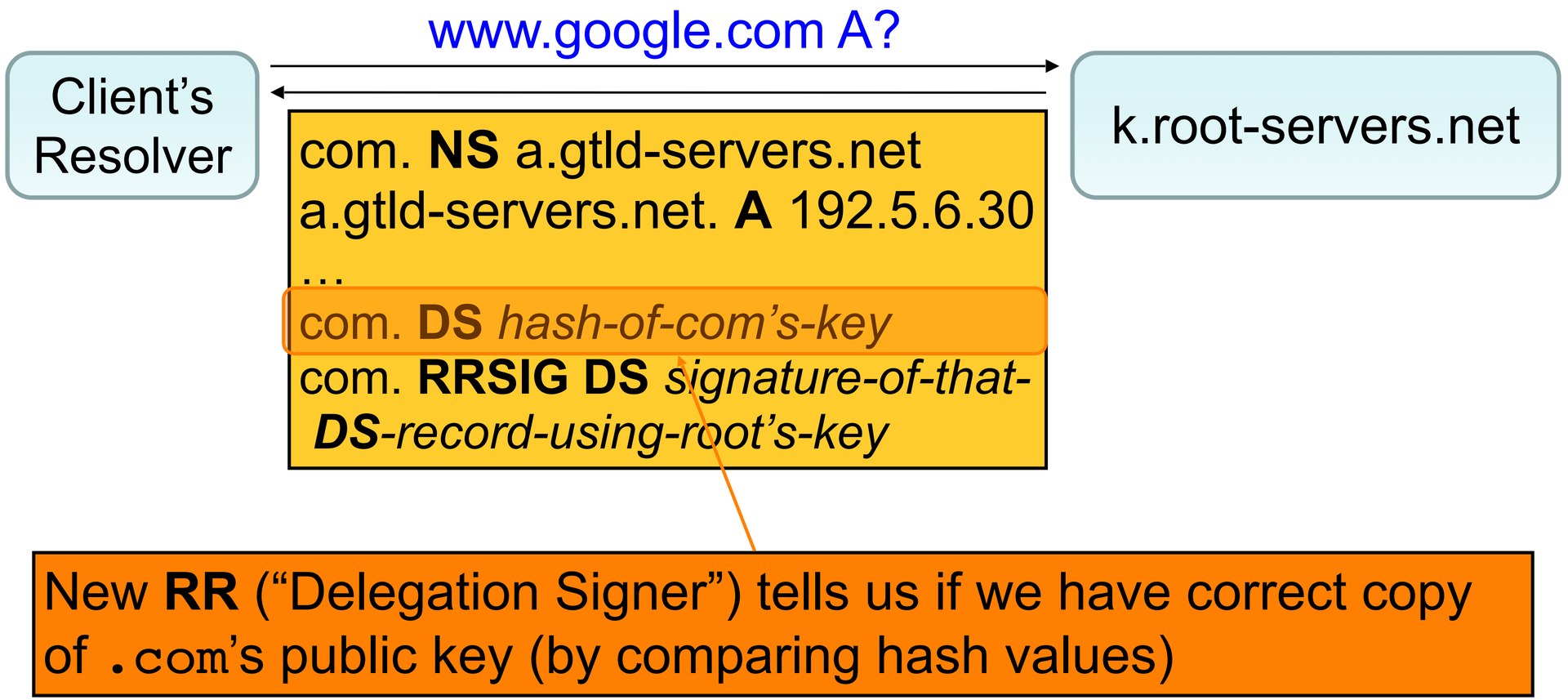
DNSSEC (with simplifications):



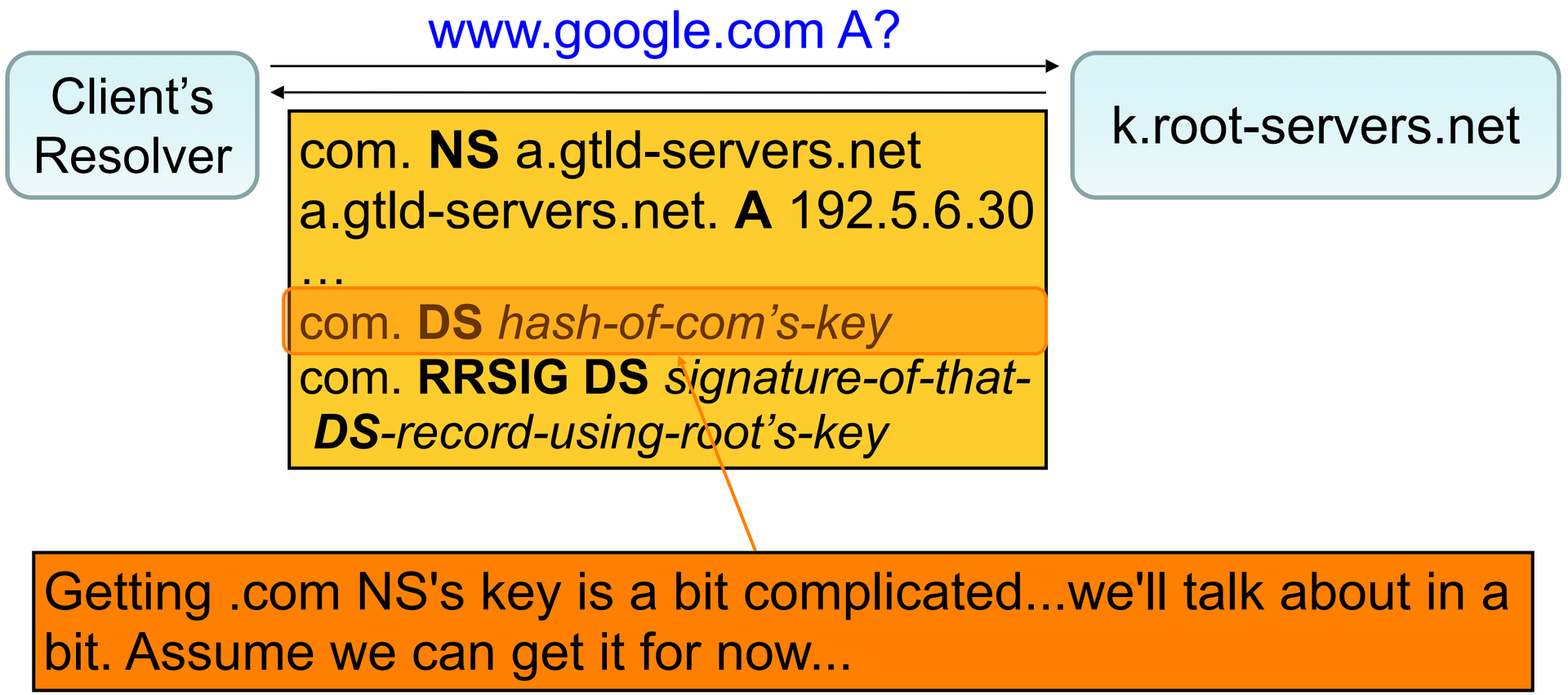
DNSSEC (with simplifications):



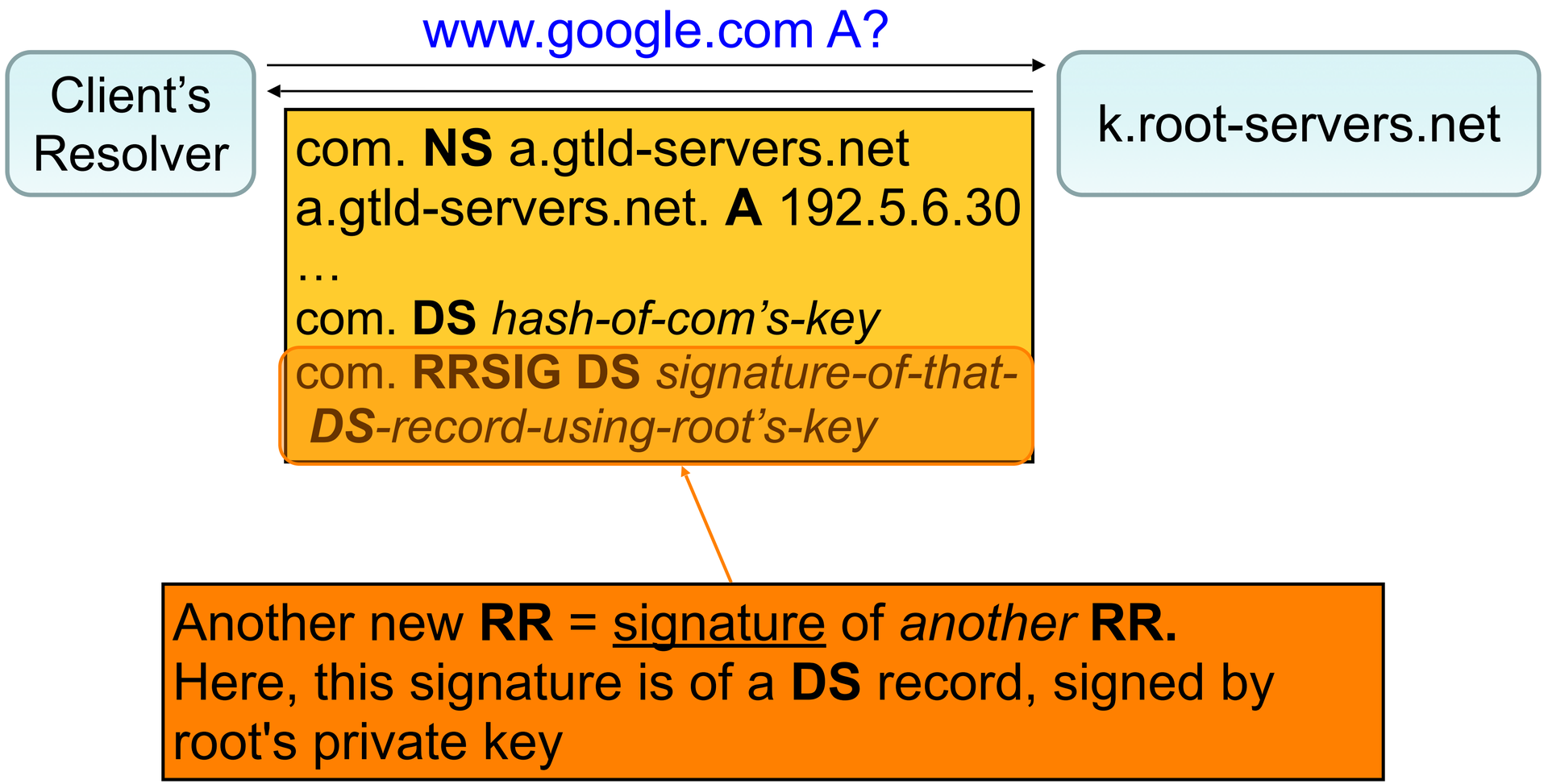
DNSSEC (with simplifications):



