

INTERNET-DRAFT  
Intended Status: Proposed Standard

Donald Eastlake  
Huawei  
Mark Andrews  
ISC  
November 30, 2014

Expires: May 29, 2015

Domain Name System (DNS) Cookies  
<[draft-ietf-dnsop-cookies-00.txt](#)>

## Abstract

DNS cookies are a lightweight DNS transaction security mechanism that provides limited protection to DNS servers and clients against a variety of increasingly common denial-of-service and amplification / forgery or cache poisoning attacks by off-path attackers. DNS Cookies are tolerant of NAT, NAT-PT, and anycast and can be incrementally deployed.

## Status of This Document

This Internet-Draft is submitted to IETF in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Distribution of this document is unlimited. Comments should be sent to the author or the DNSEXT mailing list <dnsext@ietf.org>.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/lid-abstracts.html>. The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

INTERNET-DRAFT

DNS Cookies

## Table of Contents

<a href="#">1. Introduction.....</a>	<a href="#">3</a>
<a href="#">1.1 Contents of This Document.....</a>	<a href="#">3</a>
<a href="#">1.2 Definitions.....</a>	<a href="#">4</a>
<a href="#">2. Threats Considered.....</a>	<a href="#">5</a>
<a href="#">2.1 Denial-of-Service Attacks.....</a>	<a href="#">5</a>
<a href="#">2.1.1 DNS Amplification Attacks.....</a>	<a href="#">5</a>
<a href="#">2.1.2 DNS Server Denial-of-Service.....</a>	<a href="#">5</a>
<a href="#">2.2 Cache Poisoning and Answer Forgery Attacks.....</a>	<a href="#">6</a>
<a href="#">3. Comments on Existing DNS Security.....</a>	<a href="#">7</a>
<a href="#">3.1 Existing DNS Data Security.....</a>	<a href="#">7</a>
<a href="#">3.2 DNS Message/Transaction Security.....</a>	<a href="#">7</a>
<a href="#">3.3 Conclusions on Existing DNS Security.....</a>	<a href="#">7</a>
<a href="#">4. The COOKIE OPT Option.....</a>	<a href="#">8</a>
<a href="#">4.1 Client Cookie.....</a>	<a href="#">9</a>
<a href="#">4.2 Server Cookie.....</a>	<a href="#">9</a>
<a href="#">4.3 Error Code.....</a>	<a href="#">10</a>
<a href="#">5. DNS Cookies Protocol Description.....</a>	<a href="#">11</a>
<a href="#">5.1 Originating Requests.....</a>	<a href="#">11</a>
<a href="#">5.2 Responding to Requests.....</a>	<a href="#">11</a>
<a href="#">5.2.1 No OPT RR.....</a>	<a href="#">12</a>
<a href="#">5.2.2 No Valid Client Cookie.....</a>	<a href="#">12</a>
<a href="#">5.2.3 Bad or Absent Server Cookie.....</a>	<a href="#">13</a>
<a href="#">5.2.4 A Correct Server Cookie.....</a>	<a href="#">13</a>
<a href="#">5.3 Processing Responses.....</a>	<a href="#">14</a>
<a href="#">5.4 Client and Server Secret Rollover.....</a>	<a href="#">14</a>
<a href="#">5.5 Implementation Requirement.....</a>	<a href="#">15</a>
<a href="#">6. NAT Considerations and AnyCast Server Considerations...</a>	<a href="#">16</a>
<a href="#">7. Deployment.....</a>	<a href="#">18</a>
<a href="#">8. IANA Considerations.....</a>	<a href="#">19</a>
<a href="#">9. Security Considerations.....</a>	<a href="#">20</a>
<a href="#">9.1 Cookie Algorithm Considerations.....</a>	<a href="#">20</a>

Acknowledgements.....	<a href="#">21</a>
Normative References.....	<a href="#">22</a>
Informative References.....	<a href="#">22</a>
 <a href="#">Appendix A</a> : Example Client Cookie Algorithms.....	<a href="#">24</a>
<a href="#">A.1</a> A Simple Algorithm.....	<a href="#">24</a>
<a href="#">A.2</a> A More Complex Algorithm.....	<a href="#">24</a>
 <a href="#">Appendix B</a> : Example Server Cookie Algorithms.....	<a href="#">25</a>
<a href="#">B.1</a> A Simple Algorithm.....	<a href="#">25</a>
<a href="#">B.2</a> A More Complex Algorithm.....	<a href="#">25</a>

## [1](#). Introduction

As with many core Internet protocols, the Domain Name System (DNS) was originally designed at a time when the Internet had only a small pool of trusted users. As the Internet has grown exponentially to a global information utility, the DNS has increasingly been subject to abuse.

