DoS attack on recursive resolvers with DNSSEC key-tag collisions

D.A. Bleeker
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
dex@bleeker.nl

ABSTRACT

DNSSEC was implemented to strengthen DNS and enable resolvers and end-users to validate the the integrity and origin of responses by using digital signatures. To speed up this verification, key-tags were introduced. In this paper we analyse an attack that uses key-tag collisions to generate enough computational overhead to render a recursive resolver unavailable (DoS attack). A zone with 65 keys with the same key-tag was set up on an authoritative name server, along with a resolver (Unbound and BIND) and an attacker, to simulate this attack. This paper concludes attempting to DoS a recursive resolver using DNSSEC key-tag collisions is viable, at least in theory.

Keywords

DNS, DNSSEC, key-tag collision, attack, DoS, resolver, RSA, CPU utilisation, Unbound, BIND

1. INTRODUCTION

DNS is hierarchical and decentralised which makes it possible to use the internet with more easily memorizable names, rather than using IPv4 (or even worse, IPv6) addresses. This system was developed in the 80's[10, 11] to replace the cumbersome lists for this mapping, which had to be updated and maintained by hand.

The Domain Name System works properly, but was not developed with security in mind as in the 1980s the internet was very small. With the explosive growth of the internet (and the World Wide Web), the lack of proper security poses more and more of a problem[8]. For example, by changing the response from a DNS-server (resolver), one could lead a user to an other server than intended. This server could be a malicious server in the hands of the 'hacker' that is intended to pilfer the user's password, as detailed in 1990 by Steven M. Bellovin[6]. Since the Domain Name System is hierarchical, a 'user' in the previous example could also be another resolver. When a (recursive) resolver queries an authoritative server, it has no way of validating the validity and authenticity of the response. If a resolver higher in the chain is compromised, not just one user is redirected to a wrong server, but *all* users that query any of the resolvers lower in the chain from the compromised resolver are redirected wrongly. It is easy to see why this is a significant problem. Hence the need for DNSSEC (DNS Security Extensions).

The development of DNSSEC started in the 1990s, with the initial RFC by Eastlake & Kaufman[2] and improves security on DNS by using signatures based on public-key cryptography. The DNS data itself that is transferred between resolvers and users is cryptographically signed, from the 'top' (highest name server) to 'bottom' end user. This is not to be confused with techniques like DNS over TLS[9], which secure the transmission only from one single point to another.

This paper will address the following points:

Research goal Can we render a recursive resolver unavailable (exhaust its resources, DoS-attack) by abusing the added computational overhead that occurs when querying a DNS zone containing multiple keys with the same key-tag?

- **RQ 1** What is the maximum computing overhead generated by key-tag collisions in a crafted DNS zone? This zone will contain a specific amount of RSA keys (variable per test) with identical key-tags.
- **RQ 2** How do the different signing algorithms compare in generated computing overhead when dealing with key-tag collisions?
- **RQ 3** Can we increase the impact of a key-collision attack by combining other attack vectors, like a sub-domain attack or a phantom-domain attack?
- **RQ 4** What can be done to mitigate key-tag collision attacks? Are the attacks completely preventable or is the impact reducible?

2. BACKGROUND

2.1 DNSSEC

Every DNSSEC-signed zone (specific and distinct name space for which the administrative responsibility is in hands of one or a few nameservers, the authoritative name servers), for example the .nl-zone, has a public/private key pair. This key pair is used by the zone owner (SIDN) to sign the zone's data and to generate corresponding signatures. Any resolver that queries the authoritative name server receives the response (with signatures) and the public key with its corresponding key-tag. The public key can be used to validate the received response. To do this, the querying resolver calculates the hash of the DNS data and compares that to the -with the public key decrypted- signature. If they match, the DNS data is verified. See also figure 1:

31th Twente Student Conference on IT July 5th, 2019, Enschede, The Netherlands.

