Hands on 

tutorial

Willem Toorop, NLNet Labs
Shumon Huque, Verisign
Glen Wiley, Verisign

About getdns

- getdns API = a DNS API specification – resolving names
- getdns API = created by and for applications developers

- *getdns* = the first implementation of this specification

- *getdns* highlighted feature : Parry pervasive monitoring and man in the middle attacks by bootstrapping encrypted channels
- *getdns* mission slogan : Security Begins with a Name

About DNSSEC

- A globally distributed database with authenticated data

About DNSSEC

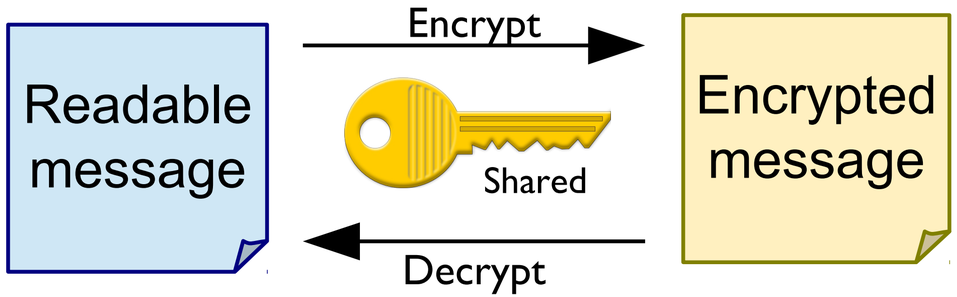
- A global distributed database with authenticated data
- Wasn't it about protecting users against domain hijacking?

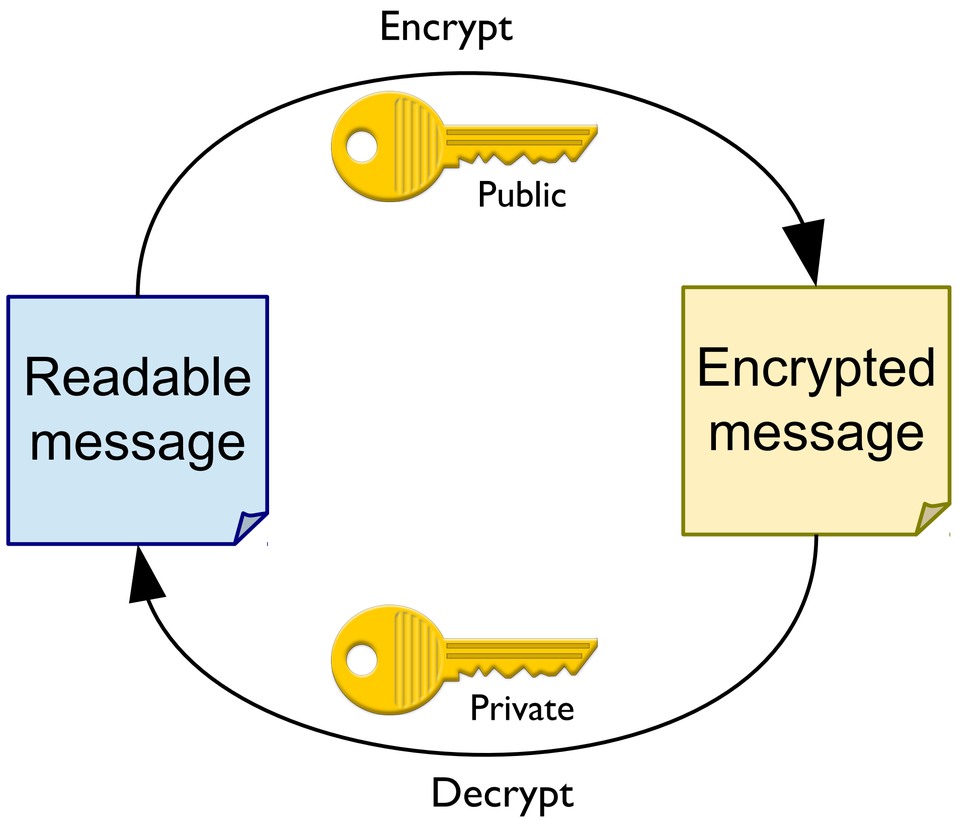
  - DNS = the phone book of the Internet
  - Data unauthenticated
  - DNSSEC to the rescue

About DNSSEC

- A global distributed database with authenticated data
- Wasn't it about protecting users against domain hijacking?
  - DNS = the phone book of the Internet
  - Data unauthenticated
  - DNSSEC to the rescue
- Yes, but it does so by giving (origin) authenticated answers
  - where *origin* means that the authoritative party for a zone authenticates the domain names within that zone

