DNSOP Working Group
Internet-Draft
Updates: 7873 (if approved)
Intended status: Standards Track
Expires: 17 July 2021

O. Sury
Internet Systems Consortium
W. Toorop
NLnet Labs
D. Eastlake 3rd
Futurewei Technologies
M. Andrews
Internet Systems Consortium

13 January 2021

Interoperable Domain Name System (DNS) Server Cookies
draft-ietf-dnsop-server-cookies-05

Abstract

DNS Cookies, as specified in [RFC7873], are a lightweight DNS transaction security mechanism that provide limited protection to DNS servers and clients against a variety of amplification denial of service, forgery, or cache poisoning attacks by off-path attackers.

This document updates [RFC7873] with precise directions for creating Server Cookies so that an anycast server set including diverse implementations will interoperate with standard clients, suggestions for constructing Client Cookies in a privacy preserving fashion, and suggestions on how to update a Server Secret. An IANA registry listing the methods and associated pseudo random function suitable for creating DNS Server Cookies is created, with the method described in this document as the first and as of yet only entry.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

DNS Cookies, as specified in [RFC7873], are a lightweight DNS transaction security mechanism that provide limited protection to DNS servers and clients against a variety of denial of service amplification, forgery, or cache poisoning attacks by off-path attackers. This document specifies a means of producing interoperable Cookies so that an anycast server set including diverse implementations can be easily configured to interoperate with standard clients. Also single implementation or non-anycast services can benefit from a well-studied standardized algorithm for which the behavioural and security characteristics are more widely known.

The threats considered for DNS Cookies and the properties of the DNS Security features other than DNS Cookies are discussed in [RFC7873].

In [RFC7873] in Section 6 it is "RECOMMENDED for simplicity that the same Server Secret be used by each DNS server in a set of anycast servers." However, how precisely a Server Cookie is calculated from this Server Secret, is left to the implementation.

This guidance has led to a gallimaufry of DNS Cookie implementations, calculating the Server Cookie in different ways. As a result, DNS Cookies are impractical to deploy on multi-vendor anycast networks, because even when all DNS Software share the same secret, as RECOMMENDED in Section 6 of [RFC7873], the Server Cookie constructed by one implementation cannot generally be validated by another.

There is no need for DNS client (resolver) Cookies to be interoperable across different implementations. Each client need only be able to recognize its own cookies. However, this document does contain recommendations for constructing Client Cookies in a client protecting fashion.

1.1. Terminology and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

* "IP address" is used herein as a length independent term covering both IPv4 and IPv6 addresses.

2. Changes to [RFC7873]

In its Appendices A.1 and B.1, [RFC7873] provides example "simple" algorithms for computing Client and Server Cookies, respectively. These algorithms MUST NOT be used as the resulting cookies are too weak when evaluated against modern security standards.

In its Appendix B.2, [RFC7873] provides an example "more complex" server algorithm. This algorithm is replaced by the interoperable specification in Section 4 of this document, which MUST be used by Server Cookie implementations.

This document has suggestions on Client Cookie construction in Section 3. The previous example in Appendix A.2 of [RFC7873] is NOT RECOMMENDED.

3. Constructing a Client Cookie

The Client Cookie acts as an identifier for a given client and its IP address, and needs to be unguessable. In order to provide minimal authentication of the targeted server, a client MUST use a different Client Cookie for each different Server IP address. This complicates a server's ability to spoof answers for other DNS servers. The Client Cookie SHOULD have 64-bits of entropy.

When a server does not support DNS Cookies, the client MUST NOT send the same Client Cookie to that same server again. Instead, it is recommended that the client does not send a Client Cookie to that server for a certain period, for example five minutes, before it retries with a new Client Cookie.

When a server does support DNS Cookies, the client should store the Client Cookie alongside the Server Cookie it registered for that server.

Except for when the Client IP address changes, there is no need to change the Client Cookie often. It is reasonable to change the Client Cookie then only if it has been compromised or after a relatively long implementation-defined period of time. The time period should be no longer than a year, and in any case Client Cookies are not expected to survive a program restart.

