SIPCORE Working Group
INTERNET-DRAFT
Intended Status: Standards Track
Expires: April 17, 2015

R. Shekh-Yusef, Ed.
Avaya
V. Pascual
Quobis

October 14, 2014

The Session Initiation Protocol (SIP) OAuth
draft-yusef-sipcore-sip-oauth-01

Abstract

This document defines an authorization framework for SIP that is based on the OAuth 2.0 framework, and adds a simple identity layer on top of that, based on the OpenID Connect Core 1.0, to enable Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User.

Status of this Memo

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

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/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Table of Contents

1 Introduction

The SIP protocol [RFC3261] uses the framework used by the HTTP protocol for authenticating users, which is a simple challenge-response authentication mechanism that allows a server to challenge a client request and allows a client to provide authentication information in response to that challenge.

The SIP protocol does not have an authorization framework to allow the system to control access to various services provided by the system.

OAuth 2.0 [RFC6749] defines a token based authorization framework to allow clients to access resources on behalf of their user. It also defines four types of authorization grants, which the client uses to request the access token.

The OpenID Connect 1.0 [OPENID] specifications defines a simple identity layer on top of the OAuth 2.0 protocol, which enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User.

This document defines an authorization framework for SIP that is based on the OAuth 2.0 framework, and adds the identity layer on top of that, based on the OpenID Connect Core 1.0 specification.

1.1 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 RFC 2119 [RFC2119].

1.2 Definitions

Types of SIP services:
- o Basic SIP Services: make/receive call, transfer, call forward, etc.
- o Advanced SIP Services: services provided by SIP application servers, e.g. Voice Mail, Conference Services, Video Services, Presence, IM, ...

Single Sign-On (SSO)
: SSO is a property that allows the user to be authenticated once and as a result have access to multiple services in the system.

Authentication
: The process of verifying the identity of a user trying to get access to some network services.

Authorization
: The process of controlling a user access to network services and the level of service provided to the user.

1.3 Roles

resource owner
: An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

resource server
: The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

OAuth 2.0 client
: An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

SIP client
: An application making requests to access SIP services on behalf of the end-user.

authorization server
: The server issuing access tokens to the OAuth 2.0 client after successfully authenticating the resource owner and obtaining authorization, or the server issuing ID tokens to the SIP client after successfully authenticating the end-user.

proof-of-possession (pop)
A hash used by one party to prove to another party that it is in possession of some shared credentials, without sending the credentials on the wire.

1.4 ID Token

RFC6749 defines two types of tokens: access token and refresh token. This document defines a new token: ID Token as defined in [OPEN-ID].

ID tokens are credentials used by the SIP client to access SIP services on behalf of the end-user.

An ID token is a string representing an authorization issued to the SIP client. The string is usually opaque to the SIP client. Tokens represent specific scopes and durations of access, granted by the SIP system, and enforced by the SIP proxy, SIP application servers, and the authorization server.

2 Benefits

This section describes the benefit of this authorization framework:

2.1 Challenges

With the existing mechanism, the proxy and application servers might need to challenge many of the requests sent by a client, which adds traffic that could be avoided with this authorization mechanism.

2.2 Single Sign-On

Single Sign-On is a property that allows the user to be authenticated once and as a result have access to multiple services in the system.

This authorization mechanism would enable Single Sign-On, as the user will be authenticated once and as a result given a token and a refresh token to allow the user access to various services based on the token scope.

2.3 Level of Service

This authorization mechanism allows the application server to control the level of service provided to the user based on the token scope.

2.4 3rd Party Authorization

This authorization mechanism allows the user to be authenticated and obtain tokens using some 3rd Party Authorization mechanism and still get services from the system.

3 Authorization Code Grant type

3.1 Enterprise SSO Use Case

An enterprise is interested in providing its users with an SSO capability to the corporate various services. The enterprise has an authorization server for controlling the user access to their network and would like to extend that existing authorization server to control the user access to the various services provided by their SIP network.

The user is expected to provide his corporate credentials to login to the corporate network and get different types of services, regardless of the protocol used to provide the service, and without the need to create different accounts for these different types of services.