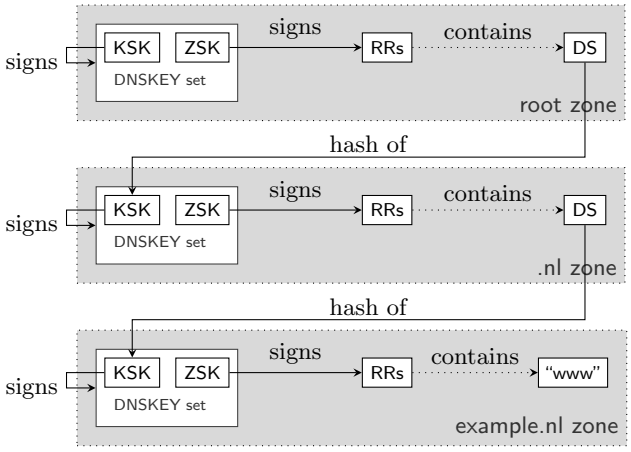

Figure 1. A schematic overview of DNSSEC[12].

The security advantages of DNSSEC can be enumerated in the following two points:

Origin authentication DNSSEC allows a resolver to verify the source of the response.

Integrity protection DNSSEC allows the resolver to verify that the data has not been modified (during transmission) since it was originally signed with the public/private key pair of the zone's manager.

2.2 DNSSEC key-tags

The public key is accompanied by a *key-tag*[1], a 16-bit value that indicates between resolvers which key to match a signature to. Because it is very inefficient to check a signature against every key in a zone, the key-tag (hint) enables much faster matching of signatures to keys. This in turn means that a resolver that aims to verify some received DNS data, can more effectively operate and does not have to check keys that are more or less guaranteed to not match the received signature.

It is assumed that it is unlikely for key-tag collisions to occur. However, they do occur (often), as only 25% of the available key-tags are used (16k of 64k)[4, 16]. This is because the key-tag is not completely random. During calculation, the protocol, algorithm and exponent are incorporated in generating the key-tag.

When key-tag collisions do occur, the resolver has to validate the signature against all keys that have a matching key-tag, until the correct key is found. This has an impact, as this results in a lot of extra computational overhead because cryptography operations are very CPU intensive.

The extra computational overhead is small in itself, but quantifiable. Even more so when ECDSA (Elliptic Curve Digital Signature Algorithm) is used rather than RSA, as verifying signatures with this algorithm is significantly slower[17]. In this research, it will be explored whether this extra computational overhead can be misused by some attacker in a DoS-attack to render the resolver unavailable by crumbling it with workload. The attacker could query the resolver very often and the resolver in turn would have to check many keys (the amount depends on the amount of key-tag collisions) for every query.

To further intensify the attack, the attacker could query the resolver for some domain name of which the authoritative name server is also in the hands of the attacker. If the attacker has a DNS zone set up at the authoritative name server with many key-tag collisions, the resolver would likely have to check significantly more keys as opposed to when the attacker only uses the key-tag collisions that occur in the wild by default.

3. RELATED WORK

3.1 Attacks with DNSSEC

In the paper 'DNSSEC and its potential for DDoS attacks: a comprehensive measurement study', by van Rijswijk-Deij et al.[18], the potential for abuse of DNSSEC-signed domains by using amplification attacks is analysed and discussed, as well as a number of mitigation strategies.

One of these amplification attacks is executed by sending many DNS queries to one or more open resolver(s), of which the sender's IP address is spoofed to be the target's IP address. An open resolver is a resolver that is misconfigured; it does not restrict which clients can query them. The resolver(s) will then send their response to the target. The amplification comes from the fact that usually the queries are small, whereas the responses are large. This attack can be intensified by setting up a domain for which certain DNS queries return even larger responses. The attacks discussed in the paper from van Rijswijk-Deij et al., are *Distributed* Denial-of-Service attacks and any system can be the target. This is different from the specific attack discussed in this paper, because this attack is a DoS attack (rather than DDoS) and only a resolver can be the target of this attack.

3.2 Attacks on recursive resolvers

In their paper, R. Perdisc et al. discuss 'WSEC DNS: Protecting recursive DNS resolvers from poisoning attacks'[13], a new attack for poisoning the cache of recursive resolvers. A mitigation for this problem -Wild-card SECure DNS (WSEC DNS)- is proposed, which decreases the probability of success for this attack by several orders of magnitude. The attack discussed in their paper (poisoning attack) is different from the DoS attack discussed in this paper, as the cache poisoning attack impacts all clients of a resolver, rather than the resolver itself. All clients are given the wrong information (when they query the resolver for `example.com`) and will be redirected to the IP address given by the attacker for as long as the poisoned cache is used by the resolver. The resolver is the direct target of the poisoning attack however, as the attack is performed on the resolver instead of its clients.