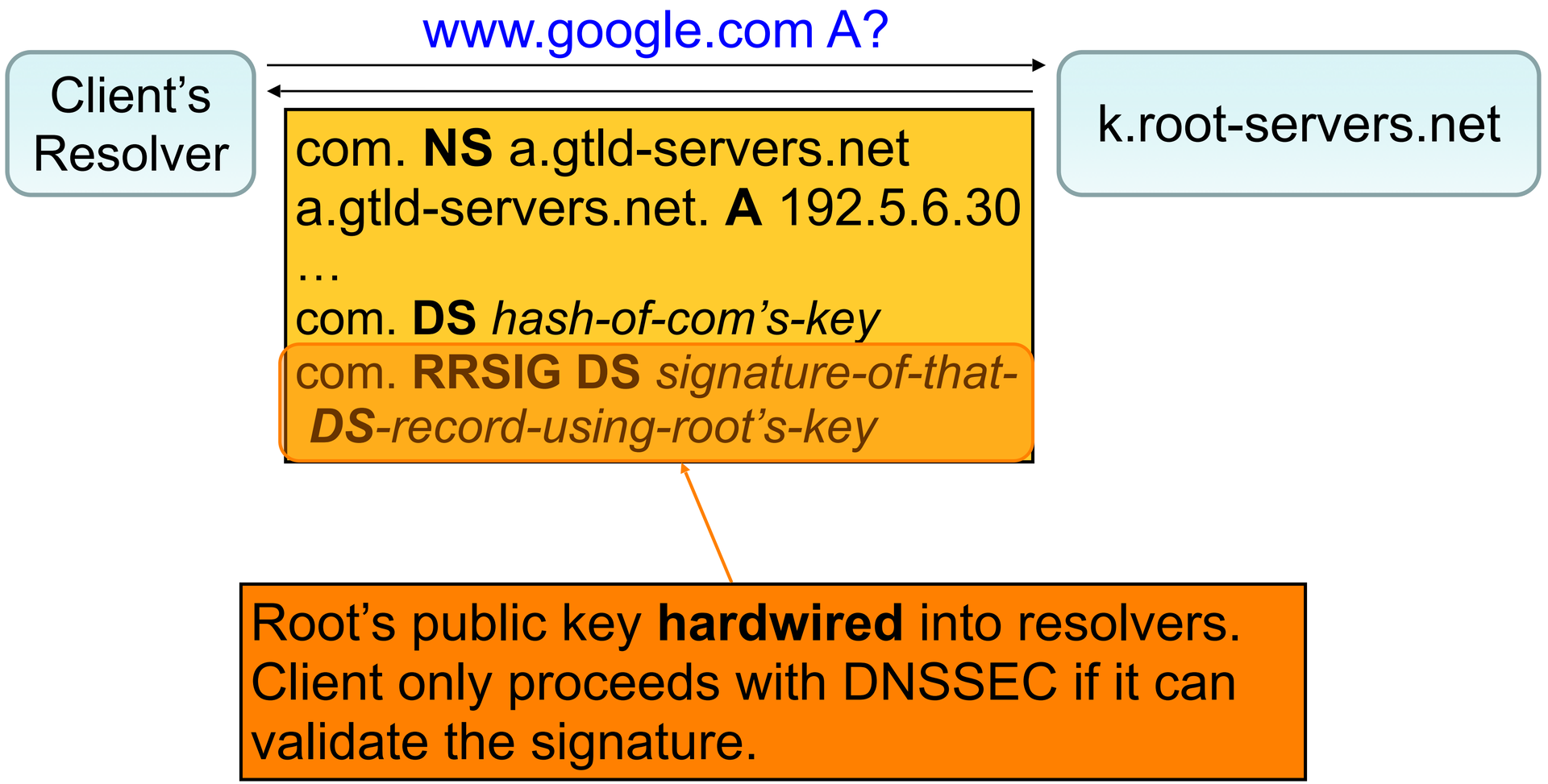
DNSSEC (with simplifications):



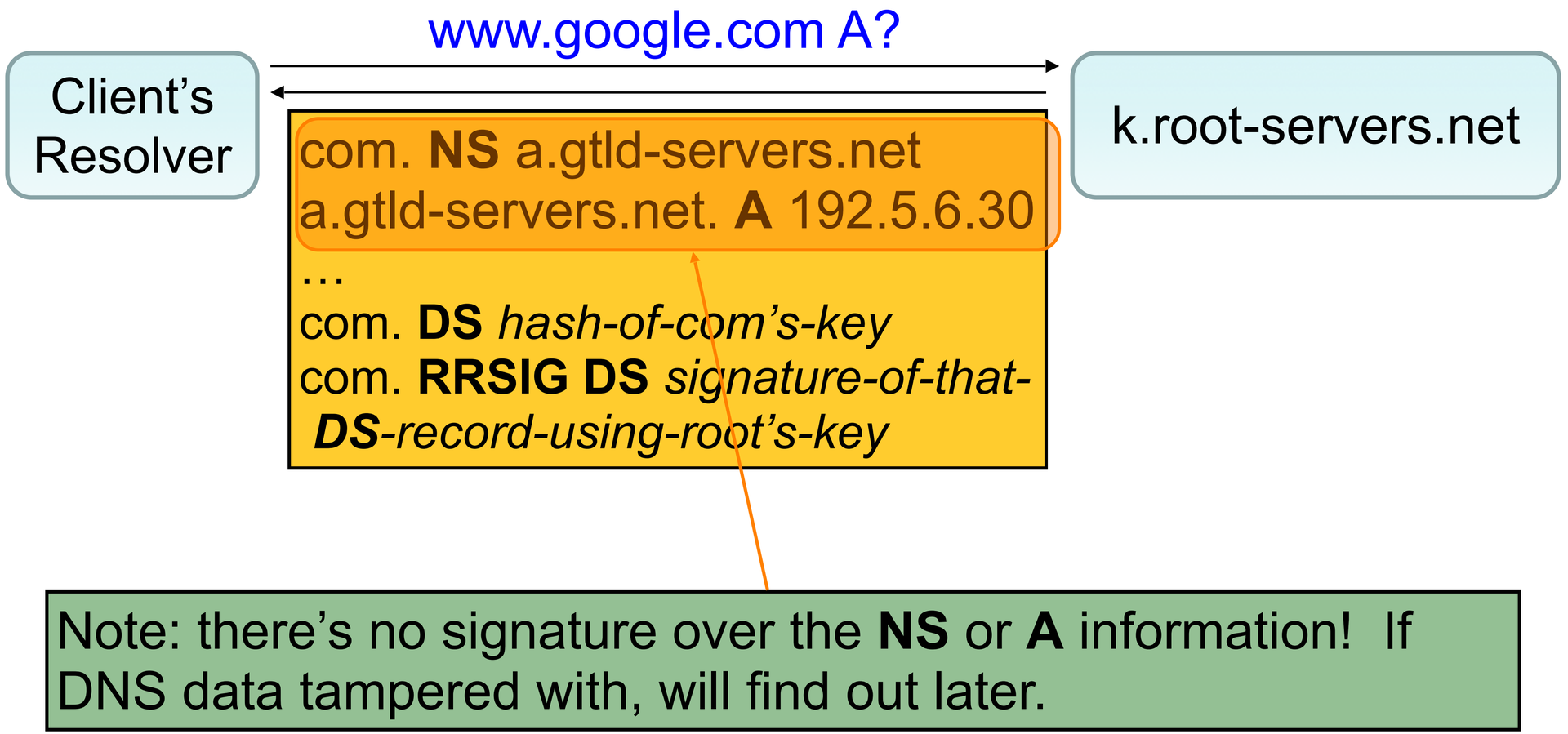
DNSSEC (with simplifications):



DNSSEC (with simplifications):

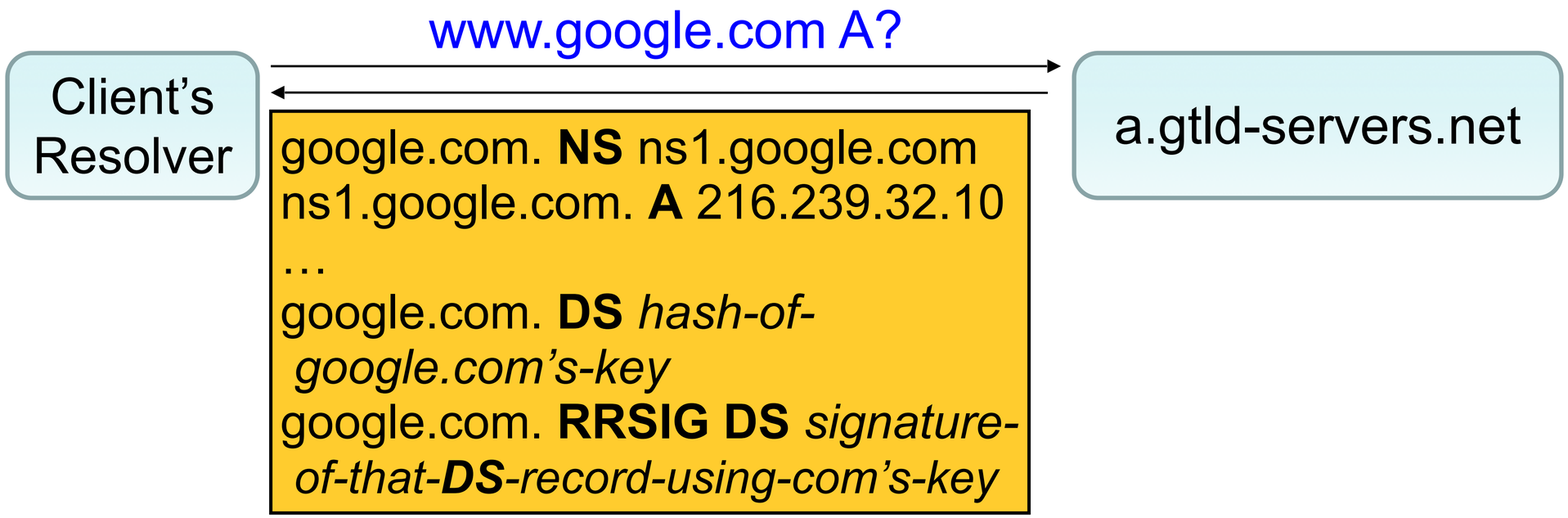


DNSSEC (with simplifications):

