A Reproduction of "A Longitudinal End-to-End View of the DNSSEC Ecosystem"

Sean Decker
Stanford University
skdecker@stanford.edu

David Liedtka
Stanford University
dliedtka@stanford.edu

1 INTRODUCTION

We reproduce the client-side results of Chung et al.'s "A Longitudinal End-to-End View of the DNSSEC Ecosystem"[4]. Chung et al. explores the state of DNS Security Extensions (DNSSEC) infrastructure in both client and server implementations in 2017. DNSSEC, introduced in the 2000s, was a response to DNS security issues. Perhaps the most notable of these security issues is DNS cache poisoning, which is an exploit that "poisons" DNS lookup servers to divert traffic away from legitimate servers to malicious ones[7]. This poisoning can effect any website on the internet that doesn't implement DNSSEC since most every website that is not in the dark web relies on DNS for users to find their content. DNSSEC adoption increased over time, especially after highly publicized attacks like 2008's Kaminsky attack[8]. However, DNSSEC is a complex system, and security guarantees can only be assumed if all aspects of the system, on both the client and server side, are implemented correctly.

While much web traffic was still knowingly not using DNSSEC in 2017, Chung et al. sought to evaluate the suspicion that many using DNSSEC and assuming a certain level of security were still vulnerable due to using an incorrect implementation. Chung et al. is the first efforts to study DNSSEC at scale, or across many domains and resolvers, and longitudinally, by monitoring behavior over time. For the purposes of our reproduction, we are interested in the client-side, so we examine their findings concerning resolvers. Specifically, Chung et al. find that while 82% of resolvers request DNSSEC records during their queries, only 12% properly validate the records. This means that a large number of internet users are susceptible to DNS security issues, and they are likely entirely unaware of it.

While Chung et al. takes an end-to-end look at DNSSEC, we focus on the client-side for our reproduction. Our project timeline is limited. Thus any server-side measurements we collected would likely not have enough "longitude" (have observations over a long enough time frame) to be interesting. Additionally, observations regarding resolvers (as opposed to DNS servers) are likely closer to most readers and therefore also likely of greater interest to readers.

In this write-up, we first give further background information on DNS and on DNSSEC.

Then, we explain how we use Python code to reproduce Chung et al.'s client-side experiments[3]. We first validate Chung et al.'s publicly available dataset by attempting to create our own version through a dataset of the domains that nameservers are authoritative for hosted on Host.io and then scanning these domains using the dig utility. Through this process, we evaluate which nameservers are correctly implementing DNSSEC. Then, we attempt to match Chung et al.'s experimental evaluations of the proportion of resolvers that properly implement DNSSEC. Our code is publicly available on GitHub[5].

Next, we present and then analyze our results. We address limitations and ideas for future work, and then we conclude.

In the process of validating the dataset, we find that despite the trend of wider DNSSEC adoption continuing, today the vast majority (more than 96%) of DNS traffic still fails to use DNSSEC and is therefore vulnerable. While we find inconsistencies between the data published by Chung et al. and their presented results and while we struggle with underspecified methodology, we find the trends they present generally hold true, and we examine some of the reasons that resolvers may fail to properly validate DNSSEC.

2 BACKGROUND

We begin by giving a more in-depth description of DNS and DNSSEC.

2.1 DNS

The Domain Name System (DNS) is the naming system that allows computers and services on the internet to interconnect by associating various pieces of information with domain names. Most prominently, DNS associates human readable domain names (.com, .org, .net, ect) to IP addresses which can then be used to locate computer services and devices on the IP networking layer. Most generally, DNS represents a mapping from the domain name hierarchy to the IP address space.

DNS to an extent is decentralized, delegating the responsibility of assigning domain names to authoritative name servers for each domain. These authoritative name servers are in charge of deciding the mapping from domain name to Internet resource, and are able to delegate authority over sub domains to other name servers.

