Data Communication and Networks	Project 4 (Bonus): Simple DNSSEC Client
CSCI-351 Fall 2019	November 12, 2019

This project is due at 11:59:59pm on December 10, 2019 and is worth 40% of your project scores. You must complete it with a partner. You may only complete it alone or in a group of three if you have the instructor's explicit permission to do so for this project.

Note that there is no milestone deadline for this project.

1 Description

The Domain Name System (DNS) is a hierarchical system for converting domain names (e.g., `www.google.com`) to Internet Protocol (IP) addresses (e.g., `209.85.129.99`). DNS is often referred to as a "phone book" for the Internet, translating human-friendly domain names into machine-friendly IP addresses.

However, DNS has long been fraught with security issues such as DNS spoofing and cache poisoning. To solve these limitations, DNS Security Extensions (DNSSEC) were introduced nearly two decades ago. DNS's Security Extensions (DNSSEC) allows clients and resolvers to verify that DNS responses have not been forged or modified in-flight. DNSSEC uses a public key infrastructure (PKI) to achieve this integrity, without which users can be subject to a wide range of attacks. In this project, you will implement a DNSSEC client program, which handles DNS requests with the DNSSEC option and verify if the DNS response is valid or not.

2 Background

DNSSEC provides integrity for DNS records using three primary record types[1]:

`DNSKEY` records, which are public keys used in DNSSEC. Typically, each zone uses two `DNSKEY` records to sign DNS records, as discussed below.

`RRSIG` (Resource Record Signature) records, which are cryptographic signatures of other records. Each `RRSIG` is a signature over all records of a given type for a certain name; this set is called an RRSet. For example, all `A` records for `example.org` will be authenticated by a single `RRSIG` (i.e., the `example.org A RRSIG`). Each `RRSIG` is created using the private key that matches a public key in `DNSKEY` records.

`DS` (Delegation Signer) records, which are essentially hashes of `DNSKEY`s. These records are uploaded to the parent zone, which establishes the chain of trust reaching up to the root zone. The `DS` records in the parent zone are authenticated using `RRSIG`s, just like any other record type.

Most Internet hosts do not do iterative DNS lookups themselves, but instead are configured to use a local DNS *resolver*. When a host wishes to look up a domain name, it sends a query to its resolver;

[1] There are other record types for expressing the non-existence of records (`NSEC` and `NSEC3` records) and for a child zone to request an update to their `DS` record (`CDNSKEY` and `CDS` records). As these are not integral to our study, we do not discuss them in detail.

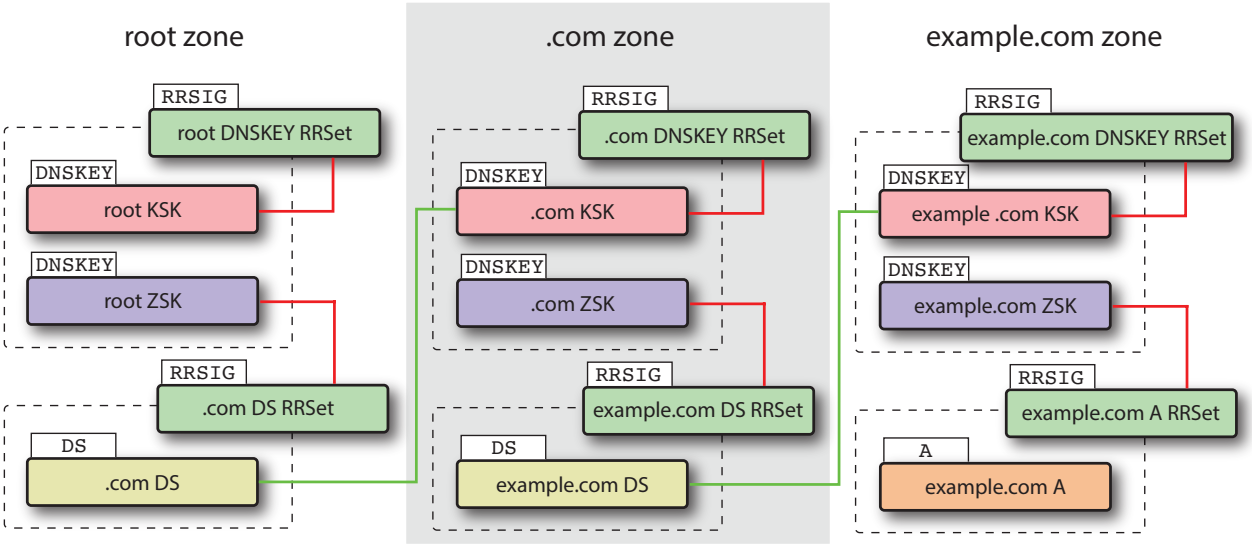

Figure 1: Overview of DNSSEC records necessary to validate `example.com`'s `A` record. Each `RRSIG` is the signature of a record set (dashed lines) verified with a `DNSKEY` (red lines). Each `DS` record is the hash of a child zone's KSK (green lines).

