OAuth 2.0 Proof-of-Possession (PoP) Security Architecture
draft-hunt-oauth-pop-architecture-01.txt

Abstract

   The OAuth 2.0 bearer token specification, as defined in RFC 6750,
   allows any party in possession of a bearer token (a "bearer") to get
   access to the associated resources (without demonstrating possession
   of a cryptographic key). To prevent misuse, bearer tokens must to be
   protected from disclosure in transit and at rest.

   Some scenarios demand additional security protection whereby a client
   needs to demonstrate possession of cryptographic keying material when
   accessing a protected resource. This document motivates the
   development of the OAuth 2.0 proof-of-possession security mechanism.

Status of This Memo

Table of Contents

1. Introduction

   At the time of writing the OAuth 2.0 protocol family ([RFC6749],
   [RFC6750], and [RFC6819]) offer a single standardized security
   mechanism to access protected resources, namely the bearer token.
   RFC 6750 [RFC6750] specifies the bearer token mechanism and defines
   it as follows:

"A security token with the property that any party in possession
of the token (a "bearer") can use the token in any way that any
other party in possession of it can. Using a bearer token does
not require a bearer to prove possession of cryptographic key
material."

The bearer token meets the security needs of a number of use cases
the OAuth 2.0 protocol had originally been designed for. There are,
however, other scenarios that require stronger security properties
and ask for active participation of the OAuth client in form of
cryptographic computations when presenting an access token to a
resource server.

This document outlines additional use cases requiring stronger
security protection in Section 3, identifies threats in Section 4,
proposes different ways to mitigate those threats in Section 5,
outlines an architecture for a solution that builds on top of the
existing OAuth 2.0 framework in Section 6, and concludes with a
requirements list in Section 7.

2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT',
'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this
specification are to be interpreted as described in [RFC2119], with
the important qualification that, unless otherwise stated, these
terms apply to the design of the protocol, not its implementation or
application.

3. Use Cases

The main use case that motivates better-than-bearer token security is
the desire of resource servers to obtain additional assurance that
the client is indeed authorized to present this access token. The
expectation is that the use of additional credentials (symmetric or
asymmetric keying material) will encourage developers to take
additional precautions when transferring or storing the access token
in combination with these credentials.

Additional use cases listed below provide further requirements for
the solution development.

3.1. Access to an 'Unprotected' Resource

This use case is for a web client that needs to access a resource
that makes data available (such as videos) without offering integrity
and confidentiality protection using TLS. Still, the initial
resource request using OAuth, which includes the access token, must

be protected against various threats (e.g., token replay, token
modification).

While it is possible to utilize bearer tokens in this scenario with
TLS protection when the request to the protected resource is made, as
described in [RFC6750], there may be the desire to avoid using TLS
between the client and the resource server at all. In such a case
the bearer token approach is not possible since it relies on TLS for
ensuring integrity and confidentiality protection of the access token
exchange since otherwise replay attacks are possible: First, an
eavesdropper may steal an access token and represent it at a
different resource server. Second, an eavesdropper may steal an
access token and replay it against the same resource server at a
later point in time. In both cases, if the attack is successful, the
adversary gets access to the resource owners data or may perform an
operation selected by the adversary (e.g., sending a message). Note
that the adversary may obtain the access token (if the
recommendations in [RFC6749] and [RFC6750] are not followed) using a
number of ways, including eavesdropping the communication on the
wireless link.

Consequently, the important assumption in this use case is that a
resource server does not have TLS support and the security solution
should work in such a scenario. Furthermore, it may not be necessary
to provide authentication of the resource server towards the client.

3.2. Offering Application Layer End-to-End Security

In Web deployments resource servers are often placed behind load
balancers, which are deployed by the same organization that operates
the resource servers. These load balancers may terminate the TLS
connection setup and HTTP traffic is transmitted in the clear from
the load balancer to the resource server. With application layer
security independent of the underlying TLS security it is possible to
allow application servers to perform cryptographic verification on an
end-to-end basis.

The key aspect in this use case is therefore to offer end-to-end
security in the presence of load balancers via application layer
security.

3.3. Preventing Access Token Re-Use by the Resource Server

Imagine a scenario where a resource server that receives a valid
access token re-uses it with other resource server. The reason for
re-use may be malicious or may well be legitimate. In a legitimate
use case consider chaining of computations whereby a resource server
needs to consult other third party resource servers to complete the

requested operation. In both cases it may be assumed that the scope
of the access token is sufficiently large that it allows such a re-
use. For example, imagine a case where a company operates email
services as well as picture sharing services and that company had
decided to issue access tokens with a scope that allows access to
both services.