Client-Cookie = 64 bits of entropy

Previously, the recommended algorithm to compute the Client Cookie included Client IP address as an input to a hashing function. However, when implementing the DNS Cookies, several DNS vendors found impractical to include the Client IP as the Client Cookie is typically computed before the Client IP address is known. Therefore, the requirement to put Client IP address as input was removed.

However, for privacy reasons, in order to prevent tracking of devices across links and to not circumvent IPv6 Privacy Extensions [RFC4941], clients MUST NOT re-use a Client or Server Cookie after the Client IP address has changed.

One way to satisfy this requirement for non-re-use is to register the Client IP address alongside the Server Cookie when it receives the Server Cookie. In subsequent queries to the server with that Server Cookie, the socket MUST be bound to the Client IP address that was also used (and registered) when it received the Server Cookie. Failure to bind MUST then result in a new Client Cookie.

4. Constructing a Server Cookie

The Server Cookie is effectively a Message Authentication Code (MAC).

The Server Cookie, when it occurs in a COOKIE option in a request, is intended to weakly assure the server that the request came from a client that is both at the source IP address of the request and using the Client Cookie included in the option. This assurance is provided by the Server Cookie that the server (or any other server from the anycast set) sent to that client in an earlier response appearing as the Server Cookie field in the request (see Section 5.2 of [RFC7873]).

DNS Cookies do not provide protection against "on-path" adversaries (see Section 9 of [RFC7873]). An on path observer that has seen a Server Cookie for a client, can abuse that Server Cookie to spoof request for that client within the timespan a Server Cookie is valid (see Section 4.3).

The Server Cookie is calculated from the Client Cookie, a series of Sub-Fields specified below, the Client IP address, and a Server Secret known only to the server, or servers responding on the same address in an anycast set.

For calculation of the Server Cookie, a pseudorandom function is RECOMMENDED with the property that an attacker that does not know the Server Secret, cannot find (any information about) the Server Secret and cannot create a Server Cookie for any combination of - the Client Cookie, the series of Sub-Fields specified below and the client IP address - for which it has not seen a Server Cookie before. Because

DNS servers need to calculate in order to verify Server Cookies, it is RECOMMENDED for the pseudorandom function to be performant. The [SipHash-2-4] pseudorandom function introduced in Section 4.4 fit these recommendations.

Changing the Server Secret regularly is RECOMMENDED but, when a secure pseudorandom function is used, it need not be changed too frequently. For example once a month would be adequate. See Section 5 on operator and implementation guidelines for updating a Server Secret.

The 128-bit Server Cookie consists of Sub-Fields: a 1 octet Version Sub-Field, a 3 octet Reserved Sub-Field, a 4 octet Timestamp Sub-Field and an 8 octet Hash Sub-Field.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Version    |                    Reserved                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Hash                              |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.1. The Version Sub-Field

The Version Sub-Field prescribes the structure and Hash calculation formula. This document defines Version 1 to be the structure and way to calculate the Hash Sub-Field as defined in this Section.

4.2. The Reserved Sub-Field

The value of the Reserved Sub-Field is reserved for future versions of server side Cookie construction. On construction it MUST be set to zero octets. On Server Cookie verification the server MUST NOT enforce those fields to be zero and the Hash should be computed with the received value as described in Section 4.4.

4.3. The Timestamp Sub-Field

The Timestamp value prevents Replay Attacks and MUST be checked by the server to be within a defined period of time. The DNS server SHOULD allow Cookies within 1 hour period in the past and 5 minutes into the future to allow operation of low volume clients and some limited time skew between the DNS servers in the anycast set.

The Timestamp value specifies a date and time in the form of a 32-bit *unsigned* number of seconds elapsed since 1 January 1970 00:00:00 UTC, ignoring leap seconds, in network byte order. All comparisons involving these fields MUST use "Serial number arithmetic", as defined in [RFC1982]. The [RFC1982] specifies how the differences should be handled. This handles any relative time window less than 68 years, at any time in the future (2038 or 2106 or 2256 or 22209 or later.)

The DNS server SHOULD generate a new Server Cookie at least if the received Server Cookie from the client is more than half an hour old, but MAY generate a new cookie more often than that.