'Mitigating DNS DoS Attacks'[5] by Ballani et al. is a paper discussing a proposal of the cache implementations of resolvers, to mitigate a DoS attack. The DoS attack in their paper is not defined; they refer to any DoS attack on (authoritative) name servers. When an authoritative name server is subject to a successful DoS attack, no recursive resolver can resolve the domain the authoritative name server is responsible for. This means that the entire domain name is not resolvable once the caches of the resolvers expire. Ballini et al. propose that resolvers change their caching implementations slightly, to accommodate a 'stale cache' where records are stored when they have expired. If the resolver does not get a response from one of the name servers during its recursive resolving for a query, it can use information stored in the stale cache to answer the query. The research from Ballini et al. is relevant when the DoS attack discussed in this paper is performed successfully.

4. METHODOLOGY

To analyse the potential of this type of DoS attack, the attack was simulated. For this, three (virtual) machines

and a hypervisor to run them on were needed. The needed virtual machines are (see also figure 2):

- Authoritative resolver
- Recursive resolver
- Attacker

Figure 2. Overview of which system queries which.

4.1 The hypervisor

Each virtual machine should have all its resources available for the only program that runs (the resolver, the authoritative name server or the attack script). The amount of available processing power or RAM should not fluctuate (because processes have to compete for resources), like the case in *type-2*- hypervisors[20]. A *type-1* hypervisor is strongly preferred. Initially VMware ESXi was chosen, but because the installer of the newest version would not run and there would be need to obtain and activate a license, Xen[1] was used instead. Xen is highly tweakable, easy to install and supports 'full-virtualisation' because it is a type-1 hypervisor. This means that, when the resolver is under attack, the attacker and authority can continue to do their work.

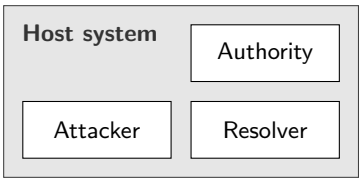

Figure 3. Schematic overview of the virtual machines on the host machine.

Debian Stretch (version 9.9) was installed on the test system. This system is equipped with a Intel Core i3-6100 processor[2] (2 cores, 2 threads per core), has 1GB of RAM at its disposal and a 256GB SSD for storage. The required virtual machines are set up and have one thread and 256MB of RAM dedicated to them (figure 3), which leaves one thread and 256MB RAM for the host operating system. The file systems of the VM's are located as an .img-file on the file system of the host, which makes it easy to clone or backup the virtual machines.

4.2 The virtual machines

Four virtual machines are set up. An authoritative name server, an attacker, a resolver running Unbound[3] and a resolver running BIND[4]. The virtual machines run Debian Buster. This version is only due for release on July 6th 2019 but is more than stable enough for this kind of usage. This version was chosen as it includes more recent software, so packages like BIND or Unbound are more up to date. Some debugging tools (like `dnsutils`, `htop`, `telnet` and `sudo`) were installed on all machines, alongside the specific software for each machine. This means BIND for the authoritative name server and one of the resolvers on the resolver machine. The first one used was Unbound, version `1.9.0`. The next resolver that was used was BIND, version `9.11.5.P4`. The attacker does not need any specific software, as the 'dig'-command was already installed with the debugging tools. The virtual machines are managed through virt-manager[5].

4.3 The zone file

The next step is creating the zone (appendix B.1), from which a signed zone could be created. Setting up the zone was fairly straightforward, as it only needed to include one (TXT) record. This record was later changed from an @-record (TXT record on highest level) to a `*`-record (TXT record on all levels). This is necessary to perform a subdomain- or phantom-domain attack. It is worth noting that this zone has to have a TTL (time to live) as small as possible to prevent caching reducing the impact of the attack. A TTL of one second was used.