DNS maintains both the hierarchical structure of authority over domains and the mappings from domain names to Internet resources through records. State of Authority (SOA) records contain administrative information about the authority over a particular domain. A and AAAA records map domain names to IP addresses, IPv4 and IPv6 respectively. And then MX records map domains to a mail server client. NS records map domains to authoritative name servers.

Because DNS records are often static, a time to live for a record is often on the order of minutes, and because a record being stale isn't a security concern, DNS servers cache records aggressively. Caching increases fault tolerance and availability. But, because of the lack of authentication within DNS, this aggressive caching can be exploited by an adversary. In the event that a query is made for say the A record for example.com to a DNS name server and the name server doesn't hold that record, the name server will make a recursive query for the record. If an adversary is listening for this recursive call, or knows it is being made (the adversary will often

be the one querying the nameserver for example.com's A record), the adversary can respond to a DNS query itself and send a false A record for example.com that actually points to a malicious IP address that corresponds to a server that serves ads or attempts to get personal information from the user. This A record will then be cached at the name server and the name server will direct all of its queries for example.com to this malicious server as long as the TTL on the faulty A record hasn't expired. This is glossing over a lot of the details of the attack; more detailed description of this attack, called cache poisoning can be found at [3].

2.2 DNSSEC

In the early 2000s, DNS cache poisoning because an optimization to the attack, this new vulnerability was named the Kaminsky DNS Vulnerability [8] , and it's goal was to serve faulty NS records instead of A records. In theory then, this attack would allow an attack to divert not just all domain queries for a particular domain say example.com, but instead all domain queries for a particular zone, for example all .com domains. In response to their vulnerability, DNSSEC was designed to provide authentication to the DNS system.

DNSSEC utilized public key cryptography in order to allow name servers to sign records. DNSSEC associates a public and private key to an authoritative name servers. Authoritative name servers then sign their DNS records, A, MX, SOA records, ect, using their private key and then this signature is saved in an RRSIG record. The RRSIG record can then be verified with the name server's public key, which is kept as another record called the DSNKEY record. But the authenticity of this DNSKEY record cannot be trusted by itself. This DNSKEY record is signed with a special key signing key for the authoritative name server, and this key signing key's hash is then stored in a DS record in the parent zone for this name server. This transfers the burden of trust from the authoritative name server for the record of interest to the name server for the parent zone. (for example, example.com's parent zone is .com.). The parent zone itself signs this DS record with its own private zone signing key, and the corresponding public zone signing key is stored in a DNSKEY record, which is then signed by a key signing key and then the DS record for this key signing key is kept at the parent zone of .com, the root zone, represented by a dot (i.e. '.' represents the parent zone). The DNSKEY for the root zone is something that is hard-coded into DNS query utilities. [2] The root zone's DNSKEY represents the 'root of trust' in a chain of trust that ends in the record of interest (in this example that would be the A record for example.com); a check of authenticity in DNSSEC thus represents a recursive verification of records all the way until the root zone. For a more thorough description of DNSSEC see [6].

DNSSEC as specified in its RFCs, does provide authentication, but keeping with the design of DNS, it is implemented entirely through records and its function is controlled entirely by name servers. Because of the complexity of DNSSEC and the the fact that its incorrect implementation doesn't harm correctness, DNS still works without DNSSEC, oftentimes name servers either ignore DNSSEC or implement it incorrectly, as was found in Chung et al.[4].

3 METHODOLOGY

Because we do not have the time or capacity to reproduce Chung et al.'s longitudinal study, which took daily snapshots of all DNSSEC records under all .com, .org, and .net domains over a period of months, we instead narrow the focus of our project on the client-side studies in Chung et al. We validate a publicly available dataset published by Chung et al. with their paper, and then we recreate the client-side experiments. Again, our code is available on GitHub, with the link provided in the references[5]. Our reproduction methodology follows.

3.1 Validation of Dataset

Chung et al. studied client side behavior through the use of Luminati, an HTTP/S proxy service that they used to emulate 403,355 end hosts, thereby allowing them to study 59,513 distinct DNS resolvers.