4.4. The Hash Sub-Field

It's important that all the DNS servers use the same algorithm for computing the Server Cookie. This document defines the Version 1 of the server side algorithm to be:

```
Hash = SipHash-2-4(
    Client Cookie | Version | Reserved | Timestamp | Client-IP,
    Server Secret )
```

where "|" indicates concatenation.

Notice that Client-IP is used for hash generation even though it is not included in the cookie value itself. Client-IP can be either 4 bytes for IPv4 or 16 bytes for IPv6. The length of all the concatenated elements (the input into [SipHash-2-4]) MUST be either precisely 20 bytes in case of an IPv4 Client-IP or precisely 32 bytes in case of an IPv6 Client-IP.

When a DNS server receives a Server Cookie version 1 for validation, the length of the received COOKIE option MUST be precisely 24 bytes: 8 bytes for the Client Cookie plus 16 bytes for the Server Cookie. Verification of the length of the received COOKIE option is REQUIRED to guarantee the length of the input into [SipHash-2-4] to be precisely 20 bytes in case of an IPv4 Client-IP and precisely 32 bytes in case of an IPv6 Client-IP. This ensures that the input into [SipHash-2-4] is an injective function of the elements making up the input, and thereby prevents data substitution attacks. More specifically, this prevents a 36 byte COOKIE option coming from an IPv4 Client-IP to be validated as if it were coming from an IPv6 Client-IP.

The Server Secret MUST be configurable to make sure that servers in an anycast network return consistent results.

5. Updating the Server Secret

Changing the Server Secret regularly is RECOMMENDED. All servers in an anycast set must be able to verify the Server Cookies constructed by all other servers in that anycast set at all times. Therefore it is vital that the Server Secret is shared among all servers before it is used to generate Server Cookies on any server.

Also, to maximize maintaining established relationships between clients and servers, an old Server Secret should be valid for verification purposes for a specific period.

To facilitate this, deployment of a new Server Secret MUST be done in three stages:

Stage 1
: The new Server Secret is deployed on all the servers in an anycast set by the operator.

  Each server learns the new Server Secret, but keeps using the previous Server Secret to generate Server Cookies.

  Server Cookies constructed with the both the new Server Secret and with the previous Server Secret are considered valid when verifying.

  After stage 1 completed, all the servers in the anycast set have learned the new Server Secret, and can verify Server Cookies constructed with it, but keep generating Server Cookies with the old Server Secret.

Stage 2
: This stage is initiated by the operator after the Server Cookie is present on all members in the anycast set.

  When entering Stage 2, servers start generating Server Cookies with the new Server Secret. The previous Server Secret is not yet removed/forgotten about.

  Server Cookies constructed with the both the new Server Secret and with the previous Server Secret are considered valid when verifying.

Stage 3
: This stage is initiated by the operator when it can be assumed that most clients have obtained a Server Cookie derived from the new Server Secret.

With this stage, the previous Server Secret can be removed and MUST NOT be used anymore for verifying.

We RECOMMEND the operator to wait at least a period to be the longest TTL in the zones served by the server plus 1 hour after it initiated Stage 2, before initiating Stage 3.

The operator SHOULD wait at least longer than the period clients are allowed to use the same Server Cookie, which SHOULD be 1 hour, see Section 4.3.

6. Cookie Algorithms

[SipHash-2-4] is a pseudorandom function suitable as Message Authentication Code. This document REQUIRES compliant DNS server to use SipHash-2-4 as a mandatory and default algorithm for DNS Cookies to ensure interoperability between the DNS Implementations.

The construction method and pseudorandom function used in calculating and verifying the Server Cookies are determined by the initial version byte and by the length of the Server Cookie. Additional pseudorandom or construction algorithms for Server Cookies might be added in the future.

7. IANA Considerations

IANA is requested to create a registry on the "Domain Name System (DNS) Parameters" IANA web page as follows:

Registry Name: DNS Server Cookie Methods
Assignment Policy: Expert Review
Reference: [this document], [RFC7873]
Note: Server Cookie method (construction and pseudorandom algorithm) are determined by the Version in the first byte of the Cookie and by the Cookie size. Server Cookie size is limited to the inclusive range of 8 to 32 bytes.