4.4 Key generation

Now the zone file is ready to be signed. This turned out to be not as easy as anticipated. It is not possible to use arbitrary keys in the collision attack, by manually modifying the key-tag. This is because the integrity of the key becomes compromised and it will not be validated[15]. Proper keys with the same key-tag needed to be generated. The script used for this can be found in Appendix A.3. It generates keys in a temporary directory and moves it to directory '1'. If this directory already contains a key with the same key-tag, it will be placed in directory '2', and so on. Multiple keys with the same key-tag in the same directory is not possible, as the file name is identical. In order to be able to generate a significant amount of keys, a file system with lot of inodes[21] was needed. After creating the .img-file, partitioning it and creating the tweaked file system, the generation could begin. The script from appendix A.3 has been run for about 16 hours, to create 1051435 RSA and 536743 ECDSA keys. The highest collision count for the RSA keys is 67, while the 'best' ECDSA key only collided 16 times.

4.5 The signed zone file

The zone should contain at least two keys: The Key Signing Key (KSK) and the Zone Signing Key (ZSK). The KSK is used to sign the ZSK, and the ZSK is used to sign all other records.

To generate different signed zones, a different amount of RSA keys (keys with key-tag 21033) were included in the zone (0, 5, 10, 25, 50 and 65). Then, the `dnssec-signzone` command was run, to generate a signed zonefile which can be loaded into BIND. The entire command is as follows:

```
dnssec-signzone -x -A -N INCREMENT -o
    collision.example -f temp.signed
    collision.example.zone
```

The file `temp.signed` is moved to sub directory `zonefiles`, with a name corresponding to the amount of keys included. Then a symbolic link from `/var/cache/bind/collision.example.zone.signed` is created to this file, which is loaded into BIND.

[1] https://xenproject.org/
[2] https://ark.intel.com/content/www/us/en/ark/products/90729/intel-core-i3-6100-processor-3m-cache-3-70-ghz.html
[3] https://nlnetlabs.nl/projects/unbound/about/
[4] https://www.isc.org/downloads/bind/
[5] https://virt-manager.org/

The given flags are:

- **-x** results in that the DNSKEY record is signed using only the KSK, not with ZSK's.
- **-A** makes sure that NSEC3 is not used. NSEC3 only introduces more complexity[7], so the simpler alternative NSEC is used by using this opt-out.
- **-N** means, together with `INCREMENT`, that the SOA serial format of the signed zone file is incremented.
- **-o** indicates the zone origin, the name of the zonefile.
- **-f** writes the output to the given file (`temp.signed` in this case).

By overwriting the symbolic link with a symbolic link to another file in the `zonefiles` directory and restarting BIND, a zone with another amount of key-tag collisions is loaded.

4.6 Establishing trust

For the attack to work, the resolver must be configured to perform the lookup for the custom domain name (`collision.example`) at the authoritative name server, and not the default nameserver it has configured (the default nameserver in the virtual network of the hypervisor). This part of the configuration is called a `stub-zone`. The non-default section of the configuration for the Unbound resolver can be found in appendix B.2. After configuring this, the resolver will query the authoritative name server when it itself is queried for `collision.example`, but it cannot yet verify the responses.

Since in this local, virtual setup, this zone is only signed locally and not part of the global DNSSEC tree, the resolver can not verify the responses automatically by looking higher in the chain (see figure 1). This means that there is no trust established between the resolver and the authoritative name server. To still be able to perform the attack, trust must be established. This can be done by configuring a `trust-anchor`. The trust-anchor should contain the KSK of the zone. This key is copied and put in the configuration of the resolver (appendix B.2). Now the resolver can be queried for the domain, it will perform a lookup by querying the authoritative name server (as configured per `stub-zone`) and then give you the response. This response should not have the status `SERVFAIL`. This is the status for when something has gone wrong, like malformed or unvalidatable. Instead, the status should now be `NOERROR` and the `ad` (authentic data) flag set, which indicates that the DNSSEC validation has succeeded[14].