Refresher – Public Key Crypto

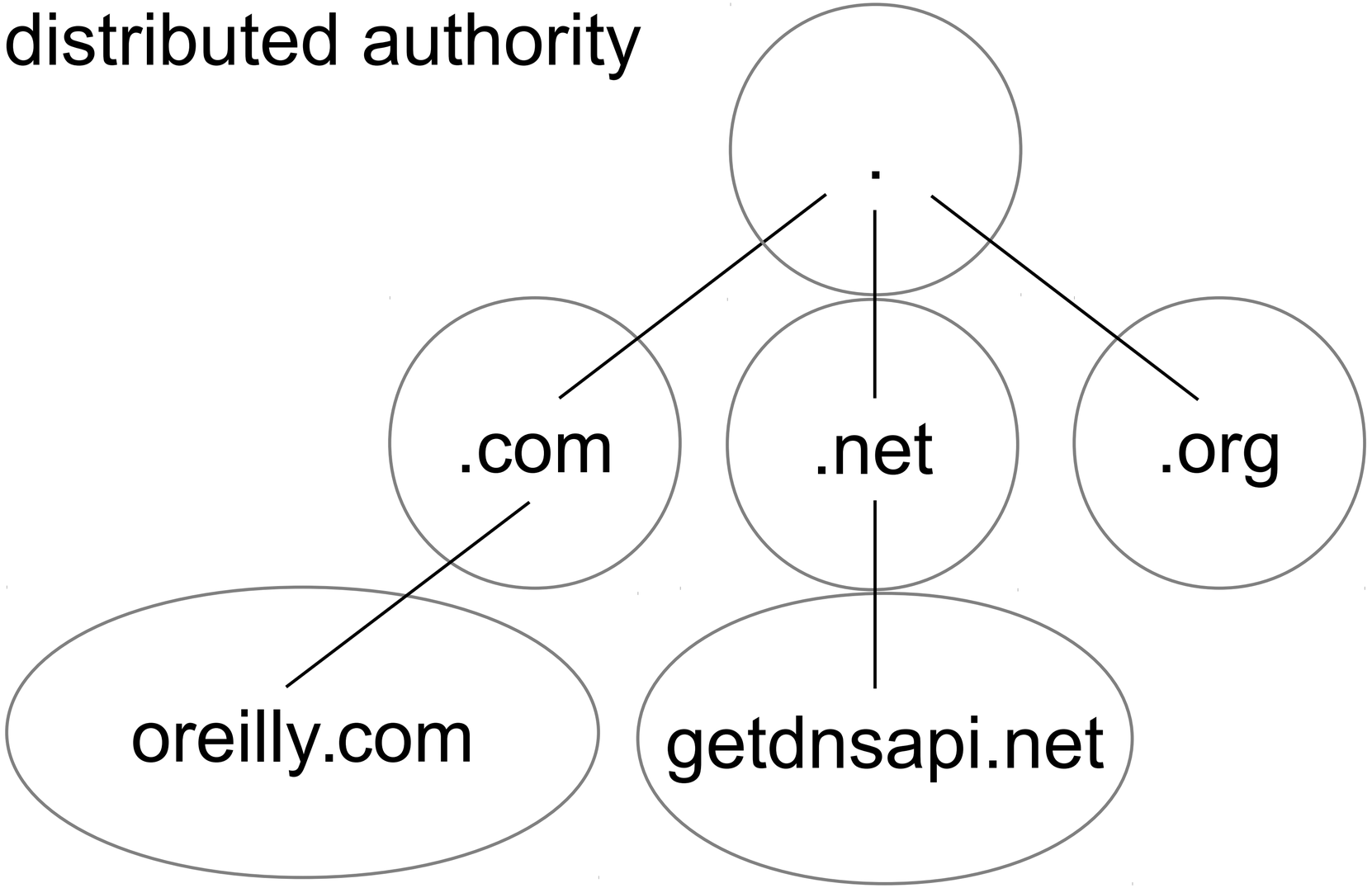




- Symmetric encryption
- Asymmetric encryption

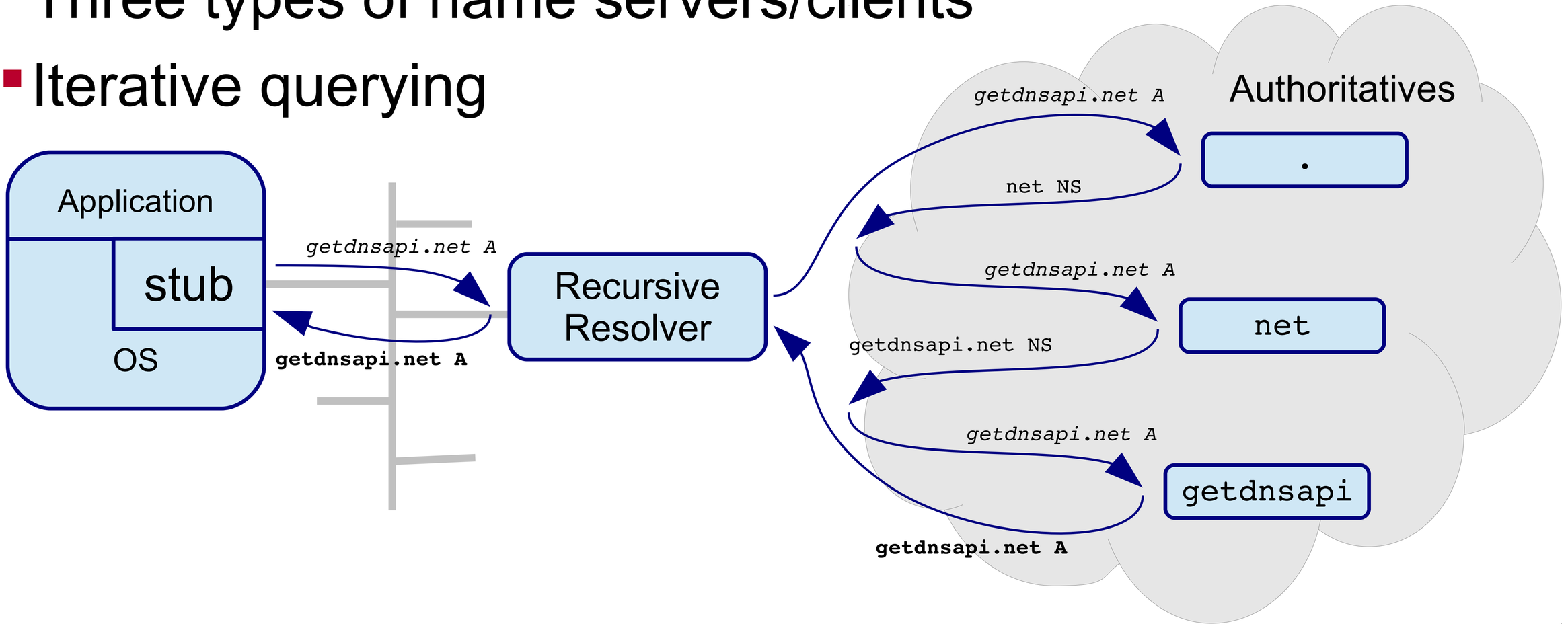
Refresher – DNS in two slides

- Zones with distributed authority

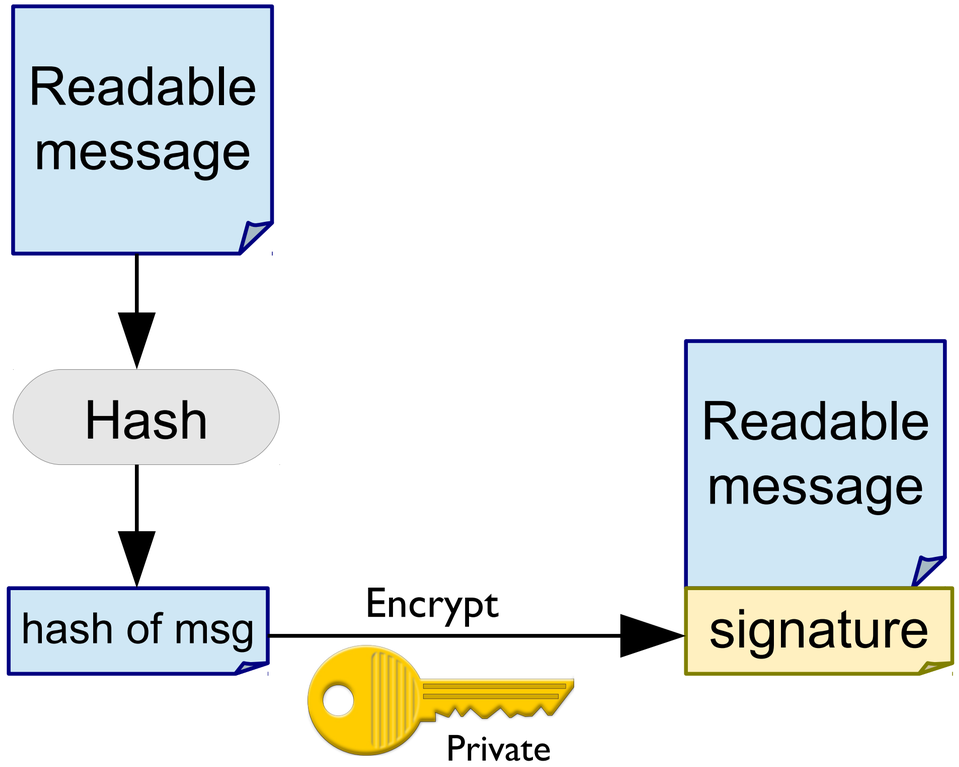


Refresher – DNS in two slides

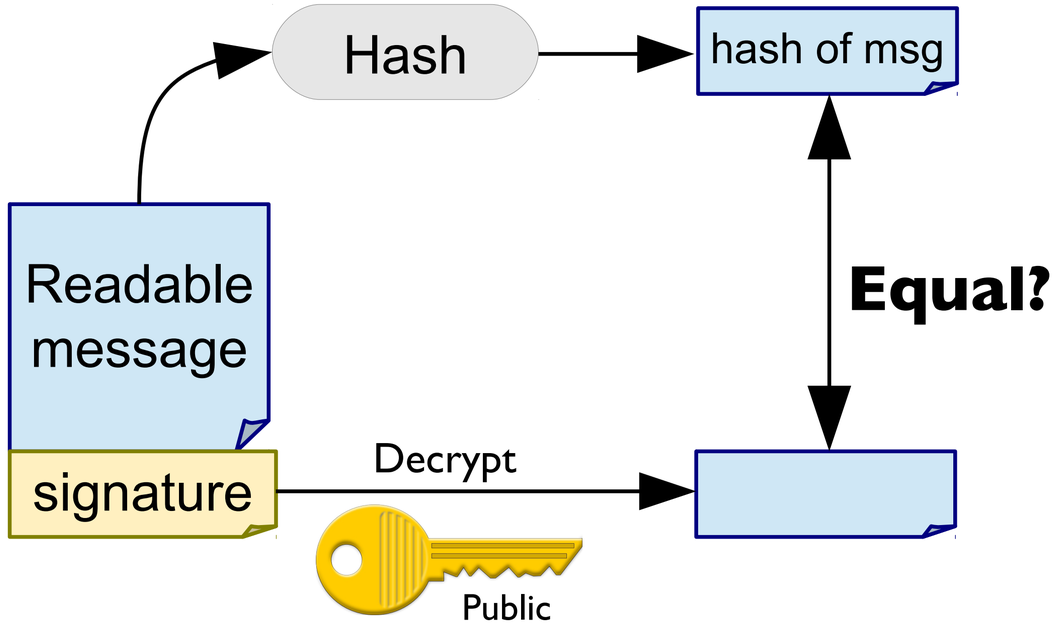
- Zones with distributed authority
- Three types of name servers/clients
- Iterative querying



DNSSEC – Public Key Crypto – Signing

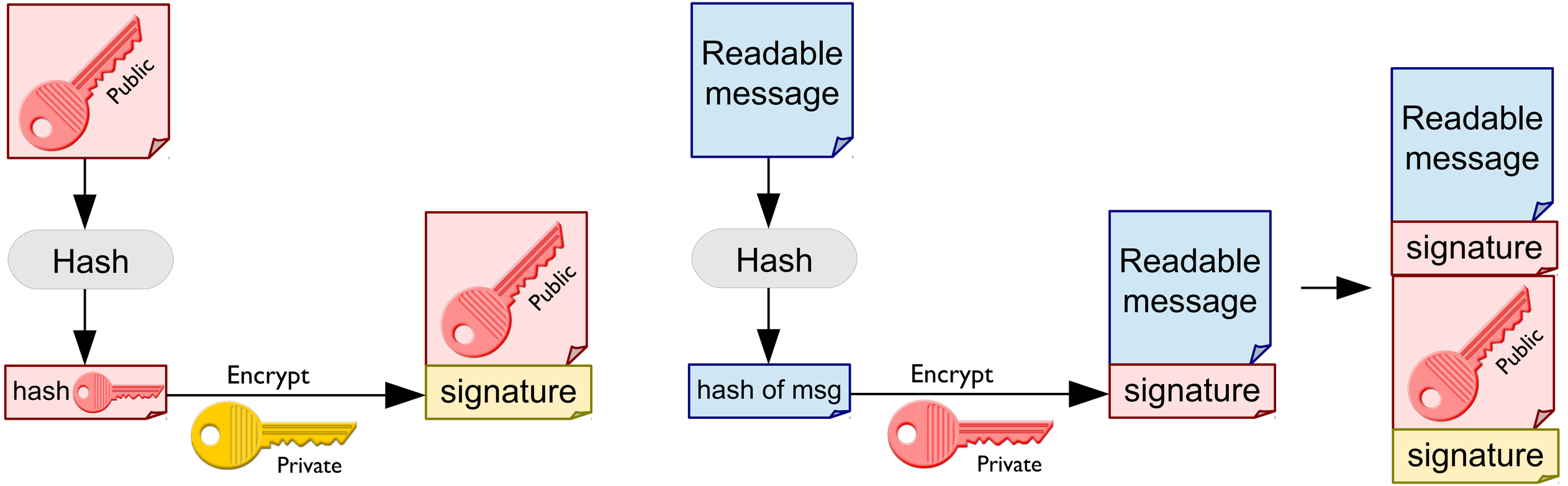


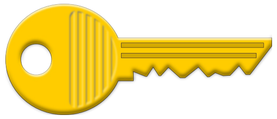
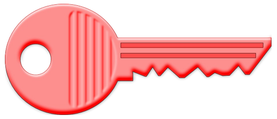
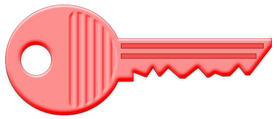
- Create signature