This document describes DNS cookies, a lightweight DNS transaction security mechanism specified as an OPT [[RFC6891](#)] option. This mechanism provides limited protection to DNS servers and clients against a variety of increasingly common abuses by off-path attackers. It is compatible with and can be used in conjunction with other DNS transaction forgery resistance measures such as those in [[RFC5452](#)].

The protection provided by DNS cookies bears some resemblance to that provided by using TCP for DNS transactions. To bypass the weak protection provided by using TCP requires an off-path attacker guessing the 32-bit TCP sequence number in use. To bypass the weak protection provided by DNS Cookies requires such an attacker to guess a 64-bit pseudo-random quantity. Where DNS Cookies are not available but TCP is, a fall back to using TCP is a reasonable strategy.

If only one party to a DNS transaction supports DNS cookies, the mechanism does not provide a benefit or significantly interfere; but, if both support it, the additional security provided is automatically available.

The DNS cookies mechanism is designed to work in the presence of NAT and NAT-PT boxes and guidance is provided herein on supporting the

DNS cookies mechanism in anycast servers.

## [1.1](#) Contents of This Document

In [Section 2](#), we discuss the threats against which the DNS cookie mechanism provides some protection.

[Section 3](#) describes existing DNS security mechanisms and why they are not adequate substitutes for DNS cookies.

[Section 4](#) describes the COOKIE OPT option.

[Section 5](#) provides a protocol description.

[Section 6](#) discusses some NAT and anycast related DNS Cookies design considerations.

Donald Eastlake & Mark Andrews

[Page 3]

---

INTERNET-DRAFT

DNS Cookies

[Section 7](#) discusses incremental deployment considerations.

Sections [8](#) and [9](#) describe IANA and Security Considerations.

## [1.2](#) Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)].

An "off-path attacker", for a particular DNS client and server, is defined as an attacker who cannot observe the DNS request and response messages between that client and server.

"Soft state" indicates information learned or derived by a host which may be discarded when indicated by the policies of that host but can be later re-instantiated if needed. For example, it could be discarded after a period of time or when storage for caching such data becomes full. If operations requiring that soft state continue after it has been discarded, it will be automatically re-generated, albeit at some cost.

"Silently discarded" indicates that there are no DNS protocol message consequences; however, it is RECOMMENDED that appropriate network management facilities be included in implementations, such as a counter of the occurrences of each such event type.

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

## [2.](#) Threats Considered

DNS cookies are intended to provide significant but limited protection against certain attacks by off-path attackers as described below. These attacks include denial-of-service, cache poisoning and answer forgery.

### [2.1](#) Denial-of-Service Attacks

The typical form of the denial-of-service attacks considered herein is to send DNS requests with forged source IP addresses to a server. The intent can be to attack that server or some other selected host as described below.

### [2.1.1](#) DNS Amplification Attacks

A request with a forged IP address generally causes a response to be sent to that forged IP address. Thus the forging of many such requests with a particular source IP address can result in enough traffic being sent to the forged IP address to interfere with service to the host at the IP address. Furthermore, it is generally easy in the DNS to create short requests that produce much longer responses, thus amplifying the attack.

The DNS Cookies mechanism can severely limit the traffic amplification obtained by attackers off path for the server and the attacked host. Enforced DNS cookies would make it hard for an off path attacker to cause any more than rate-limited short error responses to be sent to a forged IP address so the attack would be attenuated rather than amplified. DNS cookies make it more effective to implement a rate limiting scheme for error responses from the server. Such a scheme would further restrict selected host denial-of-service traffic from that server.

### [2.1.2](#) DNS Server Denial-of-Service

DNS requests that are accepted cause work on the part of DNS servers. This is particularly true for recursive servers that may issue one or more requests and process the responses thereto, in order to determine their response to the initial request. And the situation can be even worse for recursive servers implementing DNSSEC ([[RFC4033](#)] [[RFC4034](#)] [[RFC4035](#)]) because they may be induced to perform burdensome cryptographic computations in attempts to verify the authenticity of data they retrieve in trying to answer the

request.

The computational or communications burden caused by such requests may not dependent on a forged IP source address, but the use of such addresses makes

- + the source of the requests causing the denial-of-service attack harder to find and

- + restriction of the IP addresses from which such requests should be honored hard or impossible to specify or verify.

Use of DNS cookies should enables a server to reject forged queries from an off path attacker with relative ease and before any recursive queries or public key cryptographic operations are performed.

## [2.2](#) Cache Poisoning and Answer Forgery Attacks

The form of the cache poisoning attacks considered is to send forged replies to a resolver. Modern network speeds for well-connected hosts are such that, by forging replies from the IP addresses of a DNS server to a resolver for common names or names that resolver has been induced to resolve, there can be an unacceptably high probability of randomly coming up with a reply that will be accepted and cause false DNS information to be cached by that resolver (the Dan Kaminsky attack). This can be used to facilitate phishing attacks and other diversion of legitimate traffic to a compromised or malicious host such as a web server.