4.7 The attack

With a signed zone file loaded and working, the authoritative name server (BIND) is ready to assist the attacker in attacking the resolver. Just before attacking, the following command is run on the host machine to continuously capture the CPU utilisation of the resolver and write it to a file:

```
while true do
    xentop -bi2 | awk '$1 == "resolver" {
        print $4 }' >> cpu.txt || break
done
```

With this loop running, the attack (see appendix A.1) can be started. After the attack, the loop can be stopped by interrupting the command. Now the file `cpu.txt` will contain all the CPU utilisation percentages. The values from an idling machine (from just before and just after the attack) are filtered and the file is archived with a name that indicates the resolver and amount of key-tag collisions used in the attack. Then the `cpu.txt` is cleared for the next use. For example:

```
cat cpu.txt | grep -v '0.0\|0.1' > unbound/
    baseline.txt && echo "" > cpu.txt
```

4.8 Wrapping up

After running the attack on both resolvers, with all collision counts, the results are processed by GNU datamash[6]. With this program, the mean and standard deviation for all files are calculated. These values can be found in figure 4. The response times of the queries are processed for all attacks on the Unbound resolver. This is done by the script that can be found in appendix A.2. This script filters all query times from the Unbound logs and writes them to a file. Again, GNU datamash is used to calculate the mean and standard deviation for the results of all attacks. These results can be found in figure 5.

All virtual machine disk images and an archive of all generated keys are available for download here[7].

5. RESULTS

This section is arranged in accordance with the research questions of section 1. The research goal is covered first, after which RQ 1 is covered. The other research questions are discussed in section 6.

5.1 Research goal

The research goal: *Can we render a recursive resolver unavailable (exhaust its resources, DoS-attack) by abusing the added computational overhead that occurs when querying a DNS zone containing multiple keys with the same key-tag?*

5.1.1 CPU utilisation

Rendering a resolver unavailable with this attack is possible in theory, because the CPU utilisation of the resolver increases significantly when under attack, as demonstrated in figure 4. The CPU utilisation did not increase enough in this setup to be rendered unavailable, but it could be. This is discussed some more in section 6.4.1.

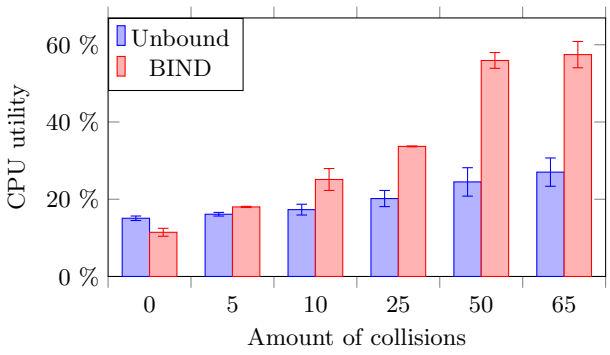

Figure 4. Average CPU utilisation of resolvers for 10000 queries.

5.1.2 Response times

The response times increased only very marginally (see figure 5). It seems therefore that legitimate queries are not hindered by this attack, at least not before the CPU utilisation of the resolver reaches 100% during the attack. Accurately measuring the effect on legitimate queries is future work.

[6] https://www.gnu.org/software/datamash/
[7] https://cloud.dexbleeker.nl/s/rMKPFYn8o9AKqSB

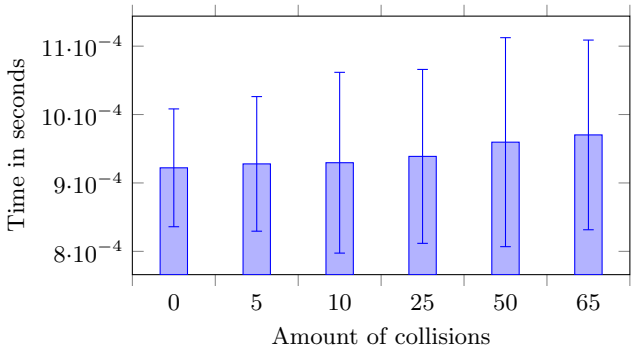

Figure 5. Average response time and error for 10000 queries.

5.2 Research question 1

What is the maximum computing overhead generated by key-tag collisions in a crafted DNS zone? The maximum computing overhead is significant. Without collisions, the CPU utilisation of Unbound is 15.07% on average and 11.4% on average for BIND (see figure 4 again). With 65 collisions, the CPU utilisation reaches 27.02% and 57.45% on average. This means an overhead of 79.26% and 402.28% for Unbound and BIND, respectively. These results, including the result of an independent sample T-test, are displayed more clearly in table 1 and figure 4.

Table 1. Average CPU utilisation for 0 and 65 collisions, with the results of an independent sample T-test.