With this use case the desire is to prevent such access token re-use.
This also implies that the legitimate use cases require additional
enhancements for request chaining.

3.4. TLS Channel Binding Support

In this use case we consider the scenario where an OAuth 2.0 request
to a protected resource is secured using TLS but the client and the
resource server demand that the underlying TLS exchange is bound to
additional application layer security to prevent cases where the TLS
connection is terminated at a TLS intermediary, which splits the TLS
connection into two separate connections.

In this use case additional information is conveyed to the resource
server to ensure that no entity entity has tampered with the TLS
connection.

4. Security and Privacy Threats

The following list presents several common threats against protocols
utilizing some form of tokens. This list of threats is based on NIST
Special Publication 800-63 [NIST800-63]. We exclude a discussion of
threats related to any form of identity proofing and authentication
of the resource owner to the authorization server since these
procedures are not part of the OAuth 2.0 protocol specification
itself.

Token manufacture/modification:

   An attacker may generate a bogus tokens or modify the token
   content (such as authentication or attribute statements) of an
   existing token, causing resource server to grant inappropriate
   access to the client. For example, an attacker may modify the
   token to extend the validity period. A client may modify the
   token to have access to information that they should not be able
   to view.

Token disclosure: Tokens may contain personal data, such as real
   name, age or birthday, payment information, etc.

Token redirect:

An attacker uses the token generated for consumption by the
resource server to obtain access to another resource server.

Token reuse:

An attacker attempts to use a token that has already been used
once with a resource server. The attacker may be an eavesdropper
who observes the communication exchange or, worse, one of the
communication end points. A client may, for example, leak access
tokens because it cannot keep secrets confidential. A client may
also re-use access tokens for some other resource servers.
Finally, a resource server may use a token it had obtained from a
client and use it with another resource server that the client
interacts with. A resource server, offering relatively
unimportant application services, may attempt to use an access
token obtained from a client to access a high-value service, such
as a payment service, on behalf of the client using the same
access token.

We excluded one threat from the list, namely 'token repudiation'.
Token repudiation refers to a property whereby a resource server is
given an assurance that the authorization server cannot deny to have
created a token for the Client. We believe that such a property is
interesting but most deployments prefer to deal with the violation of
this security property through business actions rather than by using
cryptography.

5. Threat Mitigation

A large range of threats can be mitigated by protecting the content
of the token, for example using a digital signature or a keyed
message digest. Alternatively, the content of the token could be
passed by reference rather than by value (requiring a separate
message exchange to resolve the reference to the token content). To
simplify the subsequent description we assume that the token itself
is digitally signed by the authorization server and therefore cannot
be modified.

To deal with token redirect it is important for the authorization
server to include the identifier of the intended recipient - the
resource server. A resource server must not be allowed to accept
access tokens that are not meant for its consumption.

To provide protection against token disclosure two approaches are
possible, namely (a) not to include sensitive information inside the
token or (b) to ensure confidentiality protection. The latter
approach requires at least the communication interaction between the
client and the authorization server as well as the interaction

between the client and the resource server to experience
confidentiality protection. As an example, TLS with a ciphersuite
that offers confidentiality protection has to be applied (which is
currently true for all ciphersuites, except for one). Encrypting the
token content itself is another alternative. In our scenario the
authorization server would, for example, encrypt the token content
with a symmetric key shared with the resource server.

To deal with token reuse more choices are available.

5.1. Confidentiality Protection

In this approach confidentiality protection of the exchange is
provided on the communication interfaces between the client and the
resource server, and between the client and the authorization server.
No eavesdropper on the wire is able to observe the token exchange.
Consequently, a replay by a third party is not possible. An
authorization server wants to ensure that it only hands out tokens to
clients it has authenticated first and who are authorized. For this
purpose, authentication of the client to the authorization server
will be a requirement to ensure adequate protection against a range
of attacks. This is, however, true for the description in
Section 5.2 and Section 5.3 as well. Furthermore, the client has to
make sure it does not distribute (or leak) the access token to
entities other than the intended the resource server. For that
purpose the client will have to authenticate the resource server
before transmitting the access token.

5.2. Sender Constraint

Instead of providing confidentiality protection the authorization
server could also put the identifier of the client into the protected
token with the following semantic: 'This token is only valid when
presented by a client with the following identifier.' When the
access token is then presented to the resource server how does it
know that it was provided by the client? It has to authenticate the
client! There are many choices for authenticating the client to the
resource server, for example by using client certificates in TLS
[RFC5246], or pre-shared secrets within TLS [RFC4279]. The choice of
the preferred authentication mechanism and credential type may depend
on a number of factors, including

o security properties

o available infrastructure

o library support

o credential cost (financial)

o performance

o integration into the existing IT infrastructure

o operational overhead for configuration and distribution of
   credentials

This long list hints to the challenge of selecting at least one
mandatory-to-implement client authentication mechanism.