the resolver then iteratively determines the authoritative name server for that domain and obtains the record. If the resolver supports DNSSEC, it will also fetch all DNSSEC records (`DNSKEY`s and `RRSIG`s) necessary to validate the record. Finally, the resolver returns the (validated) record back to the requesting host. It is important to note that resolvers make heavy use of caching, and will typically avoid re-requesting any unexpired records that have already been obtained.

DNSSEC is designed to be backwards-compatible, while enabling resolvers who support DNSSEC to specifically request DNSSEC records. A resolver indicates that it would like DNSSEC records by setting the `DO` ("DNSSEC OK") bit in its DNS request. If the responding authoritative name server has `RRSIG`s corresponding to the record type of the request, it is obligated to include them. Should the resolver also need `DNSKEY`s to validate the record, it may need to request them separately.

DNSSEC keys Each zone in DNSSEC typically has *two* public/private key pairs: one called a Key Signing Key (KSK) and another called a Zone Signing Key (ZSK). Typically, the KSK is used only to produce `RRSIG`s for `DNSKEY` records (hence the name). In contrast, the ZSK is used to produce the `RRSIG`s for all other record types.

There is no key revocation (apart from root authorities) in the DNSSEC PKI; Rather, to mitigate potential effects of key compromise, ZSKs are intended to be *rolled over* (i.e., replaced) daily or weekly, and the KSKs monthly or yearly (the intention is that the KSK can be stored separately from, and in a safer location than, the ZSK).

Validating a DNSSEC record The DNSSEC PKI is rooted at the KSK of the DNS root zone. This KSK is well-known by DNSSEC-aware resolvers. Validating a DNS response starts at the root and continues down the DNS hierarchy: A resolver begins by using the KSK to validate the root `DNSKEY` `RRSIG`, which validates the root zone's ZSK. The resolver can then validate the child zone's `DS` record (and thereby the child zone's KSK) using the `RRSIG` for the `DS` records in the root zone,

as this is signed with the root zone's ZSK. This process continues until the record in question is authenticated. Figure 1 shows example records and how they are related.

3 Requirements

Your will write a DNSSEC client program which, given a name to query for and a DNS server to query will:

- Construct a DNS query packet with DNSSEC option (`DO` bit) for the specified name
- Send the DNS queries to the specified DNS server using UDP
- Wait for the response to be returned from the server
- Interpret the responses, *verify* them and output the result to `STDOUT`

Your client must support the following features:

- Queries for `A`, `DNSKEY`, `RRSIG`, and `DS` records.
- Verify all records using crypto libraries.

You should be strict; if the returned message does not conform to the DNS specification, you should assert an error. You may receive other packets that are not responses to your query; you should ignore these and continue to wait for a response to your query. Remember that network-facing code should be written defensively. We will test your code by sending corrupted packets to your client; you should handle these errors gracefully and *not* crash.

Also, you need to send multiple DNSSEC-relavent queries internally to return and validate the `A` records.

4 Your client program

For this project, you can choose your language. You may use any third party libraries for encryption and decryption purposes, but are *not* allowed to use any DNS libraries in your project (e.g., `getaddrinfo` or `gethostbyname`). **You must construct the DNS request packet yourself, and interpret the reply yourself based on your project2.** Also, your code MUST work on glados. If you used third party libraries you may need to setup your experiment environment using virutal environment tools such as `virtualenv`. In such case, your `runme.sh` script file `MUST` generate your environment as well. If your code does not compile or run on the glados server then it is your fault.

4.1 Input and output

The command line syntax for your client is given below. The client program takes command line argument of the domain name to interpret and the IP address of the domain server to query. The syntax for launching your program is therefore:

```
./351dnsclient @<server:port> <domain-name> <record>
```

`server` (Required) The IP address of the DNS server, in `a.b.c.d` format.

`port` (Optional) The UDP port number of the DNS server. Default value: 53.