Resolver	CPU utilisation 0 coll.	65 coll.	Difference	Significance
Unbound	15.07%	27.02%	11.95%	$p < 0.0001$
BIND	11.4%	57.45%	46.01%	$p < 0.0001$

Another interesting observation from figure 4 is that Unbound has a higher CPU utilisation in general than BIND (without collisions). The significance of this is tested with an independent sample T-test: The mean difference is 3.635% with a significance of $p < 0.0001$. Unbound, however, is much better at withstanding the attack as its CPU utilisation does not increase as much as that of the BIND resolver.

6. DISCUSSION

This section covers the discussion of the research. This includes the sub-questions not mentioned in section 5, some notes for implementers of resolvers to mitigate this attack and the limitations of this research.

6.1 Different algorithms

Research question 2 was as follows: *How do the different signing algorithms compare in generated computing overhead when dealing with key-tag collisions?* Comparing the algorithms in terms of generated computing overhead was not possible due to time constraints. However, there are other metrics with which we can compare the algorithms. The RSA keys take longer to generate (around 100ms on average) than the ECDSA keys (around 12ms on average). Generation and this analysis was done on the authority virtual machine, with `haveged` installed. But keys generated with the RSA algorithm tend to collide more often: There was a key-tag that occurred 67 times when over one million RSA keys were generated, while the most common key-tag for keys generated with ECDSA occurred only 16 times. There were more than half a million ECDSA keys generated, so while it is not very straightforward to directly compare the two, it seems like RSA keys more often use the same key-tag.

6.2 Other attack vectors

Research question 3 was: *Can we increase the impact of a key-collision-attack by combining other attack vectors, like a subdomain attack or a phantom-domain attack?* This attack is pointless without a combined attack vector like a subdomain attack. Without querying a different subdomain every iteration, the resolver will cache the validation and the key-tag collision that is setup in the zone will be of no influence. Therefore, this attack always has to be accompanied by the subdomain attack.

6.3 Mitigation

Lastly, research question 4: *What can be done to mitigate key-tag collision attacks? Are the attacks completely preventable or is the impact reducible?* The impact is significantly reduced by having a computing capacity surplus. The machine used in this research only has one processing thread of 1.85Ghz and it was not even close to its limit. Resolvers used in production are likely to have much more resources available and they are likely to be able to withstand attacks like these.

Also, since a subdomain-attack is essential for this attack to work as discussed in section 6.2, the same countermeasures used against a 'normal' subdomain-attack could be implemented and used against this attack as well.

Finally, resolvers should have a smart storage for DNS keys. They should store keys with an index of the domain name, so a key with a key-tag from one domain does not collide with a key with the same key-tag from another domain. They could also implement that the resolver will stop validating after four keys for example, because it is very unlikely that there are more than four keys with the same key-tag in the same zone. Then the attack is significantly limited in its impact, as only four collisions can occur per query. To analyse whether this countermeasure can be implemented without further consequences and whether four is the ideal number for the limit, is future work.

6.4 Limitations

6.4.1 *The attack*

The success of the attack is still limited, however. The CPU utilisation of the attacker increases much more (to around 75%) than that of the resolver (only to around 25%), with the naive way of attacking (see the attack script in appendix A.1). This could be mitigated by using a botnet, as the workload would be divided amongst a lot of machines. Also, the attack is not very straightforward to setup, given the fact that more than one million RSA keys were needed to get 67 keys with the same-key tag. The attack is still very much possible, at least in theory.

6.4.2 *Response times*

The response times (figure 5) are not measured for resolver BIND as there was no reliable and accurate method of obtaining query times. Unbound logs the times each query took and this was parsed and compiled to come up with this graph, but BIND has no such feature.

6.4.3 *Potential targets*

Not all resolvers can be the target of the DoS attack discussed in this paper, as this attack only works on validat-

ing DNSSEC resolvers. Currently, 25.21% of all queries in The Netherlands is validated with DNSSEC[3]. This is about the same as the average for the world, which is 25.40% currently. While this percentage does not directly correspond to the amount of resolvers with DNSSEC validation enabled, it suggests that there are not that many relatively, yet.

7. FUTURE WORK

Given that the response times for the queries only increase marginally (again figure 5), at least when the CPU utilisation does not reach 100%, one could conclude that legitimate queries are not significantly impacted by this attack. This assumption has to be measured and analysed properly. This could be done by attacking a resolver that already has to handle additional query load, or by setting up a 'genuine client' that queries the resolver for common domain names, thus simulating users.