5.3. Key Confirmation

A variation of the mechanism of sender authentication, described in
Section 5.2, is to replace authentication with the proof-of-
possession of a specific (session) key, i.e., key confirmation. In
this model the resource server would not authenticate the client
itself but would rather verify whether the client knows the session
key associated with a specific access token. Examples of this
approach can be found with the OAuth 1.0 MAC token [RFC5849], and
Kerberos [RFC4120] when utilizing the AP_REQ/AP_REP exchange (see
also [I-D.hardjono-oauth-kerberos] for a comparison between Kerberos
and OAuth).

To illustrate key confirmation the first examples borrow from
Kerberos and use symmetric key cryptography. Assume that the
authorization server shares a long-term secret with the resource
server, called K(Authorization Server-Resource Server). This secret
would be established between them out-of-band. When the client
requests an access token the authorization server creates a fresh and
unique session key Ks and places it into the token encrypted with the
long term key K(Authorization Server-Resource Server). Additionally,
the authorization server attaches Ks to the response message to the
client (in addition to the access token itself) over a
confidentiality protected channel. When the client sends a request
to the resource server it has to use Ks to compute a keyed message
digest for the request (in whatever form or whatever layer). The
resource server, when receiving the message, retrieves the access
token, verifies it and extracts K(Authorization Server-Resource
Server) to obtain Ks. This key Ks is then used to verify the keyed
message digest of the request message.

Note that in this example one could imagine that the mechanism to
protect the token itself is based on a symmetric key based mechanism
to avoid any form of public key infrastructure but this aspect is not
further elaborated in the scenario.

A similar mechanism can also be designed using asymmetric cryptography. When the client requests an access token the authorization server creates an ephemeral public / privacy key pair (PK/SK) and places the public key PK into the protected token. When the authorization server returns the access token to the client it also provides the PK/SK key pair over a confidentiality protected channel. When the client sends a request to the resource server it has to use the privacy key SK to sign the request. The Resource Server, when receiving the message, retrieves the access token, verifies it and extracts the public key PK. It uses this ephemeral public key to verify the attached signature.

5.4. Summary

As a high level message, there are various ways how the threats can be mitigated and while the details of each solution is somewhat different they all ultimately accomplish the goal.

The three approaches are:

Confidentiality Protection:

   The weak point with this approach, which is briefly described in Section 5.1, is that the Client has to be careful to whom it discloses the access token. What can be done with the token entirely depends on what rights the token entitles the presenter and what constraints it contains. A token could encode the identifier of the Client but there are scenarios where the Client is not authenticated to the Resource Server or where the identifier of the Client rather represents an application class rather than a single application instance. As such, it is possible that certain deployments choose a rather liberal approach to security and that everyone who is in possession of the access token is granted access to the data.

Sender Constraint:

   The weak point with this approach, which is briefly described in Section 5.2, is to setup the authentication infrastructure such that Clients can be authenticated towards Resource Servers. Additionally, Authorization Server must encode the identifier of the Client in the token for later verification by the Resource Server. Depending on the chosen layer for providing Client-side authentication there may be additional challenges due Web server load balancing, lack of API access to identity information, etc.

Key Confirmation:

The weak point with this approach, see Section 5.3, is the
increased complexity: a complete key distribution protocol has to
be defined.

In all cases above it has to be ensured that the Client is able to
keep the credentials secret.

6. Architecture

The proof-of-possession security concept assumes that the
authorization server acts as a trusted third party that binds keys to
access tokens. These keys are then used by the client to demonstrate
the possession of the secret to the resource server when accessing
the resource. The resource server, when receiving an access token,
needs to verify that the key used by the client matches the one
included in the access token.

There are slight differences between the use of symmetric keys and
asymmetric keys when they are bound to the access token and the
subsequent interaction between the client and the authorization
server when demonstrating possession of these keys. Figure 1 shows
the symmetric key procedure and Figure 2 illustrates how asymmetric
keys are used.