We take a different approach because of the (monetary) cost of the Luminati service. First we rely on the public dataset from Host.io [1] for retrieving a list of domains that certain large name servers are authoritative for. We need to rely on Host.io because although one can retrieve NS records for a specific domain to learn the authoritative name server for the domain, records for the reverse direction, records from a name server that lists the domains it is authoritative for, are not available oftentimes unless you own a domain under that name server. With a dataset of domains and their respective name servers, we then we able to make repeated dig queries for these domains and extract the associated DNSSEC records. In particular, we queried for domains' DNSKEY and A RRSIG records (A RRSIG records are a signature on the collection of A records for a particular domain), and domains' associated DS records which are stored in parent zones (the authoratative name server level).

It is important to note that our scale is much larger than that of Chung et al., because of the continued growth of domains on the internet. For example, the largest name server, in terms of number of domains it was authoritative for, that Chung et al. scanned was ovh.net which contained 316,960 domains. The largest name server that we scan is godaddy.com, with over 60,000,000 domains. As we will discuss in our limitations section however, we were not able to scan such a large number of domains because our repeated nameserver queries resemble a DDOS attack.

3.2 Reproduction of Experiments

The dataset used used by Chung et al. that is publicly available is a JSON-formatted file of Luminati key-value pairs. A pair consists of a unique scenario ID, or *zID*, as the key and as the value a "scenario" object constructed from information such as the resolver a host used and the responses the host received after sending a series of DNS requests to Chung et al.'s DNS server. For each host, if its resolver supported DNSSEC, the host would first fetch a valid record and then nine misconfigured records. The violations associated with these misconfigured records are a missing key signing key (KSK), a missing RRset signature (RRSIG) KSK, a mismatched delegation signer (DS) record, an invalid RRSIG A record, an invalid RRSIG KSK, a missing zone signing key (ZSK), a missing RRSIG A record, a past (expired) RRSIG A record, and a future (not yet valid) RRSIG

A record. If the the object indicates that the host received one of the misconfigured records, then we know that the resolver that host is using is not properly validating DNSSEC. The construction of the dataset is explained in further detail in Sections 5.1 and 5.2 of Chung et al., and the meaning of the misconfigured record types can be surmised from Section 2[4].

The dataset is too large to work with practically, as it is larger than 14 GB and contains 10,435,705 scenarios. Chung et al. filter out many of these scenarios, as they consider only pairs in which the exit node is configured with a single resolver and resolvers which are measured at least 10 times. Therefore, rather than work with a 14 GB object, we filter down to a much more manageable filtered list of 71,899 scenarios.

Chung et al. classify resolvers into three categories: those that correctly validate DNSSEC, those that incorrectly validate DNSSEC, and those whose policies are ambiguous. We discuss this further in Sections 5 and 6, but we struggle to interpret how exactly they assign the resolvers into these categories. For example, Chung et al. state that an incorrectly validating resolver is one through which more than 90% of exit nodes receive a response when the resolver receives in invalid DNSSEC record. We interpret this as meaning out of all scenarios a resolver is involved in in which it receives an invalid DNSSEC record, if more than 90% of the time it validates the response, it is an incorrectly validating resolver. If more than 90% of the time it does not validate the response, then it is a correctly validating resolver. Otherwise, the policy is ambiguous from the data. We interpret the 90% benchmark leaves room for other reasons why a resolver could validate or fail to validate, such as network failure. We examine our filtered scenarios to make these classifications.

From the list of resolvers that incorrectly validate DNSSEC, we examine the associated ISPs and their prevalence in a manner similar to Chung et al. Additionally, we take Chung et al.'s work a step further by examining specifically which misconfiguration caused these resolvers to improperly validate.

4 RESULTS

The results of our dataset validation and experimental reproduction follow.

4.1 Validation

Table 1: The percentage of all .com, .org, .net, and .* (wildcard, any domain) second level domains that have a DNSKEY record. Chung et al. did not provide data on the overall number of domains with a DNSKEY.

