Workload Identity in Multi System Environments
Internet-Draft
Intended status: Standards Track
Expires: 25 March 2026

Y. Sheffer
Intuit

21 September 2025

WIMSE Workload-to-Workload Authentication - Proposed Revision draft-sheffer-wimse-s2s-reduced-00

Abstract

This document is a proposed revision that simplifies the working group's workload-to-workload protocol document. If accepted by the working group, these changes will be back-ported to that document. Otherwise, it was a good try.

The WIMSE architecture defines authentication and authorization for software workloads in a variety of runtime environments, from the most basic ones up to complex multi-service, multi-cloud, multi-tenant deployments. This document defines the simplest, atomic unit of this architecture: the protocol between two workloads that need to verify each other's identity in order to communicate securely. The scope of this protocol is a single HTTP request-and-response pair. To address the needs of different setups, we propose two protocols, one at the application level and one that makes use of trusted TLS transport. These two protocols are compatible, in the sense that a single call chain can have some calls use one protocol and some use the other. Workload A can call Workload B with mutual TLS authentication, while the next call from Workload B to Workload C would be authenticated at the application level.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://yaronf.github.io/draft-sheffer-wimse-s2s-reduced/draft-sheffer-wimse-s2s-reduced.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-sheffer-wimse-s2s-reduced/.

Discussion of this document takes place on the Workload Identity in Multi System Environments Working Group mailing list (mailto:wimse@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/wimse/. Subscribe at https://www.ietf.org/mailman/listinfo/wimse/.

Source for this draft and an issue tracker can be found at https://github.com/yaronf/draft-sheffer-wimse-s2s-reduced.

Status of This Memo

This Internet-Draft will expire on 25 March 2026.

Copyright Notice

Table of Contents

1. Foreword

This document is a proposed revision that simplifies the working group’s workload-to-workload protocol document. The main changes compared to draft-ietf-wimse-s2s-protocol-06 are:

* Only one application-layer security solution is retained, the approach based on HTTP Message Signatures.

* The alternative DPoP-inspired solution and the WPT construct have been moved to an appendix, under the assumption that other protocols may leverage WPT in future WIMSE profiles.

All other changes (such as removal of the comparison between the two options, IANA registrations, and security considerations) stem from these two primary modifications. If accepted by the working group, these changes will be back-ported to the working group's main document.

2. Introduction

The WIMSE architecture defines authentication and authorization for software workloads This document defines authentication and authorization in the context of interaction between two workloads. This is the core component of the WIMSE architecture [I-D.ietf-wimse-arch]. For simplicity, this document focuses on HTTP-based services, and the workload-to-workload call consists of a single HTTP request and its response. We define the credentials that both workloads should possess and how they are used to protect the HTTP exchange.

There are multiple deployment styles in use today, and they result in different security properties. We propose to address them differently.

* Many use cases have various middleboxes inserted between pairs of workloads, resulting in a transport layer that is not end-to-end encrypted. We propose to address these use cases by protecting the HTTP messages at the application level (Section 4).

* The other commonly deployed architecture has a mutual-TLS connection between each pair of workloads. This setup can be addressed by a simpler solution (Section 5).

It is an explicit goal of this protocol that a workload deployment can include both architectures across a multi-chain call. In other words, Workload A can call Workload B with mutual TLS protection, while the next call to Workload C is protected at the application level.

2.1. Extending This Protocol to Other Use Cases

The protocol defined here is narrowly scoped, targeting only HTTP-based request/response services. To secure workloads communicating over other transports, new protocol bindings will need to be defined. We note though that this protocol is designed to allow some level of reuse. In particular, we expect that the Workload Identity Token (WIT) construct will be reusable in other settings. The Workload Proof Token (WPT) may be adaptable with some changes to different environments.

2.2. Deployment Architecture and Message Flow

Regardless of the transport between the workloads, we assume the following logical architecture (numbers refer to the sequence of steps listed below):

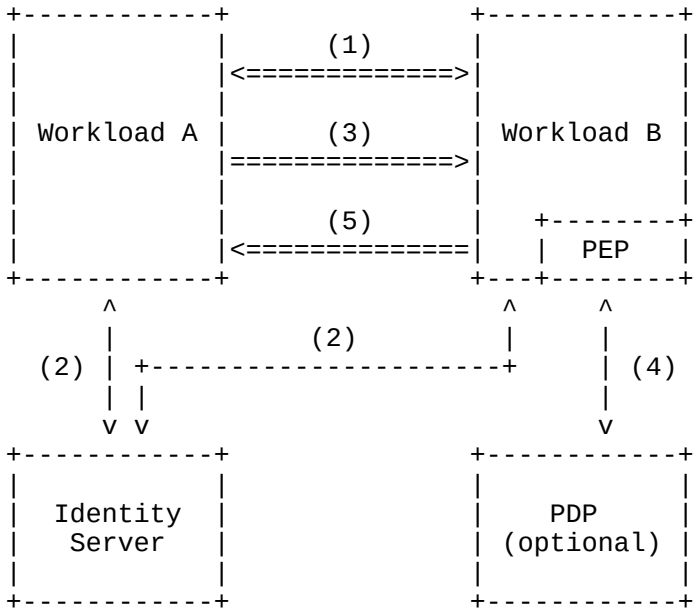

Figure 1: Sequence of Operations

The Identity Server provisions credentials to each of the workloads. At least Workload A (and possibly both) must be provisioned with a credential before the call can proceed. Details of communication with the Identity Server are out of scope of this document, however we do describe the credential received by the workload.

PEP is a Policy Enforcement Point, the component that allows the call to go through or blocks it. PDP is an optional Policy Decision Point, which may be deployed in architectures where policy management is centralized. All details of policy management and message authorization are out of scope of this document.

The high-level message flow is as follows:

1. A transport connection is set up. In the case of mutual TLS, this includes authentication of both workloads to one another. In the case of application-level security, the TLS connection is typically one-way authenticated, and workload-level authentication does not yet take place.

2. Workload A (and similarly, Workload B) obtains a credential from the Identity Server. This happens periodically, e.g. once every 24 hours.

3. Workload A makes an HTTP call into Workload B. This is a regular HTTP request, with the additional protection mechanisms defined below.