Version	Size	Method
0	8-32	reserved
1	8-15	unassigned
1	16	SipHash-2-4 [this document] Section 4
1	17-32	unassigned
2-239	8-32	unassigned
240-254	8-32	private use
255	8-32	reserved

Table 1

8. Security and Privacy Considerations

DNS Cookies provide limited protection to DNS servers and clients against a variety of denial of service amplification, forgery or cache poisoning attacks by off-path attackers. They provide no protection against on-path adversaries that can observe the plaintext DNS traffic. An on-path adversary that can observe a Server Cookie for a client and server interaction, can use that Server Cookie for denial of service amplification, forgery or cache poisoning attacks directed at that client for the lifetime of the Server Cookie.

8.1. Client Cookie construction

In [RFC7873] it was RECOMMENDED to construct a Client Cookie by using a pseudorandom function of the Client IP address, the Server IP address, and a secret quantity known only to the client. The Client IP address was included to ensure that a client could not be tracked if its IP address changes due to privacy mechanisms or otherwise.

In this document, we changed Client Cookie construction to be just 64 bits of entropy newly created for each new upstream server the client connects to. As a consequence additional care needs to be taken to prevent tracking of clients. To prevent tracking, a new Client Cookie for a server MUST be created whenever the Client IP address changes.

Unfortunately, tracking Client IP address changes is impractical with servers that do not support DNS Cookies. To prevent tracking of clients with non DNS Cookie supporting servers, a client MUST NOT send a previously sent Client Cookie to a server not known to support DNS Cookies. To prevent the creation of a new Client Cookie for each query to an non DNS Cookies supporting server, it is RECOMMENDED to not send a Client Cookie to that server for a certain period, for example five minutes.

Summarizing:

* In order to provide minimal authentication, a client MUST use a different Client Cookie for each different Server IP address.

* To prevent tracking of clients, a new Client Cookie MUST be created when the Client IP address changes.

* To prevent tracking of clients by a non DNS Cookie supporting server, a client MUST NOT send a previously sent Client Cookie to a server in the absence of an associated Server Cookie.

Note that it is infeasible for a client to detect change of the public IP address when the client is behind a routing device performing Network Address Translation (NAT). A server may track the public IP address of that routing device performing the NAT. Preventing tracking of the public IP of a NAT performing routing device is beyond the scope of this document.

8.2. Server Cookie construction

[RFC7873] did not give a precise recipe for constructing Server Cookies, but did recommend usage of a pseudorandom function strong enough to prevent guessing of cookies. In this document SipHash-2-4 is assigned as the pseudorandom function to be used for version 1 Server Cookies. SipHash-2-4 is considered sufficiently strong for the immediate future, but predictions about future development in cryptography and cryptanalysis are beyond the scope of this document.

The precise structure of version 1 Server Cookies is defined in this document. Portion of the structure is made up of unhashed data elements which are exposed in clear text to an on-path observer. These unhashed data elements are taken along as input to the SipHash-2-4 function of which the result is the other portion of the Server Cookie, so the unhashed portion of the Server Cookie can not by changed by an on-path attacking without also recalculating the hashed portion for which the Server Secret needs to be known.

One of the elements in the unhashed portion of version 1 Server Cookies is a Timestamp used to prevent Replay Attacks. Servers verifying version 1 Server Cookies need to have access to a reliable time value to compare with the Timestamp value, that cannot be altered by an attacker. Furthermore, all servers participating in an anycast set that validate version 1 Server Cookies need to have their clocks synchronized.

The cleartext Timestamp data element reveal to an on-path adversary using an observed Server Cookie to attack the client for which the Server Cookie was constructed (as shown in the first paragraph of this Section), the lifetime the observed Server Cookie can be used for the attack.

In addition to the Security Considerations in this section, the Security Considerations section of [RFC7873] still apply.