Top Level Domain	Chung et al.	% domains with DNSKEY
.com	0.7 %	19.84%
.net	0.8 %	31.56%
.org	0.6 %	00.43%
.*	n/a	37.19%

From our scans using Host.io's API and dig, we find a general increase in the inclusion of DNSKEY records for domains across the board 1 except for .org domains which remains similar. The table at 1 represents simply the percentage of domains that posses a DNSKEY record (ignoring if it is verifiable or not), subdivided by their top level domain. We also include a wildcard domain since we scanned more top level domains besides .com, .net, or .org domains. Again the DNSKEY record represents the domain's public zone signing key, the counter part to the private zone signing key which is used to sign zone records such as A, MX, and SOA records for the domain and create RRSIG records. Verifying the RRSIG records of an A, MX, or SOA record is the first step in the DNSSEC routine, and so without a verifiable DNSKEY record, DNSSEC is broken (though having a verifiable DNSKEY record does not in itself mean that DNSSEC is implemented).

Table 2: The percentage of signed domains that fail to publish a DS record in the parent zone.

Top Level Domain	Chung et al.	% of domains missing DS
.com	30%	93.01%
.net	25%	00.43%
.org	30 %	13.63%
.*	n/a	22.87%

On the other hand see a similar overall (wildcard .* domains) percentage of domains which have DNSKEY records but have not published a DS record in their parent domain 2. Our results for .com, .net, and .org domains themselves vary wildly. Recall that DS records are hashes of key signing keys and are published in a parent zone for the domain. The DS record represents a link in the chain of trust that is required to verify a DNSSEC record, and without a DS record in a parent domain, an adversary could simply replicate all DNSSEC records for a domain in a way that is indistinguishable from a valid set of DNSSEC records. In short, without a DS record, a domain is insecure under DNSSEC.

Table 3: Table showing the most popular 15 authoritative name servers (in 2016 according to Chung et al.), the number of domains with DS records, and the total number of signed domains. n/a implies that we were not able to scan the name server, often this means that the name server was migrated or there is some other reason it doesn't exist.

Name Servers	# signed doms	# w/DS	% w/ DS	Chung
*.ovh.net	1152	1152	99.13%	99.45%
*.loopia.se	8530	5688	66.68%	0.00%
*.hyp.net	n/a	n/a	n/a	99.85%
*.transip.net	6676	6650	99.61%	99.90%
*.domainmonster.com	3541	0	0.00%	0.01%
*.anycast.me	n/a	n/a	n/a	98.13%
*.transip.nl	4606	4601	99.89%	99.92%
*.binero.se	3757	3756	99.97%	38.30%
*.ns.cloudflare.com	266	266	100%	60.42%
*.is.nl	4041	2166	53.60%	0.07%
*.pcextreme.nl	442	398	90.04%	98.89%
*.webhostingserver.nl	328	327	99.69%	71.96%
*.citynetwork.se	112	4	3.57%	0.11%

In 3 we show the number of domains with a DNSKEY record and the number of domains with DS records under various popular

name servers that Chung et al. focused on because they were the most popular name servers at the time the paper was written. We see that we find very similar publishing ratios to Chung et al. for many name servers, and overall we see the same pattern of most name servers publishing close to 100% of DS records and a few name servers that fail to publish any DS records for their signed domains.

4.2 Reproduction

Table 4: Observed dataset measurements vs. those reported in Chung et al.

Measurement	Chung et al.	Observed
Unique Hosts	403,355	420,329
Authoritative Servers	8,842	9,306
Host Countries	177	177
Days Observed	13	14
Unique Resolvers	59,513	61,593
Unique DNSSEC Resolvers	49,424	51,268
Filtered Resolvers	4,427	2,200

Table 4 presents dataset measurements we observe compared to those reported by Chung et al. We expected our observations of the dataset to match Chung et al.'s observations, but we find that they do not. We further analyze the differences here and all tables in Section 5.2.

Table 5: Observed experimental results vs. those reported in Chung et al.