4. In the case of application-level security, Workload B authenticates Workload A (when using mutual TLS, this happened in step 1). In either case, Workload B decides whether to authorize the call. In certain architectures, Workload B may need to consult with an external server when making this decision.

5. Workload B returns a response to Workload A, which may be an error response or a regular one.

2.3. Workload Identifiers and Authentication Granularity

The specific format of workload identifiers (see [I-D.ietf-wimse-arch]) is set by local policy for each deployment, and this choice has several implications.

Prior to WIMSE, many use cases did not allow for fully granular authentication in containerized runtime platforms. For instance, with mutual TLS, there’s often no clear way to map the request’s external access reference (e.g., Kubernetes Ingress path, service name, or host header) to the SubjectAltName value in the server certificate. This means that the client could only verify if the server certificate is valid within a trust domain, not if it’s tied to a specific workload.

To enable mutual and granular authentication between workloads, two things must be in place:

* Each workload must know its own identifier.

* There needs to be an explicit mapping from the external handle used to access a workload (such as an Ingress path or service DNS name) to its workload identifier.

Once these conditions are met, the methods described in this document can be used for the caller and callee to mutually authenticate.

Implementations MUST allow for defining this mapping between the workload's access path and the workload identifier (e.g., through callback functions). Deployments SHOULD use these features to establish a consistent set of identifiers within their environment.

3. Conventions and Definitions

All terminology in this document follows [I-D.ietf-wimse-arch].

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

4. Application Level Workload-to-Workload Authentication

As noted in the Introduction, for many deployments communication between workloads cannot use end-to-end TLS. For these deployment styles, this document proposes application-level protection based on the Workload Identity Token defined below, and HTTP Message Signatures [RFC9421].

4.1. The Workload Identity Token

The Workload Identity Token (WIT) is a JWS [RFC7515] signed JWT [RFC7519] that represents the identity of a workload. It is issued by the Identity Server and binds a public key to the workload identity. See Section 6.3 for security considerations.

A WIT MUST contain the following claims, except where noted:

* in the JOSE header:

  - alg: An identifier for a JWS asymmetric digital signature algorithm (registered algorithm identifiers are listed in the IANA JOSE Algorithms registry [IANA.JOSE.ALGS]). The value none MUST NOT be used.

- typ: the WIT is explicitly typed, as recommended in Section 3.11 of [RFC8725], using the wimse-id+jwt media type.

* in the JWT claims:

- iss: The issuer of the token, which is the Identity Server, represented by a URI. The iss claim is RECOMMENDED but optional, see Section 4.1.3 for more.

- sub: The subject of the token, which is the identity of the workload, represented by a URI. See [I-D.ietf-wimse-arch] for details of the Workload Identifier. And see Section 2.3 for security implications of these identifiers.

- exp: The expiration time of the token (as defined in Section 4.1.4 of [RFC7519]). WITs should be refreshed regularly, e.g. on the order of hours.

- jti: A unique identifier for the token. This claim is OPTIONAL. The jti claim is frequently useful for auditing issuance of individual WITs or to revoke them, but some token generation environments do not support it.

- cnf: A confirmation claim referencing the public key of the workload.

  o jwk: Within the cnf claim, a jwk key MUST be present that contains the public key of the workload as defined in Section 3.2 of [RFC7800]. The workload MUST prove possession of the corresponding private key when presenting the WIT to another party, which can be accomplished by using it in conjunction with one of the methods in Appendix A or Section 4.2. As such, it MUST NOT be used as a bearer token and is not intended for use in the Authorization header.

    + alg: Within the jwk object, an alg field MUST be present. Allowed values are listed in the IANA "JSON Web Signature and Encryption Algorithms" registry established by [RFC7518]. The presented proof (WPT or http-sig) MUST be produced with the algorithm specified in this field. The value none MUST NOT be used. Algorithms used in combination with symmetric keys MUST NOT be used. Also encryption algorithms MUST NOT be used as this would require additional key distribution outside of the WIT. To promote interoperability, the ES256 signing algorithm MUST be supported by general purpose implementations of this document.

As noted in [I-D.ietf-wimse-arch], a workload identifier is a URI with a trust domain component. The runtime environment often determines which URI scheme is used, e.g. if SPIFFE is used to authenticate workloads, it mandates "spiffe" URIs. However for those deployments where this is not the case, this document (Section 7.4) defines the "wimse" URI scheme which can be used by any deployment that implements this protocol.

An example WIT might look like this:


```
=============== NOTE: '\' line wrapping per RFC 8792 ================

eyJhbGciOiJFUzI1NiIsImtpZCI6Ikp1bmUgNSIsInR5cCI6IndpbXNlLWlkK2p3dCJ9\
.eyJjbmYiOnsiandrIjp7ImFsZyI6IkVkRFNBIiwiY3J2IjoiRWQyNTUxOSIsImt0eSI\
6Ik9LUCIsIngiOiIxQ1hYdmZsTl9MVlZzSXNZWHNVdkIwM0ptbEdXZUNIcVFWdW91Q0Y\
5MmJnIn19LCJleHAiOjE3NDU1MTI1MTAsImlhdCI6MTc0NTUwODkxMCwianRpIjoiYmQ\
yYTdiNWJmODU3M2E0MWFkYjRjYmYzY2ZhMDFlMTUiLCJzdWIiOiJ3aW1zZTovL2V4YW1\
wbGUuY29tL3NwZWNpZmljLXdvcmtsb2FkIn0.xpODXCUhZ2zk-1-W3VEqbqWhBX6_OJI\
l7vtjahgwJStMOCRn6J6is6f5mz-Pi5-Xk6FmV44k48NzulqMDVJbAw
```


Figure 2: An example Workload Identity Token (WIT)

The decoded JOSE header of the WIT from the example above is shown here:


```
{
  "alg": "ES256",
  "kid": "June 5",
  "typ": "wimse-id+jwt"
}
```


Figure 3: Example WIT JOSE Header

The decoded JWT claims of the WIT from the example above are shown here:

```
{
  "cnf": {
    "jwk": {
      "alg": "EdDSA",
      "crv": "Ed25519",
      "kty": "OKP",
      "x": "1CXXvflN_LVVsIsYXsUvB03JmlGWeCHqQVuouCF92bg"
    }
  },
  "exp": 1745512510,
  "iat": 1745508910,
  "jti": "bd2a7b5bf8573a41adb4cbf3cfa01e15",
  "sub": "wimse://example.com/specific-workload"
}
```


Figure 4: Example WIT Claims

The claims indicate that the example WIT:

* is valid until Thu Apr 24 2025 16:35:10 GMT (represented as NumericDate Section 2 of [RFC7519] value 1745512510).