`domain-name` (Required) The name to query for

`record` (Required) The DNS record to query for, which can be either

- A: `A` records
- DNSKEY: `DNSKEY` records
- DS: `DS` records

After sending the request, your client should wait for a reply for 5 seconds. If no reply is heard within this time window, you should exit indicating that a timeout occurred, by printing out the `NORESPONSE` message.
To help us compare with the reference solution, your code must print out the packet to standard output by implementing `dump_packet` function. For example, if you have your packet in `buf` and it is `size` bytes, long, you should call `dump_packet(buf, size)` right before you call `sendto()`. You should see output like

```
[0000]   68 78 01 00 00 01 00 00   00 00 00 00 03 77 77 77   hx...... .....www
[0010]   06 67 6F 6F 67 6C 65 03   63 6F 6D 00 00 01 00 01   .google. com.....
```

Your client must then wait for a response from the server, and print the result to standard output using the following format:

```
IP <tab> <IP address(es)> <tab> <RRSIG record(s)> <VALID|INVALID>
DS <tab> <DS record(s)> <tab> <RRSIG record(s)> <VALID|INVALID>
DNSKEY <tab> <DNSKEY record> <tab> <RRSIG> <VALID|INVALID>
ERROR <tab> <NONE|MISSING-DS|MISSING-RRSIG|EXPIRED-RRSIG|INVALID-RRSIG|OTHERS>
NOTFOUND
NORESPONSE
```

If the response to a query contains multiple answers (such as multiple IP addresses or aliases), your client must print an `IP` line for each one of these. If the requested name does not exist, your client must print a `NOTFOUND` line. If no response is ever received from the server (i.e., you've waited 5 seconds and not received anything), your client must print a `NORESPONSE` line. Finally, if any other error occurs, your client should print an `ERROR` line containing a description of the error either of NONE (no errors), MISSING-DS (no `DS` records in the parent zone), MISSING-RRSIG (no `RRSIG` records for the A record, EXPIRED-RRSIG (`RRSIG` records are expired), INVALID-RRSIG (The signatures in `RRSIG` records are invalid), and OTHERS (The other non-dnssec related errors such as wrong format of IP addresses).

5 Extra credit (15 points)

For extra credit, you can also support queries for non-existent domain, which returns `NXDOMAIN` response. Generally, you need to be able to interpret `NSEC`, `NSEC3`, or `NSEC5` records[2] to check the

[2] `https://tools.ietf.org/html/rfc5155`

signature of the NXDOMAIN response. For the extra credit, however, it is okay to assume that your code expects only NSEC3 records. Therefore, your program should property show and validate the NXDOMAIN response when you query the domain that does not exist as shown below;

```
IP <tab> <NSEC3 record(s)> <tab> <VALID|INVALID>
```

The below is an example of the dig application output when you request a domain name that does not exist:

```
bash$ dig tijayisawesome.com A +dnssec +multiline

; <<>> DiG 9.10.6 <<>> tijayisawesome.com +dnssec +multiline
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 54707
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 8, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1452
;; QUESTION SECTION:
;tijayisawesome.com.        IN A

;; AUTHORITY SECTION:
com.                       900 IN SOA a.gtld-servers.net. nstld.verisign-grs.com. (
                               1542509138 ; serial
                               1800       ; refresh (30 minutes)
                               900        ; retry (15 minutes)
                               604800     ; expire (1 week)
                               86400      ; minimum (1 day)
                               )

com.                       900 IN RRSIG SOA 8 1 900 (
                               20181125024538 20181118013538 37490 com.
                               gQV7VPcxE5Ol8RVefpCd8bjt+AWvXDp+S0okc79yhElx
                               [ ... ]
                               7taLnqOpcJtaHvzYdfOQI4YuxQs+c810x1+aCl0= )
ck0pojmg874ljref7efn8430qvit8bsm.com. 86400 IN NSEC3 1 1 0 - (
                               CK0Q1GIN43N1ARRC9OSM6QPQR81H5M9A
                               NS SOA RRSIG DNSKEY NSEC3PARAM )
ck0pojmg874ljref7efn8430qvit8bsm.com. 86400 IN RRSIG NSEC3 8 2 86400 (
                               20181124054257 20181117043257 37490 com.
                               Xwm87dpZsxrn4howMYqCtaadUDX5pHW79Jj+KoMu7VmM
                               [ ... ]
                               XEIx3+iXYaQeJS7FFJ639BuDvd4j0a9gG8ZIXi4= )
s1sf8qvm5qlsjb3rue89e3k02qojrcq4.com. 86400 IN NSEC3 1 1 0 - (
                               S1SFF40N33U17O3IDH2JO5JSI8DT7M2U
```

```
                                NS DS RRSIG )
s1sf8qvm5qlsjb3rue89e3k02qojrcq4.com. 86400 IN RRSIG NSEC3 8 2 86400 (
                                20181124060733 20181117045733 37490 com.
                                sjUYhzeNyRhYOiEz/jZXW/4aPXQxMRUOqhDcJDDt7Qxx
                                [ ... ]
                                3FHqY+V3CvLbUoktdonMAM43QYnVQJMXSbEQgZM= )
3rl2q58205687c8i9kc9mv46dghcns45.com. 86400 IN NSEC3 1 1 0 - (
                                3RL3ODP8D910939I655B97GAQU6VE1Q7
                                NS DS RRSIG )
3rl2q58205687c8i9kc9mv46dghcns45.com. 86400 IN RRSIG NSEC3 8 2 86400 (
                                20181123061045 20181116050045 37490 com.
                                2+z7JifBOvIQq4otHgH8NocD8B+Qat46/XOnLanUNkt4
                                [ ... ]
                                Z1x+EOfrUMfqjT2IwJQZhoru82Ke/ZWzGyhyy40= )
```