9. Acknowledgements

Thanks to Witold Krecicki and Pieter Lexis for valuable input, suggestions and text and above all for implementing a prototype of an interoperable DNS Cookie in Bind9, Knot and PowerDNS during the hackathon of IETF104 in Prague. Thanks for valuable input and suggestions go to Ralph Dolmans, Bob Harold, Daniel Salzman, Martin Hoffmann, Mukund Sivaraman, Petr Spacek, Loganaden Velvindron, Bob

Harold, Philip Homburg, Tim Wicinski and Brian Dickson.

10. Normative References

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, DOI 10.17487/RFC1982, August 1996, <https://www.rfc-editor.org/info/rfc1982>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC7873] Eastlake 3rd, D. and M. Andrews, "Domain Name System (DNS) Cookies", RFC 7873, DOI 10.17487/RFC7873, May 2016, <https://www.rfc-editor.org/info/rfc7873>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SipHash-2-4] Aumasson, J. and D. J. Bernstein, "SipHash: a fast short-input PRF", Progress in Cryptology - INDOCRYPT 2012. Lecture Notes in Computer Science, vol 7668. Springer., 2012, <https://doi.org/10.1007/978-3-642-34931-7_28>.

11. Informative References

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <https://www.rfc-editor.org/info/rfc4941>.

Appendix A. Test vectors

A.1. Learning a new Server Cookie

A resolver (client) sending from IPv4 address 198.51.100.100, sends a query for "example.com" to an authoritative server listening on 192.0.2.53 from which it has not yet learned the server cookie.

The DNS requests and replies shown in this Appendix, are in a "dig" like format. The content of the DNS COOKIE Option is shown in hexadecimal format after "; COOKIE:".

```
;; Sending:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57406
;; flags:; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2464c4abcf10c957
;; QUESTION SECTION:
;example.com.                IN      A

;; QUERY SIZE: 52
```

The authoritative nameserver (server) is configured with the following secret: e5e973e5a6b2a43f48e7dc849e37bfcf (as hex data).

It receives the query at Wed Jun 5 10:53:05 UTC 2019.

The content of the DNS COOKIE Option that the server will return is shown below in hexadecimal format after "; COOKIE:".

The Timestamp field Section 4.3 in the returned Server Cookie has value 1559731985. In [RFC3339] format this is 2019-06-05 10:53:05+00:00.

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57406
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
```

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2464c4abcf10c957010000005cf79f111f8130c3eee29480 (good)
;; QUESTION SECTION:
;example.com.                IN      A

;; ANSWER SECTION:
example.com.         86400   IN      A       192.0.2.34

;; Query time: 6 msec
;; SERVER: 192.0.2.53#53(192.0.2.53)
;; WHEN: Wed Jun  5 10:53:05 UTC 2019
;; MSD SIZE  rcvd: 84
```

A.2. The same client learning a renewed (fresh) Server Cookie

40 minutes later, the same resolver (client) queries the same server for "example.org". It reuses the Server Cookie it learned in the previous query.

The Timestamp field in that previously learned Server Cookie, which is now send along in the request, was and is 1559731985. In [RFC3339] format this is 2019-06-05 10:53:05+00:00.

```
;; Sending:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50939
;; flags:; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2464c4abcf10c957010000005cf79f111f8130c3eee29480
;; QUESTION SECTION:
;example.org.                IN      A

;; QUERY SIZE: 52
```


The authoritative nameserver (server) now generates a new Server Cookie. The server SHOULD do this because it can see the Server Cookie send by the client is older than half an hour Section 4.3, but it is also fine for a server to generate a new Server Cookie sooner, or even for every answer.

The Timestamp field in the returned new Server Cookie has value 1559734385, which in [RFC3339] format is 2019-06-05 11:33:05+00:00.