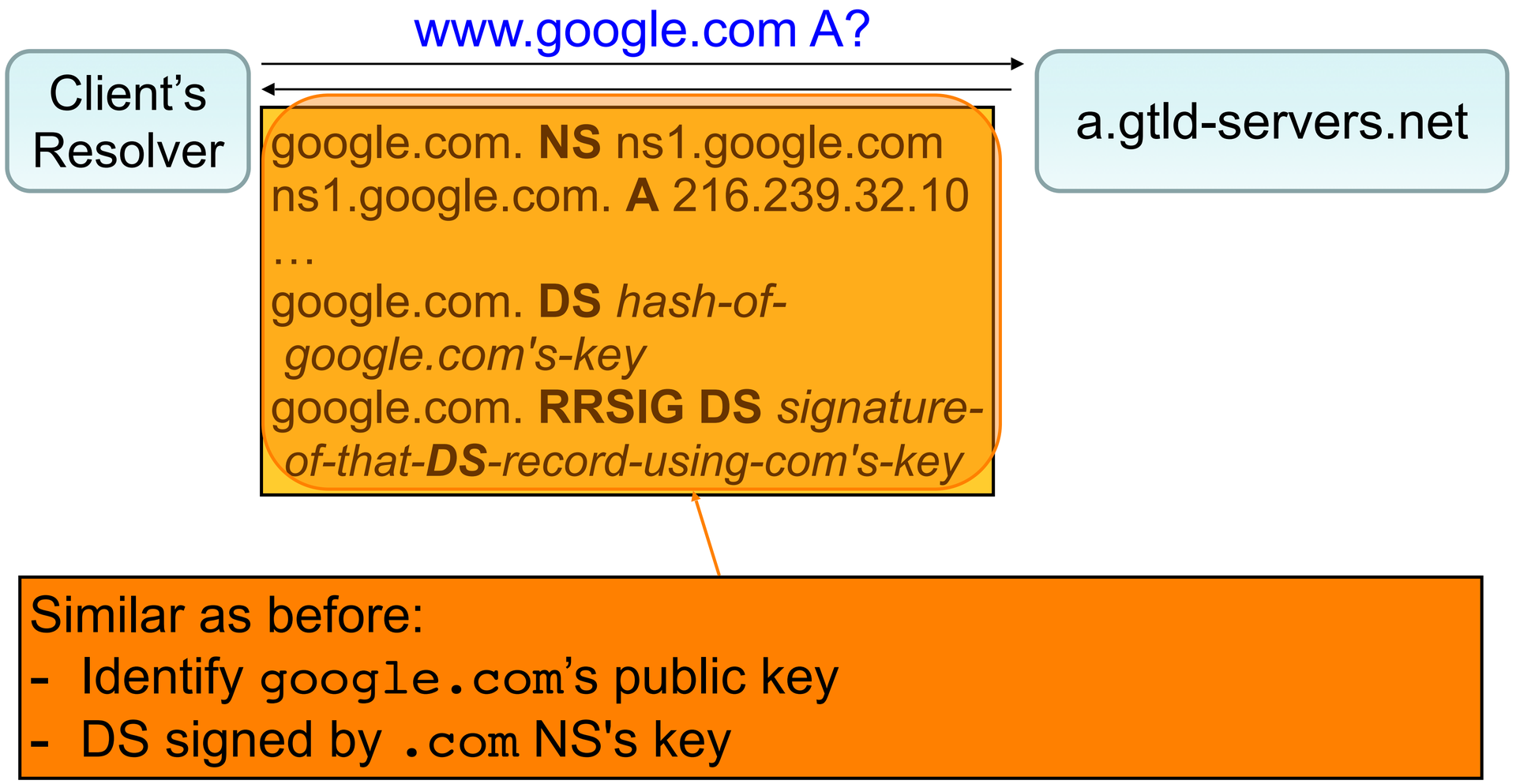


DNSSEC (with simplifications):

DNSSEC (with simplifications):

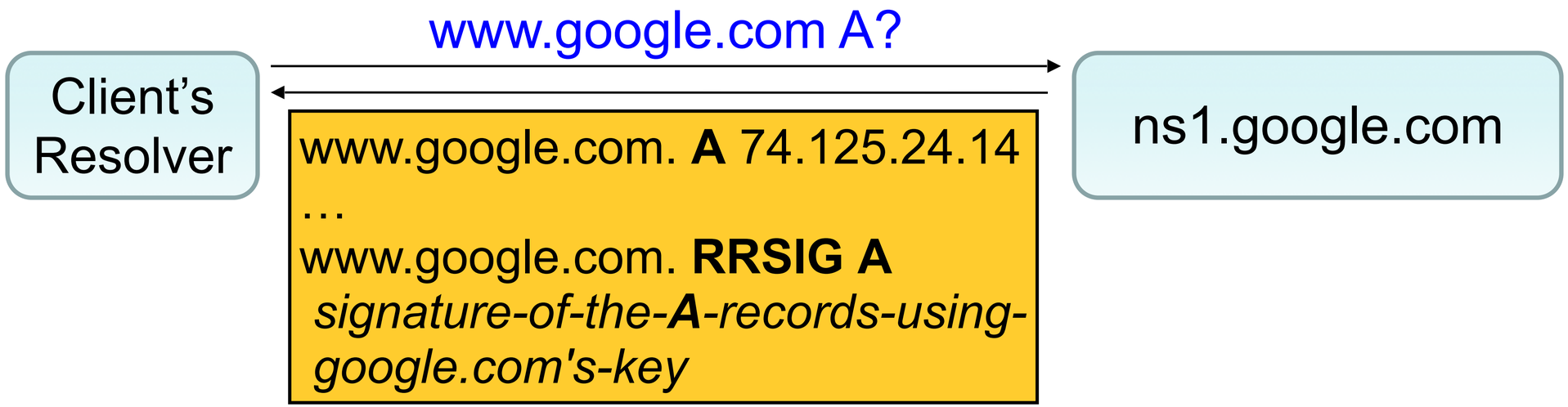


DNSSEC (with simplifications):

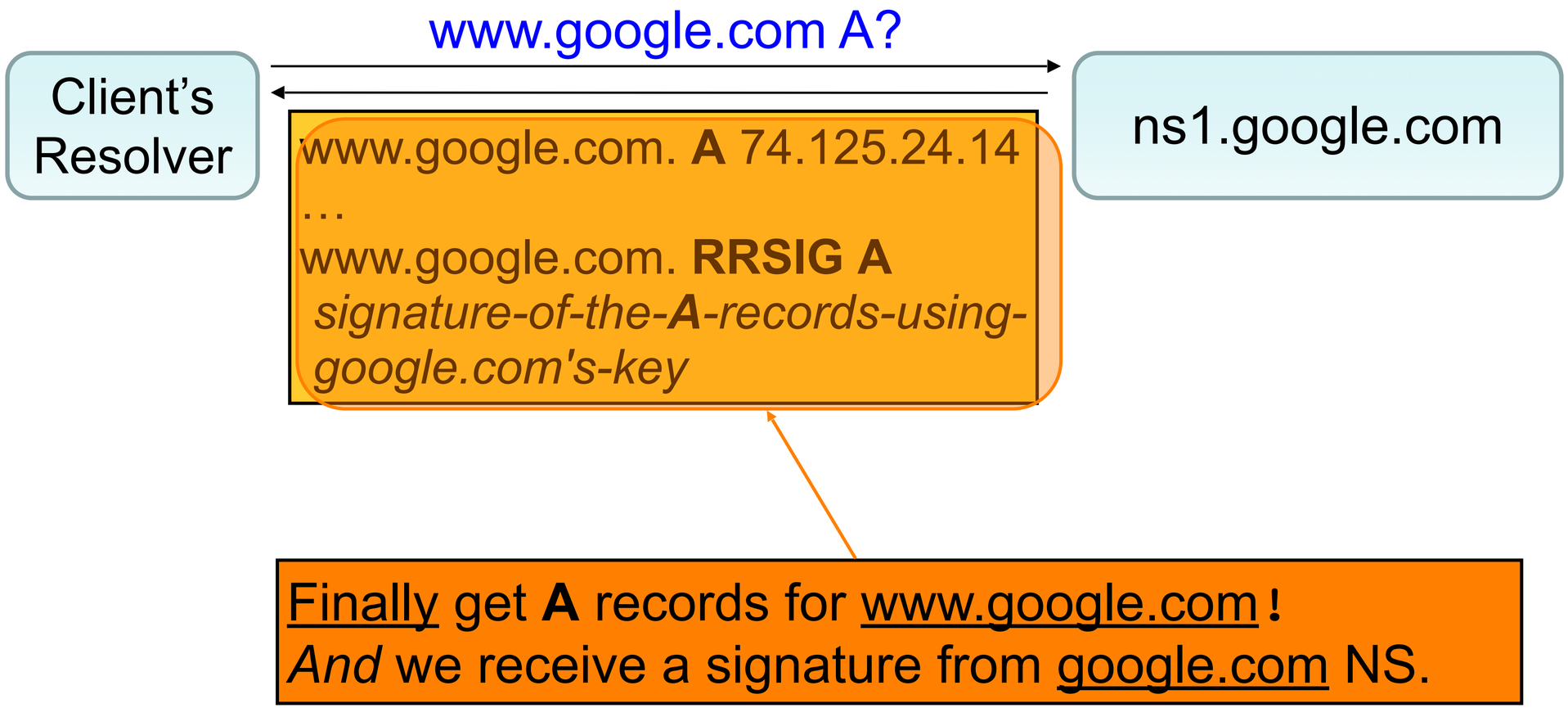


DNSSEC (with simplifications):