Measurement	Chung et al.	Observed
DNSSEC Resolvers	83.0%	83.2%
Correctly Validating Resolvers	12.2%	3.59%
Incorrectly Validating Resolvers	82.1%	94.9%
Resolvers With Ambiguous Policy	5.7%	1.55%

Table 5 presents the results of our reproduction experiments compared to the results reported by Chung et al. Again, we discuss further in Section 5.2, but the root of the differences likely lies in differences in the data we are working with and perhaps misunderstanding Chung et al.'s methodology.

Table 6: Our reproduction of Table 5 from Chung et al. We display our top five ISPs in terms of number of DNS resolvers that do not validate DNSSEC.

Country	Hosting ISP	DNS Resolvers	Exit Nodes
U.K.	Sky UK Limited	75	589
Russia	PJSC Rostelecom	47	128
Denmark	Vodafone GmbH	31	78
Austria	Liberty Global Operations B.V.	30	296
Italy	Telecom Italia S.p.A.	27	141

Table 6 is our reproduction of Table 5 from Chung et al.[4], which shows the specific ISPs that are most prevalent in terms of sheer number of incorrectly validating resolvers. Chung et al. displays their top 15, but as we filtered evidently to a smaller dataset, we display just our top five.

Table 7: We examine the DNSSEC violations that incorrectly validating resolvers fail to handle.

DNSSEC Violation	Prevalence
Missing KSK	86.8%
Missing RRSIG KSK	89.0%
Mismatch DS	94.6%
Invalid RRSIG A	85.2%
nvalid RRSIG KSK	91.5%
Missing ZSK	79.6%
Missing RRSIG A	80.3%
Past RRSIG A	84.3%
Future RRSIG A	82.5%

In Table 7, we measure how prevalent specific DNSSEC violations are amongst the resolvers we identify as incorrectly validating DNSSEC.

5 ANALYSIS

We discuss the results presented in Section 4.

5.1 Validation

As mentioned in Section 4.1, we find a higher publishing ratio than that of Chung et al. for DNSKEY records. This implies that there has been a rise in the uptake of DNSSEC over time. It is important to admit that there is wild variance in our results. This can be explained by how for our name server scans, we completely scan certain name servers as opposed to scanning random domains, and as our results from 3, we see that implementation of DNSSEC is nonuniform across name servers. It is probable that the .net and other top level domains (such as .se or .nl) name servers we scanned were disproportionately implementing DNSSEC while the .org name servers we selected were disproportionately not implementing DNSSEC.

Still, scanning over 73,520 domains in total and because there is such a steep rise in the addition of a DNSKEY record, we can be confident in our claim that implementation of DNSSEC has increased.

From our table at 2, we see that even with the uptick in implementation of DNSSEC, the complexity of DNSSEC has still led to similar levels of implementation failures. In particular we find that an the percentage of domains with a signed DNSKEY, but which do not publish a DS record in their parent zone is very comparable to the percentage that Chung et al. found. Again though, the results broken down among top level domains shows high variance.

The variance in our results can be explained by our final finding from Section 4.1. Again in table 3, we see that the implementation of DNSSEC and faults in this implementation is nonuniform across name servers. This makes sense since if a name server does not have a system in place to add DS records for domains with signed

DNSKEY records, then the server will fail to publish any records. On the flip side, if it does have such a system, then it should have close to a 100% publishing rate.

What is also fascinating about our results though is that we see that name servers that failed to publish DS records in 2016 in the Chung et al. paper often still fail to publish DS records 4 years later. This may be a consequence of how DNS still functions regardless of the correctness of DNSSEC, and so there may be little motivation for these name servers to fix this issue in their DNSSEC implementation.