- Verify signature

DNSSEC – Public Key Crypto – delegating authority

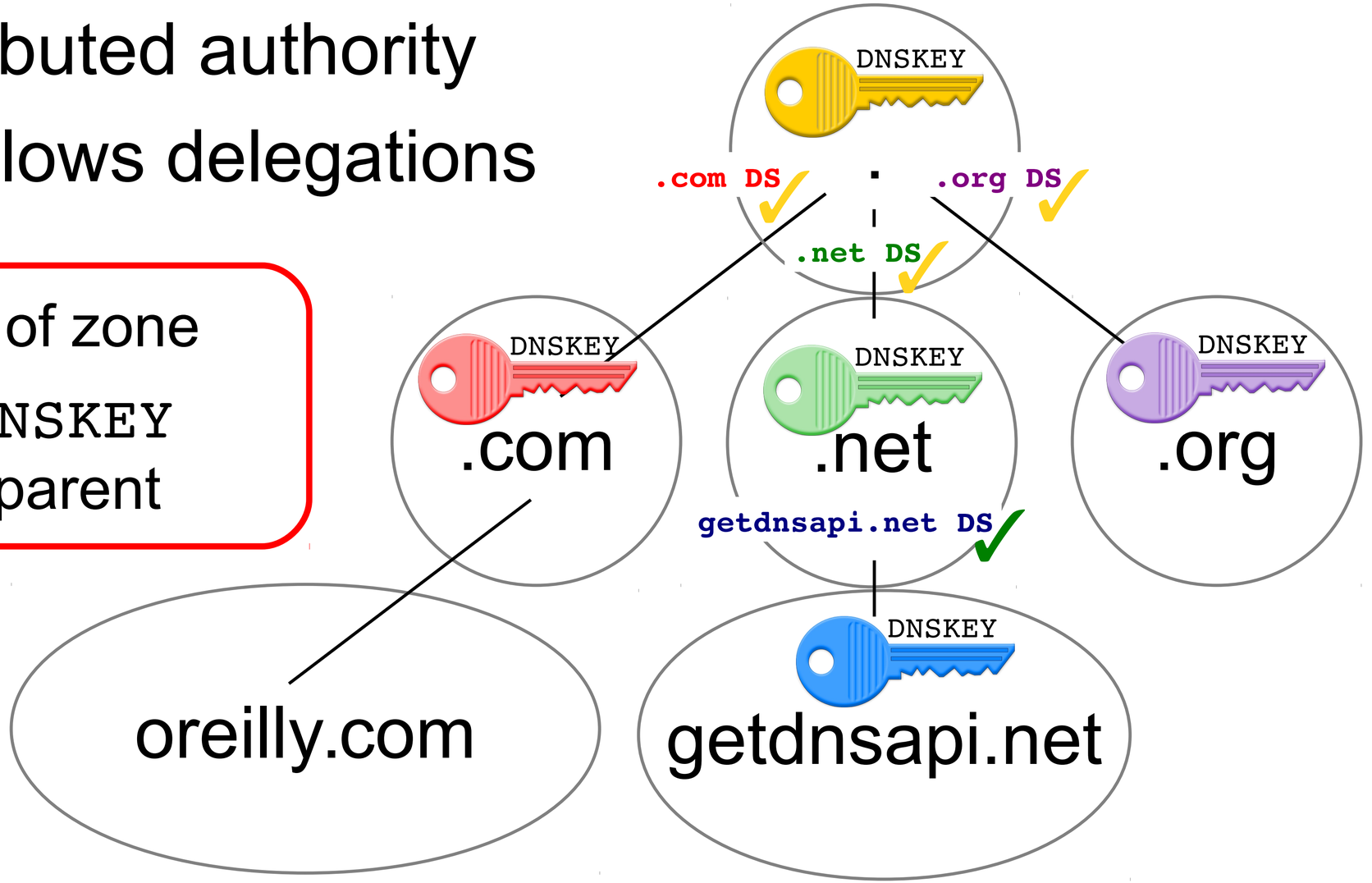


- Building the chain of trust
   authorizes 
-  signs the message

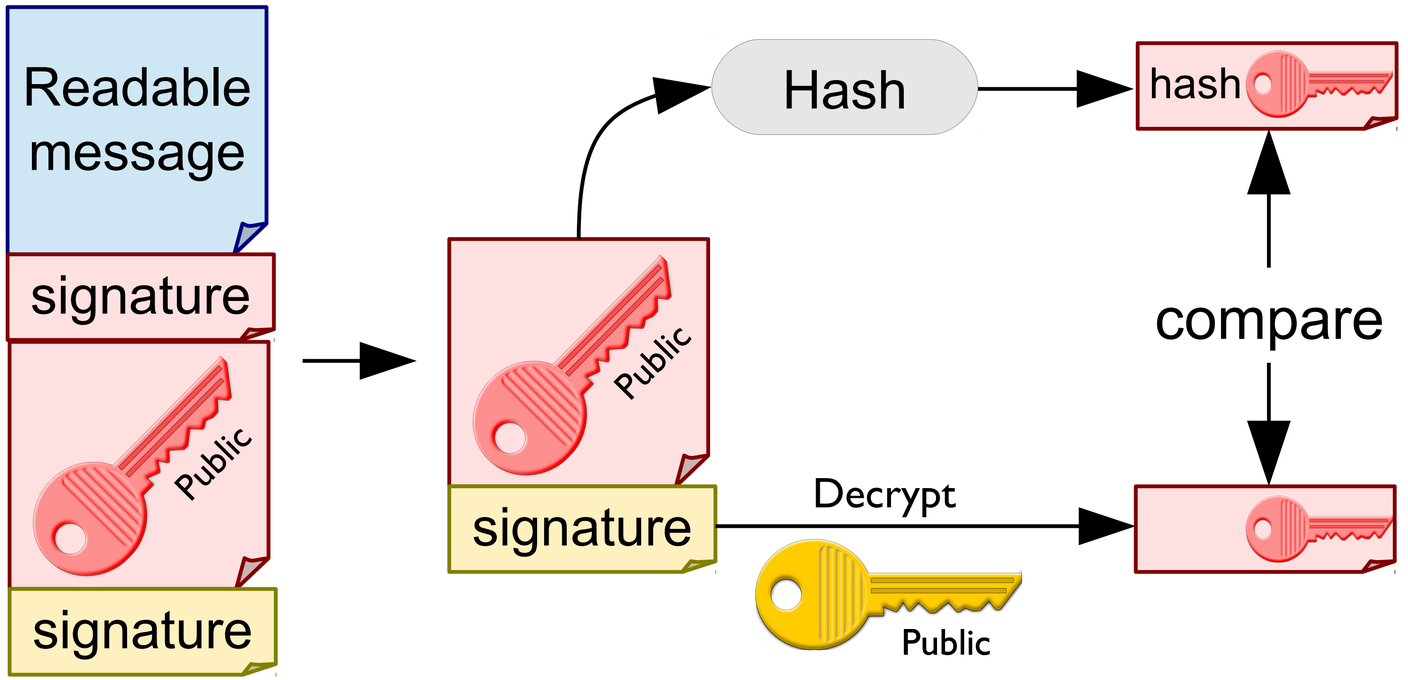
DNSSEC – Chain of Trust

- Zones with distributed authority
- Chain of trust follows delegations

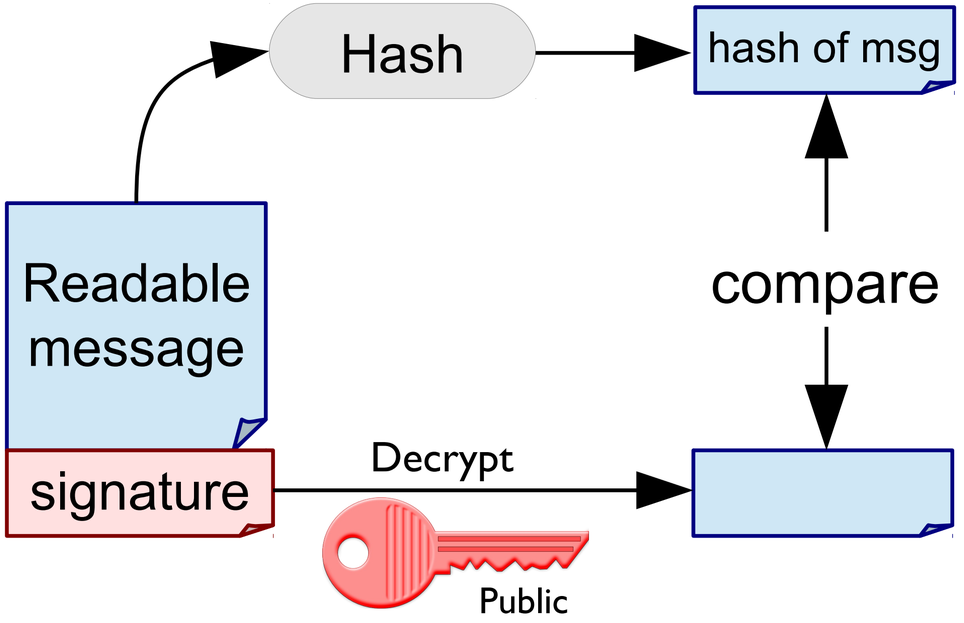
- `DNSKEY` Public key of zone
- `DS` Hash of `DNSKEY` signed by parent



DNSSEC – Public Key Crypto – Verifying delegations



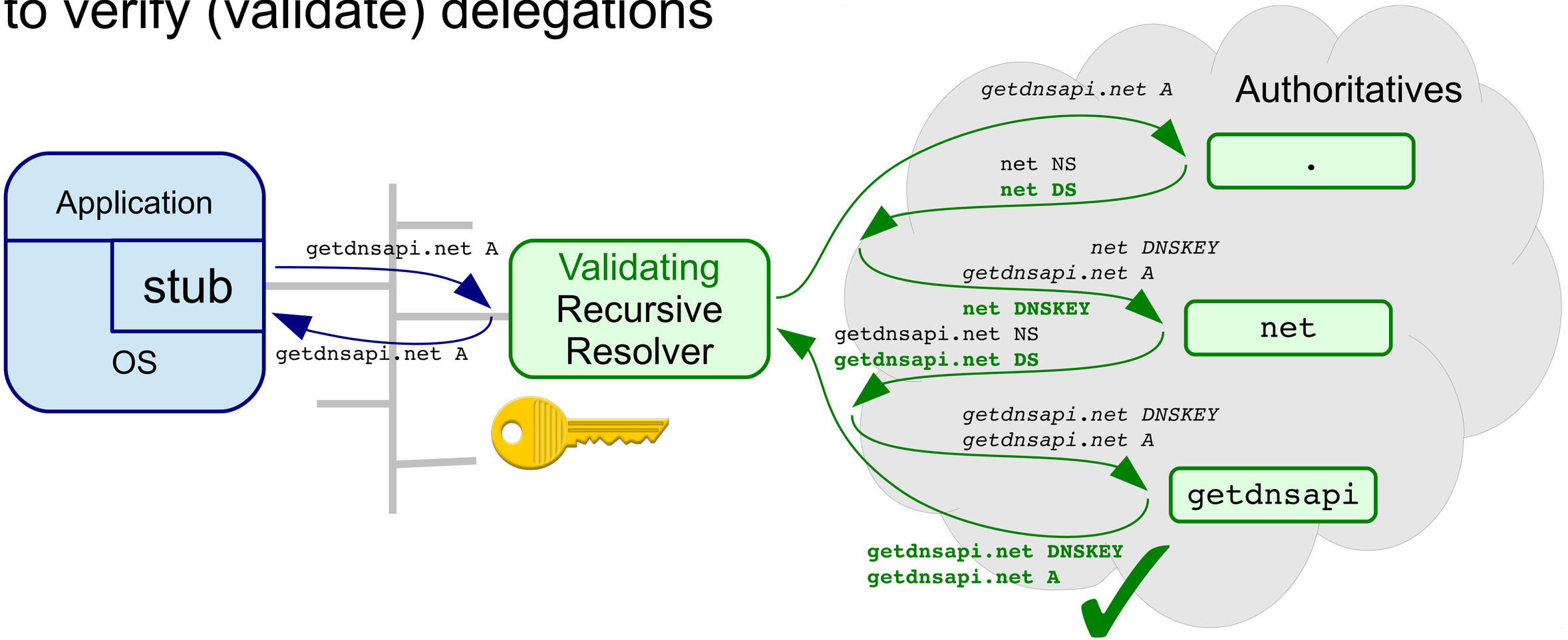
- Verify authorization



- Verify signature

DNSSEC – Validating

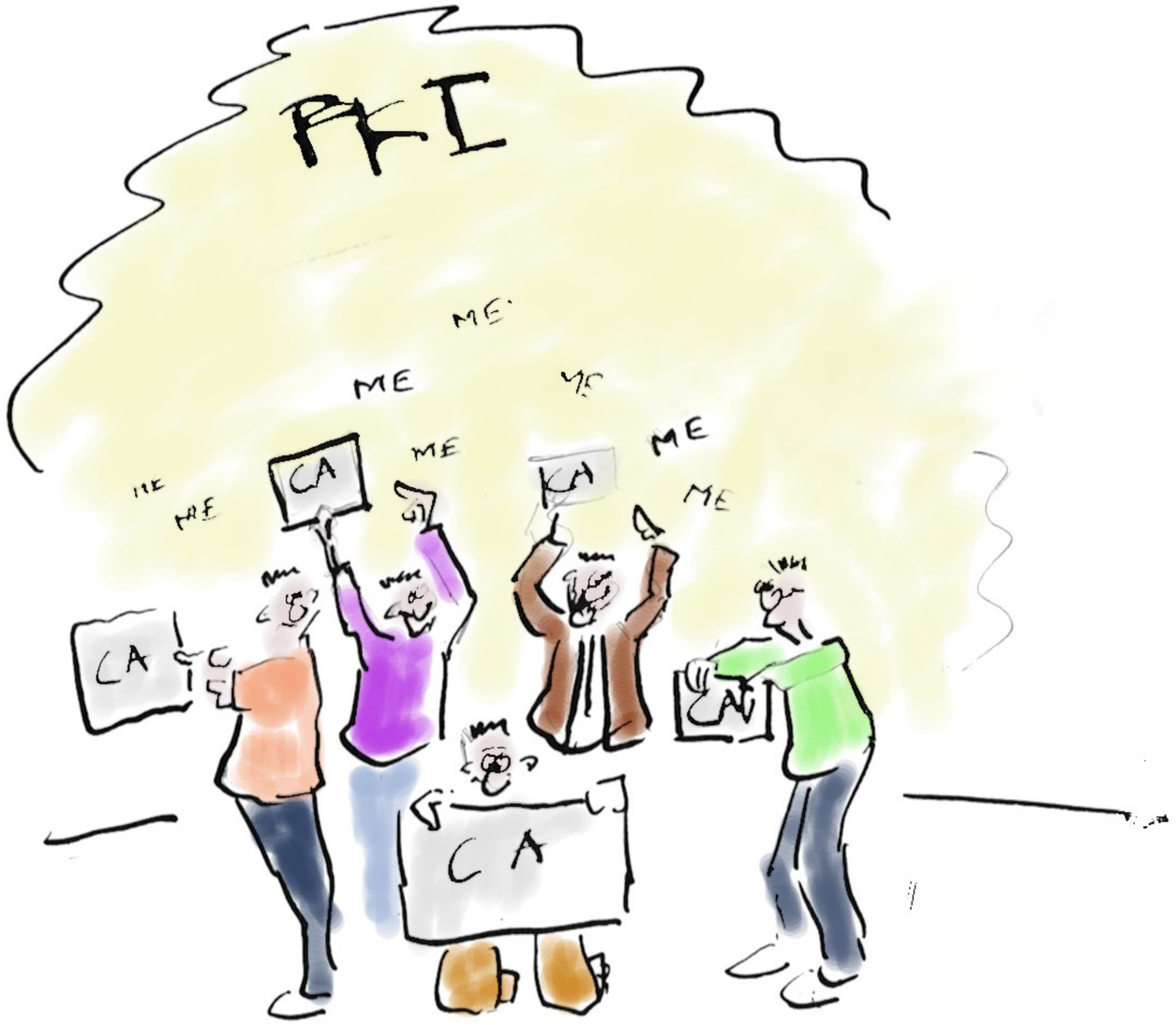
- A *Validating Recursive Resolver* uses the root's public key to verify (validate) delegations



DNSSEC for Applications – for TLS

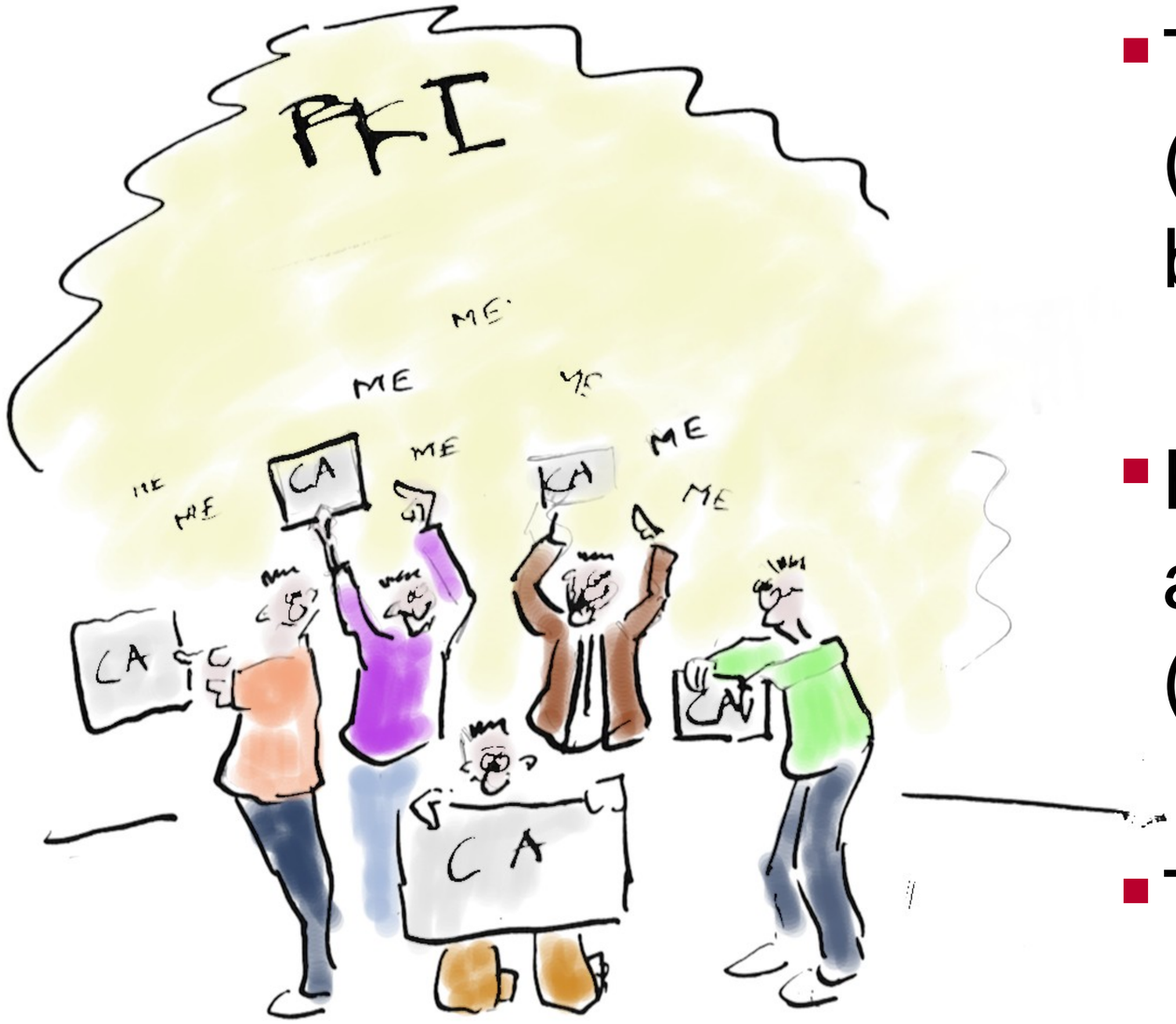
- Transport Layer Security (TLS) uses both asymmetric and symmetric encryption
- A symmetric key is sent encrypted with remote public key
- How is the remote public key authenticated?

