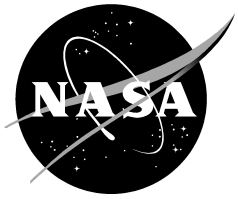
NASA/TM–2019–220364



UAS Service Supplier Framework for Authentication and Authorization

A federated approach to securing communications between service suppliers within the UAS Traffic Management system

Joseph L. Rios
Ames Research Center, Moffett Field, California

Irene Smith
Ames Research Center, Moffett Field, California

Priya Venkatesen
SGT Inc., Moffett Field, California

September 2019

NASA STI Program ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

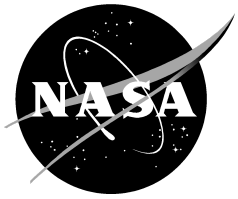
Access the NASA STI program home page at http://www.sti.nasa.gov

E-mail your question to help@sti.nasa.gov

Phone the NASA STI Information Desk at 757-864-9658

Write to:
NASA STI Information Desk
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199

NASA/TM–2019–220364



UAS Service Supplier Framework for Authentication and Authorization

A federated approach to securing communications between service suppliers within the UAS Traffic Management system

Joseph L. Rios
Ames Research Center, Moffett Field, California

Irene Smith
Ames Research Center, Moffett Field, California

Priya Venkatesen
SGT Inc., Moffett Field, California

National Aeronautics
and Space Administration

Ames Research Center
Moffett Field, California 94035-1000

September 2019

This report is available in electronic form at

https://www.sti.nasa.gov

1. Introduction **7**

2. Supporting UTM Documents **7**

3. Overview **7**
3.1. UAS Traffic Management 7
3.2. Authentication 8
3.3. Authorization 9

4. Threat Modeling and Testing **10**

5. Key Concepts **11**
5.1. Public Key Infrastructure (PKI) 11
5.2. USS Naming 12
5.3. Additional X.509 Elements 13
5.4. JSON 13
5.5. .well-known 14
5.6. JWK 14
5.7. JWS 15
5.8. Base64url 16
5.9. JWT 16
5.10. Token Revocation 17
5.11. Token Checking 17
5.12. Message Signing 17
5.12.1 Header 18
5.12.2 Payload 18
5.12.3 Signature 18
5.12.4 Supplying the Signature 18
5.12.5 Example 19
5.12.6 Discussion 22
5.13. Application Programming Interface (API) Documentation 23
5.14. Discovery of Authorization Server 23
5.15. OAuth 2.0 Token Request 23
5.15.1. Signing for Authentication 25

6. Exercising UFAA **25**
6.1. Token Request 26
6.2. Certificate Fetch 27
6.3. Application-level Security Checks 27
6.3.1. Name checks 28
6.3.1.1. Implications of Relaxing Payload to Access Token 28
6.3.1.2. Implications of Relaxing Payload to Signature Check 28
6.3.2. Access Token Time Checks 29
6.4. Potential Algorithm for USS Data Exchange 29

Appendices **31**
A1. Open Issues 31
A1.1. Message Signing 31
A1.2. Authentication to the Authorization Server 31
A1.3. Use of .well-known 32
A1.4. Use of aud Claim 32
A1.5. Implementation of JWK 32
A1.6. PKI 32
A1.7. Varying Token Timeouts 33
A1.8. Encrypted Messages 33
A1.9. Order of Checks 33
A1.10. Additional Uses of Signatures 34
A1.11. JWT Best Current Practices 34
A2. Role-Based Access Control 34
A2.1. RBAC Overview 34
A2.2. Operations 35
A2.3. Objects 35
A2.4. Permissions/Scopes 35
A2.5. Roles 37
A2.6. Users/Subjects 38
A2.7. Subject Assignments 38
A2.8. Permission Assignments 38
A2.9. Example 38
A3. Design Decisions Considered **39**
A3.1. HTTP Signing for Token Requests 39
A3.1.1. JWS Details in HTTP Signing 39
A3.1.2. Supplying the Signature 40
A3.1.3. Reasons for Exclusion from UFAA 40
A3.2. UTM-Specific Certificate Authority 41
A3.3. Mutual TLS 41
A3.4. Use of Access Tokens Outside USS Network. 41
A3.5. Multiple Authorization Server Providers 41
A3.6 Token Checking and Revocation 42

Acronyms **43**

References **45**
IETF Request for Comment (RFC) Documents (in order of RFC #) 45
National Institute of Standards and Technology (NIST) Documents 47
UTM Documents 47
Other Documents 48

1. Introduction

The Unmanned Aircraft Systems (UAS) Traffic Management (UTM) Service Supplier (USS) Framework for Authentication and Authorization (UFAA) is the basis for secure and confident data exchanges between the Flight Information Management System (FIMS) and the USS Network and within the USS Network itself. UFAA is built upon the OAuth 2.0 approach to federated authorization, with details supplied by various Internet Engineering Task Force (IETF) Request for Comment documents (RFCs). Whenever possible, a specific standard is used to support a design decision within UFAA, and when an applicable standard is not found/used, the technical decision is explained as much as possible. This document describes UFAA in detail. It is intended to serve as a reference for other key UTM documents and should be useful to many UTM stakeholders. This is an informative document. The requirements stated or implied in this document will need to be fully vetted and reviewed by appropriate stakeholders before becoming operational. It is highly encouraged that in any design decisions for future applicable standards that are counter to or missing from the decisions presented herein be justified in that future standard.

2. Supporting UTM Documents

ID	Title	Status
UTM_Accounting	UTM Accounting	Internal draft only
USS_Spec	USS Specification	Partner draft
UTM_ConOps_NASA	NASA UTM Concept of Operations	Final
UTM_ConOps_FAA	FAA UTM Concept of Operations, v1.0	Final
UTM_ConOps_FAA_v2	FAA UTM Concept of Operations, v2.0	Internal draft

3. Overview

3.1. UAS Traffic Management

This document is focused on the architecture and requirements of an authentication and authorization system within the UTM System. UTM has the stated goal of providing safe, efficient, and fair access to the low-altitude airspace for small Unmanned Aircraft Systems (sUAS). The management of these sUAS operations is envisioned to take a different form than

the management of traditional aviation in the National Airspace System (NAS). In UTM, management of the airspace is a partnership between the Air Navigation Service Provider (ANSP) and industry. Some of the key services that might have been provided by the ANSP for traditional aviation are provided instead by a federated set of UAS Service Suppliers (USS). A new, UTM-specific component that is implemented on the ANSP side for this system is called the Flight Information Management System (FIMS). This federated, collaborative approach to airspace management supports several key properties of the UTM System including scalability, enabling more seamless evolution, partitioning sUAS management from traditional aviation, and others. For a comprehensive description of the UTM System, both versions of the Concept of Operations ([UTM_ConOps_NASA] and [UTM_ConOps_FAA]) provide the primary starting point. The [USS_Spec] provides further details on what it means to be a USS. This document focuses on the authentication and authorization architecture and associated requirements for the sharing of data amongst the various USSs and between USSs and FIMS.

