Internet Engineering Task Force (IETF)
Request for Comments: 7635
Category: Standards Track
ISSN: 2070-1721

T. Reddy
P. Patil
R. Ravindranath
Cisco
J. Uberti
Google

August 2015

Session Traversal Utilities for NAT (STUN) Extension for Third-Party Authorization

Abstract

This document proposes the use of OAuth 2.0 to obtain and validate ephemeral tokens that can be used for Session Traversal Utilities for NAT (STUN) authentication. The usage of ephemeral tokens ensures that access to a STUN server can be controlled even if the tokens are compromised.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Solution Overview 3
  3.1. Usage with TURN 4
4. Obtaining a Token Using OAuth 7
  4.1. Key Establishment 8
    4.1.1. HTTP Interactions 8
    4.1.2. Manual Provisioning 10
5. Forming a Request 10
6. STUN Attributes 10
  6.1. THIRD-PARTY-AUTHORIZATION 10
  6.2. ACCESS-TOKEN 11
7. STUN Server Behavior 13
8. STUN Client Behavior 14
9. TURN Client and Server Behavior 14
10. Operational Considerations 15
11. Security Considerations 15
12. IANA Considerations 16
  12.1. Well-Known ’stun-key’ URI 16
13. References 16
  13.1. Normative References 16
  13.2. Informative References 17
Appendix A. Sample Tickets 20
Appendix B. Interaction between the Client and Authorization Server 22
Acknowledgements 24
Authors’ Addresses 24

1. Introduction

Session Traversal Utilities for NAT (STUN) [RFC5389] provides a mechanism to control access via ’long-term’ username/password credentials that are provided as part of the STUN protocol. It is expected that these credentials will be kept secret; if the credentials are discovered, the STUN server could be used by unauthorized users or applications. However, in web applications like WebRTC [WEBRTC] where JavaScript uses the browser functionality for making real-time audio and/or video calls, web conferencing, and direct data transfer, ensuring this secrecy is typically not possible.

To address this problem and the ones described in [RFC7376], this document proposes the use of third-party authorization using OAuth 2.0 [RFC6749] for STUN. Using OAuth 2.0, a client obtains an ephemeral token from an authorization server, e.g., a WebRTC server, and the token is presented to the STUN server instead of the traditional mechanism of presenting username/password credentials. The STUN server validates the authenticity of the token and provides required services. Third-party authorization using OAuth 2.0 for STUN explained in this specification can also be used with Traversal Using Relays around NAT (TURN) [RFC5766].

2. Terminology

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].

This document uses the following abbreviations:

- WebRTC Server: A web server that supports WebRTC [WEBRTC].
- Access Token: OAuth 2.0 access token.
- mac_key: The session key generated by the authorization server. This session key has a lifetime that corresponds to the lifetime of the access token, is generated by the authorization server, and is bound to the access token.
- kid: An ephemeral and unique key identifier. The kid also allows the resource server to select the appropriate keying material for decryption.
- AS: Authorization server.
- RS: Resource server.

Some sections in this specification show the WebRTC server as the authorization server and the client as the WebRTC client; however, WebRTC is intended to be used for illustrative purpose only.

3. Solution Overview

The STUN client knows that it can use OAuth 2.0 with the target STUN server either through configuration or when it receives the new STUN attribute THIRD-PARTY-AUTHORIZATION in the error response with an error code of 401 (Unauthorized).

This specification uses the token type ’Assertion’ (a.k.a. self-contained token) described in [RFC6819] where all the information necessary to authenticate the validity of the token is contained within the token itself. This approach has the benefit of avoiding a protocol between the STUN server and the authorization server for token validation, thus reducing latency. The content of the token is opaque to the client. The client embeds the token within a STUN request sent to the STUN server. Once the STUN server has determined the token is valid, its services are offered for a determined period of time. The access token issued by the authorization server is explained in Section 6.2. OAuth 2.0 in [RFC6749] defines four grant types. This specification uses the OAuth 2.0 grant type 'Implicit' as explained in Section 1.3.2 of [RFC6749] where the client is issued an access token directly. The string 'stun' is defined by this specification for use as the OAuth scope parameter (see Section 3.3 of [RFC6749]) for the OAuth token.