* identifies the workload to which the token was issued as wimse://example.com/specific-workload.

* has a unique identifier of bd2a7b5bf8573a41adb4cbf3cfa01e15.

* binds the public key represented by the jwk confirmation method to the workload wimse://example.com/specific-workload.

* requires the proof to be produced with the EdDSA signature algorithm.

For elucidative purposes only, the workload’s key, including the private part, is shown below in JWK [RFC7517] format:


```
{
 "kty": "OKP",
 "crv": "Ed25519",
 "x": "1CXXvflN_LVVsIsYXsUvB03JmlGWeCHqQVuouCF92bg",
 "d": "sdLX8yCYKqo_XvGBLn-ZWeKT7llYeeQpgeCaXVxb5kY"
}
```


Figure 5: Example Workload’s Key

The afore-exampled WIT is signed with the private key of the Identity Server. The public key(s) of the Identity Server need to be known to all workloads in order to verify the signature of the WIT. The Identity Server's public key from this example is shown below in JWK [RFC7517] format:

```
{
 "kty": "EC",
 "kid": "June 5",
 "crv": "P-256",
 "x": "Dy47KDeYao6kOhxSraJeJizjVxHjjo-9NsnrMqLyvOo",
 "y": "bj3s7bncoSYURzAzF0jBy0JOnnP5-5E11vx5QoYEFgk"
}
```

Figure 6: Example Identity Server Key

4.1.1. The WIT HTTP Header

A WIT is conveyed in an HTTP header field named Workload-Identity-Token.

ABNF [RFC5234] for the value of Workload-Identity-Token header field is provided in Figure 7:

```
ALPHA = %x41-5A / %x61-7A ; A-Z / a-z
DIGIT = %x30-39 ; 0-9
base64url = 1*(ALPHA / DIGIT / "-" / "_")
JWT =  base64url "." base64url "." base64url
WIT =  JWT
```

Figure 7: Workload-Identity-Token Header Field ABNF

The following shows the WIT from Figure 2 in an example of a Workload-Identity-Token header field:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

Workload-Identity-Token: eyJhbGciOiJFUzI1NiIsImtpZCI6Ikp1bmUgNSIsInR\
5cCI6IndpbXNlLWlkK2p3dCJ9.eyJjbmYiOnsiandrIjp7ImFsZyI6IkVkRFNBIiwiY3\
J2IjoiRWQyNTUxOSIsImt0eSI6Ik9LUCIsIngiOiIxQ1hYdmZsTl9MVlZzSXNZWHNVdk\
IwM0ptbEdXZUNIcVFWdW91Q0Y5MmJnIn19LCJleHAiOjE3NDU1MTI1MTAsImlhdCI6MT\
c0NTUwODkxMCwianRpIjoiYmQyYTdiNWJmODU3M2E0MWFkYjRjYmYzY2ZhMDFlMTUiLC\
JzdWIiOiJ3aW1zZTovL2V4YW1wbGUuY29tL3NwZWNpZmljLXdvcmtsb2FkIn0.xpODXC\
UhZ2zk-1-W3VEqbqWhBX6_OJIl7vtjahgwJStMOCRn6J6is6f5mz-Pi5-Xk6FmV44k48\
NzulqMDVJbAw
```

Figure 8: An example Workload Identity Token HTTP Header Field

Note that per [RFC9110], header field names are case insensitive; thus, Workload-Identity-Token, workload-identity-token, WORKLOAD-IDENTITY-TOKEN, etc., are all valid and equivalent header field names. However, case is significant in the header field value.

4.1.2. Including Additional Claims

The WIT contains JSON structures and therefore can be trivially extended by adding more claims beyond those defined in the current specification. This, however, could result in interoperability issues, which the following rules are addressing.

* To ensure interoperability in WIMSE environments, the use of Private claim names (Sec. 4.3 of [RFC7519]) is NOT RECOMMENDED.

* In closed environments, deployers MAY freely add claims to the WIT. Such claims SHOULD be collision-resistant, such as example.com/myclaim.

* A recipient that does not understand such claims MUST ignore them, as per Sec. 4 of [RFC7519].

* Outside of closed environments, new claims MUST be registered with IANA [IANA.JWT.CLAIMS] before they can be used.

4.1.3. A note on iss claim and key distribution

It is RECOMMENDED that the WIT carries an iss claim. This specification itself does not make use of a potential iss claim but also carries the trust domain in the workload identifier (see [I-D.ietf-wimse-arch] for a definition of the identifier and related rules). Implementations MAY include the iss claim in the form of a https URL to facilitate key distribution via mechanisms like the jwks_uri from [RFC8414] but alternative key distribution methods may make use of the trust domain included in the workload identifier which is carried in the mandatory sub claim.

4.2. Protecting the Workload-to-Workload Traffic

The workload uses the Workload Identity Token (Section 4.1) and the private key associated with the WIT’s public key, to sign the request and optionally, the response. See Section 6.3 for security considerations. This section defines a profile of the Message Signatures specification [RFC9421].

The request is signed as per [RFC9421]. The following derived components MUST be signed:

* @method

* @request-target

In addition, the following request headers MUST be signed when they exist:

* Content-Type

* Content-Digest

* Authorization

* Txn-Token [I-D.ietf-oauth-transaction-tokens]

* Workload-Identity-Token

If the response is signed, the following components MUST be signed:

* @status

* @method;req

* @request-target;req

* Content-Type if it exists

* Content-Digest if it exists

* Workload-Identity-Token

To ensure the message is fully integrity-protected, if the request or response includes a message body, the sender MUST include (and the receiver MUST verify) a Content-Digest header.

For both requests and responses, the following signature parameters MUST be included:

* created

* expires - expiration MUST be short, e.g. on the order of minutes. The WIMSE architecture will provide separate mechanisms in support of long-lived compute processes.

* nonce

* tag - the value for implementations of this specification is wimse-workload-to-workload

The following signature parameters in the Signature-Input header MUST NOT be used:

* keyid - The signing key is sent along with the message in the WIT. Additionally specifying the key identity would add confusion.

* alg - The signature algorithm is specified in the jwk section of the cnf claim in the WIT. See Section 4.1 and Sec. 3.3.7 of [RFC9421] for details.

It is RECOMMENDED to include only one signature with the HTTP message. If multiple ones are included, then the signature label included in both the Signature-Input and Signature headers SHOULD be wimse.

A sender MUST ensure that each nonce it generates is unique, at least among messages sent to the same recipient. To detect message replays, a recipient SHOULD reject a message (request or response) if a nonce generated by a certain peer is seen more than once.

Implementors need to be aware that the WIT is extracted from the message before the message signature is validated. Recipients of signed HTTP messages MUST validate the signature and content of the WIT before validating the HTTP message signature. They MUST ensure that the message is not processed further before it has been fully validated.

Either client or server MAY send an Accept-Signature header, but is not required to do so. When this header is sent, it MUST include the header components listed above.

Following is a non-normative example of a signed request and a signed response, where the caller is using the keys specified in Figure 5.

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

GET /gimme-ice-cream?flavor=vanilla HTTP/1.1
Host: example.com
Signature: wimse=:zyR2W7QeUEYwGT8q+jU5HyEiPhC4iiBJlAfwu6otL+kXBYOMVl\
ZEfGDnTlQiYmUWQMyu69811lIDWbCK42yWCw==:
Signature-Input: wimse=("@method" "@request-target" "workload-identi\
ty-token");created=1754558248;expires=1754558548;nonce="abcd1111";ta\
g="wimse-workload-to-workload"
Workload-Identity-Token: eyJhbGciOiJFZERTQSIsImtpZCI6Imlzc3Vlci1rZXk\
iLCJ0eXAiOiJ3aW1zZS1pZCtqd3QifQ.eyJjbmYiOnsiandrIjp7ImFsZyI6IkVkRFNB\
IiwiY3J2IjoiRWQyNTUxOSIsImtpZCI6InN2Yy1hLWtleSIsImt0eSI6Ik9LUCIsIngi\
OiJEUEhDOFdKaTZ3WEttN3BpRVdXRDRQdHBiTUdJMUcyOXVTMFl5MnNmQk5zIn19LCJl\
eHAiOjE3NTQ1NTg1NDgsImlhdCI6MTc1NDU1ODI0OCwiaXNzIjoiaHR0cHM6Ly9leGFt\
cGxlLmNvbS9pc3N1ZXIiLCJqdGkiOiJ3aXQtMTc1NDU1ODI0ODQwMjQ5NjAwMCIsInN1\
YiI6IndpbXNlOi8vZXhhbXBsZS5jb20vc3ZjQSJ9.OAPARpluzH34v4bhCtPovuOvTsx\
cVbAV9WSqFlbiKdNvfEVN4uRNQSRs7ePfMIJB2uqgX71XjBFH433nDzJZBQ
```