TLS Not Leveraging DNSSEC



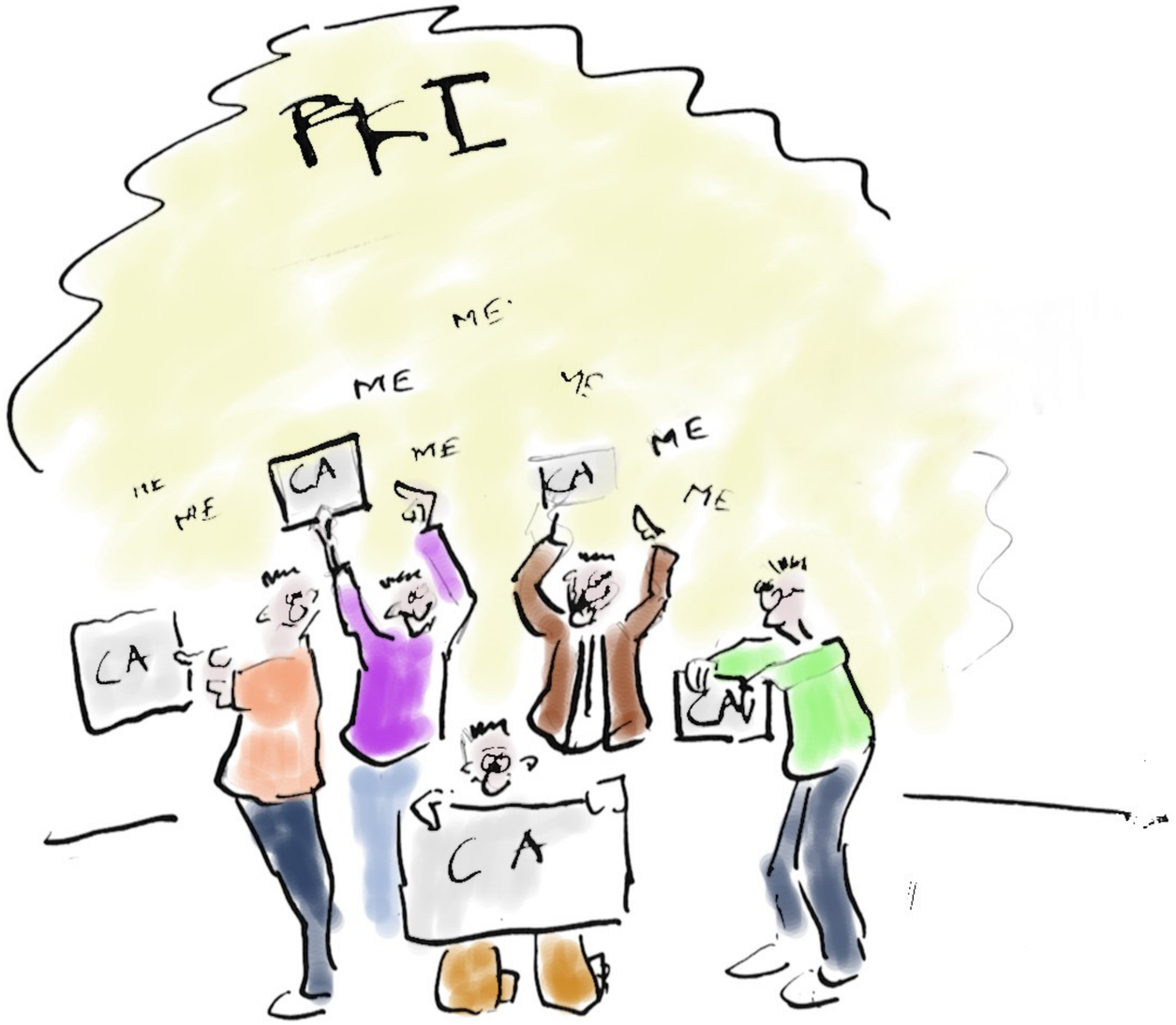
- How is the remote public key authenticated?

How is Remote Public Key Authenticated?

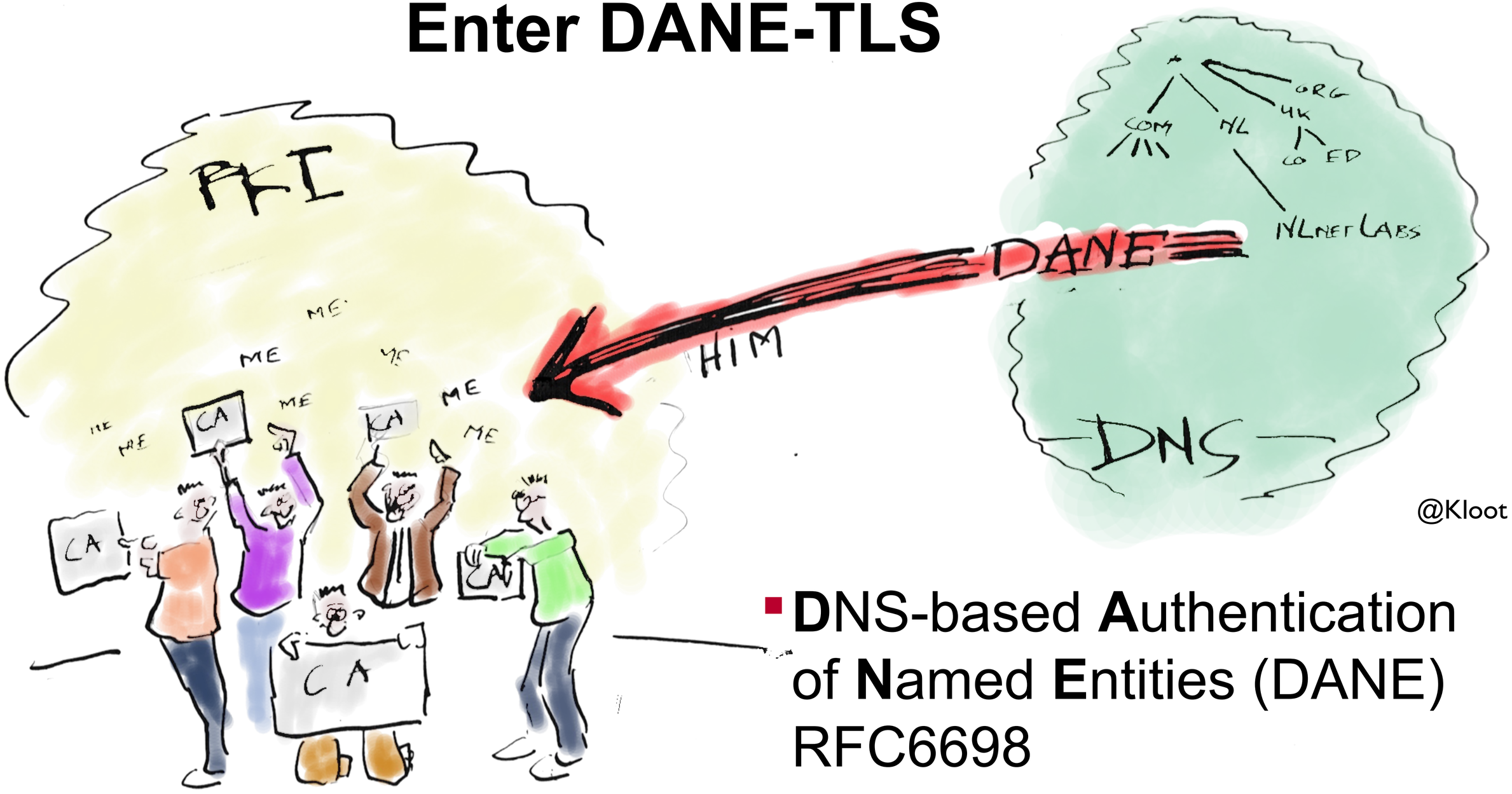


- Through Certificate Authorities (CAs), maintained in OS, browser...
- Every CA is authorized to authenticate for **any** name (as strong as the weakest link)
- There are 1000+ CAs

Enter DANE-TLS



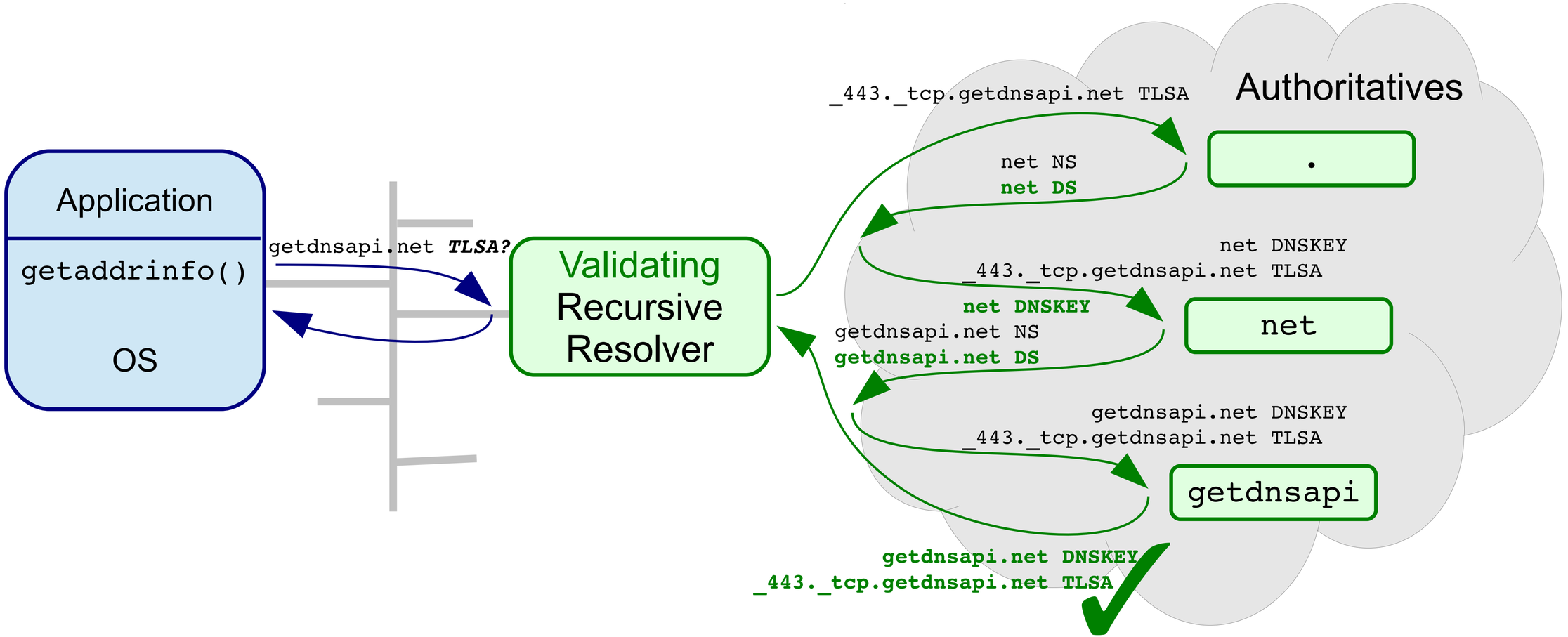
Enter DANE-TLS



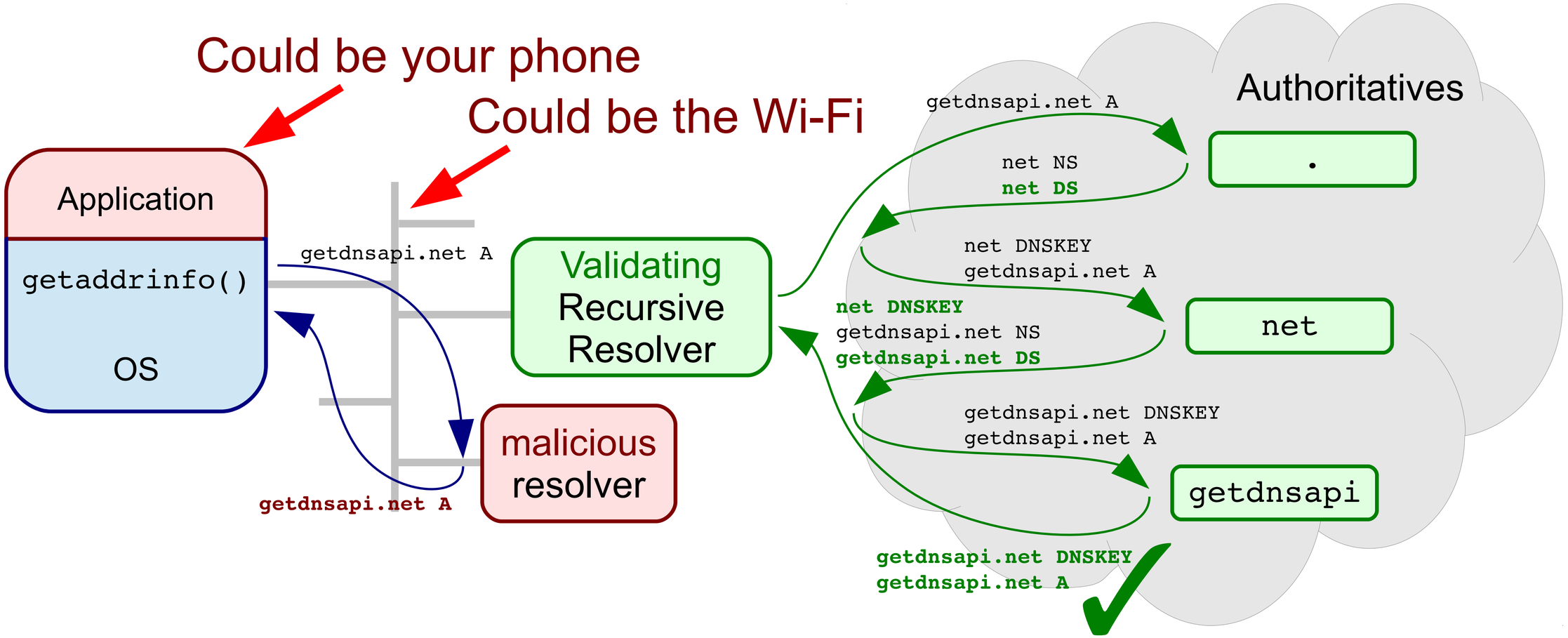
- **D**NS-based **A**uthentication of **N**amed **E**ntities (DANE) RFC6698