3.2. Authentication

Authentication is defined[1] by the National Institute of Standards and Technology (NIST) as "Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system." Authentication is primarily a machine-to-machine activity in the USS Network, including authentication to the authorization server by USSs. USS subjects are authenticated using signatures generated by public-private key pairs. The public keys are shared via X.509 certificates. These certificates are rooted at a trusted Certificate Authority (CA). If these public certificates are self-served, they are available at a well-known location on the USS server. Fetching the certificate from this location is accomplished via a secure connection (using the Transport Layer Security standards) with a requirement that a name in the fetched certificate matches a name the certificate used in creating the secure connection. The name of the subject is provisioned upon successful on-boarding[2] of the USS in the UTM System and remains static even if the certificates change in the future. The name is an X.509v3 DNS (Domain Name System) name, supplied as a Subject Alternative Name of type "DNS." More details are provided in the USS Naming section below.

The identity and roles assigned to the USS are maintained in a separate identity management system which are described in Appendix A2. The identities and roles provisioned to each identity are securely available to the UTM authorization server and are known to the identified USS.

When requests are made to another server in the USS Network, they are signed using the certificate available in the well-known location. The receiving server checks the signature using that public key in the appropriate certificate. This certificate may be cached on the receiving server or may be fetched if it is not available locally. If the signature on the request is valid, then

[1] See https://csrc.nist.gov/glossary/term/authentication, accessed 08 Aug 2019.
[2] On-boarding of USSs and provisioning of identities within the authorization server is out of scope of this document.

the name in the certificate used to check the signature represents an authenticated subject. More details on the signing and checking process are provided in the Message Signing section.

3.3. Authorization

Authorization is defined by NIST 800-82r2 and RFC 4949 as “The right or a permission that is granted to a system entity to access a system resource.” OAuth 2.0 is a framework for delegated, federated authorization. It allows for one party (the client) to act on behalf of another party (the resource owner) to yet a third party (the resource server) while protecting sensitive credential information through mediation via an authorization server. As a primary feature, OAuth 2.0 removes the requirement of identity management from the resource server for these exchanges. This helps the UTM System achieve its design goals of relinquishing a significant portion of centralized management to actors that need a method to trust each other. In UFAA, a single authorization server, called FIMS_Authz, provides tokens to USSs that can be used to access resources on other USSs[3].

This approach to resource access is achieved through the request of access tokens from FIMS_Authz. These access tokens have specific details about what the bearer of the token is allowed to access, from whom, by when, and under other such constraints. These access tokens are passed as bearer tokens in HTTP requests, however they do not work as strict bearer tokens, since UFAA leverages requires further checks on the token. These additional token constraints allow for security[4] within the system in the event that the token is obtained by a malicious party, thus breaking the pure definition of a bearer token (i.e. any entity in possession of a bearer token would have all privileges defined in the token).

Authorization is somewhat simplified from the full OAuth 2.0 specification in that the client (i.e. the systems making the requests for access tokens) and the resource owner (i.e. those that have access rights to resources stored on other systems) are the same entity, namely USSs (or FIMS). Thus, in OAuth 2.0 language, UFAA uses the client credentials grant flow wherein the client provides its own credentials to the authorization server to receive an access token. The simplified exchange is described in the following diagram:

[3] Note that for the remainder of the document we will refer only to USS-to-USS communication, but in reality, FIMS uses UFAA in the same way to allow FIMS-to-USS and USS-to-FIMS communications.
[4] Security is a broad term that implies many specific features including, but not limited to, authentication, message integrity, and non-repudiation.

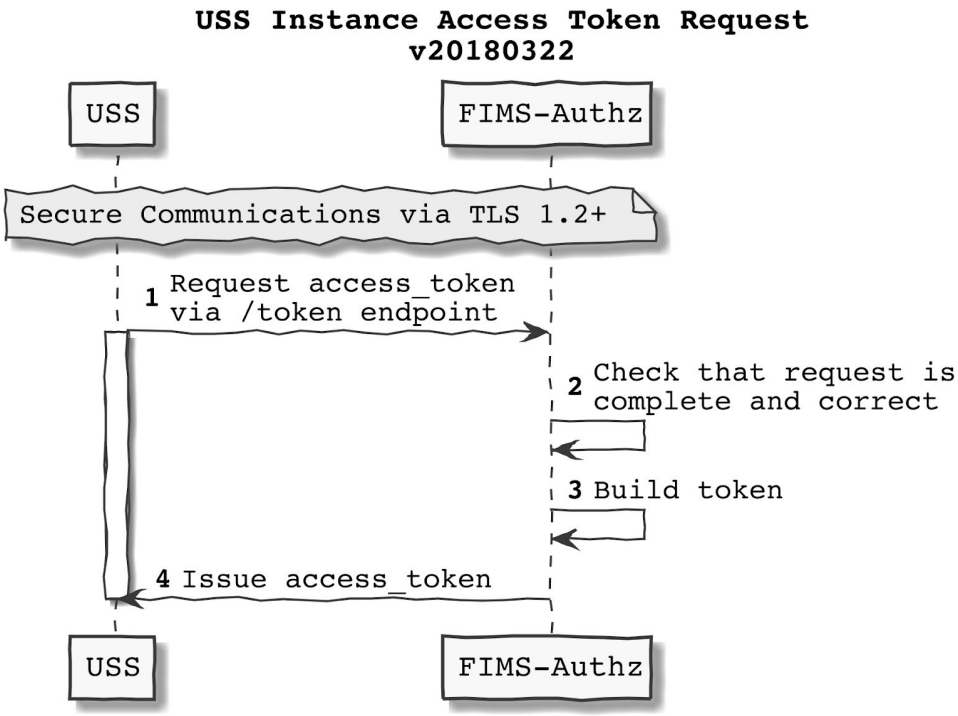


Figure 1. USS to FIMS Authorization server token request.

4. Threat Modeling and Testing

The contents of this document are the result of examination of industry best practices, threat modeling, collaborative simulations, and field testing. While the full details of the threat modeling exercises will not be published, there are key references for understanding how it was approached for this system.

The primary document for threat modeling this system is RFC 6819. That document details all known threats to OAuth 2.0 implementations as well as mitigations for those threats. Note that the overall threat tree is pruned considerably for UFAA since there are limited flows that are implemented and there are controls outside of OAuth 2.0 that mitigate certain threats.

A newer document ("JSON Web Token Best Current Practices, draft-ietf-oauth-jwt-bcp-06") describing threats and mitigations relevant to UFAA is currently a draft Best Current Practices from IETF. Those threats and mitigations have not been fully incorporated into this document. However, the further use of that Best Current Practices document are discussed in the Appendix as an open issue.

For overall threat modeling, the NASA UTM Project took an approach leveraging both the STRIDE (a mnemonic for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privileges) and DREAD (a mnemonic for Damage, Reproducibility,

Exploitability, Affected Users, and Discoverability). STRIDE aids in categorizing threats and DREAD helps quantify the potential damage of a realized threat. There are newer, more advanced approaches to threat modeling, but leveraging these two selected approaches provided reasonable coverage for the design of this system, despite debate as to the effectiveness of STRIDE and DREAD (even from the creators at Microsoft, who no longer use these approaches). This combined STRIDE/DREAD approach was influenced by the well-documented threat modeling of the OpenStack project. NASA is further developing approaches to threat modeling that could be applied to this or future systems.

As for testing, many of the elements described herein were exercised in NASA's Technical Capability Level 3 (TCL3) and TCL4 Demonstrations as well as during collaborative simulations leading up to those events.

5. Key Concepts

In this section, an overview of several key concepts is provided. By providing insight into each of these on their own merits, and as independent of each other as possible, it will ease the discussion of UFAA to follow. Many of these concepts are supported by published specifications. Those specifications are typically written such that the application or system using that concept can make design decisions as to which aspects of the specification will be required for that application or system. Thus "optional" items in the specification can become "required" by a design decision in a particular implementation, like UFAA. Additionally, the specifications allow extensions to support the application. Whenever possible, those design decisions on optional elements and extensions are described in the subsections below.