Figure 9: Signed Request

Assuming that the workload being called has the following keypair:

```
{
  "alg": "EdDSA",
  "crv": "Ed25519",
  "d": "JlNJxsZl_PC00EkoRUQbtCrzDtZ5vhFN_6qWtwghttY",
  "kid": "svc-b-key",
  "kty": "OKP",
  "x": "gz2aSJE-g9w1rbgJiNps4Gb8IPk50k5oJUEbLDusayc"
}
```

Figure 10: Callee Private Key

A signed response would be:

```
================ NOTE: '\' line wrapping per RFC 8792 ================

Response:

HTTP/1.1 404 Not Found
Connection: close
Content-Digest: sha-256=:47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU\
=:
Content-Type: text/plain
Signature: wimse=:WAjxziuCiYRqCzetetDwaTS7Ka9yMwB+dAHVJPw3VkUH+c8c4A\
5BKrCsPlD/ymy+7PgwXl3y3mVdaD4ww7WqDA==:
Signature-Input: wimse=("@status" "workload-identity-token" "content\
-type" "content-digest" "@method";req "@request-target";req);created\
=1754558248;expires=1754558550;nonce="abcd2222";tag="wimse-workload-\
to-workload"
Workload-Identity-Token: eyJhbGciOiJFZERTQSIsImtpZCI6Imlzc3Vlci1rZXk\
iLCJ0eXAiOiJ3aW1zZS1pZCtqd3QifQ.eyJjbmYiOnsiandrIjp7ImFsZyI6IkVkRFNB\
IiwiY3J2IjoiRWQyNTUxOSIsImtpZCI6InN2Yy1iLWtleSIsImt0eSI6Ik9LUCIsIngi\
OiJnejJhU0pFLWc5dzFyYmdKaU5wczRHYjhJUGs1MGs1b0pVRWJMRHVzYXljIn19LCJl\
eHAiOjE3NTQ1NTg1NTAsImlhdCI6MTc1NDU1ODI1MCwiaXNzIjoiaHR0cHM6Ly9leGFt\
cGxlLmNvbS9pc3N1ZXIiLCJqdGkiOiJ3aXQtMTc1NDU1ODI0ODQwMjY4MTAwMCIsInN1\
YiI6IndpbXNlOi8vZXhhbXBsZS5jb20vc3ZjQiJ9.yJ4gGQ1DQBX0E9Xp5fIgj6Ucpf9\
LI_gHnhURZxqu6atT-3hpbFTgw4rd-6knM7-HClok4b6N2ViZaEDcz6IMCg

No ice cream today.
```

Figure 11: Signed Response

4.3. Error Conditions

Errors may occur during the processing of the message signature. If the signature verification fails for any reason, such as an invalid signature, an expired validity time window, or a malformed data structure, an error is returned. When used with HTTP, this will be in response to an API call, so an HTTP status code such as 400 (Bad Request) is appropriate. This response could include more details as per [RFC9457], such as an indicator that the wrong key material or algorithm was used. For non-HTTP protocols, the WIMSE profile should define how errors are handled.