DANE out of reach for Applications

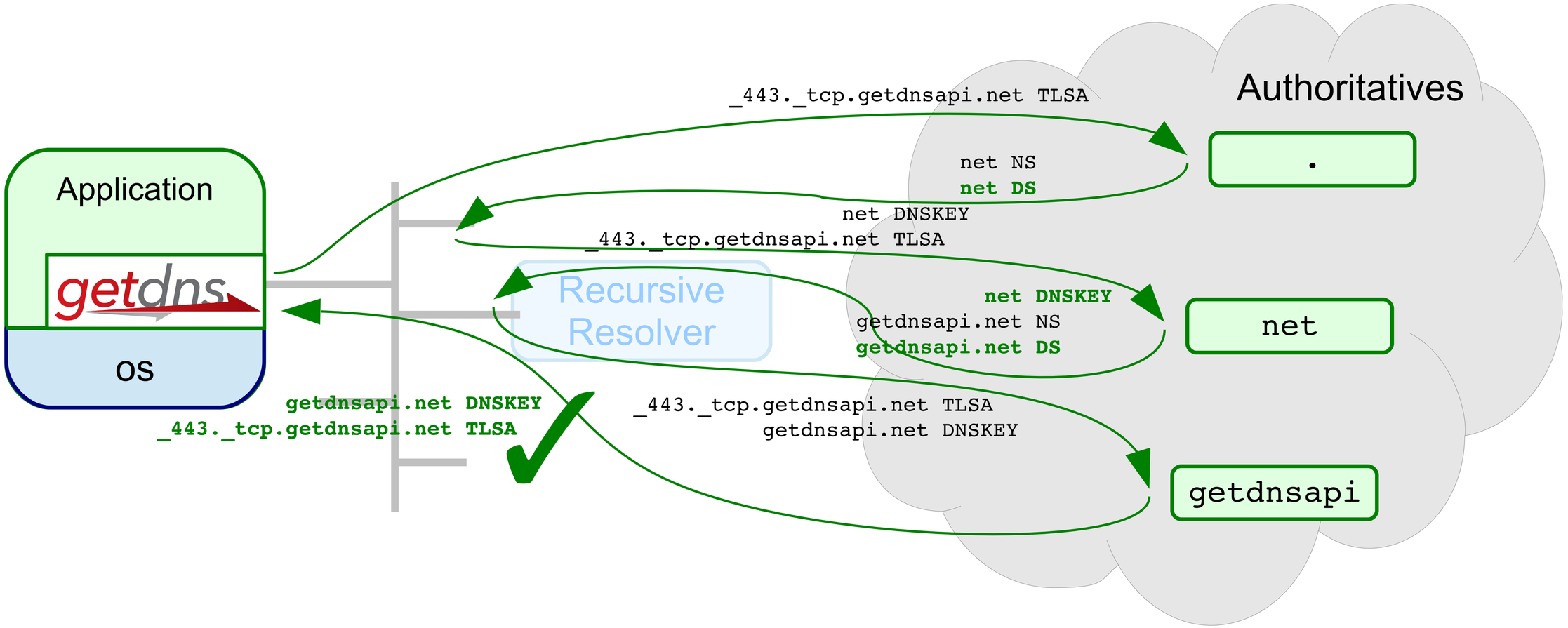
- `getaddrinfo()` returns addresses, how to ask for `TLSA`, or `SSHFP`
- `getaddrinfo()` doesn't tell if you got Authenticated Data (AD)



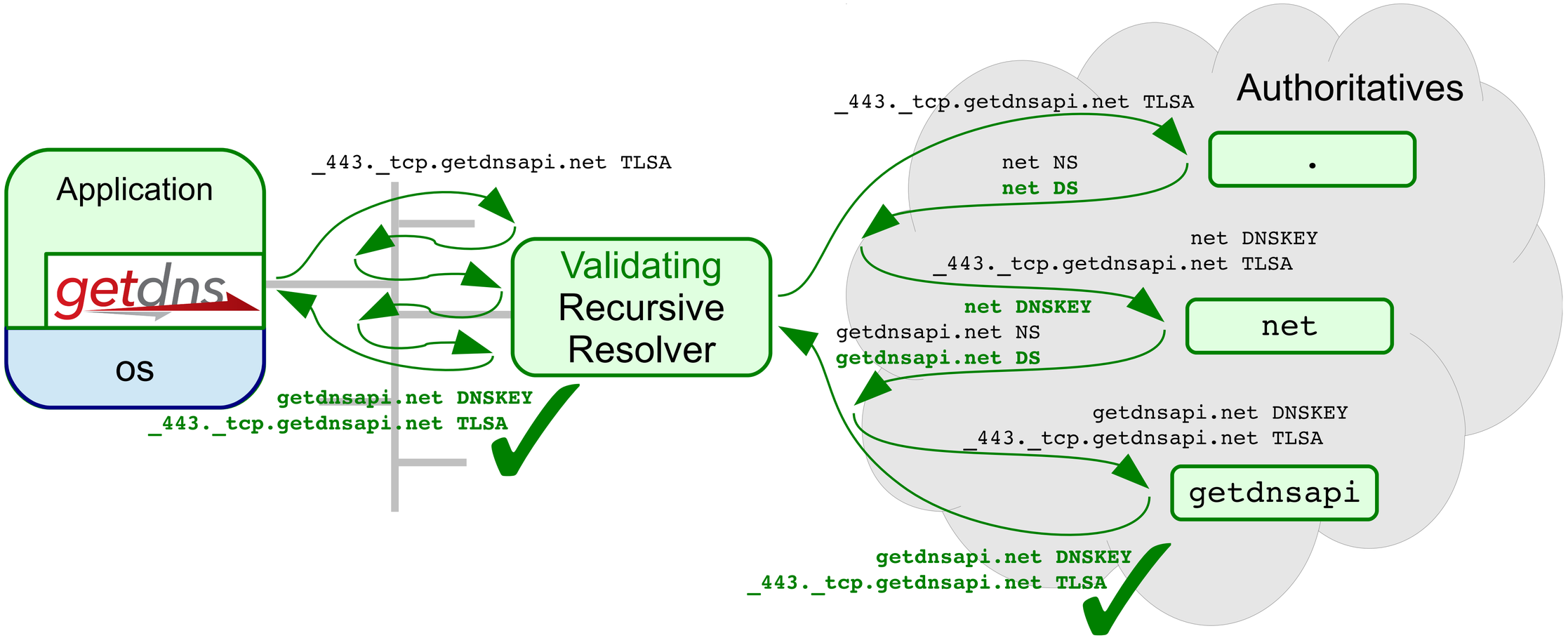
Do you trust the resolver?



Bypass resolver completely...



Or Do DNSSEC Iteration as a Stub!



Motivation – for a new DNS API

- From API Design considerations:

  … There are other DNS APIs available,
  but there has been very little uptake …

  … talking to application developers …
  … the APIs were developed by and for DNS people,
  not application developers …

Motivation – for a new DNS API

- From API Design considerations:

  … There are other DNS APIs available,
  but there has been very little uptake …

  … talking to application developers …
  … the APIs were developed by and for DNS people,
  not application developers …

- Goal

  … API design from talking to application developers …

  … create a natural follow-on to getaddrinfo() ...

Motivation – for a new DNS API

- Goal

  … API design from talking to application developers …

  … create a natural follow-on to getaddrinfo() …

  - http://www.vpnc.org/getdns-api/
  - Edited by Paul Hoffman
  - First publication April 2013
  - Updated in February 2014
    (after extensive discussion during implementation)

  - Creative Commons Attribution 3.0 Unported License

Motivation – for a new DNS API

- Goal

  … API design from talking to application developers …

  … create a natural follow-on to getaddrinfo() …

  - Implemented by Verisign Labs & NLnet Labs together
  - http://getdnsapi.net/
  - 0.1.0 release in February 2014, 0.1.1 in March,
  - 0.1.2 & 0.1.3 in June, 0.1.4 in September, 0.1.5 last Friday
  - **Node.js and Python bindings**

  - BSD 3-Clause License

Why this library (and not one of the others)

- Offers the full resolving package
  - Full recursion and DNSSEC … through libunbound
  - Access to all the resolved data … through ldns

Why this library (and not one of the others)

- Delivers a generic data structure … Response Dict
  - Lists, dicts, data, integers … ubiquitous in modern scripting languages
  - Very suitable for inspection
  - Trial and error style programming … resolve, have a look, decide how to proceed
  - Suitable for scripting language bindings … and those are very developer friendly.

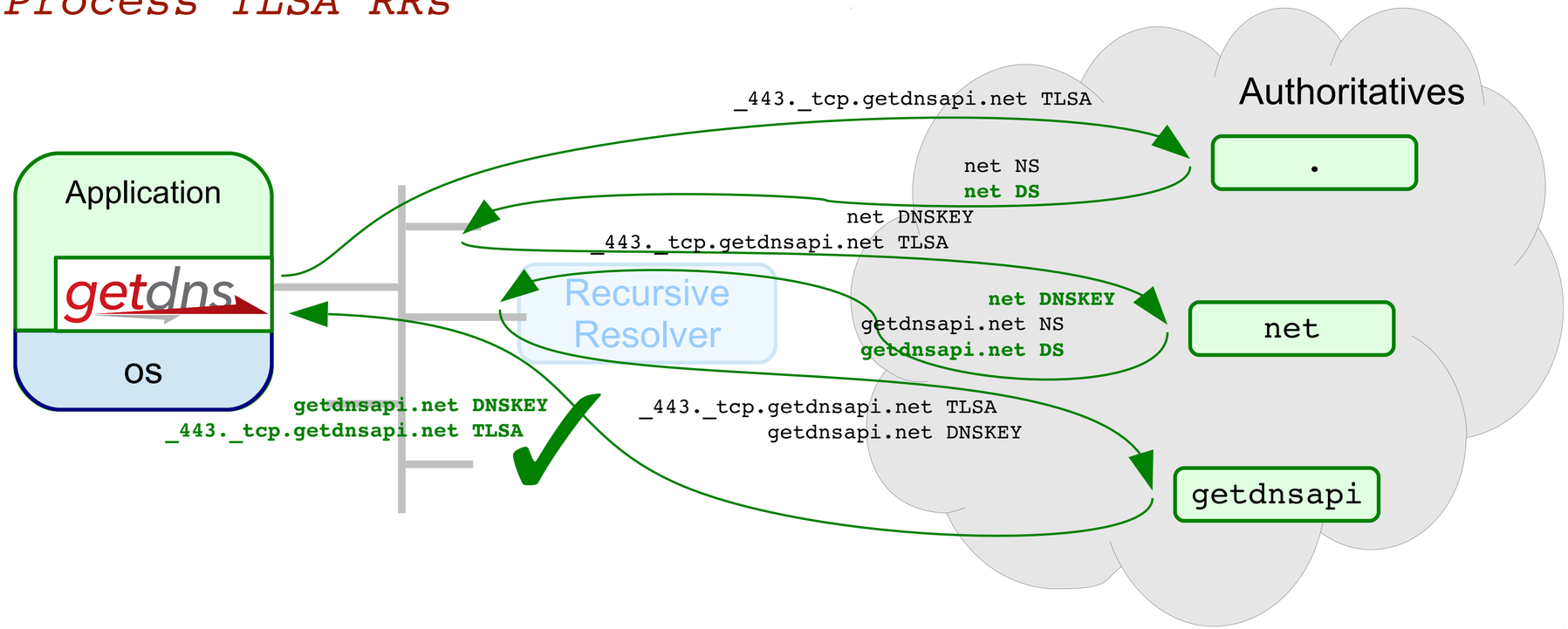
Hackathon with **Node.js** and **Python**. Ahead are **Go**, **Ruby**, **Perl** ...