The exact mechanism used by a client to obtain a token and other OAuth 2.0 parameters like token type, mac_key, token lifetime, and kid is outside the scope of this document. Appendix B provides an example deployment scenario of interaction between the client and authorization server to obtain a token and other OAuth 2.0 parameters.

Section 3.1 illustrates the use of OAuth 2.0 to achieve third-party authorization for TURN.

3.1. Usage with TURN

TURN, an extension to the STUN protocol, is often used to improve the connectivity of peer-to-peer (P2P) applications. TURN ensures that a connection can be established even when one or both sides are incapable of a direct P2P connection. However, as a relay service, it imposes a non-trivial cost on the service provider. Therefore, access to a TURN service is almost always access controlled. In order to achieve third-party authorization, a resource owner, e.g., a WebRTC server, authorizes a TURN client to access resources on the TURN server.

In this example, a resource owner, i.e., a WebRTC server, authorizes a TURN client to access resources on a TURN server.

OAuth 2.0	WebRTC
Client	WebRTC client
Resource owner	WebRTC server
Authorization server	Authorization server
Resource server	TURN server

Figure 1: OAuth Terminology Mapped to WebRTC Terminology

Using the OAuth 2.0 authorization framework, a WebRTC client (third-party application) obtains limited access to a TURN server (resource server) on behalf of the WebRTC server (resource owner or authorization server). The WebRTC client requests access to resources controlled by the resource owner (WebRTC server) and hosted by the resource server (TURN server). The WebRTC client obtains the access token, lifetime, session key, and kid. The TURN client conveys the access token and other OAuth 2.0 parameters learned from the authorization server to the TURN server. The TURN server obtains the session key from the access token. The TURN server validates the token, computes the message integrity of the request, and takes appropriate action, i.e, permits the TURN client to create allocations. This is shown in an abstract way in Figure 2.

```
                           +---------------+
                           |               +<******+
            +------------->| Authorization |       *
            |              | server        |       *
            |   +----------|(WebRTC server)|       *  AS-RS,
            |   |          |               |       *  AUTH keys
   (1)      |   |           +---------------+      *   (0)
   Access   |   |  (2)                             *
   Token    |   | Access Token                     *
   request  |   |    +                             *
            |   | Session Key                      *
            |   |                                  *
            |   V                                  V
        +-------+---+                       +-+----=-----+
        |           |         (3)           |            |
        |           | TURN request + Access |            |
        | WebRTC    | Token                 | TURN       |
        | client    |---------------------->| server     |
        | (Alice)   | Allocate response (4) |            |
        |           |<----------------------|            |
        +-----------+                       +------------+
```

User: Alice
****: Out-of-Band Long-Term Symmetric Key Establishment

Figure 2: Interactions

In the below figure, the TURN client sends an Allocate request to the TURN server without credentials. Since the TURN server requires that all requests be authenticated using OAuth 2.0, the TURN server rejects the request with a 401 (Unauthorized) error code and the STUN attribute THIRD-PARTY-AUTHORIZATION. The WebRTC client obtains an access token from the WebRTC server, provides the access token to the TURN client, and it tries again, this time including the access token in the Allocate request. This time, the TURN server validates the token, accepts the Allocate request, and returns an Allocate success response containing (among other things) the relayed transport address assigned to the allocation.

```
   +-------------------+                         +--------+  +---------+
   | .........  TURN   |                         |  TURN  |  |  WebRTC |
   | .WebRTC .  client |                         |        |  |         |
   | .client .         |                         | server |  |  server |
   | .........         |                         |        |  |         |
   +-------------------+                         +--------+  +---------+
     |       |           Allocate request                |         |
     |       |------------------------------------------>|         |
     |       |                                           |         |
     |       |         Allocate error response           |         |
     |       |         (401 Unauthorized)                |         |
     |       |<------------------------------------------|         |
     |       |         THIRD-PARTY-AUTHORIZATION         |         |
     |       |                                           |         |
     |       |                                           |         |
     |       |      HTTP request for token               |         |
     |------------------------------------------------------------>|
     |       |      HTTP response with token parameters  |         |
     |<------------------------------------------------------------|
     |OAuth 2.0                                          |         |
      attributes                                         |         |
     |------>|                                           |         |
     |       |    Allocate request ACCESS-TOKEN          |         |
     |       |------------------------------------------>|         |
     |       |                                           |         |
     |       |         Allocate success response         |         |
     |       |<------------------------------------------|         |
     |       |             TURN messages                 |         |
     |       |      ////// integrity protected //////    |         |
     |       |      ////// integrity protected //////    |         |
     |       |      ////// integrity protected //////    |         |
```