3.2 Justifications

There are 3 reasons that justify the use of the authorization code grant type:

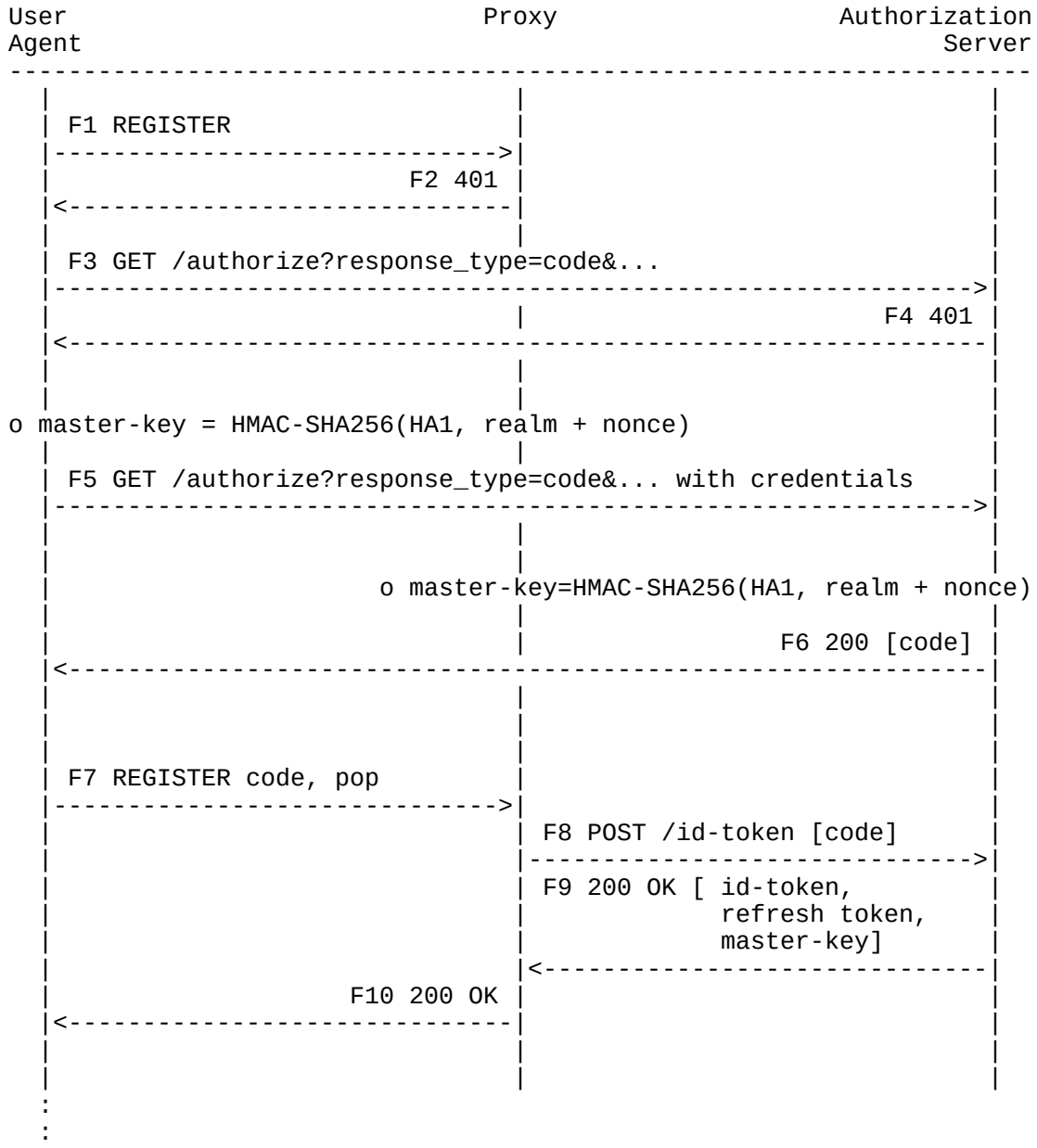
1. Minimize the potential for exposing the token.