Simple Functions – Full Recursion

```
from getdns import *

ctx = Context()
ext = { "dnssec_return_only_secure": GETDNS_EXTENSION_TRUE }
res = ctx.general( '_443._tcp.getdnsapi.net', GETDNS_RRTYPE_TLSA, ext)

if res['status'] = GETDNS_RESPSTATUS_GOOD:
        # Process TLSA RRs
```



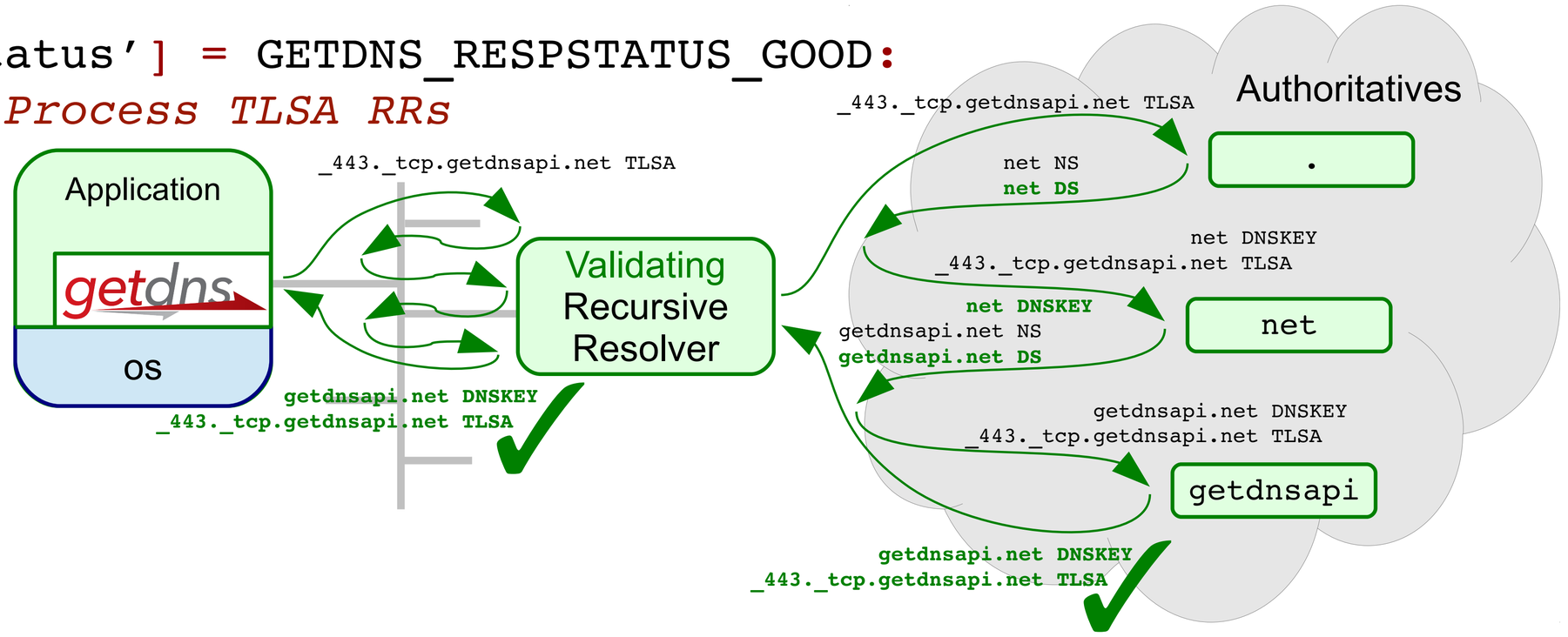
Simple Functions – Stub mode

```
from getdns import *

ctx = Context()
ctx.resolution_type = GETDNS_RESOLUTION_STUB

ext = { "dnssec_return_only_secure": GETDNS_EXTENSION_TRUE }
res = ctx.general( '_443._tcp.getdnsapi.net', GETDNS_RRTYPE_TLSA, ext)

if res['status'] = GETDNS_RESPSTATUS_GOOD:
        # Process TLSA RRs
```



Simple Functions – Fall back

```
# Determine if we have DNSSEC in stub mode

ctx = Context()
ctx.resolution_type = GETDNS_RESOLUTION_STUB

ext = { "dnssec_return_only_secure": GETDNS_EXTENSION_TRUE }
res = ctx.general('.', GETDNS_RRTYPE_DNSKEY, ext)
if res['status'] != GETDNS_RESPSTATUS_GOOD:
        # Fallback to do recursion ourselves
        ctx = Context()
```

Simple Functions – Fall back

```
# Determine if we have DNSSEC in stub mode

ctx = Context()
ctx.resolution_type = GETDNS_RESOLUTION_STUB

ext = { "dnssec_return_only_secure": GETDNS_EXTENSION_TRUE }
res = ctx.general('.', GETDNS_RRTYPE_DNSKEY, ext)
if res['status'] != GETDNS_RESPSTATUS_GOOD:
        # Fallback to do recursion ourselves
        ctx = Context()

# The root domain will never contain the wildcard.  Right?
elif ctx.general('*.', 0, ext)['status'] != GETDNS_RESPSTATUS_NO_NAME:
        # Some BIND 9.7 resolvers don't give the full NXDOMAIN proof
        # A none existent TLSA record will result in a BOGUS answer,
        # preventing the TLS connection to be setup altogether.

        # Fall back to do recursion ourselves
        ctx = Context()
```

Simple Functions – Fall back

```
# Correctly query for and process DANE records

res = ctx.general('_443._tcp.getdnsapi.net', GETDNS_RRTYPE_TLSA, ext)
if res['status'] == GETDNS_RESPSTATUS_GOOD:
        # Process TLSA RRs
        tlsas = [ answer for reply in res['replies_tree']
                         for answer in reply['answer']
                          if answer['type'] == GETDNS_RRTYPE_TLSA ]

        # Setup TLS only if the remote certificate (or CA)
        # matches one of the TLSA Rrs.
```

Simple Functions – Fall back

```
# Correctly query for and process DANE records

res = ctx.general('_443._tcp.getdnsapi.net', GETDNS_RRTYPE_TLSA, ext)
if res['status'] == GETDNS_RESPSTATUS_GOOD:
        # Process TLSA RRs
        tlsas = [ answer for reply in res['replies_tree']
                         for answer in reply['answer']
                          if answer['type'] == GETDNS_RRTYPE_TLSA ]

        # Setup TLS only if the remote certificate (or CA)
        # matches one of the TLSA Rrs.

elif res['status'] == GETDNS_RESPSTATUS_ALL_TIMEOUT or \
     res['status'] == GETDNS_RESPSTATUS_ALL_BOGUS_ANSWERS:
        # DON'T EVEN TRY!
```

Simple Functions – Fall back

```
# Correctly query for and process DANE records

res = ctx.general('_443._tcp.getdnsapi.net', GETDNS_RRTYPE_TLSA, ext)
if res['status'] == GETDNS_RESPSTATUS_GOOD:
        # Process TLSA RRs
        tlsas = [ answer for reply in res['replies_tree']
                         for answer in reply['answer']
                          if answer['type'] == GETDNS_RRTYPE_TLSA ]

        # Setup TLS only if the remote certificate (or CA)
        # matches one of the TLSA RRs.

elif res['status'] == GETDNS_RESPSTATUS_ALL_TIMEOUT or \
     res['status'] == GETDNS_RESPSTATUS_ALL_BOGUS_ANSWERS:
        # DON'T EVEN TRY!
else:
        # Conventional PKIX without DANE processing
```

The response dict

```
{
  "answer_type": GETDNS_NAMETYPE_DNS,
  "status": GETDNS_RESPSTATUS_GOOD,
  "canonical_name": <bindata of "www.getdnsapi.net.">,
  "just_address_answers":
  [ { "address_data": <bindata for 185.49.141.37>,
      "address_type": <bindata of "IPv4">
    },
    { "address_data": <bindata for 2a04:b900:0:100::37>,
      "address_type": <bindata of "IPv6">
    }
  ],
  "replies_full":
  [
    <bindata of 0x000081800001000200040010377777...>,
    <bindata of 0x000081800001000200040090377777...>
  ],
  "replies_tree":
  [
    { ... first reply ... },
    { ... second reply ... },
```

The response dict

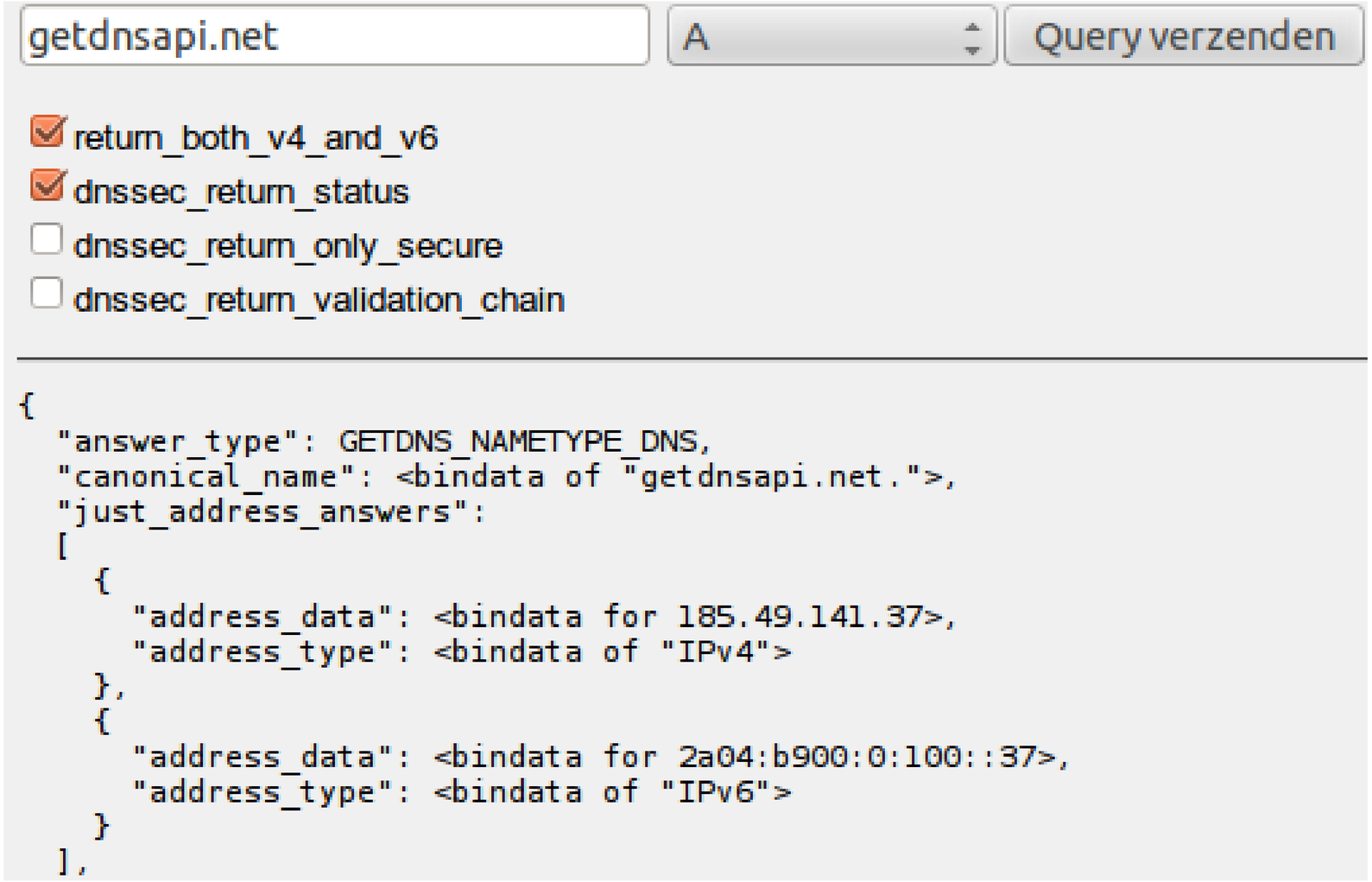
```
"replies_tree":
[
  { "header" : { "qdcount": 1, "ancount": 2, "rd": 1, "ra": 1,
                 "opcode": GETDNS_OPCODE_QUERY,
                 "rcode" : GETDNS_RCODE_NOERROR, ... },

    "question": { "qname" : <bindata for www.getdnsapi.net.>,
                  "qtype" : GETDNS_RRTYPE_A
                  "qclass": GETDNS_RRCLASS_IN, },

    "answer"  : [ { "name" : <bindata for www.getdnsapi.net.>,
                    "type" : GETDNS_RRTYPE_A,
                    "class": GETDNS_RRCLASS_IN,
                    "rdata": { "ipv4_address": <bindata for 185.49.141.37>,
                               "rdata_raw": <bindata of 0xb9318d25> },
                  }, ...
    "authority": [ ... ],
    "additional": [],
    "canonical_name": <bindata of "www.getdnsapi.net.">,
    "answer_type": GETDNS_NAMETYPE_DNS
  },
  { "header" : { ...
```

The response dict – Try It Yourself

- http://getdnsapi.net/query.html



Hands on getdns – Getting DNSSEC

- On a per query basis by setting extensions

- `dnssec_return_status`
  - Returns security assertion. Omits bogus answers
  -
```
{ # This is the response object
  "replies_tree":
  [
    { # This is the first reply
      "dnssec_status": GETDNS_DNSSEC_INSECURE,
```
  - `"dnssec_status"` can be `GETDNS_DNSSEC_SECURE`, `GETDNS_DNSSEC_INSECURE` or `GETDNS_DNSSEC_INDETERMINATE`