5.1. Public Key Infrastructure (PKI)

UTM relies on traditional certificate authorities to provide confidence in the identities of servers for the purpose of establishing secure connections. Specifically, USSs must support Transport Layer Security (TLS) (the USS Specification defers to the most recent version of NIST 800-52 to define TLS requirements, currently 1.2+ as of this writing) and they do so by establishing a secure connection using a public X.509 certificate that is assured by a chain of trust to a known CA. This is a completely independent system from UTM and is also the basis for secure connections on the Internet.

These certificates are generally not used for application-level signing or encryption. Thus, an additional certificate is required to meet the needs of UTM communication, which relies on the ability of a server to sign its requests to provide authentication, message integrity, and non-repudiation with the USS Network. This additional certificate must be issued by a valid CA, and may actually be the same certificate used to establish the TLS connections at the USSs' discretion, though this is not required nor recommended. There is a requirement that this certificate is made available in a well-known location such that clients can request it.

This directly requestable certificate will be called “the UFAA Certificate” to distinguish from “The Internet Connection Certificate” used to create the TLS connection. The UFAA Certificate is required to have at least one DNS name in common with at least one DNS name contained in The Internet Connection Certificate (see the USS Naming section below for details). Among other useful features, this naming connection requirement provides assurance that the server a USS is connected to is the same one that is allowed to use the UTM Framework Certificate being fetched, since there are appropriate matching names. This name that is common between the two certificates is also the USS Name assigned to the USS providing these certificates.

This approach to PKI leverages the existing Internet approach and chains of trust, allows for application-specific certificates, but does not require the implementation of an application-specific certificate authority.

Public-Key Infrastructure using X.509 is typically called PKIX, as defined in RFC 5280.

5.2. USS Naming

The concepts related to naming entities (specifically USSs) within the UTM System and UFAA leverages best practices and appropriate references. For example RFC 6125 related to service identity, TLS, and X.509 has specific guidelines on which names should be used by applications relying on TLS. Specifically, this RFC notes that the X.509v3 Subject Alternative Name of the DNS type should be used and that wildcards in names should be considered carefully. Within UFAA, the provisioned name of a USS MUST be free of wildcards. The provisioned name MUST NOT be matched against wildcards in any certificate (The Internet Connection Certificate nor the UFAA Certificate). This is the field UFAA will use for identity provision as used by the Authorization Server and thus, subsequently will be used for comparing names in various data objects including access tokens and certificates.

The relevant section of the X.509 certificate would look something like the following:

```
X509v3 extensions:
            ...
            X509v3 Subject Alternative Name:
                DNS:www.example.com, othername:<unsupported>,
DNS:www.sub.example.com, othername:<unsupported>
            ...
```

So if this were the certificate used to establish identity in the subject following on-boarding, the identifier for this USS would be locked in as either `www.example.com` or `www.sub.example.com` since the name must be in the Subject Alternative Name section with

a DNS type. Again, even if the certificate is changed in the future, the subject would have to ensure that the name used to provision identity in the UTM System is again included in the new certificate. The other alternative would be the retirement of the current identifier and the establishment of a new one. This process is not yet documented and is likely more intensive than having the USS obtain a new certificate with the appropriate names.

The provisioned name must not include any wildcards. The maximum number of names allowed in the Subject Alternative Name field will be limited to, say, fewer than 100, pending further insight or formailization of this requirement.

5.3. Additional X.509 Elements

The X.509v3 extension field Key Usage must be present in the UFAA Certificate and the digitalSignature and contentCommitment/nonRepudiation bits MUST be enabled. See RFC 5280 section on Key Usage for the definition and usage of these bits.

The X.509 certificate MUST be in DER (Distinguished Encoding Rules) format (ITU-T X.690 defines DER, which is a part of the ASN.1/X.680 standard) to allow for appropriate thumbprinting.

Certificates are rooted to a Certificate Authority (CA). In order to have confidence in that trusted root, there needs to be some process to manage appropriate (trusted) CAs. Currently, that process is delegated to the well-documented approach from Mozilla and their Mozilla Root Store Policy. That policy allows for an updated list of trusted CAs that is distributed with Mozilla browsers. That list is leveraged for the purposes of UFAA. The implementation of that policy allows for an updated list of trusted CAs listed online. Any CA on that list with a geographic focus of ‘global’ or ‘USA’ will be considered valid for use as a trust root for any certificate used in UFAA.

5.4. JSON

“JavaScript Object Notation (JSON) is a text format for the serialization of structured data.” Simply put, it is a structured way of organizing key-value pairs for the exchange of data. Text encoding is specified as UTF-8. See RFC 8259 for details. The following is an example provided by json-schema.org:

```
{
    "title": "Person",
    "type": "object",
    "properties": {
        "firstName": {
            "type": "string"
        },
        "lastName": {
            "type": "string"
        },
        "age": {
            "description": "Age in years",
            "type": "integer",
            "minimum": 0
        }
    },
    "required": ["firstName", "lastName"]
}
```

5.5. .well-known

A well-known Uniform Resource Identifier (URI) is a URI (RFC 3986) whose path component begins with the characters "/.well-known/", and whose scheme is "HTTP", "HTTPS" (RFC 5785). It is possible to register specific resource names that can trail .well-known in the path such that the chance of conflict in naming is reduced between applications using this scheme. For UTM, a path starting with “/.well-known/uas-traffic-management/” will be required. This may be registered with IANA in the future when some appropriate standards body is available to act as a point of contact for that registration.

5.6. JWK

A JWK is a JSON-formatted file that describes cryptographic key data. In this case the data will be an X.509 certificate as described in the PKI section above. A set of JWK objects can be formatted as a JWKS (JSON Web Key Set). See RFC 7517 for details.

There will be a file representing a JWKS called “utm.jwks” to complete the path component as “/.well-known/uas-traffic-management/utm.jwks”. This JWKS must contain a JWK describing the UFAA Certificate. That JWK must contain the following fields:

- kty: Key Type parameter, required in all JWK per specification.

- use: Use parameter, must contain the value “sig” indicating the key is used for signatures.
- alg: Algorithm parameter, the set of algorithms intended for use with this key. This set will likely be defined precisely in future versions of this document.
- kid: Key ID parameter, a unique identifier for this key. Must be a Universally Unique Identifier (UUID), version 4 (UUIDv4). This will be used to match against the ‘kid’ parameter used in Message Signing discussed below.
- x5u: X.509 URL parameter is a URI that refers to an X.509 certificate. This URL path component must begin with “/.well-known/uas-traffic-management/” and be on the same server as the JWKS. Usage rules must match “use” parameter.
- x5t#S256: X.509 Certificate SHA-256 Thumbprint for the same certificate referenced in x5u.

The following is an example of an appropriately formatted JWKS file that would be located at www.example.com/.well-known/uas-traffic-management/utm.jwks:

```
{
  "keys": [
    {
      "kty": "RSA",
      "use": "sig",
      "kid": "f6d7ac10-0ad9-4193-8cbd-453087ecd472",
      "x5u":
"www.example.com/.well-known/uas-traffic-management/my_cert.der",
      "x5t#S256":
"77ad11c0fbb461eabe35c1dde60c5a871ae80b5c019029b4cdb2493f2f2813db"
    }
  ]
}
```

Whenever the JWK or JWKS file is fetched from a server for purposes of UTM, the requirements provided in RFC 6125 will be in effect. The steps in Section 6.1 of RFC 6125 are especially relevant. Without duplicating the detail found in that reference, the process is to have a target identifier that is the UTM identifier for that USS (which is a DNS name as described above). This section will be updated as needed and as developed.