6 Testing

You can use the `wireshark` utility in order to diagnose problems with packets that you send out (these will likely be malformatted at the beginning). Wireshark will capture packets that you send and will let you view/explore the various fields. It will warn you about fields that are incorrect or missing, and can guide debugging your packets.

You can use the `dig` utility with enabling `+dnssec` option in order to help diagnose problems with interpreting responses from the DNS server that you query. To use it, see the `man` page, and an example of the output is shown below:

```
bash$ dig example.com A +dnssec +multiline

; <<>> DiG 9.10.6 <<>> example.com A +dnssec +multiline
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13953
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do[3]; udp: 1452
;; QUESTION SECTION:
;example.com.                IN A

;; ANSWER SECTION:
example.com.                10436 IN A 93.184.216.34
example.com.                10436 IN RRSIG A 8 2 86400 (
                                20181127161203 20181107025118 63855 example.com.
                                kQbzOQi13rqG33kNdGDbpITN8p/LlZkheMsTU+eng1iN
                                [ ... ]
```

```
                                AUVL0E7kkPr1KiBp6M0MItgdhm2UXAHhRUqhf0I= )
```

The explanation of each field in the RRSIG format is shown below:

- A: A type of the record of this signature.
- 8: A security algorithm number. Here the signature is generated using RSA/SHA-512 algorithm.[4]
- 2: The number of labels in the original RRSIG RR owner name. In this example, we have two labels: example and com.
- 86400: An original TTL for the covered A record.
- 20181127201150: An expiration date of this signature.
- 20181106145117: An inception date of this signature.
- 63855: A key tag.
- example.com.: A signer name.
- bhx...YXK=: A base64 encoding of the signature.

```
bash$ dig example.com DNSKEY +dnssec +multiline
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35247
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1452
;; QUESTION SECTION:
;example.com.                   IN DNSKEY

;; ANSWER SECTION:
example.com.                3334 IN        DNSKEY 257 3 8 (
                                AwEAAZ0aqu1rJ6orJynrRfNpPmayJZoAx9Ic2/Rl9VQW
                                [ ... ]
                                xjlmDNcUkF1gpNWUla4fWZbbaYQzA93mLdrng+M=
                                ) ; KSK; alg = RSASHA256 ; key id = 45620
example.com.                3334 IN        DNSKEY 257 3 8 (
                                AwEAAb0FAxl+Lkt0UMglZizKEC1AxUu8zlj65KYatR5w
                                [ ... ]
                                M+pZGhh/Yuf4RwCBgaYCi9hpiMWVvS4WBzx0/lU=
```

[3] Please note that the flags are enabled with DO (DNSSEC-OK) bit.

[4] The public-private key pair is generated using RSA algorithm, and the hash value (i.e., digest) is generated using SHA-512 function.

```
                                ) ; KSK; alg = RSASHA256 ; key id = 31406
example.com.                3334 IN        DNSKEY 256 3 8 (
                                AwEAAd3ls8XH4tS6n576cFPy9ZbtQlf8ivP29WA41Kes
                                [ ... ]
                                k2b/Ptuhkx2HRkZJKJyirRyHyg7vYQOgMIdNJ8D9munn
                                ) ; ZSK; alg = RSASHA256 ; key id = 63855
example.com.                3334 IN        RRSIG DNSKEY 8 2 3600 (
                                20181128092402 20181106205118 31406 example.com.
                                M63zIJKroqEOVN/QTKCEJsNwbctCfgCzlr7oAOK5Zaft
                                [ ... ]
                                PMNOAxQAX1vktp+kJDwirLMabOEq4OuACQ== )
example.com.                3334 IN        RRSIG DNSKEY 8 2 3600 (
                                20181128092402 20181106205118 45620 example.com.
                                btfXPLtPMEl/psB+tTjd51VNFbeeX6wDI87paXlInu2M
                                [ ... ]
                                luyJb5L1CJuM+1ocFSovMSTnSC+JNOxqXQ== )
```