Figure 3: TURN Third-Party Authorization

4. Obtaining a Token Using OAuth

A STUN client needs to know the authentication capability of the STUN server before deciding to use third-party authorization. A STUN client initially makes a request without any authorization. If the STUN server supports third-party authorization, it will return an error message indicating that the client can authorize to the STUN server using an OAuth 2.0 access token. The STUN server includes an ERROR-CODE attribute with a value of 401 (Unauthorized), a nonce value in a NONCE attribute, and a SOFTWARE attribute that gives information about the STUN server’s software. The STUN server also includes the additional STUN attribute THIRD-PARTY-AUTHORIZATION, which signals the STUN client that the STUN server supports third-party authorization.

Note: An implementation may choose to contact the authorization server to obtain a token even before it makes a STUN request, if it knows the server details beforehand. For example, once a client has learned that a STUN server supports third-party authorization from a authorization server, the client can obtain the token before making subsequent STUN requests.

4.1. Key Establishment

In this model, the STUN server would not authenticate the client itself but would rather verify whether the client knows the session key associated with a specific access token. An example of this approach can be found with the OAuth 2.0 Proof-of-Possession (PoP) Security Architecture [POP-ARCH]. The authorization server shares a long-term secret (K) with the STUN server. When the client requests an access token, the authorization server creates a fresh and unique session key (mac_key) and places it into the token encrypted with the long-term secret. Symmetric cryptography MUST be chosen to ensure that the size of the encrypted token is not large because usage of asymmetric cryptography will result in large encrypted tokens, which may not fit into a single STUN message.

The STUN server and authorization server can establish a long-term symmetric key (K) and a certain authenticated encryption algorithm, using an out-of-band mechanism. The STUN and authorization servers MUST establish K over an authenticated secure channel. If authenticated encryption with AES-CBC and HMAC-SHA (defined in [ENCRYPT]) is used, then the AS-RS and AUTH keys will be derived from K. The AS-RS key is used for encrypting the self-contained token, and the message integrity of the encrypted token is calculated using the AUTH key. If the Authenticated Encryption with Associated Data (AEAD) algorithm defined in [RFC5116] is used, then there is no need to generate the AUTH key, and the AS-RS key will have the same value as K.

The procedure for establishment of the long-term symmetric key is outside the scope of this specification, and this specification does not mandate support of any given mechanism. Sections 4.1.1 and 4.1.2 show examples of mechanisms that can be used.

4.1.1. HTTP Interactions

The STUN and AS servers could choose to use Representational State Transfer (REST) API over HTTPS to establish a long-term symmetric key. HTTPS MUST be used for data confidentiality, and TLS based on a client certificate MUST be used for mutual authentication. To retrieve a new long-term symmetric key, the STUN server makes an HTTP GET request to the authorization server, specifying STUN as the service to allocate the long-term symmetric keys for and specifying the name of the STUN server. The response is returned with content-type 'application/json' and consists of a JavaScript Object Notation (JSON) [RFC7159] object containing the long-term symmetric key.

Request

service - specifies the desired service (TURN)
name   - STUN server name associated with the key

example:

```
GET https://www.example.com/.well-known/stun-key?service=stun
&name=turn1@example.com
```

Response

k   - long-term symmetric key
exp - identifies the time after which the key expires

example:

```
{
   "k" :
"ESIzRFVmd4iZABEiM0RVZgKn6WjLaTC1FXAghRMVTzkBGNaaN496523WIISKerLi",
   "exp" : 1300819380,
   "kid" :"22BIjxU93h/IgwEb"
   "enc" : A256GCM
  }
```

The authorization server must also signal kid to the STUN server, which will be used to select the appropriate keying material for decryption. The parameter 'k' is defined in Section 6.4.1 of [RFC7518], 'enc' is defined in Section 4.1.2 of [RFC7516], 'kid' is defined in Section 4.1.4 of [RFC7515], and 'exp' is defined in Section 4.1.4 of [RFC7519]. A256GCM and other authenticated encryption algorithms are defined in Section 5.1 of [RFC7518]. A STUN server and authorization server implementation MUST support A256GCM as the authenticated encryption algorithm.