5.7. JWS

A JWS (JSON Web Signature) is a way to represent data along with a signature in a standardized way based on JSON. See RFC 7515 for details. JWS is used in several contexts within UFAA, so further details are deferred until discussion of those contexts below.

The choice of signing algorithms is broad. RFC 7518 is the key reference for algorithms to be used with JWS. Specifically Section 3.1 of that document details the algorithms that may be used with JWS. For simplicity, this specification defers to the "Required," "Recommended," and "Recommended+" algorithms as the ONLY ALLOWED algorithms for signing. See section A1.11 for further discussion on this set of algorithms. All USSs must support reading signatures that are generated with these algorithms and reject all signatures generated with any other algorithm. This limits UFAA to allowing only HMAC using SHA-256, RSASSA-PKCS1-v1_5 using SHA-256, and ECDSA using P-256 and SHA-256. Those algorithms are further detailed in RFC 7518 with appropriate references.

5.8. Base64url

Base64url is a method of encoding data and is defined in Section 2 of RFC 7515 as follows:

> *Base64 encoding using the URL- and filename-safe character set defined in Section 5 of RFC 4648 [RFC 4648], with all trailing '=' characters omitted (as permitted by Section 3.2) and without the inclusion of any line breaks, whitespace, or other additional characters. Note that the base64url encoding of the empty octet sequence is the empty string.*

5.9. JWT

A JWT (JSON Web Token, pronounced "jot" like "dot") contains a formalized set of "claims" presented as JSON provided as the body of a JWS (or JWE which is currently not part of UFAA) that is signed by some entity. See RFC 7519 for details.

The access token provided by the authorization server within UFAA is a JWT. That JWT contains the following JWT-defined claims:

- `iss`: The Issuer Claim identifies the system that issued the JWT, specifically the authorization server in UFAA.
- `sub`: The Subject Claim identifies the entity authorized to access resources and actually use the access token, value set to the provisioned name within UTM (a static DNS Name as discussed above).
- `exp`: The Expiration Claim indicates a time after which the token is to be considered invalid. In UFAA, this value is set to 30 minutes after the "issued at" time. Note this is a Unix timestamp, not an ISO timestamp per the JWT spec.
- `iat`: The Issued At Claim indicates the time at which the token was issued. Note this is a Unix timestamp, not an ISO timestamp per the JWT spec.
- `jti`: The JWT ID Claim is a unique identifier for the JWT. In UFAA, this is a UUIDv4.

And it will also contain at least the following other claim (not included in the JWT spec):

- `scopes`: The set of scopes granted by the authorization server. An array of strings.

Other fields may be present and will be further documented as they are finalized. Specifically, the `aud` claim is is discussed more in Section A1.4.

The following is an example JWT claim set for a USS, with provisioned name "uss.provider321.net" that would be signed as a JWS to be used as an access_token:

```
{
    "iss": "fims-authz.utmserver.com",
    "scope": [
        "utm.nasa.gov_write.operation"
    ]
    "iat": 1521759704,
    "exp": 1553295704,
    "sub": "uss.provider321.net",
    "jti": "c3a1b1d1-0cf8-497a-ab90-7c06805f4d86"
}
```

5.10. Token Revocation

In some implementations and within the OAuth 2.0 concept, there is the potential to revoke tokens such that they become invalid for authorization. UFAA does not support token revocation via the FIMS_Authz server. While token revocation can provide additional security for unneeded or compromised tokens, the infrastructure to support token revocation introduces new attack vectors against UFAA. In addition, token revocation as a feature is non-trivial to scale and may harm efficient growth of this overall system.

5.11. Token Checking

Similar to token revocation, the concept of token validation at the authorization server can add certain security benefits. However, this process does open UFAA to additional threats that do not currently outweigh the benefits. So, until the cost-benefit balance changes, there is no support for token validation as a service within UFAA.

5.12. Message Signing

UTM has three major security goals in the exchange of operational messages within the USS Network: message integrity, non-repudiation, and message authentication. To achieve these three goals, UTM takes the approach of having USSs digitally sign the messages that they send. A JWS is used to achieve this message signing.

5.12.1 Header

The JOSE (JSON Object Signing and Encryption) header of the JWS used in Message Signing will have the following required fields (per JWS spec and UTM needs):

- alg: The algorithm used to sign the JWS, which is required to be a registered JSON Web Algorithm (JWA). The discussion on JWS above is applicable here, with the same set of allowed algorithms. See RFC 7518 for details on JWA.
- typ: The type is used by JWS applications to declare the media type of this complete JWS. In this case, it must be “JOSE” per RFC 7515.
- x5u: The "x5u" (X.509 URL) Header Parameter is a URI [RFC 3986] that refers to a resource for the X.509 public key certificate or certificate chain [RFC 5280] corresponding to the key used to digitally sign the JWS.
- kid: The Key ID of the JWK used to sign. Note that the jku reference will provide a set of keys with unique key identifiers. Those identifiers will be searched to match against this kid.
- x5t#S256: A thumbprint of the X.509 cert used to sign.
- crit: The critical field will not be used initially, but implementers should be aware that it may be needed as we work through use cases and security concerns. This field allows for defining additional required fields in this header that MUST be understood and processed that are not defined in the JWS specification.

5.12.2 Payload

The payload of the JWS will be the same character string as the HTTP body in base64url encoding. For example this may be a complete Operation instance, base64url encoded, if the data exchange occurs on the /operations endpoint.

5.12.3 Signature

The signature is calculated on the payload based on the information in the header. See the JWS section and the JWS spec for more insight.

5.12.4 Supplying the Signature

The JWT is sent as an HTTP header of a regular Application Programming Interface (API) call. The field name is as follows:

- `x-utm-message-signature`: A JWS signature of the HTTP body (details below).

5.12.5 Example

Assume a USS needs to send the following Position to another USS:


```
{
  "altitude_gps_wgs84_ft": 1111.111,
  "altitude_num_gps_satellites": 22,
  "air_speed_source": "MEASURED",
  "enroute_positions_id": "d10cd900-086f-43c7-9d6c-BAADCAFEF00D",
  "gufi": "00000000-0000-4444-8888-000000000000",
  "hdop_gps": 1.117,
  "time_measured": "2016-10-04T09:15:40.727Z",
  "time_sent": "2016-10-04T09:15:41.491Z",
  "track_bearing": 33.44,
  "track_bearing_reference": "MAGNETIC_NORTH",
  "track_bearing_uom": "DEG",
  "track_ground_speed_kn": 33.33,
  "uss_name": "uss.provider321.net",
  "vdop_gps": 0.932,
  "location": {
    "type": "Point",
    "coordinates": [
      -122.05635935068132,
      37.41436490284069
    ]
  }
}
```