With the JSON Web Token (JWT) a standardized format for access tokens
is available. The necessary elements to bind symmetric or asymmetric
keys to the JWT is described in
[I-D.jones-oauth-proof-of-possession].

```
                      +---------------+
                    ^|               |
                   // | Authorization |
                  /   | Server        |
                //    |               |
               /      |               |
        (I)  //      /+---------------+
    Access    /     //
    Token    /     /
    Request //    //  (II) Access Token
    +Params /     /        +Symmetric Key
        //    //
       /      v
     +-----------+                    +------------+
     |           |                    |            |
     |           |                    | Resource   |
     | Client    |                    | Server     |
     |           |                    |            |
     |           |                    |            |
     +-----------+                    +------------+
```
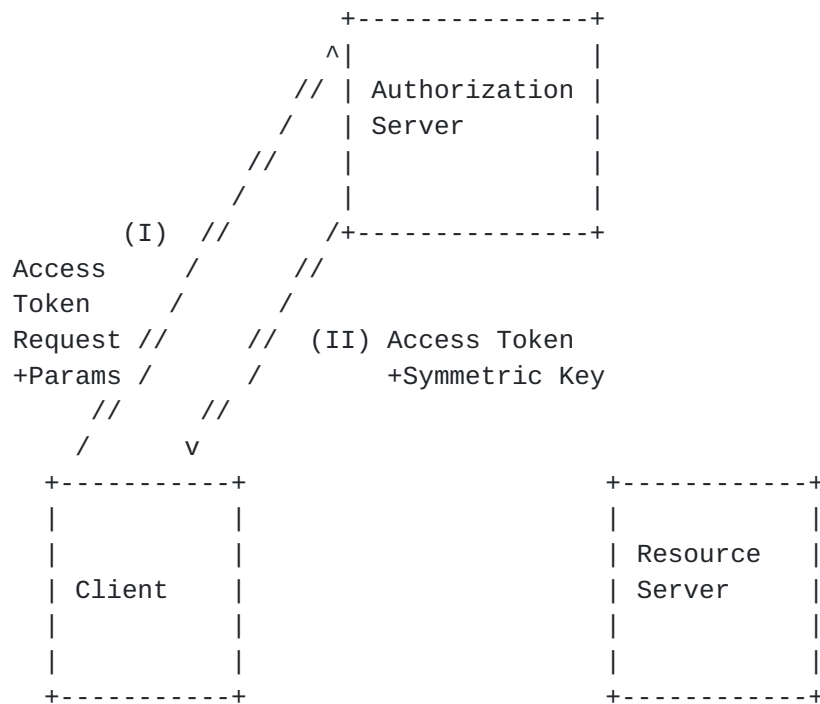
Figure 1: Interaction between the Client and the Authorization Server
(Symmetric Keys).

In order to request an access token the client interacts with the
authorization server as part of the a normal grant exchange, as shown
in Figure 1. However, it needs to include additional information
elements for use with the PoP security mechanism, as depicted in
message (I). In message (II) the authorization server then returns
the requested access token. In addition to the access token itself,
the symmetric key is communicated to the client. This symmetric key
is a unique and fresh session key with sufficient entropy for the
given lifetime. Furthermore, information within the access token
ties it to this specific symmetric key.

Note: For this security mechanism to work the client as well as the
resource server need to have access to the session key. While the
key transport mechanism from the authorization server to the client
has been explained in the previous paragraph there are three ways for
communicating this session key from the authorization server to the
resource server, namely

   Embedding the symmetric key inside the access token itself. This
   requires that the symmetric key is confidentiality protected.

The resource server queries the authorization server for the
symmetric key. This is an approach envisioned by the token
introspection endpoint [I-D.richer-oauth-introspection].

The authorization server and the resource server both have access
to the same back-end database. This is case in smaller, tightly
coupled systems.

```
                      +---------------+
                 ^|                |
Access Token Req. // | Authorization |
+Parameters      /   | Server        |
+[Fingerprint] //    |               |
              /      |               |
    (I)     //      /+---------------+
        /       //
       /       /      (II)
      //     //  Access Token
     /      /    +[ephemeral
    //     //       asymmetric key pair]
   /      v
  +-----------+                    +------------+
  |           |                    |            |
  |           |                    | Resource   |
  | Client    |                    | Server     |
  |           |                    |            |
  |           |                    |            |
  +-----------+                    +------------+
```
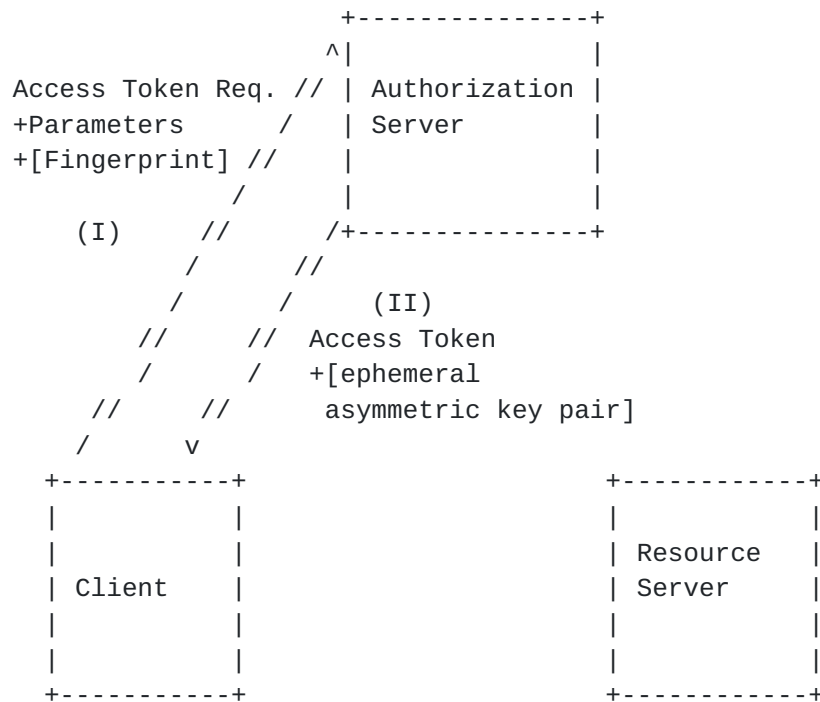
Figure 2: Interaction between the Client and the Authorization Server
(Asymmetric Keys).