2. Enable the proof-of-possession mechanism.

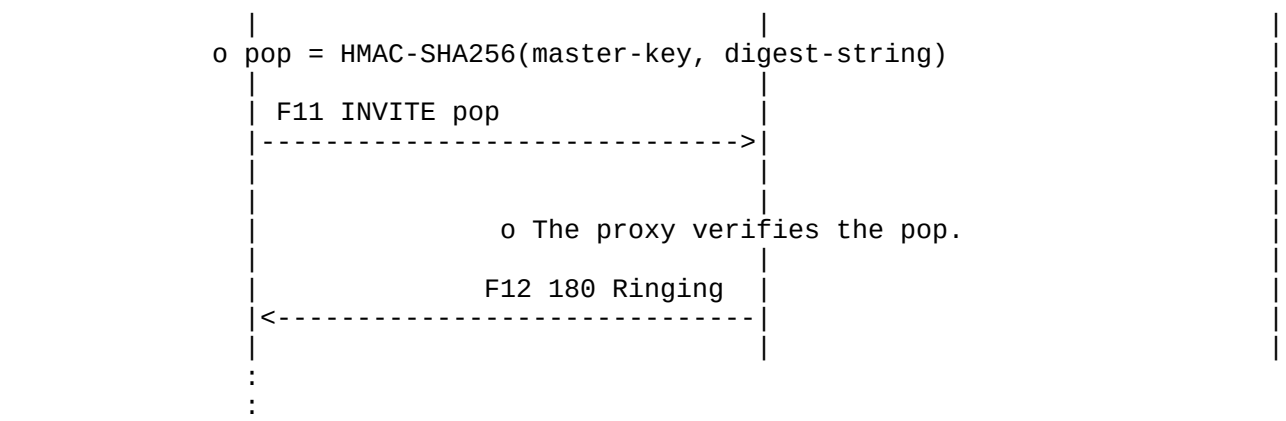
3. Re-use of existing authorization server that already supports this grant type.

3.2 Operations Overview

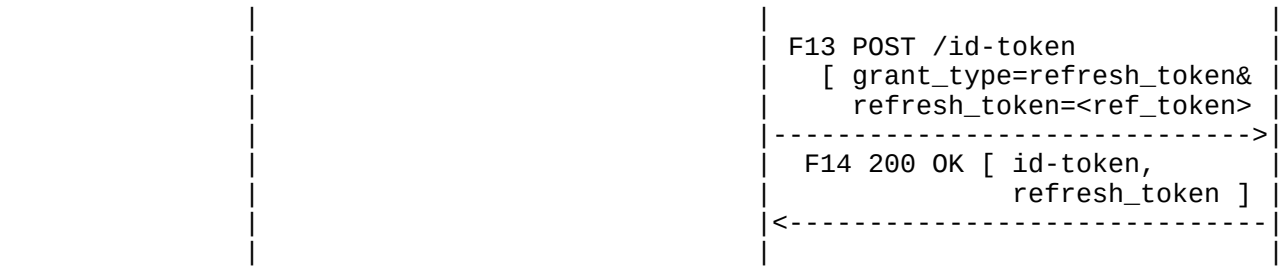
The following figure provides a high level view of flow of messages for the Authorization Code Grant type:



Subsequent Requests



Token Refresh



During registration, if the UA is in possession of a valid ID Token, the UA could use the token to register with the proxy; otherwise, the UA initially sends a REGISTER request (F1) without providing any credentials.

The proxy challenges the UA by responding with 401 (F2) that includes the address of the Authorization Server.

[[OPEN ISSUE]]

How should the UA be redirected to the Authorization Server:
1. New SIP parameter?
2. Extend the Bearer scheme?
3. Define a new Scheme?

The UA will then contact the Authorization Server without providing any credentials in the first request (F3). The Authorization Server challenges the request using the Digest scheme (F4), and the client retries the request (F5) and provide the user's credentials.

The Authorization Server verifies the request from the client; if the verification is successful, the Authorization Server responds with 200 OK (F6) includes a code in the body part.

The UA then retries the request (F7) and include the code in the body of the request. The proxy then contacts the Authorization Server and exchanges the code for a token (F8 & F9).

3.3 Registration

The UA initiates the process by sending a REGISTER request (F1) to the proxy. The proxy will redirect the UA to the Authorization Server by responding with 401 (F2) that include the address of the Authorization Server in the form of an HTTP URI.

The UA will then follow the authorization steps defined in section 3.4. At the end of the authorization process the UA will have a code that it will use to complete the registration process.