That Position instance in Base64URL encoding (as required for JWS creation) would be:


```
eyJhbHRpdHVkZV9ncHNfd2dzODRfZnQiOjExMTEuMTExLCJhbHRpdHVkZV9udW1fZ3BzX3NhdGVsbGl0ZXMiOjIyLCJhaXJfc3BlZWRfc291cmNlIjoiTUVBU1VSRUQiLCJlbnJvdXRlX3Bvc2l0aW9uc19pZCI6ImQxMGNkOTAwLTA4NmYtNDNjNy05ZDZjLUJBQURDQUZFRjAwRCIsImd1ZmkiOiIwMDAwMDAwMC0wMDAwLTQ0NDQtODg4OC0wMDAwMDAwMDAwMDAiLCJoZG9wX2dwcyI6MS4xMTcsInRpbWVfbWVhc3VyZWQiOiIyMDE2LTEwLTA0VDA5OjE1OjQwLjcyN1oiLCJ0aW1lX3NlbnQiOiIyMDE2LTEwLTA0VDA5OjE1OjQxLjQ5MVoiLCJ0cmFja19iZWFyaW5nIjozMy40NCwidHJhY2tfYmVhcmluZ19yZWZlcmVuY2UiOiJNQUdORVRJQ19OT1JUSCIsInRyYWNrX2JlYXJpbmdfdW9tIjoiREVHIiwidHJhY2tfZ3JvdW5kX3NwZWVkX2tuIjozMy4zMywidXNzX25hbWUiOiJ1c3MucHJvdmlkZXIzMjEubmV0IiwidmRvcF9ncHMiOjAuOTMyLCJsb2NhdGlvbiI6eyJ0eXBlIjoiUG9pbnQiLCJjb29yZGluYXRlcyI6Wy0xMjIuMDU2MzU5MzUwNjgxMzIsMzcuNDE0MzY0OTAyODQwNjldfX0
```

To create a header for this JWS, we would need to determine appropriate values for the fields described above. Using notional values, the JOSE header could be:

```
{
  "alg": "RS256",
  "typ": "JOSE",
  "kid": "e337ac10-0ad9-4193-8cbd-453087ece360",
  "x5u":
"uss.provider321.net/.well-known/uas-traffic-management/my_cert.der",
  "x5t#S256":
"77ad11c0fbb461eabe35c1dde60c5a871ae80b5c019029b4cdb2493f2f2813db",
  "crit": []
}
```

This is a good point to remind readers of the USS Naming requirements with this concrete example. These requirements link names between the JWS header, JWS payload, access_token, and previously persisted data.

That JOSE header in Base64URL encoding (as required for JWS creation) would be:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiJlMzM3YWMxMC0wYWQ5LTQxOTMtOGN
iZC00NTMwODdlY2UzNjAiLCJ4NXUiOiJ1c3MucHJvdmlkZXIzMjEubmV0Ly53ZWxsLWtub3duL3
Vhcy10cmFmZmljLW1hbmFnZW1lbnQvbXlfY2VydC5kZXIiLCJ4NXQjUzI1NiI6Ijc3YWQxMWMwZ
mJiNDYxZWFiZTM1YzFkZGU2MGM1YTg3MWFlODBiNWMwMTkwMjliNGNkYjI0OTNmMmYyODEzZGIi
LCJjcml0IjpbXX0
```

To create the signature, concatenate the two Base64URL encodings with a '.' and apply the noted algorithm (via the `alg` in the JOSE header, in this case HMAC using SHA-256) using the private key associated with the noted public key (pointed to by the `x5u` in the JOSE header, having the thumbprint indicated by `x5t#S256`). It is good to emphasize how this protects the header as part of the signature. Any tampering of the header after signing would invalidate the signature.

Notionally that would look like:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your_private_256_bit_private_key
)
```

The signature resulting from this function would be then concatenated with the header and body using '.' in between to produce the final JWS (highlights in this sentence match colors in JWS below):

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiJlMzM3YWMxMC0wYWQ5LTQxOTMtOGN
iZC00NTMwODdlY2UzNjAiLCJ4NXUiOiJ1c3MucHJvdmlkZXIzMjEubmV0Ly53ZWxsLWtub3duL3
Vhcy10cmFmZmljLW1hbmFnZW1lbnQvbXlfY2VydC5kZXIiLCJ4NXQjUzI1NiI6Ijc3YWQxMWMwZ
mJiNDYxZWFiZTM1YzFkZGU2MGM1YTg3MWFlODBiNWMwMTkwMjliNGNkYjI0OTNmMmYyODEzZGIi
LCJjcml0IjpbXX0.eyJhbHRpdHVkZV9ncHNfd2dzODRfZnQiOjExMTEuMTExLCJhbHRpdHVkZV9
udW1fZ3BzX3NhdGVsbGl0ZXMiOjIyLCJhaXJfc3BlZWRfc291cmNlIjoiTUVBU1VSRUQiLCJlbn
JvdXRlX3Bvc2l0aW9uc19pZCI6ImQxMGNkOTAwLTA4NmYtNDNjNy05ZDZjLUJBQURDQUZFRjAwR
CIsImd1ZmkiOiIwMDAwMDAwMC0wMDAwLTQ0NDQtODg4OC0wMDAwMDAwMDAwMDAiLCJoZG9wX2dw
cyI6MS4xMTcsInRpbWVfbWVhc3VyZWQiOiIyMDE2LTEwLTA0VDA5OjE1OjQwLjcyN1oiLCJ0aW1
lX3NlbnQiOiIyMDE2LTEwLTA0VDA5OjE1OjQxLjQ5MVoiLCJ0cmFja19iZWFyaW5nIjozMy40NC
widHJhY2tfYmVhcmluZ19yZWZlcmVuY2UiOiJNQUdORVRJQ19OT1JUSCIsInRyYWNrX2JlYXJpb
mdfdW9tIjoiREVHIiwidHJhY2tfZ3JvdW5kX3NwZWVkX2tuIjozMy4zMywidXNzX25hbWUiOiJ1
c3MucHJvdmlkZXIzMjEubmV0IiwidmRvcF9ncHMiOjAuOTMyLCJsb2NhdGlvbiI6eyJ0eXBlIjo
iUG9pbnQiLCJjb29yZGluYXRlcyI6Wy0xMjIuMDU2MzU5MzUwNjgxMzIsMzcuNDE0MzY0OTAyOD
QwNjldfX0.2pUJQJXJFepenvqMM1DedCFtFGeECAPShrLkPZu-pr8
```

Through testing with multiple implementation partners, it was discovered that including the entire JWS in an HTTP header resulted in performance and stability issues on the USS Network. Essentially, the JWS is typically too large to reasonably contain in an HTTP header since there are several weblayers that have strict limits (for good reasons) on the maximum size of an HTTP header, despite the explicit lack of a standardized limit. Thus, the preferred approach would be to include just the header and the signature as an HTTP header, with the JWS payload being exchanged for an empty string. The result of that operation would be included as the HTTP header `x-utm-message-signature`[5] during a PUT request to the other USSs' `/positions` endpoint as illustrated here:

```
x-utm-message-signature:"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpPU0UiLCJraWQiOiJlMz
M3YWMxMC0wYWQ5LTQxOTMtOGNiZC00NTMwODdlY2UzNjAiLCJ4NXUiOiJ1c3MucHJvdmlkZXIzM
jEubmV0Ly53ZWxsLWtub3duL3Vhcy10cmFmZmljLW1hbmFnZW1lbnQvbXlfY2VydC5kZXIiLCJ4
NXQjUzI1NiI6Ijc3YWQxMWMwZmJiNDYxZWFiZTM1YzFkZGU2MGM1YTg3MWFlODBiNWMwMTkwMjl
iNGNkYjI0OTNmMmYyODEzZGIiLCJjcml0IjpbXX0..2pUJQJXJFepenvqMM1DedCFtFGeECAPSh
rLkPZu-pr8"
```

[5] Note that RFC 7230 states that HTTP header labels are case-insensitive. We use all lower-case as a stylistic choice still compatible with the HTTP standard.

Note the two dots separating the JWS header from the JWS signature since the JWS payload is technically still included as an empty string.

5.12.6 Discussion

Message signing is a key mitigation against several threats in the UTM System. At a high level, these threats include, but are not limited to:

- Repudiation of messages
- Message re-use
- Message tampering en route or at the endpoint
- Message forgery

The header is of a relatively predictable and manageable size. The only elements that may add length to the HTTP header are the x5u field and any potential extensions in the future since all other fields including the JWS signature are of known size.

The approach presented here leverages a design choice described in RFC 7515. For clarity, Appendix F from that document is copied here:

> Appendix F. Detached Content
>
> In some contexts, it is useful to integrity-protect content that is not itself contained in a JWS. One way to do this is to create a JWS in the normal fashion using a representation of the content as the payload but then delete the payload representation from the JWS and send this modified object to the recipient rather than the JWS. When using the JWS Compact Serialization, the deletion is accomplished by replacing the second field (which contains BASE64URL(JWS Payload)) value with the empty string; when using the JWS JSON Serialization, the deletion is accomplished by deleting the "payload" member. This method assumes that the recipient can reconstruct the exact payload used in the JWS. To use the modified object, the recipient reconstructs the JWS by re-inserting the payload representation into the modified object and uses the resulting JWS in the usual manner. Note that this method needs no support from JWS libraries, as applications can use this method by modifying the inputs and outputs of standard JWS libraries.

There are difficulties and potential pitfalls with such an approach related to the separation of payload and signature, but those issues are outweighed by the benefits of leveraging existing

JWS libraries, REST architectures and libraries, and the ability to use the signature separately from the payload for other purposes within the UTM System.

5.13. Application Programming Interface (API) Documentation

There are many details provided in this document on UFAA. Most details that a development team would need are centered around JSON data schemas and endpoints. There are a growing number of ways to detail such API-centered systems. For UFAA, the OpenAPI Specification version 2.0 is used to document the schemas and endpoints and is a format that is human-readable and machine-parseable. The ability of software to use the API documentation as input allows for the automatic checking of semantics and the automatic generation of code. The API documentation should be made available publicly, as the security of UFAA does not rely on obscurity in any way.

The API is important to know what data to send to what endpoint and what data to expect in return, but it does not describe when endpoints should be called and what calls should follow other calls. For those purposes, UFAA will rely on this document when possible, and sometimes the UTM USS Specification when appropriate. Between these two documents, USS implementers should have sufficient insight to USS requirements to build a USS to the correct specifications.

An additional element that is documented in the API is the authorization scope set for each endpoint. The scopes are taken directly from the Role-Based Access Control scheme for UTM, which is described in the appendices.

5.14. Discovery of Authorization Server

It is assumed that stakeholders within the UTM System will know the host address of the authorization server. Further details of the server are provided in a .well-known location on that server as specified in RFC 8414. This well-known location is "/.well-known/oauth-authorization-server". The implemented schema of that JSON file is available in the authorization server API documentation. Further details are not provided in this document.