5. Using Mutual TLS for Workload-to-Workload Authentication

As noted in the introduction, for many deployments, transport-level protection of application traffic using TLS is ideal.

5.1. The Workload Identity Certificate

The Workload Identity Certificate is an X.509 certificate. The workload identity MUST be encoded in a SubjectAltName extension of type URI. There MUST be only one SubjectAltName extension of type URI in a workload certificate. If the workload will act as a TLS server for clients that do not understand workload identities it is RECOMMENDED that the workload certificate contain a SubjectAltName of type DNSName with the appropriate DNS names for the server. The certificate MAY contain SubjectAltName extensions of other types.

5.2. Workload Identity Certificate Validation

Workload certificates may be used to authenticate both the server and client side of the connections. When validating a workload certificate, the relying party MUST use the trust anchors configured for the trust domain in the workload identity to validate the peer's certificate. Other PKIX [RFC5280] path validation rules apply. WIMSE clients and servers MUST validate that the trust domain portion of the workload certificate matches the expected trust domain for the other side of the connection.

Servers wishing to use the workload certificate for authorizing the client MUST require client certificate authentication in the TLS handshake. Other methods of post handshake authentication are not specified by this document.

WIMSE server certificates SHOULD have the id-kp-serverAuth extended key usage [RFC5280] field set and WIMSE client certificates SHOULD have the id-kp-clientAuth extended key usage field set. A certificate that is used for both client and server connections may have both fields set. This specification does not make any other requirements beyond [RFC5280] on the contents of workload certificates or on the certification authorities that issue workload certificates.

5.2.1. Server Name Validation

If the WIMSE client uses a hostname to connect to the server and the server certificate contain a DNS SAN the client MUST perform standard host name validation (Section 6.3 of [RFC9525]) unless it is configured with the information necessary to validate the peer's workload identity. If the client did not perform standard host name validation then the WIMSE client SHOULD further use the workload identifier to validate the server. The host portion of the workload identifier is NOT treated as a host name as specified in section 6.4 of [RFC9525] but rather as a trust domain. The server identity is encoded in the path portion of the workload identifier in a

deployment specific way. Validating the workload identity could be a simple match on the trust domain and path portions of the identifier or validation may be based on the specific details on how the identifier is constructed. The path portion of the WIMSE identifier MUST always be considered in the scope of the trust domain. In most cases it is preferable to validate the entire workload identifier, see Section 2.3 for additional implementation advice.

5.3. Client Authorization Using the Workload Identity

The server application retrieves the workload identifier from the client certificate subjectAltName, which in turn is obtained from the TLS layer. The identifier is used in authorization, accounting and auditing. For example, the full workload identifier may be matched against ACLs to authorize actions requested by the peer and the identifier may be included in log messages to associate actions to the client workload for audit purposes. A deployment may specify other authorization policies based on the specific details of how the workload identifier is constructed. The path portion of the workload identifier MUST always be considered in the scope of the trust domain. See Section 2.3 on additional security implications of workload identifiers.

6. Security Considerations

6.1. Workload Identity

The Workload Identifier is scoped within an issuer and therefore any sub-components (path portion of Identifier) are only unique within a trust domain defined by the issuer. Using a Workload Identifier without taking into account the trust domain could allow one domain to issue tokens to spoof identities in another domain. Additionally, the trust domain must be tied to an authorized issuer cryptographic trust anchor through some mechanism such as a JWKS or X.509 certificate chain. The association of an issuer, trust domain and a cryptographic trust anchor MUST be communicated securely out of band.

6.2. Workload Identity Token and Proof of Possession

// TODO: this section needs to be reworked if WPT is removed as a
// primary option.

The Workload Identity Token (WIT) is bound to a secret cryptographic key and is always presented with a proof of possession as described in Section 4.1. The WIT is a general purpose token that can be presented in multiple contexts. The WIT and its PoP are only used in the application-level options, and both are not used in MTLS. The

WIT MUST NOT be used as a bearer token. While this helps reduce the sensitivity of the token it is still possible that a token and its proof of possession may be captured and replayed within the PoP's lifetime. The following are some mitigations for the capture and reuse of the proof of possession (PoP):

* Preventing Eavesdropping and Interception with TLS

An attacker observing or intercepting the communication channel can view the token and its proof of possession and attempt to replay it to gain an advantage. In order to prevent this the token and proof of possession MUST be sent over a secure, server authenticated TLS connection unless a secure channel is provided by some other mechanisms. Host name validation according to Section 5.2.1 MUST be performed. The WIT itself is not usable without a proof of possession.

* Limiting Proof of Possession Lifespan

The proof of possession MUST be time limited. A PoP should only be valid over the time necessary for it to be successfully used for the purpose it is needed. This will typically be on the order of minutes. PoPs received outside their validity time MUST be rejected.

* Limiting Proof of Possession Scope

In order to reduce the risk of theft and replay the PoP should have a limited scope. For example, a PoP may be targeted for use with a specific workload and even a specific transaction to reduce the impact of a stolen PoP. In some cases a workload may wish to reuse a PoP for a period of time or have it accepted by multiple target workloads. A careful analysis is warranted to understand the impacts to the system if a PoP is disclosed allowing it to be presented by an attacker along with a captured WIT.

* Replay Protection

A proof of possession includes the jti claim that MUST uniquely identify it, within the scope of a particular sender. This claim SHOULD be used by the receiver to perform basic replay protection against tokens it has already seen. Depending upon the design of the system it may be difficult to synchronize the replay cache across all token validators. If an attacker can somehow influence the identity of the validator (e.g. which cluster member receives the message) then replay protection would not be effective.

* Binding to TLS Endpoint

The POP MAY be bound to a transport layer sender such as the client identity of a TLS session or TLS channel binding parameters. The mechanisms for binding are outside the scope of this specification.

6.3. Workload Identity Key Management

Both the Workload Identity Token and the Workload Identity Certificate carry a public key. The corresponding private key:

* MUST be kept private
* MUST be individual for each Workload Identifier (see [I-D.ietf-wimse-arch])
* MUST NOT be used once the credential is expired
* SHOULD be re-generated for each new Workload Identity Token or Certificate.