The UA will send a new REGISTER request (F7) and include the code in the body of the request with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

The proxy will then use the code to get a token from the Authorization Server as defined in section 3.5. If the proxy is able to obtain the token, the proxy will respond with 200 OK (F10) to the UA to complete the registration process.

3.4 Authorization

The UA constructs the initial request (F3) without providing any user credentials, but with the following URI parameters in the query component:

response_type (REQUIRED)

Value MUST be set to "code".

user_id (REQUIRED)

The user's address-of-record (AOR).

scope (OPTIONAL)

The scope of the access request as described by Section x.x.

state (RECOMMENDED)

The value of this parameter is a nonce created by the client to prevent replay attack. The nonce is a uniquely generated value for each request. This parameter might not be included with the initial request that does not include credentials (F3).

The Authorization Server uses the user's AOR specified in the user_id parameter to verify that the user has an account in the system, and then challenges the request by responding with 401 (F4) with Digest scheme.

The UA will generate a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user's HA1 and the concatenation of realm and nonce received in the challenge from the server.

The UA will then send a new authorization request (F5), but this time include the credentials requested by the server. The UA will use the same parameters values used in the initial authorization request with the exception of the state parameter which will get a new nonce value.

When the server receives the request with the credentials (F5), the server will verify the digest provided by the UA; if that is successful, the server will respond with 302 (F6) and include a code in the body of the response with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

The server then generates a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user's HA1, and the concatenation of realm and nonce sent in the challenge (F4) to the client.

3.5 Acquiring ID Token

The proxy receives the REGISTER request (F7) that includes a body with a code obtained during authorization (section 3.4). The proxy will then contact the Authorization Server to exchange the code with an ID Token.

The proxy sends a POST request (F8) to the Authorization Server and include the following parameters in the body:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

If the request is valid and authorized, the authorization server responds with a 200 OK (F9) to complete the registration process, with id_token, token_refresh, and the master-key in the body.

3.6 Token Refresh

The proxy makes a refresh request to the token by sending a refresh POST request (F13) that includes a body with the grant_type and the refresh_token.

For example:

```
grant_type=refresh_token&refresh_token=<refresh_token>
```

If the proxy fails to refresh the token, then it MUST challenge the next request from the UA, and as a result the UA MUST go through the authorization process defined in section 3.4 to obtain new tokens.

3.7 Authenticated Requests

When the UA wants to send any request to the proxy, it MUST include the Authorization header and use the Bearer scheme to carry the proof-of-possession of the master-key.

The pop is calculated using the master-key as follows:

```
pop = HMAC-SHA256(master-key, digest-string)
```

The following is an example of an Authorization header with Bearer scheme:

```
Authorization: Bearer pop=<pop>
```

See rfc4474, section 9, for the SIP headers to hash to create digest-string.

[[OPEN ISSUE]] The Bearer scheme is used to deliver tokens without providing any proof of possession. We probably need to use different scheme later on.

3.8 Services

When the UA tries to access a service on behalf of a user, e.g. Voice Mail Service, the proxy forwards the request to the server providing the service and MUST include an Authorization header with the Bearer scheme that carries the token needed to get service, as follows:

```
Authorization: Bearer token=<token>
```

4 Resource Owner Password Credentials Grant type

4.1 SIP SSO

An enterprise is interested in providing its users with an SSO capability to the corporate various SIP services.