With the use of DNS cookies, a resolver can generally reject such forged replies.

### 3. Comments on Existing DNS Security

Two forms of security have been added to DNS, data security and message/transaction security.

#### 3.1 Existing DNS Data Security

DNS data security is one part of DNSSEC and is described in [\[RFC4033\]](#), [\[RFC4034\]](#), and [\[RFC4035\]](#) and updates thereto. It provides data origin authentication and authenticated denial of existence. DNSSEC is being deployed and can provide strong protection against forged data; however, it has the unintended effect of making some denial-of-service attacks worse because of the cryptographic computational load it can require and the increased size in DNS response packets that it tends to produce.

#### 3.2 DNS Message/Transaction Security

The second form of security that has been added to DNS provides "transaction" security through TSIG [\[RFC2845\]](#) or SIG(0) [\[RFC2931\]](#). TSIG could provide strong protection against the attacks for which the DNS Cookies mechanism provide weak protection; however, TSIG is non-trivial to deploy in the general Internet because of the burden it imposes of pre-agreement and key distribution between client-server pairs, the burden of server side key state, and because it requires time synchronization between client and server.

TKEY [\[RFC2930\]](#) can solve the problem of key distribution for TSIG but some modes of TKEY impose a substantial cryptographic computation loads and can be dependent on the deployment of DNS data security (see [Section 3.1](#)).

SIG(0) [\[RFC2931\]](#) provides less denial of service protection than TSIG or, in one way, even DNS cookies, because it does not authenticate requests, only complete transactions. In any case, it also depends on the deployment of DNS data security and requires computationally burdensome public key cryptographic operations.

#### 3.3 Conclusions on Existing DNS Security

The existing DNS security mechanisms do not provide the services provided by the DNS Cookies mechanism: lightweight message authentication of DNS requests and responses with no requirement for pre-configuration or per client server side state.

INTERNET-DRAFT

DNS Cookies

#### 4. The COOKIE OPT Option

COOKIE is an OPT RR [RFC6891] option that can be included in the RDATA portion of an OPT RR in DNS requests and responses. The option length varies depending on the circumstance in which it is being used. There are two cases as described below. Both use the same OPTION-CODE; they are distinguished by their length.

In a request sent by a client to a server when the client does not know the server cookie, its length is 10, consisting of a 2 bytes DNS error code field followed by the 8 byte Client Cookie as shown in Figure 1.

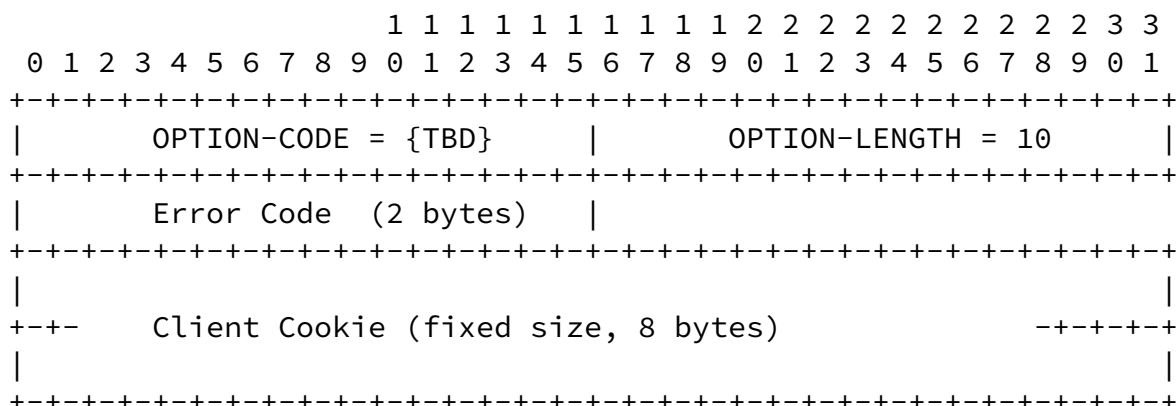


Figure 1. COOKIE Option, Unknown Server Cookie

In a request sent by a client when a server cookie is known and in all responses, the length is variable from 18 to 42 bytes, consisting of a 2 byte DNS error field followed by the 8 bytes Client Cookie and then the variable 8 to 32 bytes Server Cookie as shown in Figure 2. The variability of the option length stems from the variable length Server Cookie. The Server Cookie is an integer number of bytes with a minimum size of 64 bits for security and a maximum size of 256 bits for implementation convenience.