5.2 Reproduction

While we initially expected to produce the exact same experimental results as Chung et al. since we were using their dataset, the differences observed in Table 4 reset our expectations. It is clear that we observe "extra" data in every category until we filter resolvers. We are not sure why we observe greater numbers of hosts, authoritative servers, resolvers, etc. We initially suspected that because we had one additional "observed" day compared to Chung et al. that perhaps the dataset included an additional day of data corresponding with Section 5.4 of Chung et al. However, we tried individually isolating each day from the rest of the days, and we still got different results than those reported. We have been in contact with Chung et al.'s primary author, Professor Taejoong Chung of Rochester Institute of Technology, but at the time of writing we had yet to determine the cause of inconsistency.

While we do not have a clear idea of why the unfiltered data is different, we believe we have less filtered, "testable" DNSSEC aware resolvers because we cannot fully under Chung et al.'s methodology. It seems Chung et al. assume that all resolvers are DNSSEC capable because 83.0% of all possible resolvers use DNSSEC somewhere in the dataset. They filter hosts to only those associated with one resolver, and they filter resolvers to those that appears in at least ten different scenarios (as is our understanding). When we filter using these constraints, we end up with 6,679 resolvers, but the majority of them choose to use conventional DNS rather than DNSSEC in their requests, despite the assumption that all resolvers are DNSSEC capable. If a resolver does not set the DNSSEC bit, it requests just the valid domain and none of the misconfigured domains, and we cannot test whether it validates DNSSEC. Only 2,200 of our filtered resolvers ever send requests with the DNSSEC bit, so we can only run our experiments on these 2,200. It is unclear to us how Chung et al. obtained such a large "testable" set of resolvers.

Clearly, with a vastly different set of hosts and resolvers, we would not reproduce the results exactly, but we still hoped to observe the same trends. Generally, we achieve this goal. Looking at our results in Table 5, we find that even fewer resolvers correctly validate DNSSEC and even more resolvers incorrectly validate DNSSEC. Still, the expected trends hold. A large number of resolvers pay for the overhead of DNSSEC without bothering to even validate it, or perhaps even more alarmingly, many resolvers might think they are secure when they are not.

Table 6 can be compared directly with Table 5 from Chung et al.[4]. We see that two ISPs in our top five are in their top 15 and that as expected, we have much smaller numbers of resolvers and exit nodes. This furthers are belief that their is a significant mismatch between the filtered data we are running our experiments on and that used by Chung et al.

Despite our difficulties replicating Chung et al.'s experiments, Table 7 is perhaps our most noteworthy contribution, as Chung et al. did not explore the distribution of misconfigurations. While certainly resolvers are more prone to some problems than others, such as a mismatched delegation signer vs. a missing zone signing key, it seems likely that in most cases DNSSEC either works or does not. If a resolver is prone to one misconfiguration, it is likely prone to most if not all other possible misconfigurations. This seems reasonable, as we can imagine that if a ISP makes DNSSEC a priority, they will do their best to fully implement it themselves or will consult someone for use of a strong implementation. Many ISPs are using resolvers that are DNSSEC capable, but they entirely ignore DNSSEC.

6 LIMITATIONS

We discuss obstacles encountered during our reproduction.

6.1 Validation

Again our reproduction of the authors' dataset was done using python code that was running on a home network. This python code generated a list of domains that were under the authority of a given name server, and then these domains were queried by dig requests for records such as their DNSKEY, A RRSIG, DS, ect. This python code was threaded and thus capable of creating thousands of DNS queries in seconds. When scanning a large dataset of domains though, we noticed that our local machine would become unable to connect to the internet. Dig requests and all other network requests would stall indefinitely for about 10 minutes when internet connectivity was restored. At first we thought that this was a problem with the python code that we wrote, and thus we optimized it to have less threads, and to minimize the number of sockets that it used. Still this problem persisted. The fact that this bug rendered our python code un-runable for a long period of time after it occurred, and because it led to a failure that would disallow uses of the computer, such as Zoom calls or Stack Overflow searches, made it hard to debug.

Then, we noticed that not only did the internet for the machine running the python code halt, but connectivity for all network devices, including mobile phones and other computers connected to the same local WiFi network was halted. This problem occurred on multiple WiFi networks we tested on. This is when we realized that is wasn't that the python code was causing an issue on the computer it was running on, but it was instead causing an issue at the router level or higher.