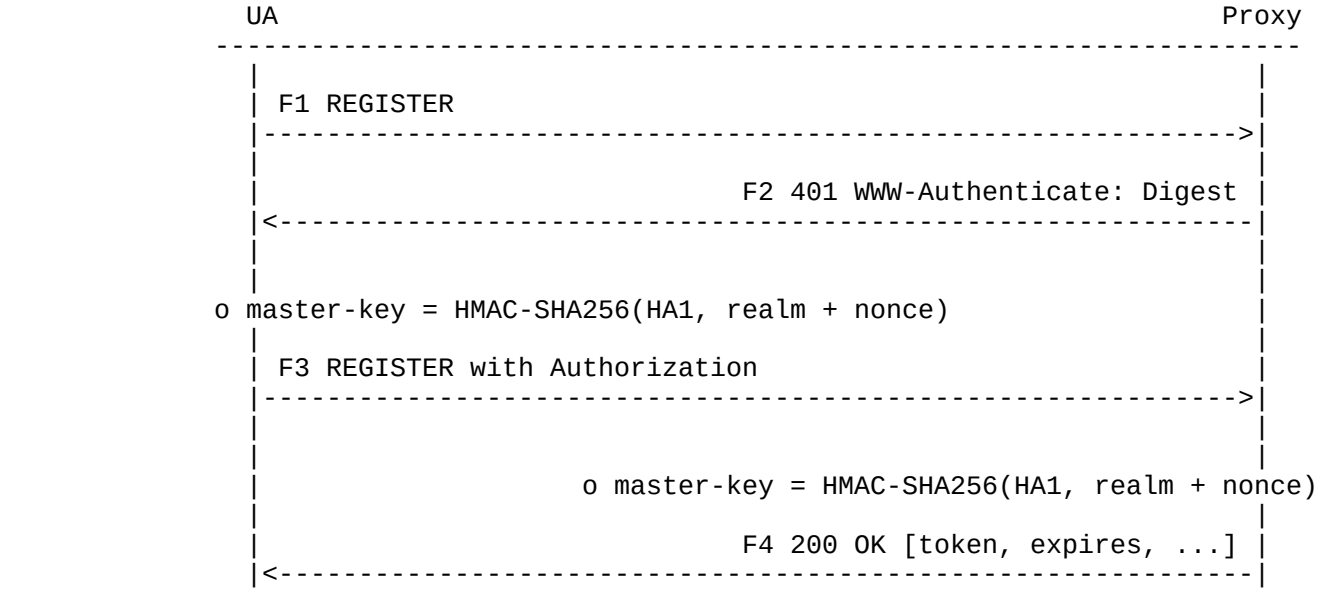
The enterprise wants to control the services provided to their SIP users and the level of service provided to the user by their SIP application servers without the need to create different accounts for these services.

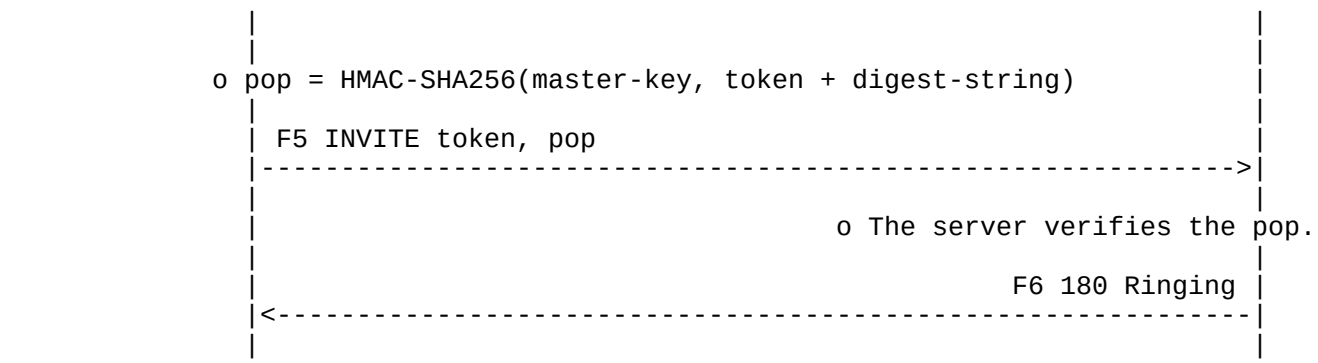
The enterprise wants to utilize an existing authentication mechanism provided by SIP, but would like to be able to control who gets access to what service and when.

The user is expected to use his SIP credentials to login to the SIP network and get access to the basic services, and to get access to the services provided by the various SIP application servers without being challenged to provide credentials for each type of service.

4.2 Operations Overview

The following figure provides a high level view of flow of messages for the Resource Owner Password Credentials Grant type:





During registration the UA initially sends a REGISTER request (F1) without providing any credentials.

The proxy then challenges the UA by responding with 401 (F2) that includes the Digest scheme in the www-authenticate header.

The UA will generate a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user’s HA1 and the concatenation of realm and nonce received in the challenge from the server. The UA will continue to use the existing operation of handling the Digest challenge and then sends a new REGISTER request (F3) with the credentials to the server.