INTERNET-DRAFT

DNS Cookies

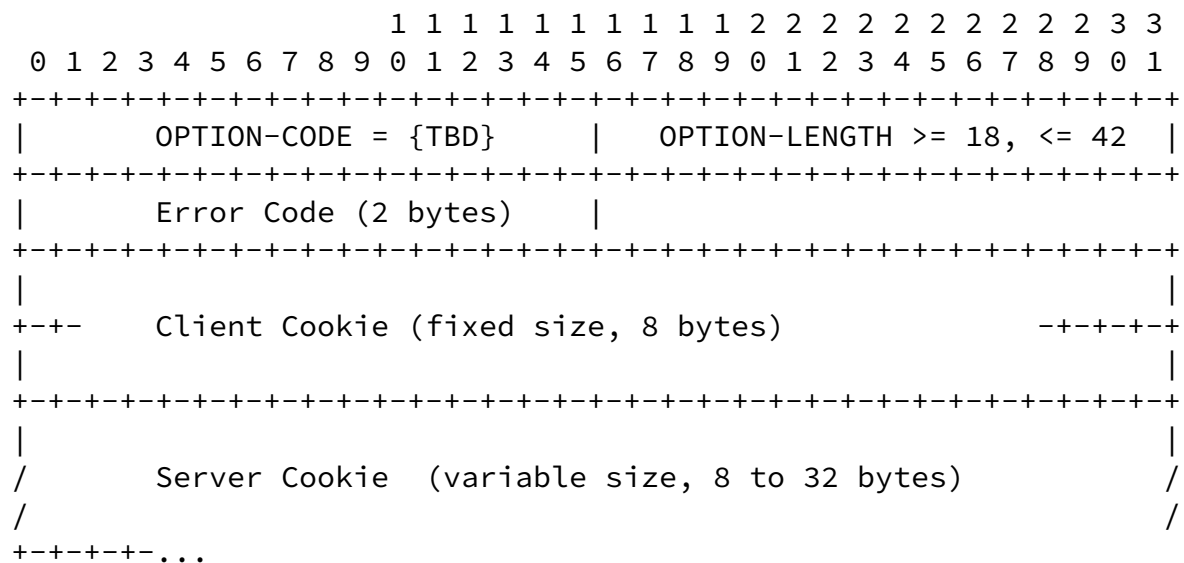


Figure 2. COOKIE Option, Known Server Cookie

#### 4.1 Client Cookie

The Client Cookie SHOULD be a pseudo-random function of the server IP address and a secret quantity known only to the client. This client secret SHOULD have at least 64 bits of entropy [RFC4086] and be changed periodically (see [Section 5.4](#)). The selection of the pseudo-random function is a matter private to the client as only the client needs to recognize its own DNS cookies.

For further discussion of the Client Cookie field, see [Section 5.1](#). For example methods of determining a Client Cookie, see [Appendix A](#).

A client MUST NOT use the same Client Cookie value for queries to all

servers.

## [4.2](#) Server Cookie

The Server Cookie SHOULD consist of or include a 64-bit or larger pseudo-random function of the request source IP address, the request Client Cookie, and a secret quantity known only to the server. (See [Section 6](#) for a discussion of why the Client Cookie is used as input to the Server Cookie but the Server Cookie is not used as an input to the Client Cookie.) This server secret SHOULD have at least 64 bits of entropy [[RFC4086](#)] and be changed periodically (see [Section 5.4](#)). The selection of the pseudo-random function is a matter private to the server as only the server needs to recognize its own DNS cookies.

Donald Eastlake & Mark Andrews

[Page 9]

---

INTERNET-DRAFT

DNS Cookies

For further discussion of the Server Cookie field see [Section 5.2](#). For example methods of determining a Server Cookie, see [Appendix B](#).

A server MUST NOT use the same Server Cookie value for responses to all clients.

## [4.3](#) Error Code

In requests, the Error Code field MUST be zero and is ignored on receipt. Replies have a COOKIE OPT with an Error Code equal to one of the following four values: Zero (if the request they respond to had a COOKIE OPT with a correct Server Cookie), NOCOOKIE, MFCOOKIE, or BADCOOKIE.

NOCOOKIE and MFCOOKIE indicate that the server did not receive a Client Cookie, either because there was no COOKIE OPT option in the request (NOCOOKIE) or one was present but the COOKIE OPT option was malformed as not being a valid length (MFCOOKIE). BADCOOKIE indicates that the server did receive a Client Cookie but did not receive the correct Server Cookie either because there was no Server Cookie present or because it was not a valid value.

A server may choose to normally process a request, for example returning the normal answer information for a QUERY, notwithstanding

a cookie error condition. For more information on error processing, see [Section 5](#).