If A256CBC-HS512 as defined in [RFC7518] is used, then the AS-RS and AUTH keys are derived from K using the mechanism explained in Section 5.2.2.1 of [RFC7518]. In this case, the AS-RS key length must be 256 bits and the AUTH key length must be 256 bits (Section 2.6 of [RFC4868]).

4.1.2. Manual Provisioning

The STUN and AS servers could be manually configured with a long-term symmetric key, an authenticated encryption algorithm, and kid.

Note: The mechanism specified in this section requires configuration to change the long-term symmetric key and/or authenticated encryption algorithm. Hence, a STUN server and authorization server implementation SHOULD support REST as explained in Section 4.1.1.

5. Forming a Request

When a STUN server responds that third-party authorization is required, a STUN client re-attempts the request, this time including access token and kid values in the ACCESS-TOKEN and USERNAME STUN attributes. The STUN client includes a MESSAGE-INTEGRITY attribute as the last attribute in the message over the contents of the STUN message. The HMAC for the MESSAGE-INTEGRITY attribute is computed as described in Section 15.4 of [RFC5389] where the mac_key is used as the input key for the HMAC computation. The STUN client and server will use the mac_key to compute the message integrity and do not perform MD5 hash on the credentials.

6. STUN Attributes

The following new STUN attributes are introduced by this specification to accomplish third-party authorization.

6.1. THIRD-PARTY-AUTHORIZATION

This attribute is used by the STUN server to inform the client that it supports third-party authorization. This attribute value contains the STUN server name. The authorization server may have tie ups with multiple STUN servers and vice versa, so the client MUST provide the STUN server name to the authorization server so that it can select the appropriate keying material to generate the self-contained token. If the authorization server does not have tie up with the STUN server, then it returns an error to the client. If the client does not support or is not capable of doing third-party authorization, then it defaults to first-party authentication. The THIRD-PARTY-AUTHORIZATION attribute is a comprehension-optional attribute (see Section 15 from [RFC5389]). If the client is able to comprehend THIRD-PARTY-AUTHORIZATION, it MUST ensure that third-party authorization takes precedence over first-party authentication (as explained in Section 10 of [RFC5389]).

6.2. ACCESS-TOKEN

The access token is issued by the authorization server. OAuth 2.0 does not impose any limitation on the length of the access token but if path MTU is unknown, then STUN messages over IPv4 would need to be less than 548 bytes (Section 7.1 of [RFC5389]). The access token length needs to be restricted to fit within the maximum STUN message size. Note that the self-contained token is opaque to the client, and the client MUST NOT examine the token. The ACCESS-TOKEN attribute is a comprehension-required attribute (see Section 15 from [RFC5389]).

The token is structured as follows:

```
      struct {
          uint16_t nonce_length;
          opaque nonce[nonce_length];
          opaque {
              uint16_t key_length;
              opaque mac_key[key_length];
              uint64_t timestamp;
              uint32_t lifetime;
          } encrypted_block;
      } token;
```

Figure 4: Self-Contained Token Format

Note: uintN_t means an unsigned integer of exactly N bits. Single-byte entities containing uninterpreted data are of type ’opaque’. All values in the token are stored in network byte order.

The fields are described below:

nonce_length: Length of the nonce field. The length of nonce for AEAD algorithms is explained in [RFC5116].

Nonce: Nonce (N) formation is explained in Section 3.2 of [RFC5116].

key_length: Length of the session key in octets. The key length of 160 bits MUST be supported (i.e., only the 160-bit key is used by HMAC-SHA-1 for message integrity of STUN messages). The key length facilitates the hash agility plan discussed in Section 16.3 of [RFC5389].

mac_key: The session key generated by the authorization server.

timestamp: 64-bit unsigned integer field containing a timestamp. The value indicates the time since January 1, 1970, 00:00 UTC, by using a fixed-point format. In this format, the integer number of seconds is contained in the first 48 bits of the field, and the remaining 16 bits indicate the number of 1/64000 fractions of a second (Native format - Unix).

lifetime: The lifetime of the access token, in seconds. For example, the value 3600 indicates one hour. The lifetime value MUST be greater than or equal to the 'expires_in' parameter defined in Section 4.2.2 of [RFC6749], otherwise the resource server could revoke the token, but the client would assume that the token has not expired and would not refresh the token.

encrypted_block: The encrypted_block (P) is encrypted and authenticated using the long-term symmetric key established between the STUN server and the authorization server.