When the server receives the request with the credentials (F3), the server will verify the digest provided by the UA; if that is successful, the server will accept the registration (F4) and include the details of the token in the response.

The server then generates a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user’s HA1, and the concatenation of realm and nonce sent in the challenge to the client.

At the end of the above process the UA would have registered with the proxy and both the UA and the proxy would have created the same master-key without sending the master-key on the wire.

Later when the UA wants to send a request to the proxy it MUST always include the token and SHOULD include the pop as defined in section 4.6.

4.3 Registration and Acquiring Tokens

The UA MUST request the access token during the registration process with the proxy, by including a body with the grant_type as "password". Initially, the UA sends a REGISTER request without providing any credentials.

The proxy MUST then challenge the UA by responding with 401 with the Digest scheme in the WWW-Authenticate header.

When the UA gets challenged by the proxy to provide its credentials, the UA MUST include its credentials in the new REGISTER request in the authorization header as it is done with the existing mechanism, and MUST include a body with the grant_type as "password".

In addition, the UA MUST generate a master-key as follows:

```
master-key = HMAC-SHA256(HA1, realm + nonce)
```

- o HA1 - this is the user's H(A1) as defined in [DIGEST].
- o realm - this is the realm that is returned by the server in the response to the initial request from the UA.
- o nonce - this is the nonce that is returned by the server in the response to the initial request from the UA.

When the server receives the request with the credentials, the server will verify the digest provided by the UA; if that is successful, the server will accept the registration and include the details of the token in the response.

[[OPEN ISSUE]]

How should the tokens be transported to the UA? in the body of the 200 OK? or a SIP header?

The server then generates a master-key following the same procedure followed by the client.

As a result of this procedure both the UA and the server would have created the same master-key without sending the master-key on the wire.

4.4 Discarding Credentials

After successfully receiving the access and refresh tokens from the proxy, the UA SHOULD discard the user credentials.

4.5 Token Refresh

The UA makes a refresh request to the token by sending a refresh REGISTER request that includes the authorization header and a body with the grant_type, the refresh_token, and the proof-of-possession of the master-key.

For example:

grant_type=refresh_token&refresh_token=<refresh_token>&pop=<pop>

4.6 Authenticated Requests

When the UA wants to send any request to the proxy, it MUST include the Authorization header and use the Bearer scheme to carry the access token, and the proof-of-possession of the master-key. For example:

Authorization: Bearer token=<token>, pop=<pop>

See rfc4474, section 9, for the SIP headers to hash to create the value for the proof.

[[OPEN ISSUE]]

The Bearer scheme is used to deliver tokens without providing any proof of possession. We probably need to use different scheme later on.

4.7 Examples

```
REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/TCP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 19

grant_type=password&pop=<pop>
```

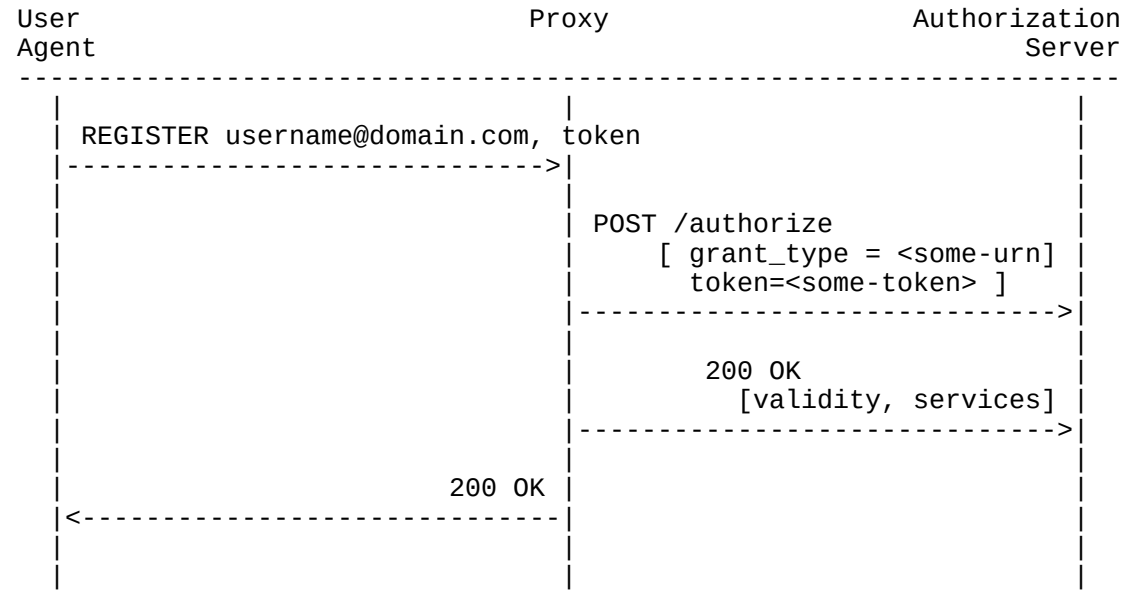
```
SIP/2.0 200 OK
Via: SIP/2.0/TCP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
     ;received=192.0.2.4
To: Bob <sip:bob@biloxi.com>;tag=2493k59kd
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0

{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"example",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
   "example_parameter":"example_value"
}
```