5.15. OAuth 2.0 Token Request

OAuth 2.0 offers various flows for providing authorization. UFAA currently only uses the client credentials grant flow. This is a simplified flow due to the fact that the client (the entity making the resource requests) and the resource owner (the entity that is capable of granting access to resource) are the same entity (see Roles in RFC 6749). The diagram as provided in the specification is reproduced here:

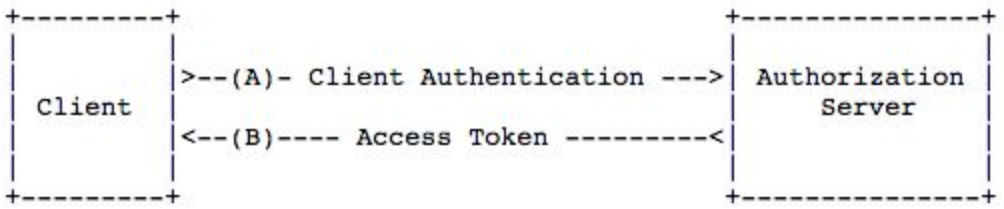


Figure 2. Client and Authorization Server exchange.

The client authentication is completed via Message Signing. The access token is a bearer token represented as a JWT.

For this flow, the client is required to provide a grant_type and scope as an `application/x-www-form-urlencoded` HTTP payload. The grant_type value must be "client_credentials" and exactly one scope MUST be provided and represents the scope desired in the access token. If this field is omitted or contains more than one scope, the request will be rejected by the authorization server. The scopes are known to the client through the appropriate API documentation or other UTM documentation provided to USS implementers and is out-of-scope[6] of this document. The client also provides a "client_id" with the value of its registered USS name. This is the same name that bridges the certificates and would appear in the 'sub' field of the access_token provided by this request. The client_id field will be used by the authorization server as a hint and sanity check as to the identity of the client. This will likely be helpful in cases where there may be many valid DNS names in the certificates. The client_id is described in the client password section of RFC 6749, but is used in the context described in this paragraph.

The authorization server will perform the following checks upon receiving a token request:

1. All required fields, properly formatted in HTTP body.
2. JWS provided in header is properly signed with a valid cert.
3. client_id is contained as a subject alternative name in the cert used to sign JWS.
4. client_id represents a known USS.
5. That USS has the appropriate role to request the given scope.

If any of these checks fails, the access_token will not be granted by the authorization server.

Note that scope is restricted to single values versus an array of values. This is in support of the Principle of Least Privilege, which is a feature of UFAA and supports mitigations against several threat models.

[6] Pun intended.

5.15.1. Signing for Authentication

The Message Signature would sign the HTTP payload (i.e. the `application/x-www-form-urlencoded` data described above) in the same manner as described in the Message Signature section of this document. This signature would be the authentication method employed by the Authorization Server. The only exception is that instead of a JSON message in the body of the JWS, the data would be `x-www-form-urlencoded` data as provided in the HTTP body. The signature would be provided as an HTTP header with the same name as previously described: `x-utm-message-signature`[7].

A successful request will result in the Authorization Server sending the client a JSON response as described in RFC 6749. Specifically, the response will contain the following fields:

- access_token: The requested token.
- token_type: The token type will always be "bearer" in UFAA.
- expires_in: Number of seconds until the 'exp' value in the token is reached.
- scope: The scope of the token, must be equal to the requested scope, else the request for the token should have been rejected.

The authorization server must be configured to adhere to the OAuth 2.0 specification in regards to error responses.

6. Exercising UFAA

In this section, sequence diagrams are used to illustrate UFAA data exchanges based on the concepts discussed in the sections above.

[7] Note: this is a design choice that should be discussed further. We could use the Authorization HTTP header, but no registered schemes fit this approach. It is reasonable and there is precedent for defining an application or domain-specific scheme to make use of the Authorization header. A potential value could be "UTM-JWS" for example. This would result in a header such as this (without the quotes): "Authorization: UTM-JWS ej34f…<the JWS>". Unclear which is better: using the standard header with a unique scheme or using a UTM-specific header while maintaining consistency with the UTM data exchanges.

6.1. Token Request

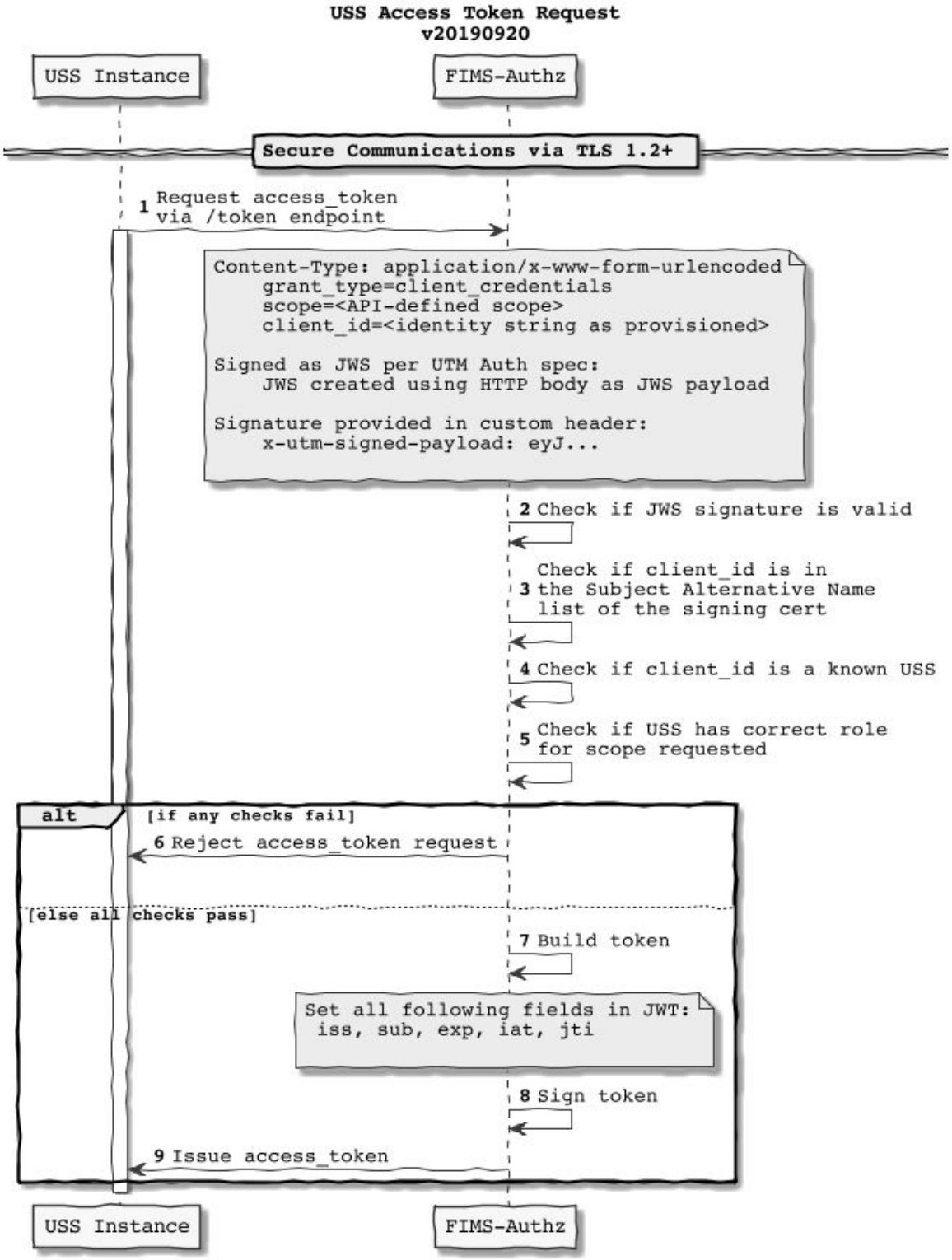


Figure 3. Detailed USS to FIMS Authorization server token request.

6.2. Certificate Fetch