The AEAD encryption operation has four inputs: K, N, A, and P, as defined in Section 2.1 of [RFC5116], and there is a single output of ciphertext C or an indication that the requested encryption operation could not be performed.

The associated data (A) MUST be the STUN server name. This ensures that the client does not use the same token to gain illegal access to other STUN servers provided by the same administrative domain, i.e., when multiple STUN servers in a single administrative domain share the same long-term symmetric key with an authorization server.

If authenticated encryption with AES-CBC and HMAC-SHA (explained in Section 2.1 of [ENCRYPT]) is used, then the encryption process is as illustrated below. The ciphertext consists of the string S, with the string T appended to it. Here, C and A denote ciphertext and the STUN server name, respectively. The octet string AL (Section 2.1 of [ENCRYPT]) is equal to the number of bits in A expressed as a 64-bit unsigned big-endian integer.

- AUTH = initial authentication key length octets of K,

- AS-RS = final encryption key length octets of K,

- S = CBC-PKCS7-ENC(AS-RS, encrypted_block),

  - The Initialization Vector is set to zero because the encrypted_block in each access token will not be identical and hence will not result in generation of identical ciphertext.

- mac = MAC(AUTH, A || S || AL),

- T = initial T_LEN octets of mac,
- C = S || T.

The entire token, i.e., the 'encrypted_block', is base64 encoded (see Section 4 of [RFC4648]), and the resulting access token is signaled to the client.

7. STUN Server Behavior

The STUN server, on receiving a request with the ACCESS-TOKEN attribute, performs checks listed in Section 10.2.2 of [RFC5389] in addition to the following steps to verify that the access token is valid:

- The STUN server selects the keying material based on kid signaled in the USERNAME attribute.

- The AEAD decryption operation has four inputs: K, N, A, and C, as defined in Section 2.2 of [RFC5116]. The AEAD decryption algorithm has only a single output, either a plaintext or a special symbol FAIL that indicates that the inputs are not authentic. If the authenticated decrypt operation returns FAIL, then the STUN server rejects the request with an error response 401 (Unauthorized).

- If AES_CBC_HMAC_SHA2 is used, then the final T_LEN octets are stripped from C. It performs the verification of the token message integrity by calculating HMAC over the STUN server name, the encrypted portion in the self-contained token, and the AL using the AUTH key, and if the resulting value does not match the mac field in the self-contained token, then it rejects the request with an error response 401 (Unauthorized).

- The STUN server obtains the mac_key by retrieving the content of the access token (which requires decryption of the self-contained token using the AS-RS key).

- The STUN server verifies that no replay took place by performing the following check:

  - The access token is accepted if the timestamp field (TS) in the self-contained token is shortly before the reception time of the STUN request (RDnew). The following formula is used:

    lifetime + Delta > abs(RDnew - TS)

The RECOMMENDED value for the allowed Delta is 5 seconds. If the timestamp is NOT within the boundaries, then the STUN server discards the request with error response 401 (Unauthorized).

- The STUN server uses the mac_key to compute the message integrity over the request, and if the resulting value does not match the contents of the MESSAGE-INTEGRITY attribute, then it rejects the request with an error response 401 (Unauthorized).
- If all the checks pass, the STUN server continues to process the request.
- Any response generated by the server MUST include the MESSAGE-INTEGRITY attribute, computed using the mac_key.

If a STUN server receives an ACCESS-TOKEN attribute unexpectedly (because it had not previously sent out a THIRD-PARTY-AUTHORIZATION), it will respond with an error code of 420 (Unknown Attribute) as specified in Section 7.3.1 of [RFC5389].

8. STUN Client Behavior

- The client looks for the MESSAGE-INTEGRITY attribute in the response. If MESSAGE-INTEGRITY is absent or the value computed for message integrity using mac_key does not match the contents of the MESSAGE-INTEGRITY attribute, then the response MUST be discarded.
- If the access token expires, then the client MUST obtain a new token from the authorization server and use it for new STUN requests.