Although the CPU utilisation increased significantly under attack, it did not reach 100%. It is not yet clear what happens when the CPU utilisation of the resolver does reach 100% and how the system and response times are influenced. This could be researched further.

Also, more research has to be done to investigate how to do the attack more efficiently. The attack script (appendix A.1) could be optimised or distributed by using a botnet (as discussed in 6.4.1). This is interesting as the effectiveness of the attack is significantly increased by performing the attack in an optimised manner.

Finally, the impact of this attack using ECDSA keys could be investigated. Since the use of ECDSA could (partially) mitigate the attack[19], it is worth analysing the computational impact of the DoS attack when using ECDSA keys and compare that to the impact it has when using RSA keys (figure 4).

8. CONCLUSION

In this paper, I have analysed and discussed the potential for using DNSSEC key-tag collisions to DoS a validating DNSSEC resolver and shown that this is possible in theory. It is possible because there is a significant CPU overhead when the resolver has to deal with many key-tag collisions, as seen in figure 4. The impact of a subdomain-attack is also discussed and why the attack is not possible without it. Furthermore, generating keys with key-tag collisions is faster with the RSA algorithm than with ECDSA, even though ECDSA keys are generated faster, because the key-tags of RSA keys collide more often. Moreover, key-tag collisions that occur 'in the wild' and are not crafted explicitly to be used as an attack, are likely of no significant influence on the performance of resolvers. This is partly because resolvers have a capacity surplus, but also because they cache very effectively. Finally, abusing key-tag collisions and a subdomain-attack has no significant influence on the response times, at least not when the resolver has a surplus of computing resources during the attack.

9. ACKNOWLEDGEMENTS

I would like to thank Moritz Müller for his supervision, help and support during this research. I would also like to thank all my (peer) reviewers and everyone else that has provided me with much appreciated feedback and constructive criticism.

10. REFERENCES

[1] D. E. E. 3rd. Domain Name System Security Extensions. RFC 2535, Mar. 1999.

[2] D. E. E. 3rd and C. W. Kaufman. Domain Name System Security Extensions. RFC 2065, Jan. 1997.

[3] A.-P. N. I. C. (APNIC). Dnssec validation rate by country, 2019. `https://stats.labs.apnic.net/dnssec`, Last accessed on 2019-06-26.

[4] R. Arends. The quest for the missing keytags. `https://indico.dns-oarc.net/event/22/contributions/315/`, April 2016.

[5] H. Ballani and P. Francis. Mitigating dns dos attacks. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 189–198. ACM, 2008.

[6] S. M. Bellovin. Using the domain name system for system break-ins. In *USENIX Security Symposium*, 1995.

[7] K. Fujiwara, A. Kato, and W. A. Kumari. Aggressive Use of DNSSEC-Validated Cache. RFC 8198, July 2017.

[8] A. Householder, K. Houle, and C. Dougherty. Computer attack trends challenge internet security. *Computer*, 35(4):sulp5–sulp7, April 2002.

[9] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, May 2016.

[10] P. Mockapetris. Domain names: Concepts and facilities. RFC 882, Nov. 1983.

[11] P. Mockapetris. Domain names: Implementation specification. RFC 883, Nov. 1983.

[12] M. Müller, T. Chung, A. Mislove, and R. van Rijswijk-Deij. Rolling with confidence: Managing the complexity of dnssec operations. *IEEE Transactions on Network and Service Management*, 2019.

[13] R. Perdisci, M. Antonakakis, X. Luo, and W. Lee. Wsec dns: Protecting recursive dns resolvers from poisoning attacks. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 3–12. IEEE, 2009.

[14] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends. Protocol Modifications for the DNS Security Extensions. RFC 4035, Mar. 2005.

[15] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends. Resource Records for the DNS Security Extensions. RFC 4034, Mar. 2005.

[16] R. van Rijswijk-Deij. Tag you're it! - revisiting the reality of dnssec keytags. `https://ripe78.ripe.net/`, May 2019 (to be published).

[17] R. van Rijswijk-Deij, K. Hageman, A. Sperotto, and A. Pras. The performance impact of elliptic curve cryptography on dnssec validation. *IEEE/ACM Transactions on Networking (TON)*, 25(2):738–750, 2017.