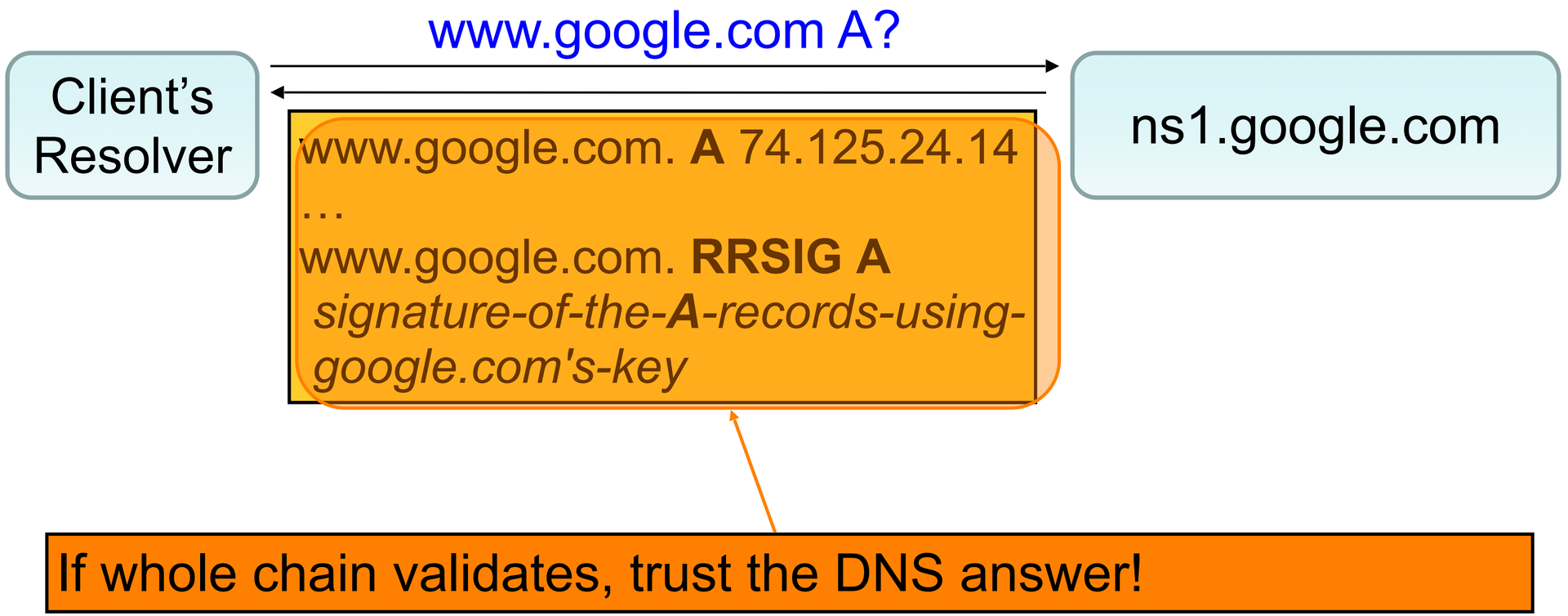
DNSSEC (with simplifications):



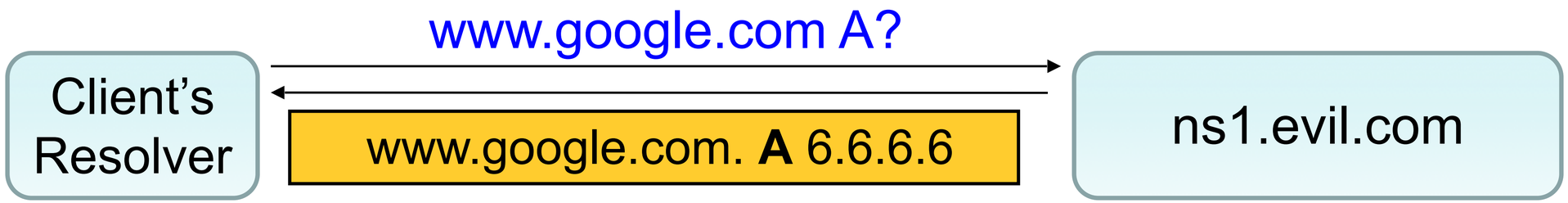
DNSSEC (with simplifications):



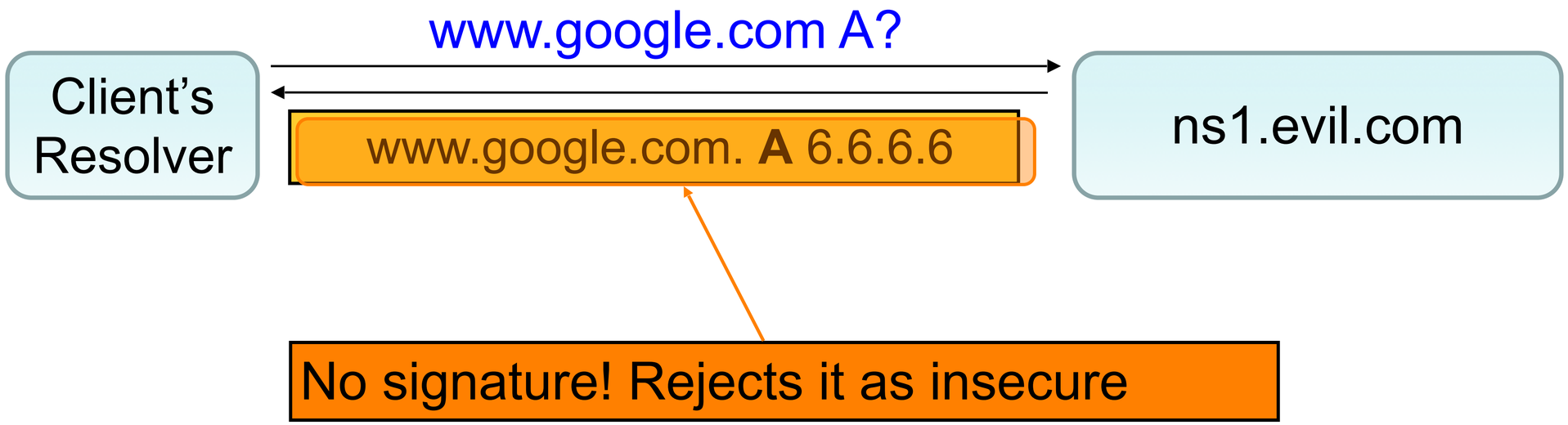
DNSSEC (with simplifications):



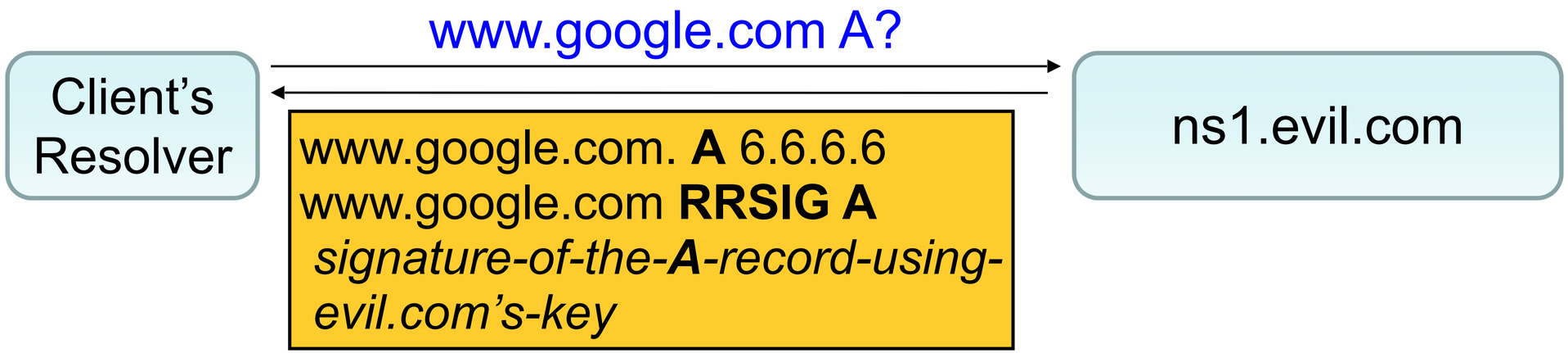
DNSSEC - Mallory attacks!



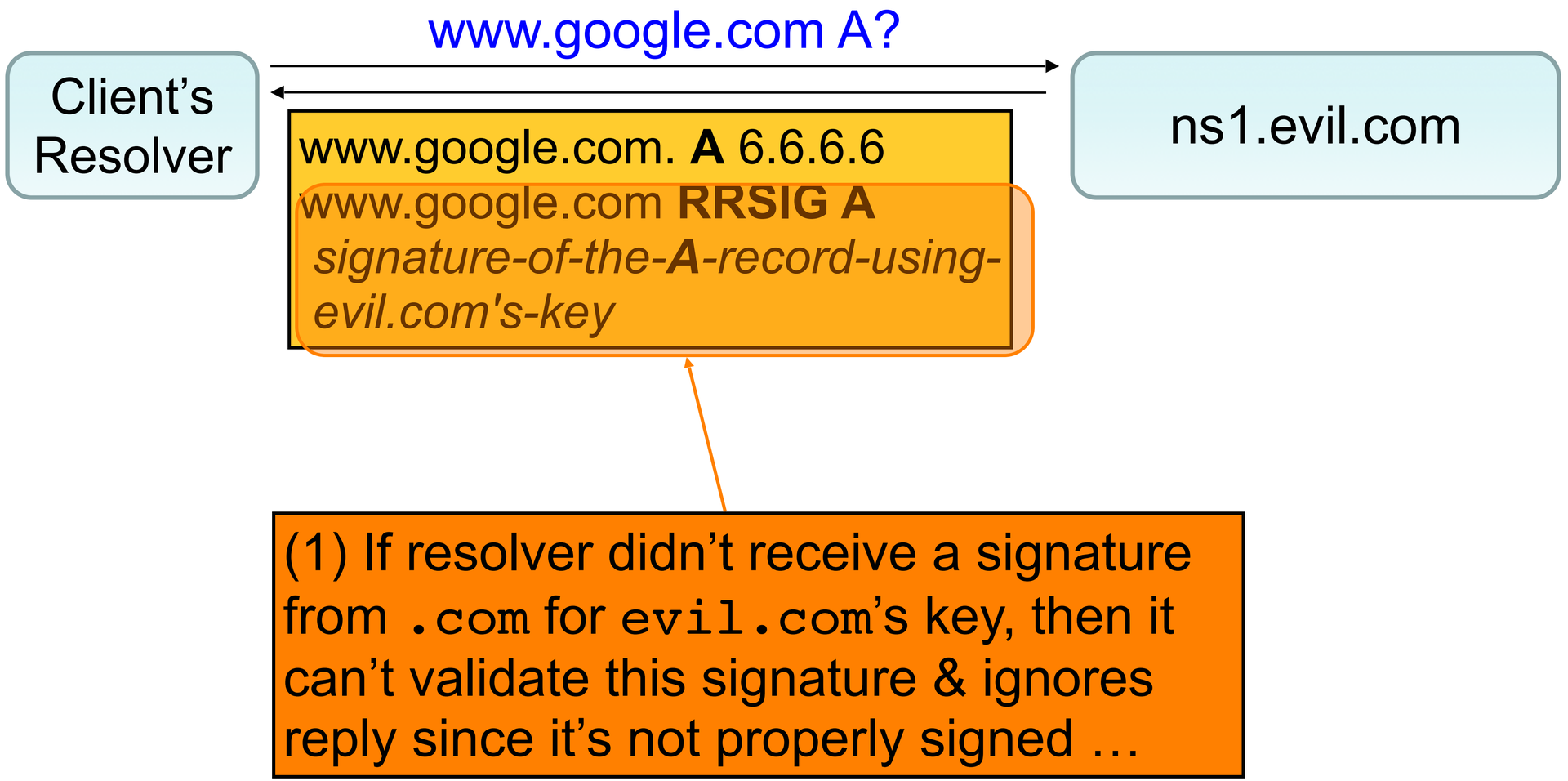
DNSSEC - Mallory attacks!