Hands on getdns – Getting DNSSEC

- On a per query basis by setting extensions

- `dnssec_return_status`
  - Returns security assertion. Omits bogus answers
  -
```
{ # This is the response object
  "replies_tree":
  [
    { # This is the first reply
      "dnssec_status": GETDNS_DNSSEC_INSECURE,
```
  - `"dnssec_status"` can be `GETDNS_DNSSEC_SECURE`, `GETDNS_DNSSEC_INSECURE` or `GETDNS_DNSSEC_INDETERMINATE`

- Our implementation provides:

```
void getdns_context_set_return_dnssec_status(context);
```

Hands on getdns – Getting DNSSEC

- `dnssec_return_only_secure` The DANE extension
  - Returns security assertion. Omits bogus and insecure answers
  -
```
{ # This is the response object
  "replies_tree": [],
  "status" : GETDNS_RESPSTATUS_NO_SECURE_ANSWERS,
```

Hands on getdns – Getting DNSSEC

- dnssec_return_validation_chain

```
- { # Response object
    "validation_chain":
    [ { "name" : <bindata for .>, "type": GETDNS_RRTYPE_DNSKEY, ... },
      { "name" : <bindata for .>, "type": GETDNS_RRTYPE_DNSKEY, ... },

      { "name" : <bindata for .>, "type": GETDNS_RRTYPE_RRSIG,
        "rdata": { "signers_name": <bindata for .>,
                   "type_covered": GETDNS_RRTYPE_DNSKEY, ... }, ... },

      { "name" : <bindata for net.>, "type": GETDNS_RRTYPE_DS, ... },
      { "name" : <bindata for net.>, "type": GETDNS_RRTYPE_RRSIG,
        "rdata": { "signers_name": <bindata for .>,
                   "type_covered": GETDNS_RRTYPE_DS, ... }, ... },
```

Hands on getdns – Getting DNSSEC

- dnssec_return_validation_chain
  - { # Response object
    "validation_chain":
    [{ "name" : <bindata for .>, "type": GETDNS_RRTYPE_DNSKEY, ... },
      { "name" : <bindata for .>, "type": GETDNS_RRTYPE_DNSKEY, ... },

      { "name" : <bindata for .>, "type": GETDNS_RRTYPE_RRSIG,
        "rdata": { "signers_name": <bindata for .>,
                   "type_covered": GETDNS_RRTYPE_DNSKEY, ... }, ... },

      { "name" : <bindata for net.>, "type": GETDNS_RRTYPE_DS, ... },
      { "name" : <bindata for net.>, "type": GETDNS_RRTYPE_RRSIG,
        "rdata": { "signers_name": <bindata for .>,
                   "type_covered": GETDNS_RRTYPE_DS, ... }, ... },
  - Can be combined with dnssec_return_status and dnssec_return_only_secure
  - No replies are omitted!
    Only now “dnssec_status” can be GETDNS_DNSSEC_BOGUS

Hands on getdns – Async DNS lookups

```
getdns_return_t getdns_general(
    getdns_context          *context,
    const char              *name,
    uint16_t                 request_type,
    getdns_dict             *extensions,
    void                    *userarg,
    getdns_transaction_t    *transaction_id,
    getdns_callback_t        callbackfn
);
```

- **context** contains configuration parameters
  - Stub or recursive modus operandi, timeout values, root-hints, forwarders, trust anchor, search path (+ how to evaluate (not implemented yet) etc.)
- **context** contains the resolver cache

Hands on getdns – Async DNS lookups

```
getdns_return_t getdns_general(
    getdns_context          *context,
    const char              *name,
    uint16_t                 request_type,
    getdns_dict             *extensions,
    void                    *userarg,
    getdns_transaction_t    *transaction_id,
    getdns_callback_t        callbackfn
);
```

- `context` contains configuration parameters
- **`name`** and **`request_type`** the name and type to lookup

Hands on getdns – Async DNS lookups

```
getdns_return_t getdns_general(
    getdns_context          *context,
    const char              *name,
    uint16_t                 request_type,
    getdns_dict             *extensions,
    void                    *userarg,
    getdns_transaction_t    *transaction_id,
    getdns_callback_t        callbackfn
);
```

- `context` contains configuration parameters
- `name` and `request_type` the name and type to lookup
- **`extensions`** additional parameters specific for this lookup
  - `return_both_v4_and_v6`, `dnssec_return_status`, `specify_class`
  - `add_opt_parameter`

Hands on getdns – Async DNS lookups

```
getdns_return_t getdns_general(
    getdns_context          *context,
    const char              *name,
    uint16_t                 request_type,
    getdns_dict             *extensions,
    void                    *userarg,
    getdns_transaction_t    *transaction_id,
    getdns_callback_t        callbackfn
);
```

- `context` contains configuration parameters
- `name` and `request_type` the name and type to lookup
- `extensions` additional parameters specific for this lookup
- **`userarg`** is passed in on the call to **`callbackfn`**
- **`transaction_id`** is set to a unique value that is also passed in on the call to **`callbackfn`**

Hands on getdns – Async DNS lookups

```
getdns_return_t getdns_general(
    getdns_context          *context,
    const char              *name,
    uint16_t                 request_type,
    getdns_dict             *extensions,
    void                    *userarg,
    getdns_transaction_t    *transaction_id,
    getdns_callback_t        callbackfn
);

typedef void (*getdns_callback_t)(
    getdns_context          *context,
    getdns_callback_type_t   callback_type,
    getdns_dict             *response,
    void                    *userarg,
    getdns_transaction_t     transaction_id
);
// callback_type = complete, cancel, timeout or error
```

Hands on getdns – Synchronous lookups

```
getdns_return_t getdns_general(
    getdns_context          *context,
    const char              *name,
    uint16_t                 request_type,
    getdns_dict             *extensions,
    void                    *userarg,
    getdns_transaction_t    *transaction_id,
    getdns_callback_t        callbackfn
);

getdns_return_t getdns_general_sync(
    getdns_context          *context,
    const char              *name,
    uint16_t                 request_type,
    getdns_dict             *extensions,
    getdns_dict            **response
);
```

Hands on getdns – Address lookups

```
getdns_return_t getdns_address(
    getdns_context          *context,
    const char              *name,
    getdns_dict             *extensions,
    void                    *userarg,
    getdns_transaction_t    *transaction_id,
    getdns_callback_t        callbackfn
);
```

- **`getdns_address`** also lookups in other name systems
  - local files, WINS, mDNS, NIS (not implemented yet)
- When `name` is found in the DNS, **`getdns_address`** returns both IPv4 and IPv6
  - i.e. the `return_both_v4_and_v6` extension is set by default

Hands on getdns – Reverse lookups

```
getdns_return_t getdns_hostname(
    getdns_context          *context,
    getdns_dict             *address,
    getdns_dict             *extensions,
    void                    *userarg,
    getdns_transaction_t    *transaction_id,
    getdns_callback_t        callbackfn
);
```

- With **address**:

```
{
  "address_type": <bindata of "IPv4">
  "address_data": <bindata for 185.49.141.37>,
}
```

will lookup `37.141.49.185.in-addr.arpa PTR`

Hands on getdns – Data structures

```
typedef struct getdns_dict getdns_dict;
typedef struct getdns_list getdns_list;
typedef struct getdns_bindata {  size_t  size;
                                 uint8_t *data; } getdns_bindata;
```

- Used to represent extensions, addresses and response objects.
- `char *getdns_pretty_print_dict(const getdns_dict *dict);`

```
{
  "answer_type": GETDNS_NAMETYPE_DNS,
  "status": GETDNS_RESPSTATUS_GOOD,
  "canonical_name": <bindata of "www.getdnsapi.net.">,
  "just_address_answers":
  [ { "address_data": <bindata for 185.49.141.37>,
      "address_type": <bindata of "IPv4">
    }
  ],
  "replies_full": [ <bindata of 0x000818000010002004...> ],
  "replies_tree": [ { … first reply …  } ],
```

Hands on getdns – Asynchronous

- From the getdns API specification:

  1.8 Event-driven Programs

  … Each implementation of the DNS API will specify an extension function that tells the DNS context which event base is being used...

- `libevent`

  Include : `#include <getdns/getdns_ext_libevent.h>`
  Use : `getdns_extension_set_libevent_base(context, base);`
  Link : -lgetdns -lgetdns_ext_event

```
struct event_base *base = event_base_new();
getdns_extension_set_libevent_base(context, base);

getdns_address(context, "getdnsapi.net", 0, 0, 0, callback);

event_base_dispatch(base);
event_base_free(base);
```

Hands on getdns – Asynchronous

- libevent

  Include : `#include <getdns/getdns_ext_libevent.h>`
  Use : `getdns_extension_set_libevent_base(context, base);`
  Link : -lgetdns -lgetdns_ext_event

- libev

  Include : `#include <getdns/getdns_ext_libev.h>`
  Use : `getdns_extension_set_libev_loop(context, loop);`
  Link : -lgetdns -lgetdns_ext_evt

- libuv

  Include : `#include <getdns/getdns_ext_libuv.h>`
  Use : `getdns_extension_set_libuv_loop(context, loop);`
  Link : -lgetdns -lgetdns_ext_uv

Hands on getdns – Asynchronous

- `void getdns_context_run(getdns_context *context);`

```
/* Call the event loop */
struct timeval tv;

while (getdns_context_get_num_pending_requests(context, &tv) > 0) {

        int fd = getdns_context_fd(context);
        fd_set read_fds;

        FD_ZERO(&read_fds);
        FD_SET(fd, &read_fds);
        select(fd + 1, &read_fds, NULL, NULL, &tv);

        if (getdns_context_process_async(context) != GETDNS_RETURN_GOOD) {
                // context destroyed
                break;
        }
}
```

Hands on getdns – Installation Instructions

Hands on getdns – Walk reverse IPv6 address space

- `7.3.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.9.b.4.0.a.2.ip6.arpa. IN PTR getdnsapi.net`
- 32 nibles 16^{32} or 2^{128} = 340282366920938463463374607431768211456 possibilities
- But empty non terminals (there is something higher up) return `NOERROR` (but no answer either) instead of `NXDOMAIN`

Hands on getdns – Walk reverse IPv6 address space

0.ip6.arpa.	NXDOMAIN	**0.2.ip6.arpa.**	**NOERROR**
1.ip6.arpa.	NXDOMAIN	1.2.ip6.arpa.	NXDOMAIN
2.ip6.arpa.	**NOERROR**	2.2.ip6.arpa.	NXDOMAIN
3.ip6.arpa.	NXDOMAIN	3.2.ip6.arpa.	NXDOMAIN
4.ip6.arpa.	NXDOMAIN	**4.2.ip6.arpa.**	**NOERROR**
5.ip6.arpa.	NXDOMAIN	5.2.ip6.arpa.	NXDOMAIN
6.ip6.arpa.	NXDOMAIN	**6.2.ip6.arpa.**	**NOERROR**
7.ip6.arpa.	NXDOMAIN	7.2.ip6.arpa.	NXDOMAIN
8.ip6.arpa.	NXDOMAIN	**8.2.ip6.arpa.**	**NOERROR**
9.ip6.arpa.	NXDOMAIN	9.2.ip6.arpa.	NXDOMAIN
a.ip6.arpa.	NXDOMAIN	**a.2.ip6.arpa.**	**NOERROR**
b.ip6.arpa.	NXDOMAIN	b.2.ip6.arpa.	NXDOMAIN
c.ip6.arpa.	NXDOMAIN	**c.2.ip6.arpa.**	**NOERROR**
d.ip6.arpa.	NXDOMAIN	d.2.ip6.arpa.	NXDOMAIN
e.ip6.arpa.	NXDOMAIN	e.2.ip6.arpa.	NXDOMAIN
f.ip6.arpa.	NXDOMAIN	f.2.ip6.arpa.	NXDOMAIN

Hands on getdns – Walk reverse IPv6 address space

- Beware!
- A wildcard will return NOERROR too.
- But we can test, because only a wildcard will match *!
- The wildcard is an anti reverse walking defense mechanism

Hands on getdns – Walk reverse IPv6 address space – javascript with node

- All queries are schedules simultaneously

```
var getdns = require('getdns');

function callback(err, result)
{
        console.log(err ? Err :
            result.canonical_name + ': ' +
            JSON.stringify(result.just_address_answers));
}


ctx = getdns.createContext();
ctx.getAddress('getdnsapi.net', callback);
ctx.getAddress('verisignlabs.com', callback);
ctx.getAddress('sinodun.com', callback);
ctx.getAddress('nomountain.net', callback);
ctx.getAddress('ripe69.ripe.net', callback);
```

Hands on getdns – Walk reverse IPv6 address space – javascript with node

- But when to destroy the context?

```
willem@bonobo:~/ripe69/walk6$ node example-1.js

getdnsapi.net.: ["185.49.141.37","2a04:b900:0:100::37"]
nomountain.net.: ["208.113.197.240","2607:f298:5:104b::b80:8f9e"]
ripe69.ripe.net.: ["193.0.19.34","2001:67c:2e8:11::c100:1322"]
verisignlabs.com.: ["72.13.58.64"]
sinodun.com.: ["88.98.24.67"]
[1414839133] libunbound[6180:0] error: tube msg write failed: Broken pipe

willem@bonobo:~/ripe69/walk6$
```

- Does not happen in stub mode (because no process is spawn)