## [5](#). DNS Cookies Protocol Description

This section discusses using DNS Cookies in the DNS Protocol.

### [5.1](#) Originating Requests

A DNS client that implements DNS cookies includes one DNS Cookie option in every DNS request it sends unless DNS cookies are disabled. The COOKIE OPT option in a request always includes a zero Error Code field and a Client Cookie as discussed in [Section 4.1](#).

If the client has no Server Cookie obtained from a previous DNS response and cached under the server's IP address, it uses the shorter form of COOKIE OPT shown in Figure 1. If the client does have such a cached Server Cookie, it uses the form of COOKIE OPT shown in Figure 2 and also includes that cached Server Cookie in the DNS

option it sends.

## [5.2](#) Responding to Requests

The Server Cookie, when it occurs in a COOKIE OPT 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 weak assurance is provided by the Server Cookie that server would send to that client in an earlier response appearing as the Server Cookie field in the request.

At a server where DNS Cookies are not implemented and enabled, the presence of a COOKIE OPT option is ignored and the server responds as before.

When DNS Cookies are implemented and enabled, there are four possibilities: (1) there is no OPT RR at all in the request; (2) there is no valid Client Cookie in the request because the COOKIE OPT option is absent from the request or one is present but not a legal length; (3) there is a valid length cookie option in the request with no Server Cookie or an incorrect Server Cookie; or (4) there is a cookie option in the request with a correct Server Cookie. The four possibilities are discussed in the subsections below.

In the case of multiple COOKIE OPT options in a request, only the first (the one closest to the DNS header) is considered. All others are ignored.

### [5.2.1](#) No OPT RR

If there is no OPT RR in the request, the client does not support EDNS since [\[RFC6891\]](#) requires that an OPT RR be included in a request if the requester supports that feature. Under these circumstances, the server cannot expect to ever receive a correct COOKIE OPT option from the client as in [Section 5.2.4](#).

The situation and server options available are the same as those in [Section 5.2.2](#) except that no OPT RR can be included in any response.

### [5.2.2](#) No Valid Client Cookie

A request with an OPT RR but no COOKIE OPT option or with a COOKIE that is not a valid length (10 or 18 through 42) could be from a client that does not implement DNS cookies or on which they are disabled or it could be some form of abuse or broken client implementation. A server on which DNS cookies are enabled has the following three choices in responding to such a request:

- (1) Silently discard the request.
- (2) Not process the request other than returning a minimal length error response. Because of the absence of a validly formatted COOKIE OPT option in the request, it cannot be assumed that the client would understand any new RCODE values. An RCODE of Refused is returned and the Error Field of the returned COOKIE OPT option is set to NOCOOKIE if there was no COOKIE OPT option in the request and set to MFCookie if such an option was present but not a valid length.
- (3) Process the request normally and provide a normal response except that a COOKIE OPT option with a non-zero Error Field is included as in point 2 above. The RCODE in the DNS Header is zero unless some non-cookie error occurs in processing the request.

Server policy determines how often the server selects each of the above response choices; however, if the request was received over TCP, the server may wish to take the weak authentication provided by the use of TCP into account, increasing the probability of choice 3 and decreasing the probability of choice 1 perhaps to the extent of never choosing 1. For both response choices 2 and 3, the server should consider setting TC=1 in the response so that future requests from the client are more likely to be received with the weak authentication that can be provided by TCP.

### [5.2.3](#) Bad or Absent Server Cookie

If a request is received with the COOKIE OPT option having no Server Cookie value (length 10) or a bad Server Cookie value (length 18 to 42), it could be some attempted abuse or it could just be that the client does not know a currently valid Server Cookie for the server to which the request was sent. For example, the client might have an old, no longer recognized Server Cookie

Servers MUST, at least occasionally, respond to such requests to inform the client of the correct Server Cookie. This is necessary so that such a client can bootstrap to the weakly secure state where requests and responses have recognized Server Cookies and Client Cookies.

In responding to such a request, the server has the following three choices:

- (1) Silently discard the request.
- (2) Not process the request other than returning a minimal length error response. Because of the correct length COOKIE OPT option in the request, the client can be assumed to understand the new error codes assigned in this document. Both the Error Field in the returned COOKIE OPT option and the extended RCODE are set to BADCOOKIE.
- (3) Processes the request normally and sends its usual response including a COOKIE OPT option with an Error field of BADCOOKIE and a zero RCODE (unless there was also a non-cookie error in processing the request).

Server policy determines how often the server selects each of the above response choices; however, if the request was received over TCP, the server may wish to take the weak authentication provided by the use of TCP into account, increasing the probability of choice 3 and decreasing the probability of choice 1 perhaps to the extent of never choosing 1.