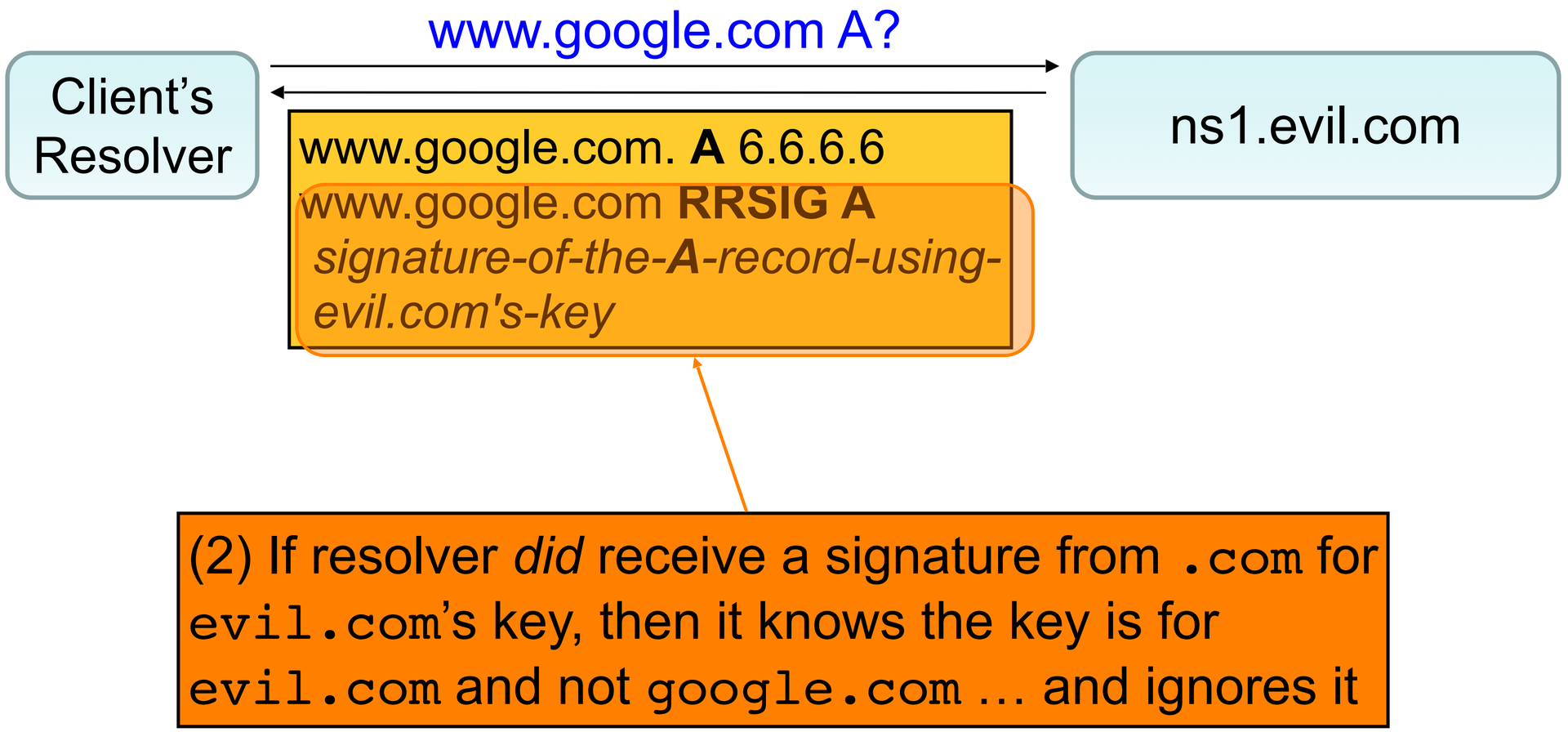
DNSSEC - Mallory attacks!



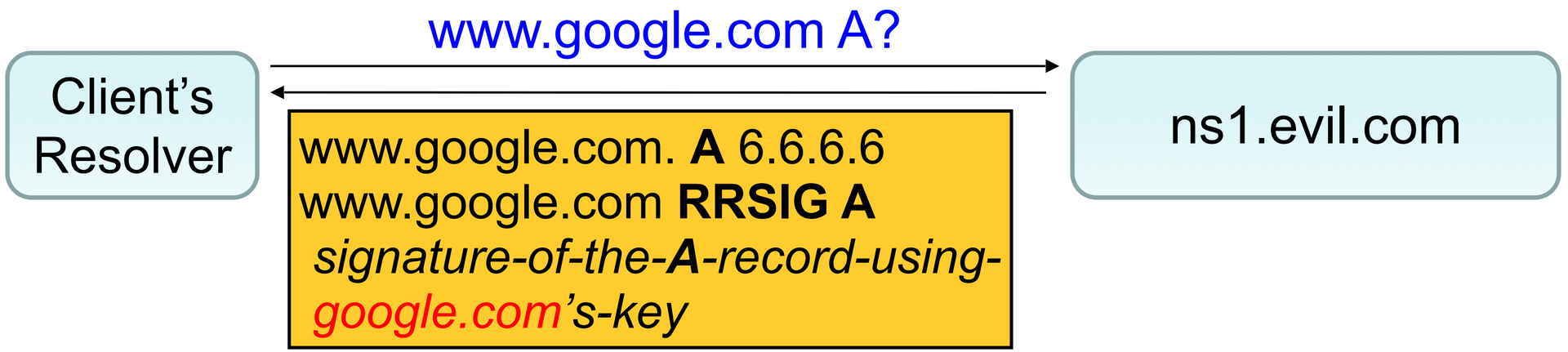
DNSSEC - Mallory attacks!



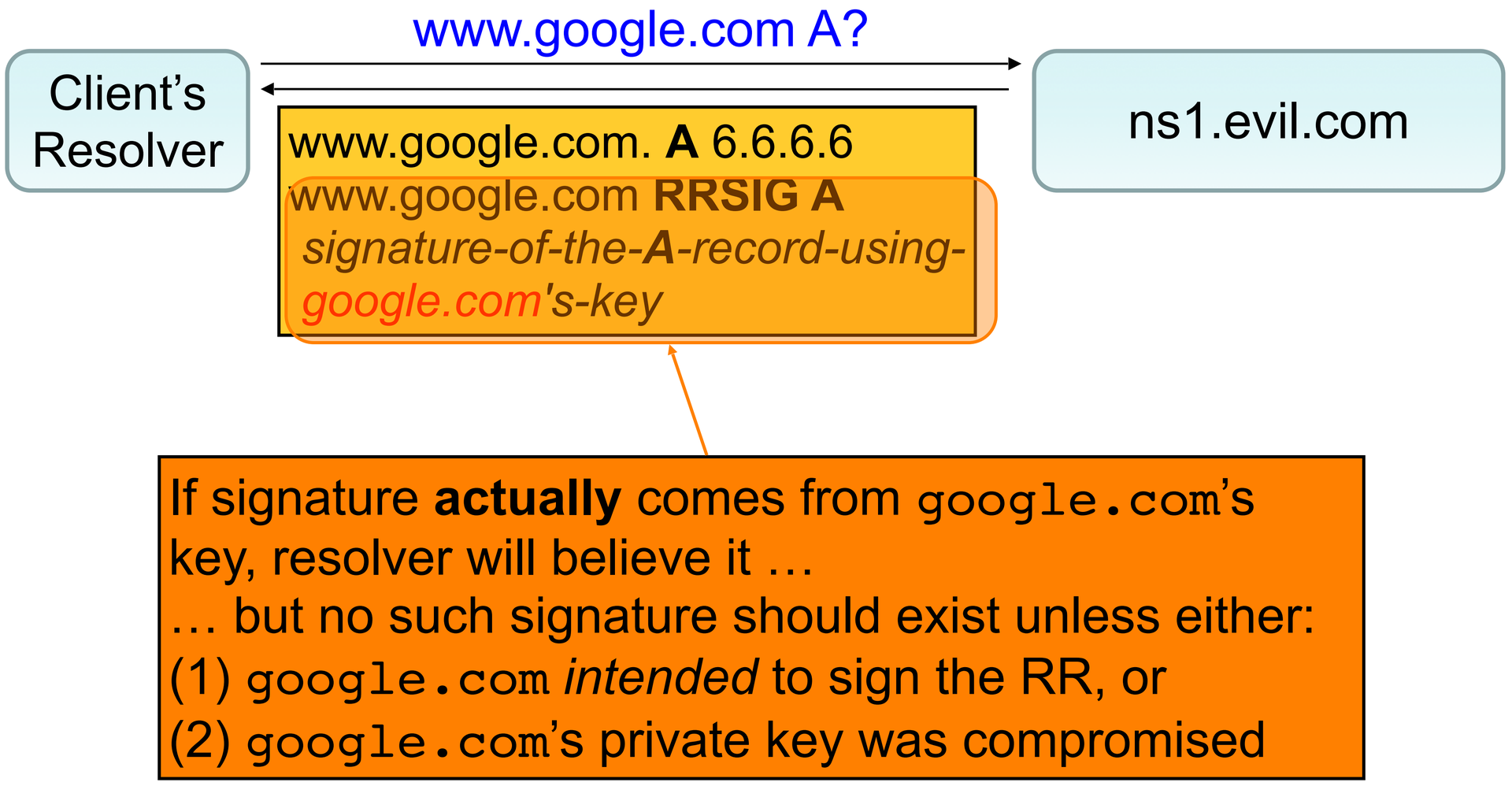
DNSSEC - Mallory attacks!