9. TURN Client and Server Behavior

Changes specific to TURN are listed below:

- The access token can be reused for multiple Allocate requests to the same TURN server. The TURN client MUST include the ACCESS-TOKEN attribute only in Allocate and Refresh requests. Since the access token is valid for a specific period of time, the TURN server can cache it so that it can check if the access token in a new allocation request matches one of the cached tokens and avoids the need to decrypt the token.

- The lifetime provided by the TURN server in the Allocate and Refresh responses MUST be less than or equal to the lifetime of the token. It is RECOMMENDED that the TURN server calculate the maximum allowed lifetime value using the formula:

  ```
  lifetime + Delta - abs(RDnew - TS)
  ```

  The RECOMMENDED value for the allowed Delta is 5 seconds.

- If the access token expires, then the client MUST obtain a new token from the authorization server and use it for new allocations. The client MUST use the new token to refresh existing allocations. This way, the client has to maintain only one token per TURN server.

10. Operational Considerations

The following operational considerations should be taken into account:

- Each authorization server should maintain the list of STUN servers for which it will grant tokens and the long-term secret shared with each of those STUN servers.

- If manual configuration (Section 4.1.2) is used to establish long-term symmetric keys, the necessary information, which includes long-term secret (K) and the authenticated encryption algorithm, has to be configured on each authorization server and STUN server for each kid. The client obtains the session key and HMAC algorithm from the authorization server in company with the token.

- When a STUN client sends a request to get access to a particular STUN server (S), the authorization server must ensure that it selects the appropriate kid and access token depending on server S.

11. Security Considerations

When OAuth 2.0 is used, the interaction between the client and the authorization server requires Transport Layer Security (TLS) with a ciphersuite offering confidentiality protection, and the guidance given in [RFC7525] must be followed to avoid attacks on TLS. The session key MUST NOT be transmitted in clear since this would completely destroy the security benefits of the proposed scheme. An attacker trying to replay the message with the ACCESS-TOKEN attribute can be mitigated by frequent changes of the nonce value as discussed in Section 10.2 of [RFC5389]. The client may know some (but not all) of the token fields encrypted with an unknown secret key, and the token can be subjected to known-plaintext attacks, but AES is secure against this attack.

An attacker may remove the THIRD-PARTY-AUTHORIZATION STUN attribute from the error message forcing the client to pick first-party authentication; this attack may be mitigated by opting for TLS [RFC5246] or Datagram Transport Layer Security (DTLS) [RFC6347] as a transport protocol for STUN, as defined in [RFC5389]and [RFC7350].

Threat mitigation discussed in Section 5 of [POP-ARCH] and security considerations in [RFC5389] are to be taken into account.

12. IANA Considerations

This document defines the THIRD-PARTY-AUTHORIZATION STUN attribute, described in Section 6. IANA has allocated the comprehension-optional codepoint 0x802E for this attribute.

This document defines the ACCESS-TOKEN STUN attribute, described in Section 6. IANA has allocated the comprehension-required codepoint 0x001B for this attribute.

12.1. Well-Known 'stun-key' URI

This memo registers the 'stun-key' well-known URI in the Well-Known URIs registry as defined by [RFC5785].

URI suffix: stun-key

Change controller: IETF

Specification document(s): This RFC

Related information: None

13. References

13.1. Normative References

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, DOI 10.17487/RFC4868, May 2007, <http://www.rfc-editor.org/info/rfc4868>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <http://www.rfc-editor.org/info/rfc5116>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <http://www.rfc-editor.org/info/rfc5389>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <http://www.rfc-editor.org/info/rfc7518>.

13.2. Informative References

[ENCRYPT] McGrew, D., Foley, J., and K. Paterson, "Authenticated Encryption with AES-CBC and HMAC-SHA", Work in Progress, draft-mcgrew-aead-aes-cbc-hmac-sha2-05, July 2014.

[POP-ARCH] Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", Work in Progress, draft-ietf-oauth-pop-architecture-02, July 2015.

[POP-KEY-DIST] Bradley, J., Hunt, P., Jones, M., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution", Work in Progress, draft-ietf-oauth-pop-key-distribution-01, March 2015.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, DOI 10.17487/RFC5766, April 2010, <http://www.rfc-editor.org/info/rfc5766>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <http://www.rfc-editor.org/info/rfc5785>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <http://www.rfc-editor.org/info/rfc6819>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, DOI 10.17487/RFC7350, August 2014, <http://www.rfc-editor.org/info/rfc7350>.