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50939
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2464c4abcf10c957010000005cf7a871d4a564a1442aca77 (good)
;; QUESTION SECTION:
;example.org.                IN      A

;; ANSWER SECTION:
example.org.         86400   IN      A       192.0.2.34

;; Query time: 6 msec
;; SERVER: 192.0.2.53#53(192.0.2.53)
;; WHEN: Wed Jun  5 11:33:05 UTC 2019
;; MSD SIZE  rcvd: 84
```

A.3. Another client learning a renewed Server Cookie

Another resolver (client) with IPv4 address 203.0.113.203 sends a request to the same server with a valid Server Cookie that it learned before (at Wed Jun 5 09:46:25 UTC 2019).

The Timestamp field in Server Cookie in the request has value 1559727985, which in [RFC3339] format is 2019-06-05 09:46:25+00:00.

Note that the Server Cookie has Reserved bytes set, but is still valid with the configured secret; the Hash part is calculated taking along the Reserved bytes.

```
;; Sending:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34736
;; flags:; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: fc93fc62807ddb8601abcdef5cf78f71a314227b6679ebf5
;; QUESTION SECTION:
;example.com.                IN      A

;; QUERY SIZE: 52
```

The authoritative nameserver (server) replies with a freshly generated Server Cookie for this client conformant with this specification; so with the Reserved bits set to zero.

The Timestamp field in the returned new Server Cookie has value 1559734700, which in [RFC3339] format is 2019-06-05 11:38:20+00:00.

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34736
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: fc93fc62807ddb86010000005cf7a9acf73a7810aca2381e (good)
;; QUESTION SECTION:
;example.com.                IN      A

;; ANSWER SECTION:
example.com.        86400   IN      A       192.0.2.34

;; Query time: 6 msec
;; SERVER: 192.0.2.53#53(192.0.2.53)
;; WHEN: Wed Jun  5 11:38:20 UTC 2019
;; MSD SIZE  rcvd: 84
```

A.4. IPv6 query with rolled over secret

The query below is from a client with IPv6 address 2001:db8:220:1:59de:d0f4:8769:82b8 to a server with IPv6 address 2001:db8:8f::53. The client has learned a valid Server Cookie before (at Wed Jun 5 13:36:57 UTC 2019) when the Server had the secret: dd3bdf9344b678b185a6f5cb60fca715. The server now uses a new secret, but it can still validate the Server Cookie provided by the client as the old secret has not expired yet.

The Timestamp field in the Server Cookie in the request has value 1559741817, which in [RFC3339] format is 2019-06-05 13:36:57+00:00.

```
;; Sending:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6774
;; flags:; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 22681ab97d52c298010000005cf7c57926556bd0934c72f8
;; QUESTION SECTION:
;example.net.                IN      A

;; QUERY SIZE: 52
```

The authoritative nameserver (server) replies with a freshly generated server cookie for this client with its new secret: 445536bcd2513298075a5d379663c962

The Timestamp field in the returned new Server Cookie has value 1559741961, which in [RFC3339] format is .

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6774
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 22681ab97d52c298010000005cf7c609a6bb79d16625507a (good)
;; QUESTION SECTION:
;example.net.                IN      A

;; ANSWER SECTION:
example.net.         86400   IN      A       192.0.2.34

;; Query time: 6 msec
;; SERVER: 2001:db8:8f::53#53(2001:db8:8f::53)
;; WHEN: Wed Jun  5 13:36:57 UTC 2019
;; MSD SIZE  rcvd: 84
```

Appendix B. Implementation status

At the time of writing, BIND from version 9.16 and Knot DNS from version 2.9.0 create Server Cookies according to the recipe described in this draft. Unbound and NSD have an Proof of Concept implementation that has been tested for interoperability during the hackathon at the IETF104 in Prague. Construction of privacy maintaining Client Cookies according to the directions in this draft

have been implemented in the getdns library and will be in the upcoming getdns-1.6.1 release and in Stubby version 0.3.1.

Authors' Addresses

Ondrej Sury
Internet Systems Consortium
Czechia

Email: ondrej@isc.org

Willem Toorop
NLnet Labs
Science Park 400
1098 XH Amsterdam
Netherlands

Email: willem@nlnetlabs.nl

Donald E. Eastlake 3rd
Futurewei Technologies
1424 Pro Shop Court
Davenport, FL 33896
United States of America

Phone: +1-508-333-2270
Email: d3e3e3@gmail.com

Mark Andrews
Internet Systems Consortium

950 Charter Street
Redwood City, CA 94063
United States of America

Email: marka@isc.org