The use of asymmetric keys is slightly different since the client or
the server could be involved in the generation of the ephemeral key
pair. This exchange is shown in Figure 1. If the client generates
the key pair it includes a fingerprint of the public key (of the
SubjectPublicKeyInfo structure, more precisely). The authorization
server would include this fingerprint in the access token and thereby
bind the asymmetric key pair to the token. If the client did not
provide a fingerprint in the request then the authorization server is
asked to create an ephemeral asymmetric key pair, binds the
fingerprint of the public key to the access token, and returns the
asymmetric key pair to the client.

The specification describing the interaction between the client and
the authorization server, as shown in Figure 1 and in Figure 2, can
be found in [I-D.bradley-oauth-pop-key-distribution].

Once the client has obtained the necessary access token and keying
material it can start to interact with the resource server. To
demonstrate possession of the key bound to the access token it needs
to apply this key to the HTTP request by computing a keyed message
digest (i.e., a symmetric key-based cryptographic primitive) or a
digital signature (i.e., an asymmetric cryptographic primitive).
When the resource server receives the request it verifies it and
decides whether access to the protected resource can be granted.
This exchange is shown in Figure 3.

```
                    +---------------+
                    |               |
                    | Authorization |
                    | Server        |
                    |               |
                    |               |
                    +---------------+




            HTTP Request
+-----------+  + Signature/MAC (a)  +------------+
|           |---------------------->|            |
|           | [+Access Token]       | Resource   |
| Client    |                       | Server     |
|           | HTTP Response (b)      |            |
|           |<----------------------|            |
+-----------+                       +------------+


     ^                                   ^
     |                                   |
     |                                   |
  Symmetric Key                     Symmetric Key
     or                                  or
  Asymmetric Key Pair               Public Key (Client)
     +                                   +
   Parameters                         Parameters
```
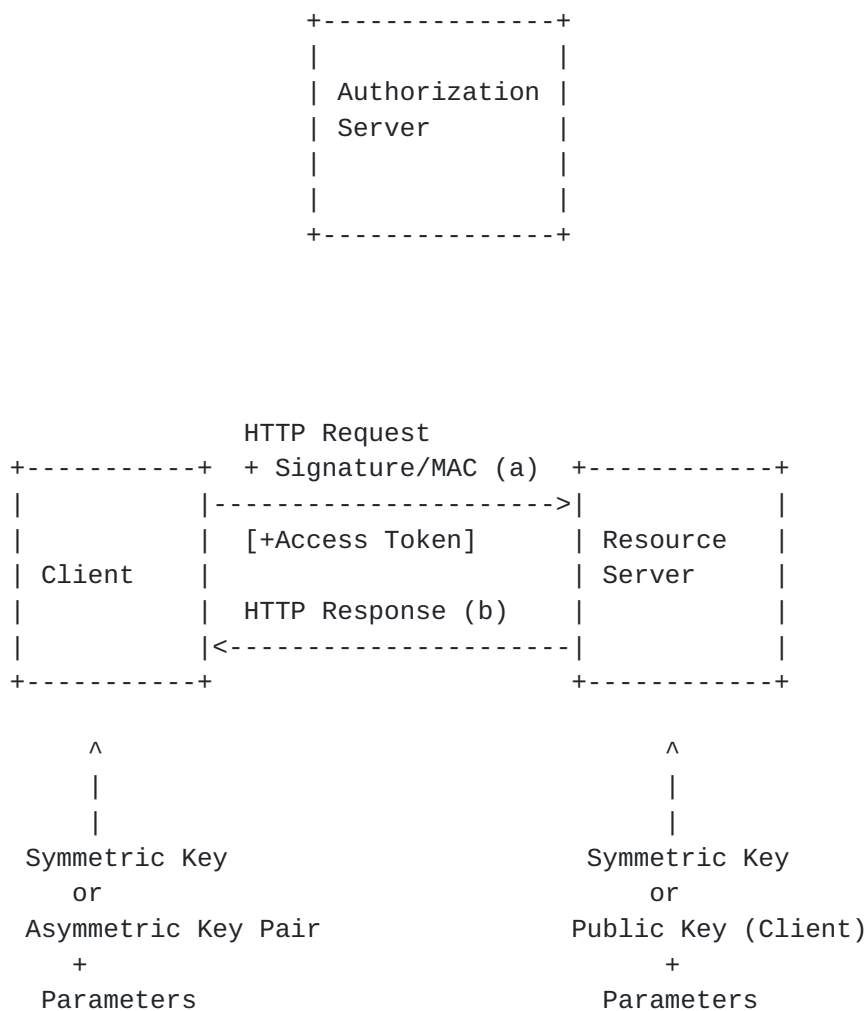
                  Figure 3: Client demonstrates PoP.