[RFC7376] Reddy, T., Ravindranath, R., Perumal, M., and A. Yegin, "Problems with Session Traversal Utilities for NAT (STUN) Long-Term Authentication for Traversal Using Relays around NAT (TURN)", RFC 7376, DOI 10.17487/RFC7376, September 2014, <http://www.rfc-editor.org/info/rfc7376>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <http://www.rfc-editor.org/info/rfc7519>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[STUN] Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal Utilities for NAT (STUN)", Work in Progress, draft-ietf-tram-stunbis-04, March 2015.

[WEBRTC] Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", Work in Progress, draft-ietf-rtcweb-overview-14, June 2015.

Appendix A. Sample Tickets

Input data (same for all samples below):

```
   //STUN SERVER NAME
   server_name = "blackdow.carleon.gov";

   //Shared key between AS and RS

   long_term_key = \x48\x47\x6b\x6a\x33\x32\x4b\x4a\x47\x69\x75\x79
                   \x30\x39\x38\x73\x64\x66\x61\x71\x62\x4e\x6a\x4f
                   \x69\x61\x7a\x37\x31\x39\x32\x33

   //MAC key of the session (included in the token)
   mac_key = \x5a\x6b\x73\x6a\x70\x77\x65\x6f\x69\x78\x58\x6d\x76\x6e
             \x36\x37\x35\x33\x34\x6d;

   //length of the MAC key
   mac_key_length  =  20;

   //The timestamp field in the token
   token_timestamp = 92470300704768;

   //The lifetime of the token
   token_lifetime = 3600;

   //nonce for AEAD
   aead_nonce = \x68\x34\x6a\x33\x6b\x32\x6c\x32\x6e\x34\x62\x35;

   Samples:

   1) token encryption algorithm = AEAD_AES_256_GCM

      Encrypted token (64 bytes = 2 + 12 + 34 + 16) =

    \x00\x0c\x68\x34\x6a\x33\x6b\x32\x6c\x32\x6e\x34\x62
    \x35\x61\x7e\xf1\x34\xa3\xd5\xe4\x4e\x9a\x19\xcc\x7d
    \xc1\x04\xb0\xc0\x3d\x03\xb2\xa5\x51\xd8\xfd\xf5\xcd
    \x3b\x6d\xca\x6f\x10\xcf\xb7\x7e\x5b\x2d\xde\xc8\x4d
    \x29\x3a\x5c\x50\x49\x93\x59\xf0\xc2\xe2\x6f\x76
   2) token encryption algorithm = AEAD_AES_128_GCM

      Encrypted token (64 bytes = 2 + 12 + 34 + 16) =

   \x00\x0c\x68\x34\x6a\x33\x6b\x32\x6c\x32\x6e\x34\x62
   \x35\x7f\xb9\xe9\x9f\x08\x27\xbe\x3d\xf1\xe1\xbd\x65
   \x14\x93\xd3\x03\x1d\x36\xdf\x57\x07\x97\x84\xae\xe5
   \xea\xcb\x65\xfa\xd4\xf2\x7f\xab\x1a\x3f\x97\x97\x4b
   \x69\xf8\x51\xb2\x4b\xf5\xaf\x09\xed\xa3\x57\xe0

Note:
[1] After EVP_EncryptFinal_ex encrypts the final data,
    EVP_CIPHER_CTX_ctrl must be called to append
    the authentication tag to the ciphertext.
    //EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_AEAD_GET_TAG, taglen, tag);

[2] EVP_CIPHER_CTX_ctrl must be invoked to set the
    authentication tag before calling EVP_DecryptFinal.
    //EVP_CIPHER_CTX_ctrl (&ctx, EVP_CTRL_GCM_SET_TAG, taglen, tag);
```