The explanation of each field in the DNSKEY format is shown below:

- 257: A key type: 256 indicates a zone signing key (ZSK) and 257 indicates a key signing key (KSK).
- 3: The fixed protocol value. It is always 3 for DNSSEC.
- 8: An algorithm type, which identifies the public key's cryptographic algorithm.[5]
- AwEAA...Q==: A base64 encoding of the public key.

```
bash$ dig example.com DS +dnssec +multiline
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55102
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 7, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1452
;; QUESTION SECTION:
;example.com.                IN DS

;; ANSWER SECTION:
example.com.                10359 IN DS 31406 8 1 (
                                189968811E6EBA862DD6C209F75623D8D9ED9142 )
example.com.                10359 IN DS 31406 8 2 (
                                F78CF3344F72137235098ECBBD08947C2C9001C7F6A0
                                85A17F518B5D8F6B916D )
```

[5] https://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml.

```
[ ... ]

example.com.                10359 IN RRSIG DS 8 2 86400 (
                                20181123052305 20181116041305 37490 com.
                                PEtWYgEZtwRfqUFCZBYBT++ZdxO6W1EoW/NYBtO2icY5
                                [ ... ]
                                oUHW3nEQ9wMcgb1Es6EOC/WgX/CJldGXr99aaXU= )
```

The explanation of each field in the DS format is shown below:

- 31406: The key tag for the corresponding "example.com.". A key tag is used to easily find the matching DNSKEY.
- 8: The algorithm used to generate the DNSKEY for the corresponding DS record.
- 2: The algorithm used to construct the digest (i.e., hash of the DNSKEY).

7 Submitting your project

7.1 Registering your team

If you want to participate in this bonus project, you should pick out a team name (no spaces or non-alphanumeric characters). One of team members should send me an email (the title is [CSCI351] Registering a team) with team name, your name, your RIT ID, partner name, partner's RIT ID.

You must register your team by 11:59:59pm on November 19, 2019. If you do not register your team, your submission will not be considered.

7.2 Final submission

For the final submission, you should submit your (thoroughly documented) code along with a plain-text (no Word or PDF) README file. In this file, you should describe your high-level approach, the challenges you faced, a list of properties/features of your design that you think is good, and an overview of how you tested your code. You MUST submit a "shell" `runme.sh` script that generates the executable file `351dnsclient` and the virtual environment you used (if you used any virtual environment): you choose your language so you have to prepare it. You should submit your project to Project4 folder in the Mycourses Dropbox. Specifically, place all of your code and README files into one folder (Project4) and zip it (TEAMNAME.zip) and upload it to the Dropbox.

You must submit your project by 11:59:59pm on December 10, 2019.

8 Grading

The grading in this project will consist of

70% Program functionality

15% Error handling

15% Style and documentation

You are, however, going to be graded on how gracefully you handle errors. In other words, what will you do if you receive a corrupted response packet? Remember, network-facing code should be graded defensively; you should always assume that everyone is trying to break your program. To paraphrase John F. Woods, "Always code as if the [the remote machine you're communicating with] will be a violent psychopath who knows where you live."

Any third party libraries usage will be considered as fail.

9 Advice

A few pointers that you may find useful while working on this project:

- Before getting your hands dirty, please fully understand how DNSSEC works by reading the background thoroughly. The course material[6] will be your friend. Also, the first 6 minutes of this talk[7] would be helpful as well.
- I STRONGLY RECOMMEND to read RFC4034[8] to understand how to calculate the signature and verify it. You may use crypto libraries (e.g., pyCrypto) to check if the signature is correct given the (1) message, (2) hash algorithm, and (3) public key. For example, if you use pyCrypto library, you can simply call `signer.verify(digest, signature)` to validate the signature.[9]
- Check the Mycourses for question and clarifications. You should post project-specific questions there first, before emailing the professor.

[6] `https://taejoong.github.io/courses/csci-351/handouts/18-DNSSEC.key`
[7] `https://www.youtube.com/watch?v=KLj4ayIi920`
[8] https://www.ietf.org/rfc/rfc4034.txt
[9] See an example: `https://gist.github.com/lkdocs/6519372`