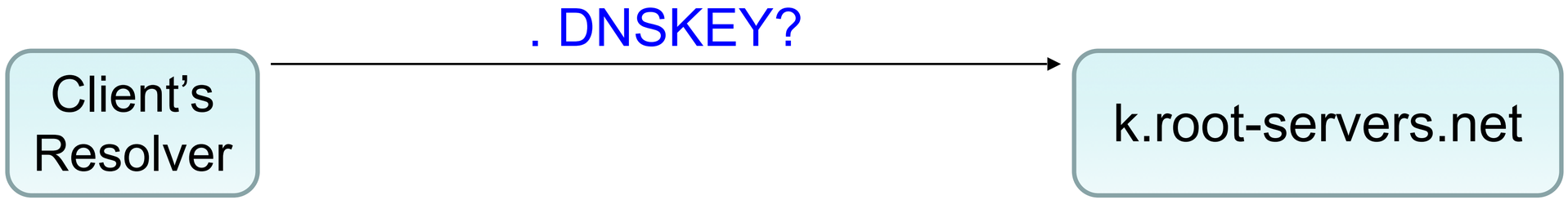
DNSSEC - Mallory attacks!



DNSSEC - Mallory attacks!



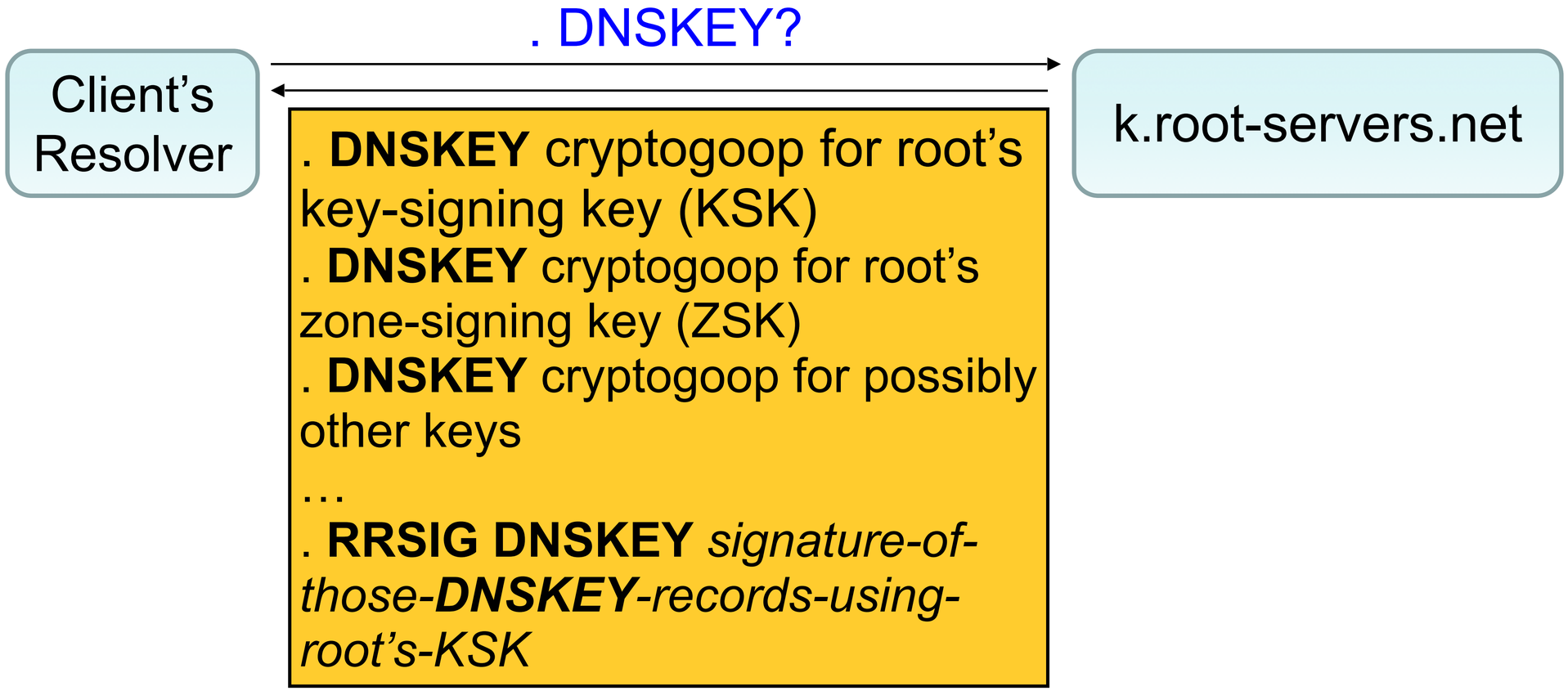
DNSSEC: Accessing keys



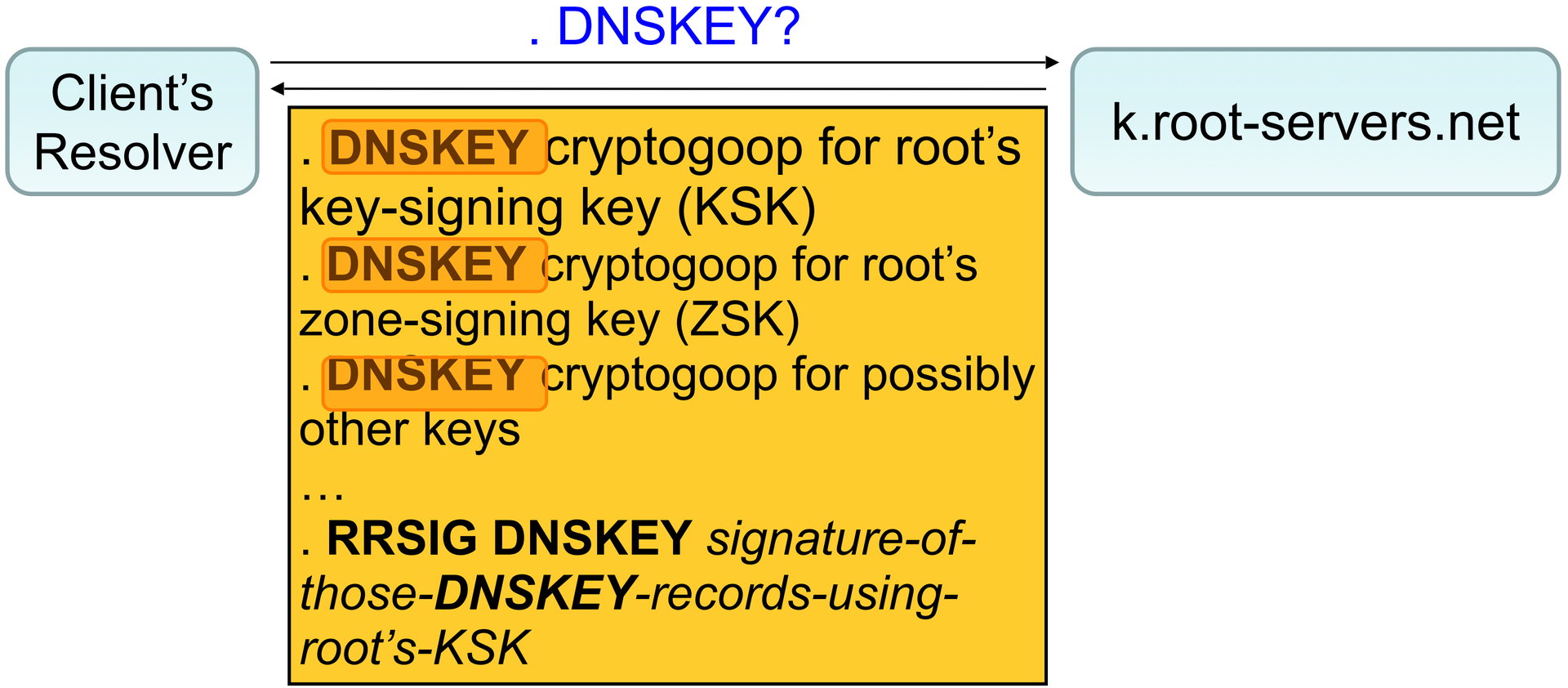
To get keys needed for validation, our client contacts each NS during the query asking for its keys.