Figure 5: Sample Tickets

Appendix B. Interaction between the Client and Authorization Server

The client makes an HTTP request to an authorization server to obtain a token that can be used to avail itself of STUN services. The STUN token is returned in JSON syntax [RFC7159], along with other OAuth 2.0 parameters like token type, key, token lifetime, and kid as defined in [POP-KEY-DIST].

```
   +-------------------+                         +--------+  +---------+
   | .........  STUN   |                         |  STUN  |  |  WebRTC |
   | .WebRTC .  client |                         |        |  |         |
   | .client .         |                         | server |  |  server |
   | .........         |                         |        |  |         |
   +-------------------+                         +--------+  +---------+
     |       |           STUN request                    |         |
     |       |------------------------------------------>|         |
     |       |                                           |         |
     |       |         STUN error response               |         |
     |       |         (401 Unauthorized)                |         |
     |       |<------------------------------------------|         |
     |       |         THIRD-PARTY-AUTHORIZATION         |         |
     |       |                                           |         |
     |       |                                           |         |
     |       |      HTTP request for token               |         |
     |------------------------------------------------------------>|
     |       |      HTTP response with token parameters  |         |
     |<------------------------------------------------------------|
     |OAuth 2.0                                          |         |
      attributes                                         |         |
     |------>|                                           |         |
     |       |    STUN request with ACCESS-TOKEN         |         |
     |       |------------------------------------------>|         |
     |       |                                           |         |
     |       |         STUN success response             |         |
     |       |<------------------------------------------|         |
     |       |             STUN messages                 |         |
     |       |      ////// integrity protected //////    |         |
     |       |      ////// integrity protected //////    |         |
     |       |      ////// integrity protected //////    |         |
```

Figure 6: STUN Third-Party Authorization

[POP-KEY-DIST] describes the interaction between the client and the authorization server. For example, the client learns the STUN server name "stun1@example.com" from the THIRD-PARTY-AUTHORIZATION attribute value and makes the following HTTP request for the access token using TLS (with extra line breaks for display purposes only):

```
     HTTP/1.1
     Host: server.example.com
     Content-Type: application/x-www-form-urlencoded
     aud=stun1@example.com
     timestamp=1361471629
     grant_type=implicit
     token_type=pop
     alg=HMAC-SHA-256-128
```

Figure 7: Request

[STUN] supports hash agility and accomplishes this agility by computing message integrity using both HMAC-SHA-1 and HMAC-SHA-256-128. The client signals the algorithm supported by it to the authorization server in the 'alg' parameter defined in [POP-KEY-DIST]. The authorization server determines the length of the mac_key based on the HMAC algorithm conveyed by the client. If the client supports both HMAC-SHA-1 and HMAC-SHA-256-128, then it signals HMAC-SHA-256-128 to the authorization server, gets a 256-bit key from the authorization server, and calculates a 160-bit key for HMAC-SHA-1 using SHA1 and taking the 256-bit key as input.

If the client is authorized, then the authorization server issues an access token. An example of a successful response:

```
     HTTP/1.1 200 OK
     Content-Type: application/json
     Cache-Control: no-store

     {
       "access_token":
"U2FsdGVkX18qJK/kkWmRcnfHglrVTJSpS6yU32kmHmOrfGyI3m1gQj1jRPsr0uBb
HctuycAgsfRX7nJW2BdukGyKMXSiNGNnBzigkAofP6+Z3vkJ1Q5pWbfSRroOkWBn",
       "token_type":"pop",
       "expires_in":1800,
       "kid":"22BIjxU93h/IgwEb",
       "key":"v51N62OM65kyMvfTI08O"
       "alg":HMAC-SHA-256-128
     }
```

Figure 8: Response

Acknowledgements

The authors would like to thank Dan Wing, Pal Martinsen, Oleg Moskalenko, Charles Eckel, Spencer Dawkins, Hannes Tschofenig, Yaron Sheffer, Tom Taylor, Christer Holmberg, Pete Resnick, Kathleen Moriarty, Richard Barnes, Stephen Farrell, Alissa Cooper, and Rich Salz for comments and review. The authors would like to give special thanks to Brandon Williams for his help.

Thanks to Oleg Moskalenko for providing token samples in Appendix A.

Authors’ Addresses

Tirumaleswar Reddy
Cisco Systems, Inc.
Cessna Business Park, Varthur Hobli
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India
Email: tireddy@cisco.com

Prashanth Patil
Cisco Systems, Inc.
Bangalore
India
Email: praspati@cisco.com

Ram Mohan Ravindranath
Cisco Systems, Inc.
Cessna Business Park,
Kadabeesanahalli Village, Varthur Hobli,
Sarjapur-Marathahalli Outer Ring Road
Bangalore, Karnataka 560103
India
Email: rmohanr@cisco.com

Justin Uberti
Google
747 6th Ave S.
Kirkland, WA 98033
United States
Email: justin@uberti.name