#### [5.2.4](#) A Correct Server Cookie

If a server with enabled DNS cookies receives a request where the COOKIE OPT option has a valid length and correct Server Cookie, it processes the request normally and includes a COOKIE OPT option with a zero Error Field in the response. Such a response might have a non-zero RCODE if a non-cookie error occurs in processing the request.

### [5.3](#) Processing Responses

The Client Cookie, when it occurs in a COOKIE OPT option in a DNS reply, is intended to weakly assure the client that the reply came from a server at the source IP address use in the response packet because the Client Cookie value is the value that client would send to that server in a request. If there are multiple COOKIE OPT options in a DNS reply, all but the first (the one closest to the DNS Header) are ignored.

A DNS client where DNS cookies are implemented and enabled examines response for DNS cookies and MUST discard the response if it contains an illegal COOKIE OPT option length or an incorrect Client Cookie value. If the COOKIE OPT option Client Cookie is correct, the client caches the Server Cookie provided even if the response is an error response (RCODE non-zero).

If the reply extended RCODE is BADCOOKIE, it means that the server was unwilling to process the request because it did not have the correct Server Cookie in it. The client should retry the request using the new Server Cookie from the response.

If the RCODE is some value other than BADCOOKIE, including zero, the response is then processed normally.

### [5.4](#) Client and Server Secret Rollover

Clients and servers MUST NOT continue to use the same secret in new queries and responses, respectively, for more than 14 days and SHOULD NOT continue to do so for more than 1 day. Many clients rolling over their secret at the same time could briefly increase server traffic and exactly predictable rollover times for clients or servers might facilitate guessing attacks. For example, an attacker might increase the priority of attacking secrets they believe will be in effect for an extended period of time. To avoid rollover synchronization and predictability, it is RECOMMENDED that pseudorandom jitter of at least 30% be applied to the time of a scheduled rollover of a DNS cookie secret.

It is RECOMMENDED that a client keep the Client Cookie it is expecting in a reply associated with the outstanding query to avoid rejection of replies due to a bad Client Cookie right after a change in the Client Secret. It is RECOMMENDED that a server retain its previous secret for a period of time not less than 1 second or more

than 3 minutes, after a change in its secret, and consider queries with Server Cookies based on its previous secret to have a correct Server Cookie during that time.

Receiving a sudden increased level of requests with bad Server Cookies or replies with bad Client Cookies would be a good reason to believe a server or client is likely to be under attack and should consider more frequent rollover of its secret.

### [5.5](#) Implementation Requirement

DNS clients and servers SHOULD implement DNS cookies to decrease their vulnerability to the threats discussed in [Section 2](#).

---

INTERNET-DRAFT

DNS Cookies

## 6. NAT Considerations and AnyCast Server Considerations

In the Classic Internet, DNS Cookies could simply be a pseudo-random function of the client IP address and a sever secret or the server IP address and a client secret. You would want to compute the Server Cookie that way, so a client could cache its Server Cookie for a particular server for an indefinitely amount of time and the server could easily regenerate and check it. You could consider the Client Cookie to be a weak client signature over the server IP address that the client checks in replies and you could extend this weak signature to cover the request ID, for example, or any other information that is returned unchanged in the reply.

But we have this reality called NAT [[RFC3022](#)], Network Address Translation (including, for the purposes of this document, NAT-PT, Network Address and Protocol Translation, which has been declared Historic [[RFC4966](#)]). There is no problem with DNS transactions between clients and servers behind a NAT box using local IP addresses. Nor is there a problem with NAT translation of internal addresses to external addresses or translations between IPv4 and IPv6 addresses, as long as the address mapping is relatively stable. Should the external IP address an internal client is being mapped to change occasionally, the disruption is little more than when a client rolls-over its DNS COOKIE secret. And normally external access to a DNS server behind a NAT box is handled by a fixed mapping which forwards externally received DNS requests to a specific host.

However, NAT devices sometimes also map ports. This can cause multiple DNS requests and responses from multiple internal hosts to be mapped to a smaller number of external IP addresses, such as one

address. Thus there could be many clients behind a NAT box that appear to come from the same source IP address to a server outside that NAT box. If one of these were an attacker (think Zombie or Botnet), that behind-NAT attacker could get the Server Cookie for some server for the outgoing IP address by just making some random request to that server. It could then include that Server Cookie in the COOKIE OPT of requests to the server with the forged local IP address of some other host and/or client behind the NAT box. (Attacker possession of this Server Cookie will not help in forging responses to cause cache poisoning as such responses are protected by the required Client Cookie.)