5 Client Credentials Grant

The following flow assumes that the UA is able to get a token using some out-of-band mechanism, and the UA wants to use the token to register, subscribe, and get service.

The flow uses a combination of the following from RFC6749:

- Client Credentials Grant defined in section 4.4
- Extensions Grants defined in section 4.5.



5.1 Registration

The UA is in possession of a token that was obtained through some out-of-band mechanism.

The UA sends a REGISTER request and include the token in the Authorization header using the Bearer scheme as defined in RFC6750.

If the proxy is able to verify the token, the proxy accepts the registration request and responds with 200 OK.

5.2 Authorization

When the proxy receives the REGISTER request with the token, the proxy will try to first validate the token before responding to the UA request.

The proxy sends a POST request and include the following parameters in the body of the request:

grant_type (REQUIRED)

Some well defined URN.

username (REQUIRED)

The resource owner username.

access_token (REQUIRED)

The token received from the UA.

scope (OPTIONAL)

The scope of the token.

If the authorization server is able to validate and authorize the request, it will respond with 200 OK with a body that contains the following parameters:

access_token, token_type, expires, refresh_token, scope

6 Outbound

RFC5626 defines a mechanism that allows a UA to simultaneously connect and establish registration with multiple outbound proxies to get service.

This section describes that impact of outbound on this authorization mechanism.

6.1 Authorization Code Grant type

During initial registration with the primary proxy, the UA is able to get an authorization code that it will use to register with the primary proxy. Assuming the authorization server is shared between the various outbound proxies, the UA will be able to use the same authorization code to register with the secondary proxies and as a result each one of the secondary proxies will get the master-key associated with the user to be used for the calculation of the proof-of-possession.

6.2 Resource Owner Password Credentials Grant type

During registration the proxy challenges the UA, and both the proxy and the UA create a master-key based on HA1, realm, and nonce. Since the nonce is not shared between the various proxies, it is not possible for the outbound proxies to use the same master-key; as a result, the UA is expected to maintain a master-key and token per outbound proxy.

6.3 Client Credentials Grant type

Since the tokens are obtained using some out-of-band mechanism, and the authorization server is shared between the outbound proxies, the UA should be able to register and get service from any one of the outbound proxies.

7 Security Considerations

<Security considerations text>

8 IANA Considerations

<IANA considerations text>

9 Acknowledgments

<Acknowledgments text>

10 References

10.1 Normative References

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[OPENID] Sakimura, N., J. Bradley, J., Jones, M., de Medeiros, B.,Mortimore, C., "OpenID Connect Core 1.0", February, 2014 http://openid.net/specs/openid-connect-core-1_0.html

[DIGEST] Shekh-Yusef, R., Ahrens, D., and Bremer, S., "HTTP Digest Access Authentication", Work in Progress, January 2014.

https://datatracker.ietf.org/doc/draft-ietf-httpauth-digest/

10.2 Informative References

Authors' Addresses

Rifaat Shekh-Yusef (Editor)
Avaya
250 Sydney Street
Belleville, Ontario
Canada

Phone: +1-613-967-5267
Email: rifaat.ietf@gmail.com

Victor Pascual
Quobis
Spain

Email: victor.pascual@quobis.com