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

DNSSEC: Accessing keys

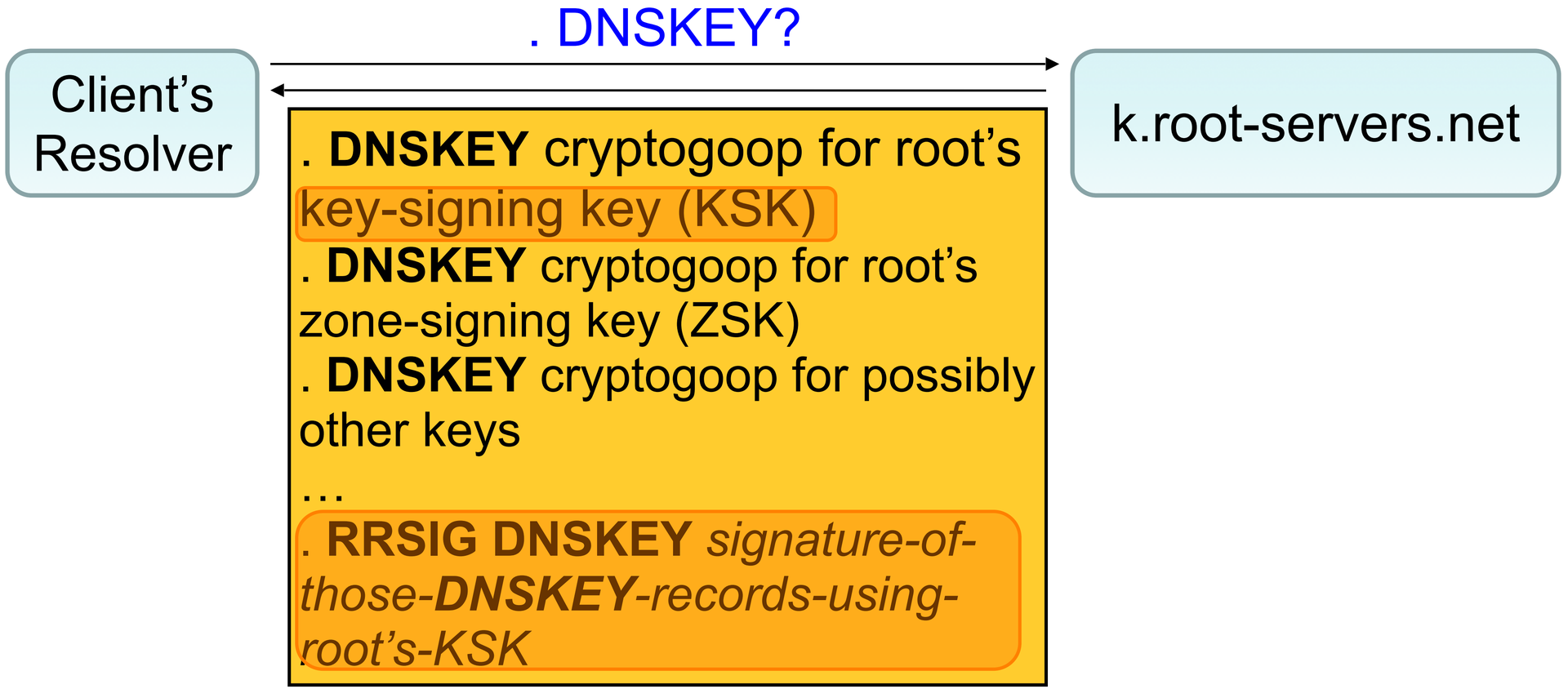


DNSSEC: Accessing keys



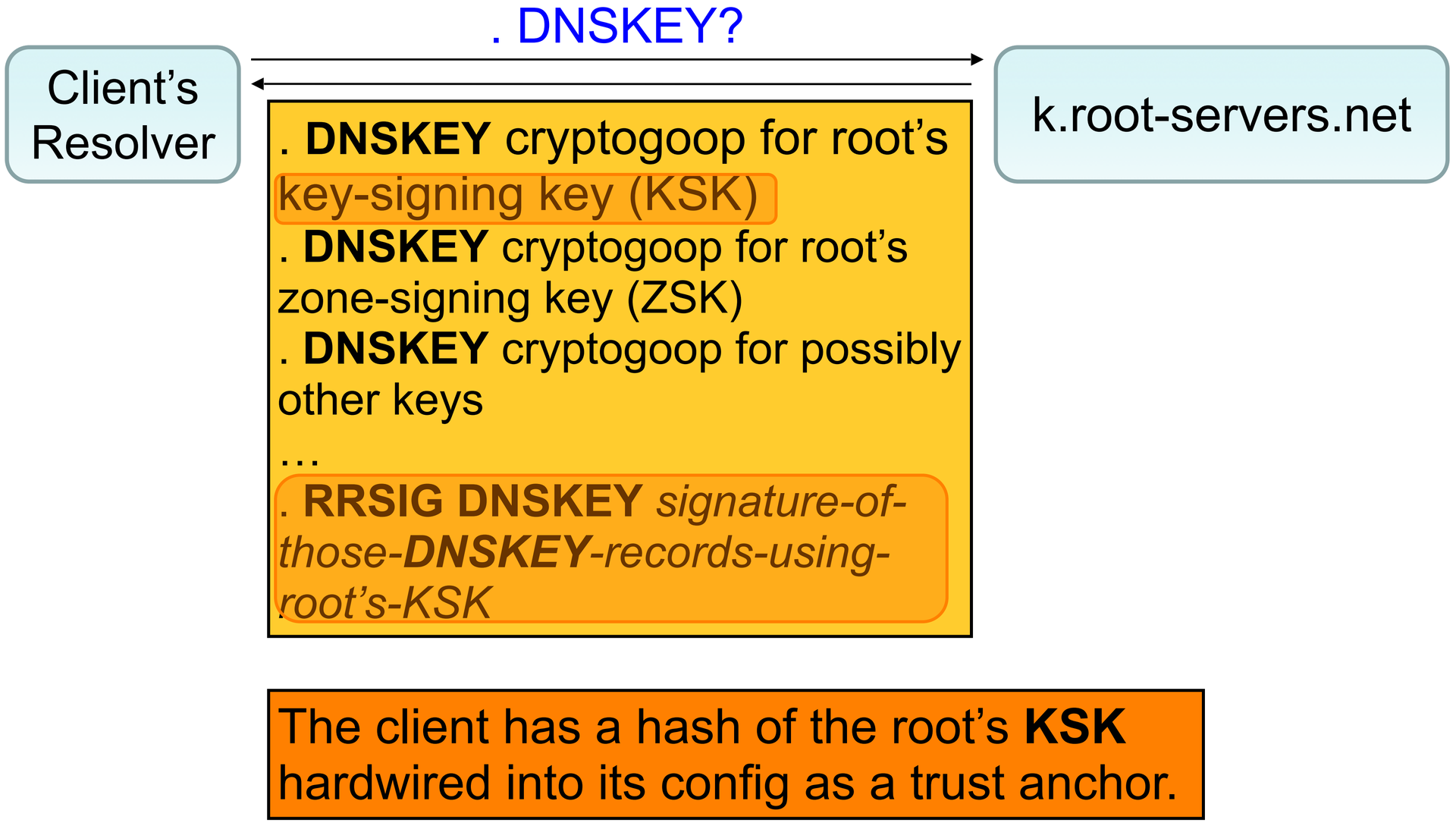
Each **DNSKEY** is a public key plus a description of the algorithms it's associated with (e.g., RSA+SHA256)

DNSSEC: Accessing keys



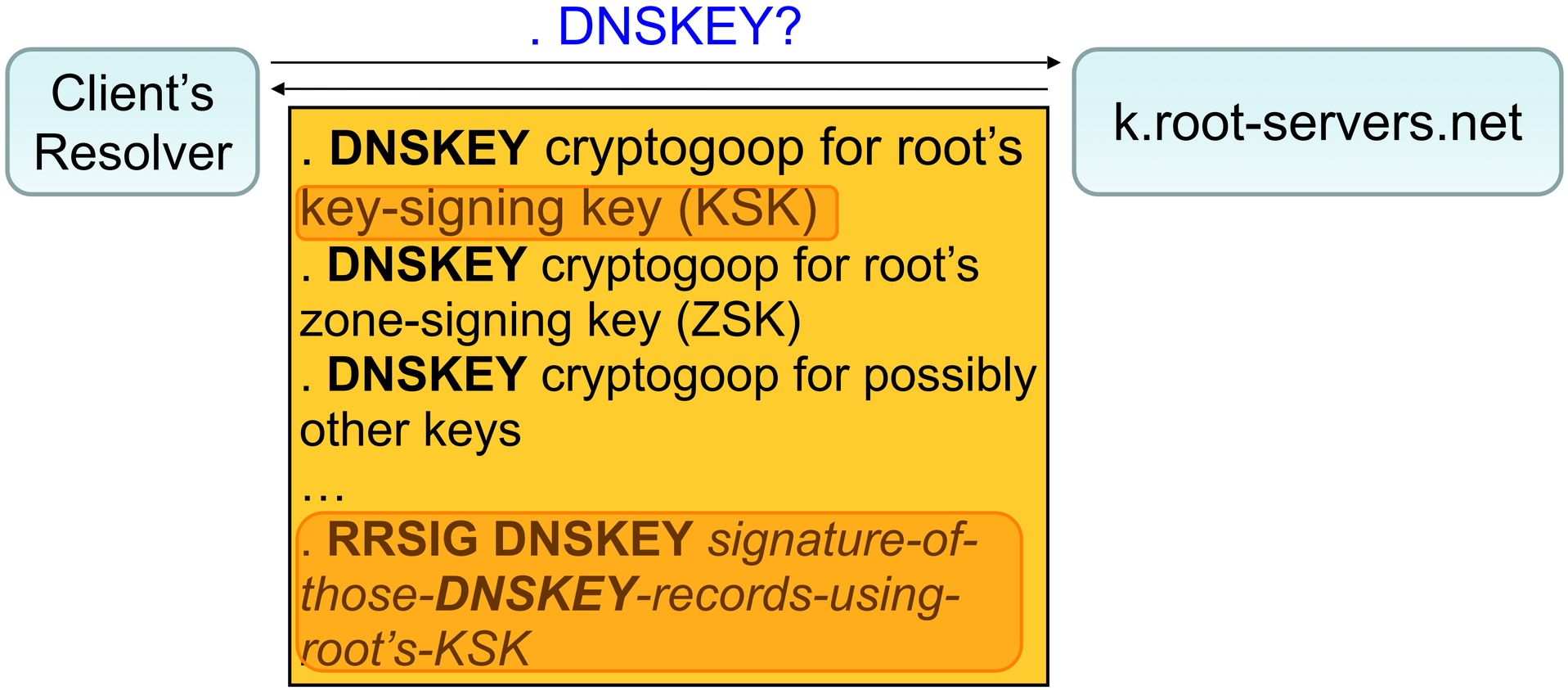
The **KSK** is used to sign all of the **DNSKEY** entries in the zone.