The specification describing the ability to sign the HTTP request
from the client to the resource server can be found in
[I-D.richer-oauth-signed-http-request].

[Editor's Note: Description about token introspection needs to be
added as well.

7. Requirements

RFC 4962 [RFC4962] gives useful guidelines for designers of
authentication and key management protocols. While RFC 4962 was
written with the AAA framework used for network access authentication
in mind the offered suggestions are useful for the design of other
key management systems as well. The following requirements list
applies OAuth 2.0 terminology to the requirements outlined in RFC
4962.

These requirements include

Cryptographic Algorithm Independent:

   The key management protocol MUST be cryptographic algorithm
   independent.

Strong, fresh session keys:

   Session keys MUST be strong and fresh. Each session deserves an
   independent session key, i.e., one that is generated specifically
   for the intended use. In context of OAuth this means that keying
   material is created in such a way that can only be used by the
   combination of a Client instance, protected resource, and
   authorization scope.

Limit Key Scope:

   Following the principle of least privilege, parties MUST NOT have
   access to keying material that is not needed to perform their
   role. Any protocol that is used to establish session keys MUST
   specify the scope for session keys, clearly identifying the
   parties to whom the session key is available.

Replay Detection Mechanism:

   The key management protocol exchanges MUST be replay protected.
   Replay protection allows a protocol message recipient to discard
   any message that was recorded during a previous legitimate
   dialogue and presented as though it belonged to the current
   dialogue.

Authenticate All Parties:

   Each party in the key management protocol MUST be authenticated to
   the other parties with whom they communicate. Authentication
   mechanisms MUST maintain the confidentiality of any secret values
   used in the authentication process. Secrets MUST NOT be sent to
   another party without confidentiality protection.

Authorization:

   Client and Resource Server authorization MUST be performed. These
   entities MUST demonstrate possession of the appropriate keying
   material, without disclosing it. Authorization is REQUIRED
   whenever a Client interacts with an Authorization Server. The
   authorization checking prevents an elevation of privilege attack,
   and it ensures that an unauthorized authorized is detected.

Keying Material Confidentiality and Integrity:

   While preserving algorithm independence, confidentiality and
   integrity of all keying material MUST be maintained.

Confirm Cryptographic Algorithm Selection:

   The selection of the "best" cryptographic algorithms SHOULD be
   securely confirmed. The mechanism SHOULD detect attempted roll-
   back attacks.

Uniquely Named Keys:

   Key management proposals require a robust key naming scheme,
   particularly where key caching is supported. The key name
   provides a way to refer to a key in a protocol so that it is clear
   to all parties which key is being referenced. Objects that cannot
   be named cannot be managed. All keys MUST be uniquely named, and
   the key name MUST NOT directly or indirectly disclose the keying
   material.

Prevent the Domino Effect:

   Compromise of a single Client MUST NOT compromise keying material
   held by any other Client within the system, including session keys
   and long-term keys. Likewise, compromise of a single Resource
   Server MUST NOT compromise keying material held by any other
   Resource Server within the system. In the context of a key
   hierarchy, this means that the compromise of one node in the key
   hierarchy must not disclose the information necessary to
   compromise other branches in the key hierarchy. Obviously, the

compromise of the root of the key hierarchy will compromise all of
the keys; however, a compromise in one branch MUST NOT result in
the compromise of other branches. There are many implications of
this requirement; however, two implications deserve highlighting.
First, the scope of the keying material must be defined and
understood by all parties that communicate with a party that holds
that keying material. Second, a party that holds keying material
in a key hierarchy must not share that keying material with
parties that are associated with other branches in the key
hierarchy.