6.4. Middle Boxes

// TODO: this section needs to be reworked or eliminated if WPT is // removed as a primary option.

In some deployments the Workload Identity Token and proof of possession may pass through multiple systems. The communication between the systems is over TLS, but the token and PoP are available in the clear at each intermediary. While the intermediary cannot modify the token or the information within the PoP they can attempt to capture and replay the token or modify the data not protected by the PoP.

Mitigations listed in Section 4 can be used to provide some protection from middle boxes. However we note that the DPoP-inspired solution (Appendix A) does not protect major portions of the request and response and therefore does not provide protection from an actively malicious middle box. Deployments should perform analysis on their situation to determine if it is appropriate to trust and allow traffic to pass through a middle box.

6.5. Privacy Considerations

WITs and the proofs of possession may contain private information such as user names or other identities. Care should be taken to prevent the disclosure of this information. The use of TLS helps protect the privacy of WITs and proofs of possession.

WITs and certificates with workload identifiers are typically associated with a workload and not a specific user, however in some deployments the workload may be associated directly to a user. While these are exceptional cases a deployment should evaluate if the disclosure of WITs or certificates can be used to track a user.

7. IANA Considerations

7.1. JSON Web Token Claims

IANA is requested to add the following entries to the "JSON Web Token Claims" registry [IANA.JWT.CLAIMS]:

Claim Name	Claim Description	Change Controller	Reference
tth	Transaction Token hash	IETF	RFC XXX, Appendix A
wth	Workload Identity Token hash	IETF	RFC XXX, Appendix A
oth	Other Tokens hashes	IETF	RFC XXX, Appendix A

Table 1

7.2. Media Type Registration

IANA is requested to register the following entries to the "Media Types" registry [IANA.MEDIA.TYPES]:

* application/wimse-id+jwt, per Section 7.2.1.

* application/wimse-proof+jwt, per Section 7.2.2.

7.2.1. application/wimse-id+jwt

Type name: application

Subtype name: wimse-id+jwt

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: Encoding considerations are identical to those specified for the "application/jwt" media type. See [RFC7519].

Security considerations: See the Security Considerations section of RFC XXX.

Interoperability considerations: N/A

Published specification: RFC XXX, Section 4.1.

Applications that use this media type: Identity servers that vend Workload Identity Tokens, and Workloads that use these tokens to authenticate to each other.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): None

Macintosh file type code(s): N/A

Person & email address to contact for further information:

See the Authors' Addresses section of RFC XXX.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC XXX.

Change controller: Internet Engineering Task Force (iesg@ietf.org).

7.2.2. application/wimse-proof+jwt

// Should we keep this? It is essential to the WPT definition so we // probably should.

Type name: application

Subtype name: wimse-proof+jwt

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: Encoding considerations are identical to those specified for the "application/jwt" media type. See [RFC7519].

Security considerations: See the Security Considerations section of RFC XXX.

Interoperability considerations: N/A

Published specification: RFC XXX, Appendix A.

Applications that use this media type: Workloads that use these tokens to integrity-protect messages in the WIMSE workload-to-workload protocol.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): None

Macintosh file type code(s): N/A

Person & email address to contact for further information:

See the Authors' Addresses section of RFC XXX.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC XXX.

Change controller: Internet Engineering Task Force (iesg@ietf.org).

7.3. Hypertext Transfer Protocol (HTTP) Field Name Registration

IANA is requested to register the following entries to the "Hypertext Transfer Protocol (HTTP) Field Name Registry" [IANA.HTTP.FIELDS]:

* Workload-Identity-Token, per Section 7.3.1.

* Workload-Proof-Token, per Section 7.3.2.

7.3.1. Workload-Identity-Token

* Field Name: Workload-Identity-Token
* Status: permanent
* Structured Type: N/A
* Specification Document: RFC XXX, Section 4.1.1
* Comments: see reference above for an ABNF syntax of this field

7.3.2. Workload-Proof-Token

// Should we keep this? It only applies to WPT-in-HTTP so probably
// not.

* Field Name: Workload-Proof-Token
* Status: permanent
* Structured Type: N/A
* Specification Document: RFC XXX, Appendix A
* Comments: see reference above for an ABNF syntax of this field

7.4. URI Scheme Registration

IANA is requested to register the "wimse" scheme to the "URI Schemes" registry [IANA.URI.SCHEMES]:

* Scheme name: wimse
* Status: permanent
* Applications/protocols that use this scheme name: the WIMSE workload-to-workload authentication protocol.
* Contact: IETF Chair chair@ietf.org (mailto:chair@ietf.org)
* Change controller: IESG iesg@ietf.org (mailto:iesg@ietf.org)
* References: Section 4 of this document (RFC XXX).

8. References

8.1. 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, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

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

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

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

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

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/rfc/rfc7800>.

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

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/rfc/rfc8414>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/rfc/rfc8725>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/rfc/rfc9421>.

[RFC9525] Saint-Andre, P. and R. Salz, "Service Identity in TLS", RFC 9525, DOI 10.17487/RFC9525, November 2023, <https://www.rfc-editor.org/rfc/rfc9525>.

8.2. Informative References

[I-D.ietf-oauth-transaction-tokens]
Tulshibagwale, A., Fletcher, G., and P. Kasselman, "Transaction Tokens", Work in Progress, Internet-Draft, draft-ietf-oauth-transaction-tokens-06, 28 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-transaction-tokens-06>.

[I-D.ietf-wimse-arch]
Salowey, J. A., Rosomakho, Y., and H. Tschofenig, "Workload Identity in a Multi System Environment (WIMSE) Architecture", Work in Progress, Internet-Draft, draft-ietf-wimse-arch-05, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-arch-05>.

[IANA.HTTP.FIELDS]
IANA, "Hypertext Transfer Protocol (HTTP) Field Name Registry", <https://www.iana.org/assignments/http-fields>.

[IANA.JOSE.ALGS]
IANA, "JSON Web Signature and Encryption Algorithms", <https://www.iana.org/assignments/jose>.

[IANA.JWT.CLAIMS]
IANA, "JSON Web Token Claims",
<https://www.iana.org/assignments/jwt>.

[IANA.MEDIA.TYPES]
IANA, "Media Types",
<https://www.iana.org/assignments/media-types>.