To fix this potential defect, it is necessary to distinguish different clients behind a NAT box from the point of view of the server. It is for this reason that the Server Cookie is specified as a pseudo-random function of both the request source IP address and the Client Cookie. From this inclusion of the Client Cookie in the calculation of the Server Cookie, it follows that a stable Client Cookie, for any particular server, is needed. If, for example, the request ID was included in the calculation of the Client Cookie, it

would normally change with each request to a particular server. This would mean that each request would have to be sent twice: first to learn the new Server Cookie based on this new Client Cookie based on the new ID and then again using this new Client Cookie to actually get an answer. Thus the input to the Client Cookie computation must be limited to the server IP address and one or more things that change slowly such as the client secret.

In principle, there could be a similar problem for servers, not due to NAT but due to mechanisms like anycast which may cause queries to a DNS server at an IP address to be delivered to any one of several machines. (External queries to a DNS server behind a NAT box usually occur via port forwarding such that all such queries go to one host.) However, it is impossible to solve this the way the similar problem was solved for NATed clients; if the Server Cookie was included in the calculation of the Client Cookie the same way the Client Cookie is included in the Server Cookie, you would just get an almost infinite series of errors as a request was repeatedly retried.

For servers accessed via anycast to successfully support DNS COOKIES, the server clones must either all use the same server secret or the mechanism that distributes queries to them must cause the queries

from a particular client to go to a particular server for a sufficiently long period of time that extra queries due to changes in Server Cookie resulting from accessing different server machines are not unduly burdensome. (When such anycast-accessed servers act as recursive servers or otherwise act as clients they normally use a different unique address to source their queries to avoid confusion in the delivery of responses.)

For simplicity, it is RECOMMENDED that the same server secret be used by each DNS server in a set of anycast servers. If there is limited time skew in updating this secret in different anycast servers, this can be handled by a server accepting requests containing a Server Cookie based on either its old or new secret for the maximum likely time period of such time skew (see also [Section 5.4](#)).

## [7](#). Deployment

The DNS cookies mechanism is designed for incremental deployment and to complement the orthogonal techniques in [[RFC5452](#)]. Either or both techniques can be deployed independently at each DNS server and client.

In particular, a DNS server or client that implements the DNS COOKIE mechanism can interoperate successfully with a DNS client or server that does not implement this mechanism although, of course, in this case it will not get the benefit of the mechanism and the server involved might choose to severely rate limit responses. When such a server or client interoperates with a client or server which also implements the DNS cookies mechanism, they get the weak security

benefits of the DNS Cookies mechanism.

## [8.](#) IANA Considerations

IANA is requested to assign the following four code points:

The OPT option value for COOKIE is <TBD> [10 suggested].

Three new DNS error codes in the range above 16 and below 3,840 as shown below:

RCODE	Name	Description	Reference
-----	-----	-----	-----
TBD1[23]	NOCOKIE	No client cookie.	[this document]
TBD2[24]	MFCOKIE	Malformed cookie.	[this document]
TBD3[25]	BADCOKIE	Bad/missing server cookie.	[this document]

## [9.](#) Security Considerations

DNS Cookies provide a weak form of authentication of DNS requests and responses. In particular, they provide no protection against "on-path" adversaries; that is, they provide no protection against any adversary that can observe the plain text DNS traffic, such as an on-path router, bridge, or any device on an on-path shared link (unless the DNS traffic in question on that path is encrypted).

For example, if a host is connected via an unsecured IEEE 802.11 link (Wi-Fi), any device in the vicinity that could receive and decode the 802.11 transmissions must be considered "on-path". On the other hand, in a similar situation but one where 802.11 Robust Security (WPAv2) is appropriately deployed on the Wi-Fi network nodes, only the Access Point via which the host is connecting is "on-path" as far as the 802.11 link is concerned.

Despite these limitations, deployment of DNS Cookies on the global Internet is expected to provide a substantial reduction in the available launch points for the traffic amplification and denial of service forgery attacks described in [Section 2](#) above.

Should stronger message/transaction security be desired, it is suggested that TSIG or SIG(0) security be used (see [Section 3.2](#)); however, it may be useful to use DNS Cookies in conjunction with these features. In particular, DNS Cookies could screen out many DNS messages before the cryptographic computations of TSIG or SIG(0) are required and, if SIG(0) is in use, DNS Cookies could usefully screen out many requests given that SIG(0) does not screen requests but only authenticates the response of complete transactions.

### [9.1](#) Cookie Algorithm Considerations

The cookie computation algorithm for use in DNS Cookies SHOULD be based on a pseudo-random function at least as strong as [\[FNV\]](#) because an excessively weak or trivial algorithm could enable adversaries to guess cookies. However, in light of the weak plain-text token security provided by DNS Cookies, a strong cryptography hash algorithm may not be warranted in many cases, and would cause an increased computational burden. Nevertheless there is nothing wrong with using something stronger, for example, HMAC-SHA256-64 [\[RFC6234\]](#), assuming a DNS processor has adequate computational resources available. DNS processors that feel the need for somewhat stronger security without a significant increase in computational load should consider more frequent changes in their client and/or server secret; however, this does require more frequent generation of a cryptographically strong random number [\[RFC4086\]](#). See Appendices A

and B for specific examples of cookie computation algorithms.