Our current theory is that the thousands of dig requests coming from one WiFi router is flagged as a part of a possible DOS attack by the ISP or by the router itself. Thus network connectivity for all machines on the WiFi network is being throttled to stop the possible attack.

Either way, this issue led to us not being able to scale our name server scans to as large of datasets as the Chung et al. Large network scans could only be done at night, when other people were not using the WiFi, and only about 2,000 queries could be sent before the network crashed. This issue would possibly be fixed by instead

running our python code in a distributed cloud service, sending queries at a slower rate from more machines. This would be more expensive though, and the traffic would still resemble a DOS attack most likely. This issue is one of the reasons why the authors of Chung et al. used a proprietary proxy service, Luminati, in order to do their scans.

In total, we scanned 73,520 domains. This was enough to get results that we believe give insight into the nature of the current state of DNSSEC, but we can see from the large variation in our results broken down by top level domain that this smaller sample size does affect our results.

6.2 Reproduction

We mentioned previously that our reproduction suffered because of inconsistencies and misunderstood methodology. We feel that some of the methodology in Chung et al. was underspecified, which is understandable as they covered both client and server-side DNSSEC, and it is difficult to delimit minor implementation details in an end-to-end study when considering page limits and readability. However, with an open communication channel with the authors, which we discuss further in the next section, we believe it would be possible to remove dataset inconsistency and methodology ambiguity, and we would be able to achieve an exact reproduction.

7 FUTURE WORK

Obvious future steps include scanning other name servers, so we can verify whether 25% of domains under .aka.net failing to provide the information needed to verify their DNSKEY was a quirk of that specific name server.

Additionally, once we have resolved inconsistencies between what we observed in the dataset and what Chung et al. reported and cleared up ambiguities in methodology, we expect that re-runs of our experiments would lead to results that match Chung et al.'s. As we mentioned, we contacted the lead author from Chung et al., but due to an upcoming deadline he was not able to look through dataset. He did offer his help later in the summer, so it is an avenue we could further explore.

Additional future work could also include following up our breakdown of what problems in particular resolvers were susceptible to (it is unlikely that susceptibility to one misconfiguration is independent of the others); although this would require new experiments and perhaps acquisition of a new dataset.

8 CONCLUSION

We attempt a reproduction of the client-side experiments of Chung et al.'s longitudinal, end-to-end study of DNSSEC, which is now three years old[4]. We first validate their dataset with our own name server scans, and then we run the same resolver-focused experiments they detail in their paper. Through our validation, we confirm their observations that while DNSSEC adoption is on the rise, it is still alarmingly low at less than 4% in 2020. Experimentally, we are generally able to reproduce the trends they observe in client-side DNSSEC, and we contribute additional insight into which problems in particular seem to most frequently afflict resolvers.

ACKNOWLEDGMENTS

We would like to thank Professor McKeown, Professor Katti, and Bruce Spang for putting together a worthwhile course and for their valuable feedback and support.

REFERENCES

[1] 2020. https://host.io/
[2] et al Arends. 2005. Resource Records for the DNS Security Extensions. *Network Working Group* (2005).
[3] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. [n.d.]. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. https://securepki.org/sec17.html. Accessed: 2020-05-03.
[4] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. 2017. A Longitudinal, End-to-End View of the {DNSSEC} Ecosystem. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 1307–1322.
[5] Sean Decker and David Liedtka. 2020. CS 244 Final Project - DNSSEC. https://github.com/seankdecker/cs244-DNSSEC-project
[6] DNSSEC. 2014. How DNSSEC Works. (2014). https://www.cloudflare.com/dns/dnssec/how-dnssec-works/
[7] Steve Friedl. 2008. An illustrated guide to the kaminsky dns vulnerability. *Unixwiz. net Tech Tips, August* (2008).
[8] Dan Kaminsky. 2008. Black ops 2008: It's the end of the cache as we know it. *Black Hat USA* 2 (2008).