[18] R. van Rijswijk-Deij, A. Sperotto, and A. Pras. Dnssec and its potential for ddos attacks: a comprehensive measurement study. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 449–460. ACM, 2014.

[19] R. van Rijswijk-Deij, A. Sperotto, and A. Pras. Making the case for elliptic curves in dnssec. *ACM SIGCOMM Computer Communication Review*, 45(5):13–19, 2015.

[20] Wikipedia. Hypervisor — Wikipedia, the free

encyclopedia. http://en.wikipedia.org/wiki/Hypervisor, 2019. [Online; accessed 02-May-2019].

[21] Wikipedia. Hypervisor — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Inode, 2019. [Online; accessed 13-June-2019].

11. APPENDICES

A. SCRIPTS

A.1 attack.sh

```
#!/bin/bash
# Loop 10000 times
for i in {0..10000}
do
    # Query the subdomain
    dig TXT $i.collision.example @192
        .168.122.96 +dnssec > /dev/null 2>&1
        || break
done
exit 0
```

A.2 parse_unbound_logs.sh

```
#!/bin/bash
# This script is to parse attack results and
    write them to a file.
FILE="results-$(date +"%m-%d-%Y--%T").txt"
# Parse query times from logfile and write
    them to a new file
grep "query took" /var/log/unbound/unbound.
    log | awk {‘ print $6 ’} > /root/$FILE
# Clear logfile
echo "" > /var/log/unbound/unbound.log
exit 0
```

A.3 generate_keys.sh

```
#!/bin/bash

DIR=/mnt/keys/rsa
SUB=1
OUTPUT=$DIR/temp

# Create temp output directory
mkdir -p $OUTPUT

# Generate the keys
while true
do
    KEY=‘dnssec-keygen -a RSASHA256 -b 2048 -
        n ZONE -K $OUTPUT collision.example‘
    # KEY=‘dnssec-keygen -a ECDSAP256SHA256 -
        b 2048 -n ZONE -K $OUTPUT collision.
        example‘
    while true
    do
        if [[ -f "$DIR/$SUB/$KEY.key" ]];
            then
        # Collision found! (key exists
            already in this subdir)
        # Increment dircounter
        SUB=$((SUB + 1))
        # Dir does not exist, so we create it
        mkdir -p $DIR/$SUB
    else
        if [[ ! -d "$DIR/$SUB" ]]; then
            mkdir -p $DIR/$SUB
        fi
        break
    fi
    done
    # Finally, move the keys to the proper (
        new or old) subdir and set first
        subdir
    mv $OUTPUT/$KEY* $DIR/$SUB
    sleep 0.1
    SUB=1
done
exit 0
```

B. OTHER

B.1 collision.example.zone

```
$ORIGIN collision.example.
$TTL 1
$include Kcollision.example.+008+27679.key ;
    KSK
$include Kcollision.example.+008+21033.key ;
    ZSK
@                       IN      SOA     ns1.
    collsion.example.   hostmaster.collison.
    example. (
    1474556905  ; serial
    10800       ; refresh after 3 hours
    3600        ; retry after 1 hour
    604800      ; expire after 1 week
    1 )         ; minimum TTL
@                       IN      NS      ns.
    collision.example.
@                       IN      A
    192.168.122.9
ns.collision.example.   IN      A
    192.168.122.9
*                       IN      TXT     "This
     is some TXT record"
```

B.2 unbound.conf

```
server:
  interface: 0.0.0.0
  access-control: 192.168.122.0/24 allow
  access-control: 127.0.0.0/8 allow
  val-log-level: 2
  val-permissive-mode: yes
  logfile: /var/log/unbound/unbound.log
  log-queries: yes
  verbosity: 4

# This part of the configuration makes sure
    that the resolver send the queries for
    the test domain (collision.example) to
    the NS that is running in the other VM.
stub-zone:
  name: "collision.example"
    stub-addr: 192.168.122.9

# This part of the configuration makes sure
    that the resolver trusts the authority of
     collision.example
# This should be the KSK!
trust-anchor:
  "collision.example. IN DNSKEY 257 3 8
      AwEAAcVhI7x+0<snip the rest of the key>
```