Bind Key to its Context:

   Keying material MUST be bound to the appropriate context. The
   context includes the following.

   * The manner in which the keying material is expected to be used.

   * The other parties that are expected to have access to the
      keying material.

   * The expected lifetime of the keying material. Lifetime of a
      child key SHOULD NOT be greater than the lifetime of its parent
      in the key hierarchy.

   Any party with legitimate access to keying material can determine
   its context. In addition, the protocol MUST ensure that all
   parties with legitimate access to keying material have the same
   context for the keying material. This requires that the parties
   are properly identified and authenticated, so that all of the
   parties that have access to the keying material can be determined.
   The context will include the Client and the Resource Server
   identities in more than one form.

Authorization Restriction:

   If Client authorization is restricted, then the Client SHOULD be
   made aware of the restriction.

Client Identity Confidentiality:

   A Client has identity confidentiality when any party other than
   the Resource Server and the Authorization Server cannot
   sufficiently identify the Client within the anonymity set. In
   comparison to anonymity and pseudonymity, identity confidentiality
   is concerned with eavesdroppers and intermediaries. A key
   management protocol SHOULD provide this property.

Resource Owner Identity Confidentiality:

   Resource servers SHOULD be prevented from knowing the real or
   pseudonymous identity of the Resource Owner, since the
   Authorization Server is the only entity involved in verifying the
   Resource Owner's identity.

Collusion:

   Resource Servers that collude can be prevented from using
   information related to the Resource Owner to track the individual.
   That is, two different Resource Servers can be prevented from
   determining that the same Resource Owner has authenticated to both
   of them. This requires that each Authorization Server obtains
   different keying material as well as different access tokens with
   content that does not allow identification of the Resource Owner.

AS-to-RS Relationship Anonymity:

   This MAC Token security does not provide AAS-to-RS Relationship
   Anonymity since the Client has to inform the resource server about
   the Resource Server it wants to talk to. The Authorization Server
   needs to know how to encrypt the session key the Client and the
   Resource Server will be using.

As an additional requirement a solution MUST enable support for
channel bindings. The concept of channel binding, as defined in
[RFC5056], allows applications to establish that the two end-points
of a secure channel at one network layer are the same as at a higher
layer by binding authentication at the higher layer to the channel at
the lower layer.

There are performance concerns with the use of asymmetric
cryptography. Although symmetric key cryptography offers better
performance asymmetric cryptography offers additional security
properties. A solution MUST therefore offer the capability to
support both symmetric as well as asymmetric keys.

There are threats that relate to the experience of the software
developer as well as operational practices. Verifying the servers
identity in TLS is discussed at length in [RFC6125].

A number of the threats listed in Section 4 demand protection of the
access token content and a standardized solution, in form of a JSON-
based format, is available with the JSON Web Token (JWT)
[I-D.ietf-oauth-json-web-token]. Hence, threat related to the
protection of the access token itself are deferred to the
[I-D.ietf-oauth-json-web-token].

8. Security Considerations

The purpose of this document is to provide use cases, requirements,
and motivation for developing an OAuth security solution extending
Bearer Tokens. As such, this document is only about security.

9. IANA Considerations

This document does not require actions by IANA.

10. Acknowledgments

This document is the result of conference calls late 2012/early 2013
and in design team conference calls February 2013 of the IETF OAuth
working group. The following persons (in addition to the OAuth WG
chairs, Hannes Tschofenig, and Derek Atkins) provided their input
during these calls: Bill Mills, Justin Richer, Phil Hunt, Prateek
Mishra, Mike Jones, George Fletcher, Leif Johansson, Lucy Lynch, John
Bradley, Tony Nadalin, Klaas Wierenga, Thomas Hardjono, Brian
Campbell

In the appendix of this document we re-use content from [RFC4962] and
the authors would like thank Russ Housely and Bernard Aboba for their
work on RFC 4962.

11. References

11.1. Normative References

[I-D.bradley-oauth-pop-key-distribution]
           Bradley, J., Hunt, P., Jones, M., and H. Tschofenig,
           "OAuth 2.0 Proof-of-Possession: Authorization Server to
           Client Key Distribution", draft-bradley-oauth-pop-key-
           distribution-00 (work in progress), April 2014.

[I-D.ietf-httpbis-p1-messaging]
           Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
           (HTTP/1.1): Message Syntax and Routing", draft-ietf-
           httpbis-p1-messaging-26 (work in progress), February 2014.