Hands on getdns – Walk reverse IPv6 address space – javascript with node

- `ctx.destroy()` at the bottom would cancel all outstanding queries
- So, track the queries manually

```
function callback(err, result)
{
        console.log(err ? err : result.canonical_name + ': ' +
                                 JSON.stringify(result.just_address_answers));

        if (--num_queries == 0)
                ctx.destroy();
}
var num_queries = 5;
ctx = getdns.createContext();
ctx.getAddress('getdnsapi.net', callback);
ctx.getAddress('verisignlabs.com', callback);
ctx.getAddress('sinodun.com', callback);
ctx.getAddress('nomountain.net', callback);
ctx.getAddress('ripe69.ripe.net', callback);
```

Hands on getdns – Walk reverse IPv6 address space – javascript with node

- Or collect results using (for example) the `async` module

```
var getdns = require('getdns');
var async = require('async');

ctx = getdns.createContext();
async.parallel([ 'getdnsapi.net', 'verisignlabs.com', 'sinodun.com'
               , 'nomountain.net', 'ripe69.ripe.net'].map(function(name) {
                   return function (result_cb) {
                       ctx.getAddress(name, function(err, result) {
                           result_cb(err, !result ? Null :
                                          result.canonical_name + ': '+
                                          result.just_address_answers);
                       });
                   }
               }), function(err, result) {
                   console.log(err ? err : result);
                   ctx.destroy(); // Everything is done
               });
```

Hands on getdns – Walk reverse IPv6 address space – javascript with node

- Or collect results using (for example) the `async` module

```
willem@bonobo:~/ripe69/walk6$ node example-3.js
[ 'getdnsapi.net.: 185.49.141.37,2a04:b900:0:100::37',
  'verisignlabs.com.: 72.13.58.64',
  'sinodun.com.: 88.98.24.67',
  'nomountain.net.: 208.113.197.240,2607:f298:5:104b::b80:8f9e',
  'ripe69.ripe.net.: 193.0.19.34,2001:67c:2e8:11::c100:1322' ]
willem@bonobo:~/ripe69/walk6$
```

Hands on getdns – Walk reverse IPv6 address space – javascript with node

- Depth first (so we get results more quickly)
- Make sure there is no wildcard
- Serialize the async way, schedule what to do next with the `next` callback

```
var getdns = require('getdns');
var async = require('async');
var ctx = getdns.createContext({'stub':true});

check_wildcard_and_walk('ip6.arpa.' function(){ctx.destroy()});
function check_wildcard_and_walk(name, next) {
    ctx.lookup('*.' + name, 0, function(err, result) {
        if (result &&
            result.replies_tree[0].header.rcode == getdns.RCODE_NXDOMAIN) {

            // Schedule 16 lookups for [0..f].<name> and process result
        }
    });
}
```

Hands on getdns – Walk reverse IPv6 address space – javascript with node

```
// Schedule 16 lookups for [0..f].<name> and process results
async.parallel(['0','1','2','3','4','5','6','7','8','9','a','b'
                ,'c','d','e','f'].map(function(digit) {
    return function(cb_result) {
        var new_name = digit + '.' + name;
        ctx.lookup( new_name, getdns.RRTYPE_PTR
                  , function(err, result) {

            cb_result(null, result && getdns.RCODE_NOERROR ==
              result.replies_tree[0].header.rcode ?
              { 'n': new_name
              , 'a': result.replies_tree[0].answer} : null);
        });
    }
}), function (err, result) {
    // Process results

    console.log(result);
    next();
});
```

Hands on getdns – Walk reverse IPv6 address space – javascript with node

```
willem@bonobo:~/ripe69/walk6$ node example-5.js
[ null,
  null,
  { n: '2.ip6.arpa.', a: [] },
  null,
  null,
  null,
  null,
  null,
  null,
  null,
  null,
  null,
  null,
  null,
  null,
  null,
  null,
  null,
  null ]
```

Hands on getdns – Walk reverse IPv6 address space – javascript with node

```
function process_results(results, next) {
    while (results && results.length) {
        var result = results.shift()
        if (result) {
            if (result.a.length) {
                console.log(result.a[0].name + ': '
                            result.a[0].rdata.ptrdname);
            } else if (result.n.length < 73) {
                check_wildcard_and_walk( result.n, function({
                    process_results(results, next)});
                return;
            }
        }
    }
    next();
}
```

- Turn direction with `results.pop()` instead of `results.shift()`

Hands on getdns – Walk reverse IPv6 address space – javascript with node

```
willem@bonobo:~/ripe69/walk6$ node example-6.js
0.8.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: l7lb.cloud.wide.ad.jp.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: ns1.v6.wide.ad.jp.
3.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: nons.wide.ad.jp.
4.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: ns.wide.ad.jp.
5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: www2.wide.ad.jp.
6.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: member.wide.ad.jp.
7.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: mail.wide.ad.jp.
8.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: ns.tokyo.wide.ad.jp.
9.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: ns.nara.wide.ad.jp.
a.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: ftp-netbsd.tokyo.wide.ad.jp.
c.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: moca.wide.ad.jp.
d.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: www3.wide.ad.jp.
f.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: ns-wide.wide.ad.jp.
1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: ns.fujisawa.wide.ad.jp.
1.2.0.0.0.0.8.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: ftp.nara.wide.ad.jp.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: cisco2.notemachi.wide.ad.jp.
1.0.0.0.0.0.8.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: lo-0.hitachi2.nara.wide.ad.jp.
2.0.0.0.0.0.8.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: lo-0-v0.juniper2.nara.wide.ad.jp.
3.0.0.0.0.0.8.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: lo-0.juniper4.nara.wide.ad.jp.
4.0.0.0.0.0.8.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: lo-0.juniper5.nara.wide.ad.jp.
5.0.0.0.0.0.8.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: lo0-1.juniper6.nara.wide.ad.jp.
1.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: juniper3.fujisawa.wide.ad.jp.
3.0.0.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: juniper2.fujisawa.wide.ad.jp.
1.0.0.0.0.0.4.1.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: pc2.komatsu.wide.ad.jp.
2.0.0.0.0.0.4.1.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: juniper1.komatsu.wide.ad.jp.
3.0.0.0.0.0.4.1.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: cisco2.komatsu.wide.ad.jp.
4.0.0.0.0.0.4.1.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: brocade1.komatsu.wide.ad.jp.
1.0.0.0.0.0.8.1.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: hitachi1.otemachi.wide.ad.jp.
2.0.0.0.0.0.8.1.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.: pc6.otemachi.wide.ad.jp.
```

Hands on getdns – Walk reverse IPv6 address space – javascript with node

- Now all lookups were serialized (with `next` callback)
- The outstanding parallel lookups were 1 (wildcard), followed by 16 (for every nibble), followed by 1, followed by 16, followed by 1, etc.
- Without the serialization (i.e. not forwarding the next callback)
  we would have complete parallel descent resulting in thousands of parallel queries
- This needs to be restrained to prevent the network to get clogged and memory exhaustion. (i.e. query rate limiting).

Hands on getdns – Setup DANE authenticated TLS session – python example

- Ingredients:

```
from getdns import *
from M2Crypto import SSL, X509
import sys
import socket
import hashlib
GETDNS_RESPSTATUS_ALL_BOGUS_ANSWERS = 904


if len(sys.argv) > 1:
        hostname = sys.argv[1]
        port = int(sys.argv[2]) if len(sys.argv) > 2 else 443
else:
        print('%s <hostname> [ <port> ]' % sys.argv[0])
        sys.exit(0)
```

- `GETDNS_RESPSTATUS_ALL_BOGUS_ANSWERS` is not in the python bindings yet...

Hands on getdns – Setup DANE authenticated TLS session – python example

- We have seen this before...

```
# Determine if we have DNSSEC in stub mode
# First initialize a context in stub mode
ctx = Context()
ctx.resolution_type = GETDNS_RESOLUTION_STUB

ext = { "dnssec_return_only_secure": GETDNS_EXTENSION_TRUE }
res = ctx.general('.', GETDNS_RRTYPE_DNSKEY, ext)
if res['status'] != GETDNS_RESPSTATUS_GOOD:
        # Fallback to do recursion ourselves
        ctx = Context()

# Root domain will never contain a wildcard.  Right?
elif ctx.general('*.', 0, ext)['status'] != GETDNS_RESPSTATUS_NO_NAME:
        # Some BIND 9.7 resolvers don't give the full NXDOMAIN proof
        # A none existent TLSA record will result in a BOGUS answer,
        # preventing the TLS connection to be setup alltogether.

        # Fall back to do recursion ourselves
        ctx = Context()
```

Hands on getdns – Setup DANE authenticated TLS session – python example

- And this too...

```
# Correctly query and process DANE records
res = ctx.general('_%d._tcp.%s' % (port, hostname), GETDNS_RRTYPE_TLSA, ext)
if res['status'] == GETDNS_RESPSTATUS_GOOD:
        # Process TLSA Rrs
        tlsas = [ answer for reply in res['replies_tree']
                         for answer in reply['answer']
                          if answer['type'] == GETDNS_RRTYPE_TLSA ]
elif res['status'] == GETDNS_RESPSTATUS_ALL_TIMEOUT:
        print('Network error trying to get DANE records for %s' % hostname)
        sys.exit(-1);
elif res['status'] == GETDNS_RESPSTATUS_ALL_BOGUS_ANSWERS:
        print('DANE records for %s were BOGUS' % hostname)
        sys.exit(-1);
else:
        tlsas = None
        # Conventional PKIX without DANE processing
```

Hands on getdns – Setup DANE authenticated TLS session – python example

```
ca_cert = None
def get_ca(ok, store):
        global ca_cert
        if store.get_current_cert().check_ca():
                ca_cert = store.get_current_cert()
        return ok
```

- We also need to find the CA vouching for the connection for PKIX-TA and DANE-TA certificate usages.
- This is not very straight forward with M2Crypto

```
# Now TLS connect to each address for the hostname and verify the cert (or CA)
for address in ctx.address(hostname)['just_address_answers']:
        sock = socket.socket(socket.AF_INET
            if address['address_type'] == 'IPv4' else socket.AF_INET6,
            socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

        print('Connecting to %s' % address['address_data']);
        ssl_ctx = SSL.Context()
        ssl_ctx.load_verify_locations(capath = '/etc/ssl/certs')
        ssl_ctx.set_verify(SSL.verify_none, 10, get_ca)
        connection = SSL.Connection(ssl_ctx, sock=sock)
```

Hands on getdns – Setup DANE authenticated TLS session – python example

- Just two more domestic affairs...

```
# set TLS SNI extension if available in M2Crypto on this platform
# Note: the official M2Crypto release does not yet (as of late 2014)
# have support for SNI, sigh, but patches exist.
Try:
        connection.set_tlsext_host_name(hostname)
except AttributeError:
        pass


# Per https://tools.ietf.org/html/draft-ietf-dane-ops, for DANE-EE
# usage, certificate identity checks are based solely on the TLSA
# record, so we ignore name mismatch conditions in the certificate.
Try:
        connection.connect((address['address_data'], port))
except SSL.Checker.WrongHost:
        pass
```

Hands on getdns – Setup DANE authenticated TLS session – python example

- Without TLSA RRs, fall back to old fashioned PKIX

```
if not tlsas:
        print( 'No TLSAS. Regular PKIX validation '
             + ('succeeded' if connection.verify_ok() else 'failed'))
        continue # next address
```

- But with TLSA RRs, try each TLSA RR in turn. First one matching makes the day!
- Note that for PKIX-TA (0) and DANE-TA (2) we set cert to the CA certificate.

```
cert = connection.get_peer_cert()
TLSA_matched = False
for tlsa in tlsas:
        rdata = tlsa['rdata']
        if rdata['certificate_usage'] in (0, 2):
                cert = ca_cert
```

Hands on getdns – Setup DANE authenticated TLS session – python example

- Put `certdata` into selector and the matching_type shape

```
if rdata['selector'] == 0:
        certdata = cert.as_der()
elif rdata['selector'] == 1:
        certdata = cert.get_pubkey().as_der()
else:
        raise ValueError('Unkown selector')


if rdata['matching_type'] == 1:
        certdata = hashlib.sha256(certdata).digest()
elif rdata['matching_type'] == 2:
        certdata = hashlib.sha512(certdata).digest()
else:
        raise ValueError('Unkown matching type')
```

Hands on getdns – Setup DANE authenticated TLS session – python example

- And see if `certdata` matches the TLSA's certificate association data
- With usage types 0 and 1 (PKIX-TA and PKIX-EE) we need to PKIX validate too (i.e. `connection.verify_ok()`)

```
                if str(certdata) == str(rdata['certificate_association_data'])\
                and (rdata['certificate_usage'] > 1 or connection.verify_ok()):
                        TLSA_matched = True
                        print('DANE validated successfully')
                        break # from “for tlsa in tlsas:” (first one wins!)
        if not TLSA_matched:
                print('DANE validation failed')
```

Hands on getdns – Setup DANE authenticated TLS session – python example

- Our DANE example in action:

```
willem@bonobo:~/ripe69/dane$ ./example-1.py getdnsapi.net

Connecting to 185.49.141.37
DANE validated successfully
Connecting to 2a04:b900:0:100::37
DANE validated successfully

willem@bonobo:~/ripe69/dane$ ./example-1.py ripe69.ripe.net

Connecting to 193.0.19.34
No TLSAS. Regular PKIX validation succeeded
Connecting to 2001:67c:2e8:11::c100:1322
No TLSAS. Regular PKIX validation succeeded
```