[IANA.URI.SCHEMES]
IANA, "Uniform Resource Identifier (URI) Schemes",
<https://www.iana.org/assignments/uri-schemes>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/rfc/rfc9449>.

[RFC9457] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", RFC 9457, DOI 10.17487/RFC9457, July 2023, <https://www.rfc-editor.org/rfc/rfc9457>.

Appendix A. DPoP-Inspired Authentication

This appendix defines an alternative mechanism that provides different security guarantees than HTTP Message Signatures. For protecting synchronous HTTP traffic, the use of HTTP Message Signatures is mandatory. However, future WIMSE profiles may adopt the DPoP-inspired mechanism and the Workload Proof Token (WPT) it defines. Since the current form of the WPT includes certain HTTP-specific dependencies (particularly in the aud and oth claims), it would need to be adapted to support non-HTTP protocols.

The mechanism defined here is inspired by the OAuth DPoP specification [RFC9449], and uses a DPoP-like mechanism to authenticate the calling workload in the context of the request. The Workload Identity Token (Section 4.1) is sent in the request as described in Section 4.1.1. An additional JWT, the Workload Proof Token (WPT), is signed by the private key corresponding to the public key in the WIT. When used with HTTP, the WPT is sent in the Workload-Proof-Token header field of the request. The ABNF syntax of the Workload-Proof-Token header field is:

```
WPT =  JWT
```

Figure 12: Workload-Proof-Token Header Field ABNF

where the JWT projection is defined in Figure 7.

A WPT MUST contain the following:

* in the JOSE header:

  - alg: An identifier for an appropriate JWS asymmetric digital signature algorithm corresponding to the confirmation key in the associated WIT. The value MUST match the alg value of the jwk in the cnf claim of the WIT. See Section 4.1 for valid values and restrictions.

  - typ: the WPT is explicitly typed, as recommended in Section 3.11 of [RFC8725], using the application/wimse-proof+jwt media type.

* in the JWT claims:

  - aud: The audience SHOULD contain the HTTP target URI (Section 7.1 of [RFC9110]) of the request to which the WPT is attached, without query or fragment parts. However, there may be some normalization, rewriting or other process that requires the audience to be set to a deployment-specific value. See also Section 2.3 for more details.

  - exp: The expiration time of the WPT (as defined in Section 4.1.4 of [RFC7519]). WPT lifetimes MUST be short, e.g., on the order of minutes or seconds.

  - jti: An identifier for the token. The value MUST be unique, at least within the scope of the sender.

  - wth: Hash of the Workload Identity Token, defined in Section 4.1. The value is the base64url encoding of the SHA-256 hash of the ASCII encoding of the WIT’s value.

  - ath: Hash of the OAuth access token, if present in the request, which might convey end-user identity and/or authorization context of the request. The value, as per Section 4.1 of [RFC9449], is the base64url encoding of the SHA-256 hash of the ASCII encoding of the access token’s value.

  - tth: Hash of the Txn-Token [I-D.ietf-oauth-transaction-tokens], if present in the request, which might convey end-user identity and/or authorization context of the request. The value MUST be the result of a base64url encoding (as defined in Section 2 of [RFC7515]) of the SHA-256 hash of the ASCII encoding of the associated token’s value.

- oth: Hash(es) of other token(s) in the request that convey end-user identity and/or authorization context of the request. The value is a JSON object with a key-value pair for each such token. For each, in the absence of an application profile specifying details, the key corresponds to the header field name containing the token, and the value is the base64url encoding of the SHA-256 hash of the ASCII bytes of the header field value with any leading or trailing spaces removed. The header field name MUST be normalized by converting it to all lower case. Header fields occurring multiple times in the request are not supported by default. An application profile may specify different behavior for a key, such as using a different hash algorithm or means of locating the token in the request.

To clarify: the ath, tth and oth claims are each mandatory if the respective tokens are included in the request.

The rules for using non-standard claims in WPTs are similar to the rules for WITs, Section 4.1.2.

An example WPT might look like the following:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

eyJhbGciOiJFZERTQSIsInR5cCI6IndpbXNlLXByb29mK2p3dCJ9.eyJhdGgiOiJDTDR\
3amZwUm1OZi1iZFlJYllMblY5ZDVyTUFSR3dLWUUxMHdVd3pDMGpJIiwiYXVkIjoiaHR\
0cHM6Ly93b3JrbG9hZC5leGFtcGxlLmNvbS9wYXRoIiwiZXhwIjoxNzQ1NTA5MjEwLCJ\
qdGkiOiJlMzI5YmI4Njk2YWE0YWVjYTA0ODg2ZGQ3NmU3OGIyNiIsInd0aCI6InJvN3h\
GT1NHX2pZeG1VV2Z3ZXFrNVgxc2M2TDBzQ2o3NVdLVDkxZ014eFUifQ.oSegRTrBxuQN\
55oyWRK5PnPEZLhgRy0Va7BpxBw-a64E3map15dbDo9ArRcJ8M4Z4QZ829CCppfnuaLI\
ei1bBQ
```

Figure 13: Example Workload Proof Token (WPT)

The decoded JOSE header of the WPT from the example above is shown here:

```
{
  "alg": "EdDSA",
  "typ": "wimse-proof+jwt"
}
```

Figure 14: Example WPT JOSE Header

The decoded JWT claims of the WPT from the example above are shown here:

```
{
  "ath": "CL4wjfpRmNf-bdYIbYLnV9d5rMARGwKYE10wUwzC0jI",
  "aud": "https://workload.example.com/path",
  "exp": 1740755048,
  "jti": "0c740386ca1dcad37de1b5f9de1b0705",
  "wth": "aA0W_oFJK7qV7zYhcmzR1KOXVCHjd2x6c4sOQLvE90Y"
}
```

Figure 15: Example WPT Claims

An example of an HTTP request with both the WIT and WPT from prior examples is shown below:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

POST /path HTTP/1.1
Host: workload.example.com
Content-Type: application/json
Authorization: Bearer 16_mAd0GiwaZokU26_0902100
Workload-Identity-Token: eyJhbGciOiJFUzI1NiIsImtpZCI6Ikp1bmUgNSIsInR\
5cCI6IndpbXNlLWlkK2p3dCJ9.eyJjbmYiOnsiandrIjp7ImNydiI6IkVkMjU1MTkiLC\
JrdHkiOiJPS1AiLCJ4Ijoiclp3VUEwVHJIazRBWEs5MkY2Vll2bUhIWDN4VU0tSUdsck\
11VkNRaG04VSJ9fSwiZXhwIjoxNzQwNzU4MzQ4LCJpYXQiOjE3NDA3NTQ3NDgsImp0aS\
I6IjRmYzc3ZmNlZjU3MWIzYmIzM2I2NzJlYWYyMDRmYWY0Iiwic3ViIjoid2ltc2U6Ly\
9leGFtcGxlLmNvbS9zcGVjaWZpYy13b3JrbG9hZCJ9.j-WlF3bufTwWeVZQntPhlzvST\
Pwf37-4wfazJZARdHYmW9S_olB5nKEqwqTZpIX_LoVVIcyK0VBE7Fa0CMvw2g
Workload-Proof-Token: eyJhbGciOiJFZERTQSIsInR5cCI6IndpbXNlLXByb29mK2\
p3dCJ9.eyJhdGgiOiJDTDR3amZwUm1OZi1iZFlJYllMblY5ZDVyTUFSR3dLWUUxMHdVd\
3pDMGpJIiwiYXVkIjoiaHR0cHM6Ly93b3JrbG9hZC5leGFtcGxlLmNvbS9wYXRoIiwiZ\
XhwIjoxNzQwNzU1MDQ4LCJqdGkiOiIwYzc0MDM4NmNhMWRjYWQzN2RlMWI1ZjlkZTFiM\
DcwNSIsInd0aCI6ImFBMFdfb0ZKSzdxVjd6WWhjbXpSMUtPWFZDSGpkMng2YzRzT1FMd\
kU5MFkifQ.W9RZqieXeD-UgdtbYf8ZNkf2_6_6b_kJSfkODQdq3_QDSSGOhVbRAR3qQo\
Ou0SzihiG6HCsGwslfo4WdvnH5AQ

{"do stuff":"please"}
```

Figure 16: Example HTTP Request with WIT and WPT

To validate the WPT in the request, the recipient MUST ensure the following:

* There is exactly one Workload-Proof-Token header field in the request.

* The Workload-Proof-Token header field value is a single and well-formed JWT.

* The signature algorithm in the alg JOSE header string-equal matches the alg attribute of the jwk in the cnf claim of the WIT.

* The WPT signature is valid using the public key from the confirmation claim of the WIT.

* The typ JOSE header parameter of the WPT conveys a media type of wimse-proof+jwt.

* The aud claim of the WPT matches the target URI, or an acceptable alias or normalization thereof, of the HTTP request in which the WPT was received, ignoring any query and fragment parts. See also Section 2.3 for implementation advice on this verification check.

* The exp claim is present and conveys a time that has not passed. WPTs with an expiration time unreasonably far in the future SHOULD be rejected.

* The wth claim is present and matches the hash of the token value conveyed in the Workload-Identity-Token header.

* It is RECOMMENDED to check that the value of the jti claim has not been used before in the time window in which the respective WPT would be considered valid.

* If presented in conjunction with an OAuth access token, the value of the ath claim matches the hash of that token’s value.

* If presented in conjunction with a Txn-Token, the value of the tth claim matches the hash of that token’s value.

* If presented in conjunction with a token conveying end-user identity or authorization context, the value of the oth claim matches the hash of that token’s value.

* If the oth claim is present, verify the hashes of all tokens listed in the oth claim per the default behavior defined in Appendix A or as specified by an application specific profile. If the oth claim contains entries that are not understood by the recipient, the WPT MUST be rejected. Conversely, additional tokens not covered by the oth claim MUST NOT be used by the recipient to make authorization decisions.

Appendix B. Document History

// RFC Editor: please remove before publication.

B.1. draft-sheffer-wimse-s2s-reduced-00

* A proposal to restructure the document.

B.2. draft-ietf-wimse-s2s-protocol-07

* Rework the WPT’s oth claim

B.3. draft-ietf-wimse-s2s-protocol-06

* Explicit definition of the Workload Identity Certificate.

* Definition of the validation of workload identifiers as part of workload authentication. Still work in progress.

B.4. draft-ietf-wimse-s2s-protocol-05

* Removed the entire Workload Identity section which is now covered in the Architecture document.

* Content-Digest is mandatory with HTTP-Sig.

* Some wording on extending the protocol beyond HTTP.

* IANA considerations.

B.5. draft-ietf-wimse-s2s-protocol-04

* Require cnf.jwk.alg in WIT which restricts signature algorithm of WPT or HTTP-Sig.

* Replay protection as a SHOULD for both WPT and HTTP-Sig.

* Consolidate terminology with the Architecture draft.

B.6. draft-ietf-wimse-s2s-protocol-03

* Consistently use "workload".

* Implement comments from the SPIFFE community.

* Make iss claim in WIT optional and add wording about its relation to key distribution.

* Remove iss claim from WPT.

* Make jti claim in WIT optional.

* Error handling for the application level methods.

B.7. draft-ietf-wimse-s2s-protocol-02

* Coexistence with bearer tokens.

* Improve the architecture diagram.

* Some more ABNF.

* Clarified identifiers and URIs.

* Moved an author to acknowledgments.

B.8. draft-ietf-wimse-s2s-protocol-01

* Addressed multiple comments from Pieter.

* Clarified WIMSE identity concepts, specifically "trust domain" and "workload identifier".

* Much more detail around mTLS, including some normative language.

* WIT (the identity token) is now included in the WPT proof of possession.

* Added a section comparing the DPoP-inspired app-level security option to the Message Signature-based alternative.

B.9. draft-ietf-wimse-s2s-protocol-00

* Initial WG draft, an exact copy of draft-sheffer-wimse-s2s-protocol-00

* Added this document history section

Acknowledgments

The authors would like to thank Pieter Kasselman for his detailed comments.

We thank Daniel Feldman for his contributions to earlier versions of this document.

Author’s Address

Yaron Sheffer
Intuit

Email: yaronf.ietf@gmail.com