[I-D.ietf-jose-json-web-encryption]
           Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)",
           draft-ietf-jose-json-web-encryption-25 (work in progress),
           March 2014.

   [I-D.ietf-oauth-json-web-token]
              Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token
              (JWT)", draft-ietf-oauth-json-web-token-19 (work in
              progress), March 2014.

   [I-D.jones-oauth-proof-of-possession]
              Jones, M., Bradley, J., and H. Tschofenig, "Proof-Of-
              Possession Semantics for JSON Web Tokens (JWTs)", draft-
              jones-oauth-proof-of-possession-00 (work in progress),
              April 2014.

   [I-D.richer-oauth-introspection]
              Richer, J., "OAuth Token Introspection", draft-richer-
              oauth-introspection-04 (work in progress), May 2013.

   [I-D.richer-oauth-introspection]
              Richer, J., "OAuth Token Introspection", draft-richer-
              oauth-introspection-04 (work in progress), May 2013.

   [I-D.richer-oauth-signed-http-request]
              Richer, J., Bradley, J., and H. Tschofenig, "A Method for
              Signing an HTTP Requests for OAuth", draft-richer-oauth-
              signed-http-request-01 (work in progress), April 2014.

   [I-D.tschofenig-oauth-audience]
              Tschofenig, H., "OAuth 2.0: Audience Information", draft-
              tschofenig-oauth-audience-00 (work in progress), February
              2013.

   [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
              Extensions (MIME) Part One: Format of Internet Message
              Bodies", RFC 2045, November 1996.

   [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-
              Hashing for Message Authentication", RFC 2104, February
              1997.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
              Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
              Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

   [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S.,
              Leach, P., Luotonen, A., and L. Stewart, "HTTP
              Authentication: Basic and Digest Access Authentication",
              RFC 2617, June 1999.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
           Resource Identifier (URI): Generic Syntax", STD 66, RFC
           3986, January 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an
           IANA Considerations Section in RFCs", BCP 26, RFC 5226,
           May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
           (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265,
           April 2011.

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

[W3C.REC-html401-19991224]
           Raggett, D., Hors, A., and I. Jacobs, "HTML 4.01
           Specification", World Wide Web Consortium Recommendation
           REC-html401-19991224, December 1999,
           <http://www.w3.org/TR/1999/REC-html401-19991224>.

11.2. Informative References

[I-D.hardjono-oauth-kerberos]
           Hardjono, T., "OAuth 2.0 support for the Kerberos V5
           Authentication Protocol", draft-hardjono-oauth-kerberos-01
           (work in progress), December 2010.

[NIST-FIPS-180-3]
           National Institute of Standards and Technology, "Secure
           Hash Standard (SHS). FIPS PUB 180-3, October 2008",
           October 2008.

[NIST800-63]
           Burr, W., Dodson, D., Perlner, R., Polk, T., Gupta, S.,
           and E. Nabbus, "NIST Special Publication 800-63-1,
           INFORMATION SECURITY", December 2008.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness
           Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The
           Kerberos Network Authentication Service (V5)", RFC 4120,
           July 2005.

    [RFC4279]   Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites
                for Transport Layer Security (TLS)", RFC 4279, December
                2005.

    [RFC4962]   Housley, R. and B. Aboba, "Guidance for Authentication,
                Authorization, and Accounting (AAA) Key Management", BCP
                132, RFC 4962, July 2007.

    [RFC5056]   Williams, N., "On the Use of Channel Bindings to Secure
                Channels", RFC 5056, November 2007.

    [RFC5849]   Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849,
                April 2010.

    [RFC5929]   Altman, J., Williams, N., and L. Zhu, "Channel Bindings
                for TLS", RFC 5929, July 2010.

    [RFC6125]   Saint-Andre, P. and J. Hodges, "Representation and
                Verification of Domain-Based Application Service Identity
                within Internet Public Key Infrastructure Using X.509
                (PKIX) Certificates in the Context of Transport Layer
                Security (TLS)", RFC 6125, March 2011.

    [RFC6750]   Jones, M. and D. Hardt, "The OAuth 2.0 Authorization
                Framework: Bearer Token Usage", RFC 6750, October 2012.

    [RFC6819]   Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0
                Threat Model and Security Considerations", RFC 6819,
                January 2013.

Authors' Addresses

   Phil Hunt (editor)
   Oracle Corporation

   Email: phil.hunt@yahoo.com

   Justin Richer
   The MITRE Corporation

   Email: jricher@mitre.org

   William Mills
   Yahoo! Inc.

   Email: wmills@yahoo-inc.com

Prateek Mishra
Oracle Corporation

Email: prateek.mishra@oracle.com

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at