INTERNET-DRAFT

DNS Cookies

#### Acknowledgements

The contributions of the following are gratefully acknowledged:

Tim Wicinski

---

INTERNET-DRAFT

DNS Cookies

## Normative References

- [RFC2119] - Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC4086] - Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", [BCP 106](#), [RFC 4086](#), June 2005.
- [RFC6891] - Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, [RFC 6891](#), April 2013.

## Informative References

- [FNV] - G. Fowler, L. C. Noll, K.-P. Vo, D. Eastlake, "The FNV Non-Cryptographic Hash Algorithm", [draft-eastlake-fnv](#), work in progress.
- [RFC2845] - Vixie, P., Gudmundsson, O., Eastlake 3rd, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", [RFC 2845](#), May 2000.
- [RFC2930] - Eastlake 3rd, D., "Secret Key Establishment for DNS (TKEY RR)", [RFC 2930](#), September 2000.
- [RFC2931] - Eastlake 3rd, D., "DNS Request and Transaction Signatures ( SIG(0)s )", [RFC 2931](#), September 2000.
- [RFC3022] - Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", [RFC 3022](#), January 2001.

- [RFC4033] - Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", [RFC 4033](#), March 2005.
- [RFC4034] - Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", [RFC 4034](#), March 2005.
- [RFC4035] - Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", [RFC 4035](#), March 2005.
- [RFC4966] - Aoun, C. and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status", [RFC 4966](#), July 2007.
- [RFC5452] - Hubert, A. and R. van Mook, "Measures for Making DNS More

- Resilient against Forged Answers", [RFC 5452](#), January 2009.
- [RFC6234] - Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", [RFC 6234](#), May 2011.

## Appendix A: Example Client Cookie Algorithms

### [A.1](#) A Simple Algorithm

An simple example method to compute Client Cookies is the FNV-64 [\[FNV\]](#) of the server IP address and the client secret. That is

$$\text{Client Cookie} = \text{FNV-64} ( \text{Client Secret} \mid \text{Server IP Address} )$$

where " $\mid$ " indicates concatenation.

### [A.2](#) A More Complex Algorithm

A more complex algorithm to calculate Client Cookies is give below. It uses more computational resources than the simpler algorithm shown in A.1.

Client Cookie = HMAC-SHA256-64 ( Client Secret, Server IP Address )

## Appendix B: Example Server Cookie Algorithms

### [B.1](#) A Simple Algorithm

An example simple method producing a 64-bit Server Cookie is the FNV-64 [[FNV](#)] of the request IP address, the Client Cookie, and the server secret. That is

Server Cookie =  
FNV-64 ( Server Secret | Request IP Address | Client Cookie )

where "|" represents concatenation.

## [B.2](#) A More Complex Algorithm

Since the Server Cookie is variable size, the server can store various information in that field as long as it is hard for an adversary to guess the entire quantity used for weak authentication. There should be 64 bits of entropy in the Server Cookie; for example it could have a sub-field of 64-bits computed pseudo-randomly with the server secret as one of the inputs to the pseudo-random function. Types of additional information that could be stored include a time stamp and/or a nonce.

The example below is one variation for the Server Cookie that has been implemented in a beta release of BIND where the Server Cookie is 128 bits composed as follows:

Sub-field	Size
-----	-----
Nonce	32 bits
Time	32 bits
Hash	64 bits

With this algorithm, the server sends a new 128-bit cookie back with every request. The Nonce field assures a low probability that there would be a duplicate.

The Time field gives the server time and makes it easy to reject old cookies.

The Hash part of the Server Cookie is the hard-to-guess part. In the beta release of BIND, its computation can be configured to use AES, HMAC-SHA1, or, as shown below, HMAC-SHA256:

hash =

HMAC-SHA256-64 ( Server Secret,  
(Client Cookie | nonce | time | client IP Address) )

where "|" represents concatenation.

INTERNET-DRAFT

DNS Cookies

Author's Address

Donald E. Eastlake 3rd  
Huawei Technologies  
155 Beaver Street  
Milford, MA 01757 USA

Telephone: +1-508-333-2270  
EMail: d3e3e3@gmail.com

Mark Andrews  
Internet Systems Consortium  
950 Charter Street  
Redwood City, CA 94063 USA

Email: marka@isc.org

Copyright, Disclaimer, and Additional IPR Provisions

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](http://trustee.ietf.org/license-info) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