DNSSEC: Accessing keys



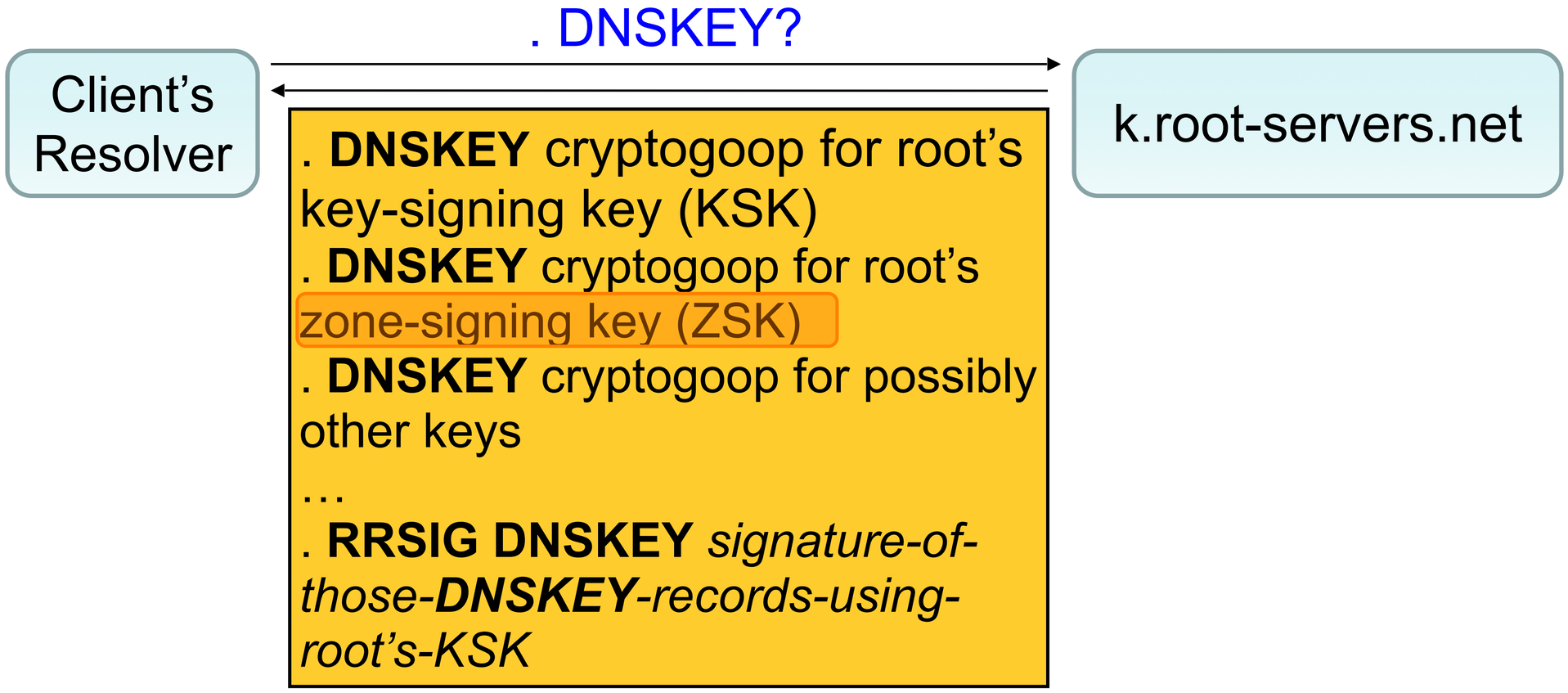
The client has a hash of the root's **KSK** hardwired into its config as a trust anchor.

DNSSEC: Accessing keys



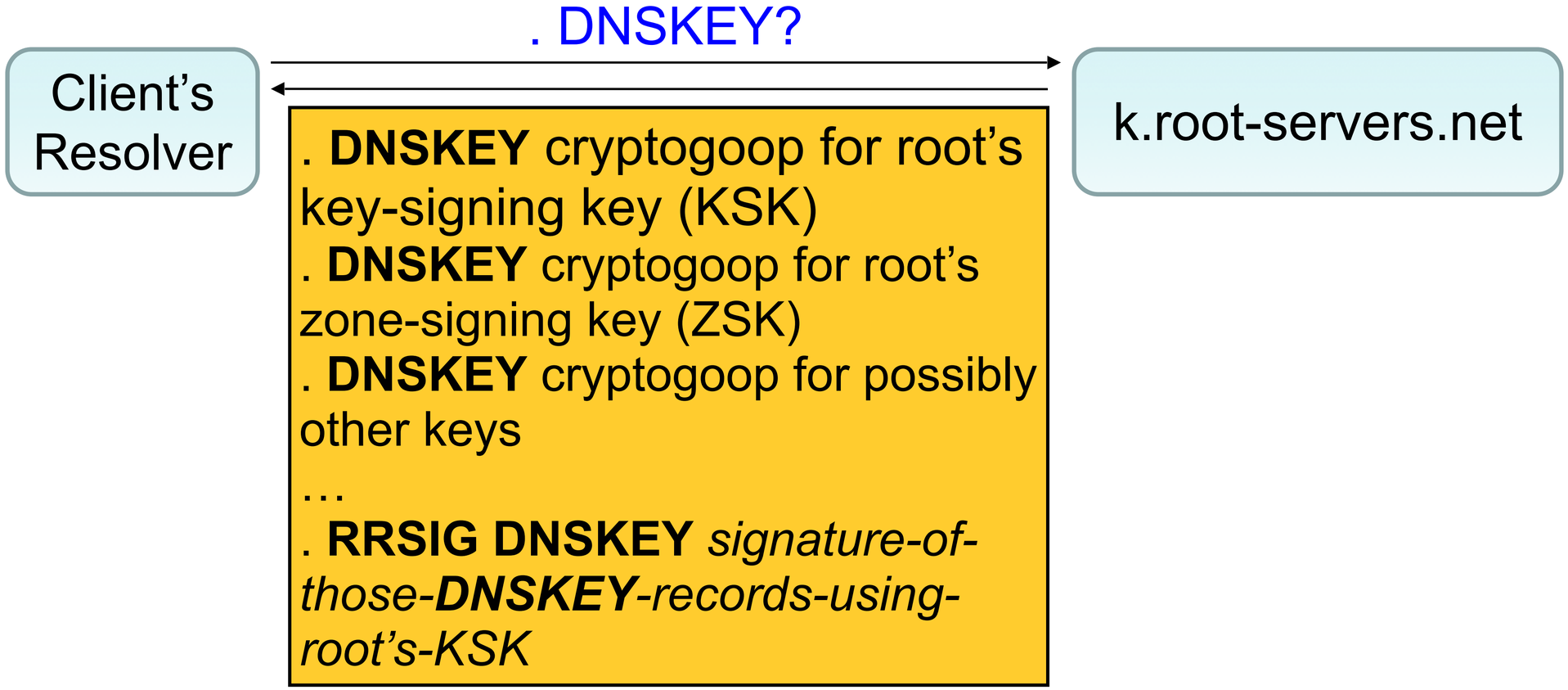
For everything below the root (e.g., `.com` and `google.com`) we get a hash of the KSK via a **DS** record, as shown earlier, so we can tell if we get the right KSK in a **DNSKEY** entry.

DNSSEC: Accessing keys



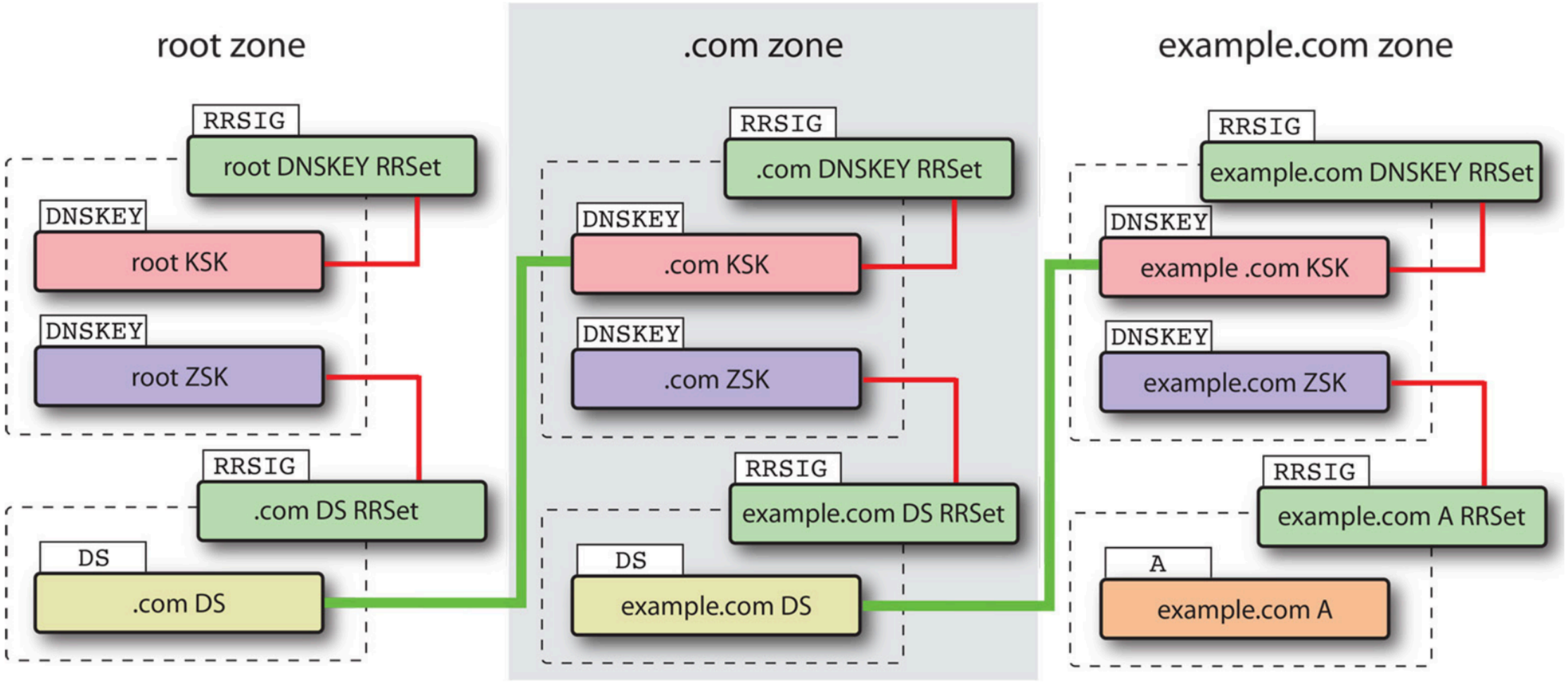
The **ZSK** is used for signing all of the other RRSIG entries in the zone, including DS records for subzones.
(E.g., `.com` signs its **DS** record for `google.com` using `.com`'s **ZSK**)

DNSSEC: Accessing keys



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

Issues With DNSSEC ?

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

Issues With DNSSEC, con't

- 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

Issues With DNSSEC, con't

- 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)

Issues With DNSSEC, con't

- 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.

Issues With DNSSEC, con't

- 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).