Note that certificates will also be available from Certificate Authorities on behalf of a USS.

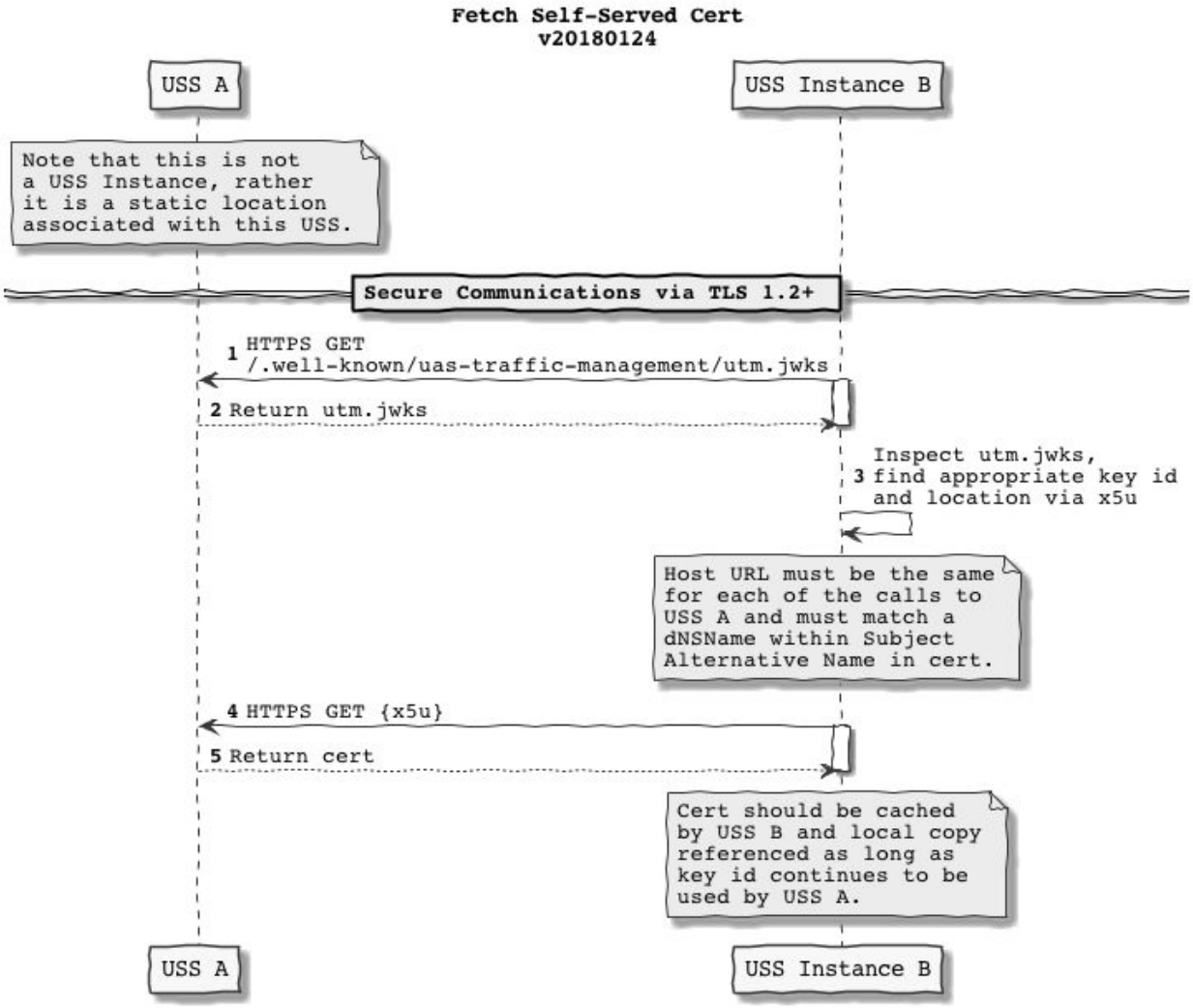


Figure 4. Fetching certificate from another USS.

6.3. Application-level Security Checks

With all of the infrastructure and mechanisms described throughout this document in place, it is now possible to describe specific checks that the USSs will be required to perform to maintain the security of the USS Network and UTM as a whole.

6.3.1. Name checks

A USS receiving a data payload via an API call must check that the sub claim in the access token matches the uss_name in the data payload. This protects against several attacks including token re-use and USS spoofing. In addition, a USS receiving a data payload via an API call must check that the uss_name matches a name in the certificate used to sign the payload. This check prevents elevation of privilege attacks, amongst other potential threats. For further insight into abuse and misuse within an OAuth 2.0 implementation, RFC 6819 is an excellent resource and was a key guiding document for the development of UFAA.

6.3.1.1. Implications of Relaxing Payload to Access Token

If this name check is not required, then a USS can re-use an access token that it obtained from a valid data exchange with a USS. For example assume USS A uses the HTTP PUT method to send data to USS B with a valid access token (call it Token X). USS B may store Token X. Then USS B may use Token X to access an endpoint on USS C. If USS C does not check the sub claim in the access token for a match against the supplied data, then USS B is provided unauthorized access to a resource. This can be an example of escalation of privilege if the endpoint USS B is accessing with Token X is not typically accessible by USS B's role(s). Escalation of privilege is one of the most dangerous vulnerabilities in any computer system or network. See Figure 5 as an illustration of this example.

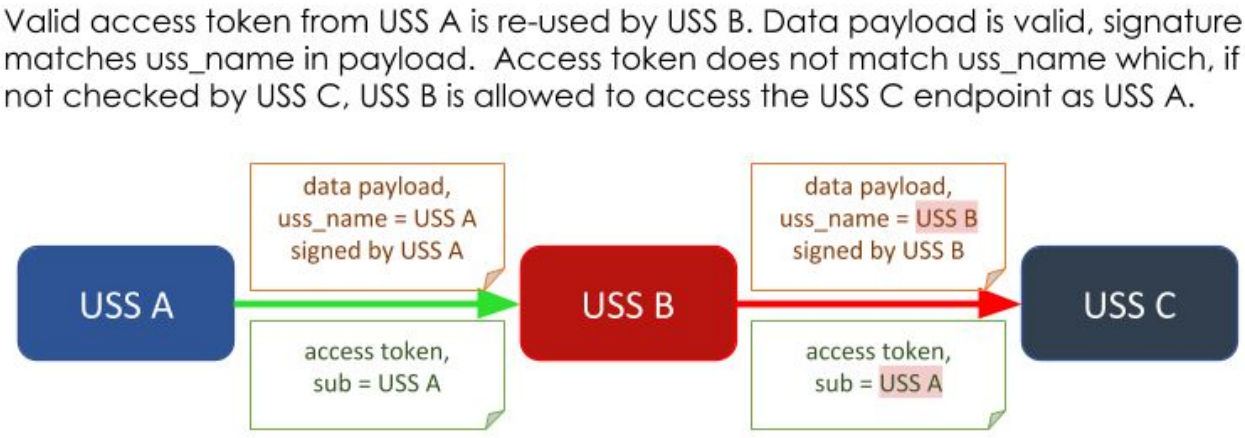


Figure 5. Checking names between access token and payload.

6.3.1.2. Implications of Relaxing Payload to Signature Check

If the check between the payload data and the UFAA Certificate (used to create the JWS) is not required, then a USS can send data to another USS with a different identity. For example assume USS A uses the HTTP PUT method to send data to USS B with a valid access token (call it Token X). USS B may store Token X. Then USS B may use Token X to access an

endpoint on USS C. If USS B creates a valid JWS of the exchanged data and includes USS A as the uss_name in the data payload, then USS C would recognize the access token as matching the data payload (both reference USS A). USS C would also recognize the JWS as being valid, though signed by USS B. If the check that the UFAA Certificate does not appropriately include USS A's name, then USS B has successfully spoofed data to USS C. This undermines trust in the entire UFAA. See Figure 6 as an illustration of this example.

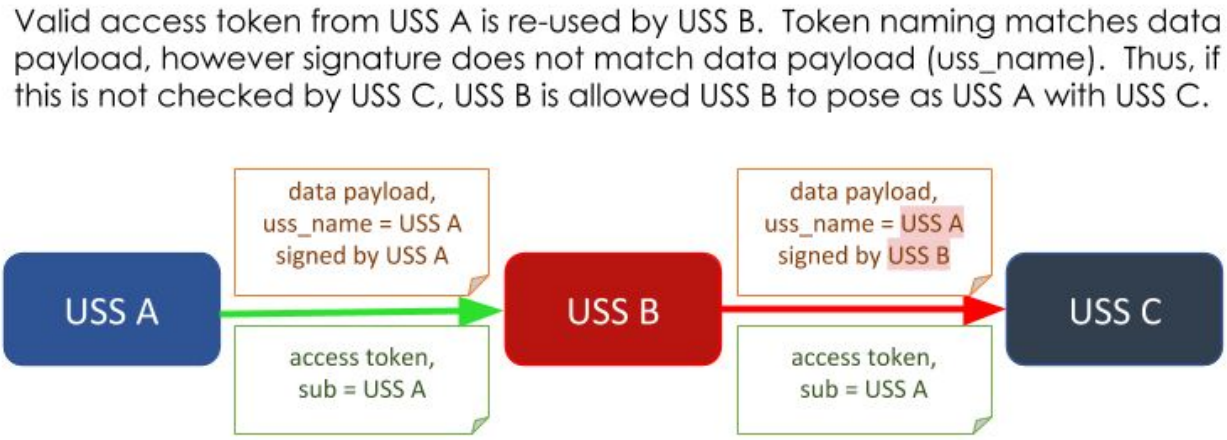


Figure 6. Checking names between signature and HTTP payload.

6.3.2. Access Token Time Checks

A USS receiving any request requiring an access token must ensure that the times are valid. The timeout of tokens is a major mitigation against token abuse. Since there may be issues with clock synchronization, tokens should be generated with time limits that are well clear of reasonable uncertainty in clocks across systems. Else, an assumption could be made that clock synchronization within the USS Network is a solved problem (note it is NOT as of this writing).

6.4. Potential Algorithm for USS Data Exchange

A formal requirements document based on the information and analysis provided in this document may be produced by NASA in the future. Meanwhile, by synthesizing the information provided herein, some requirements can be inferred. An algorithm can subsequently be defined for validating data requests within the USS Network. A nominal version of such an algorithm is provided below and will be referred to as Algorithm 1. Note this flow assumes that data are PUT or POSTed to a USSs endpoint. Similar steps and security arguments can be documented for GET requests from server, but the GET flow will not be documented further here.

1. **Validate certificates.** Note that this initial "step" is inferred for all uses of certificates and is not a separate "active" step for every data exchange. See section X.X for discussion on Certificate Authorities and which CAs should be recognized.

2. **Check Access Token.** When a server receives a data request, the initial step should be to check the access token. This is often handled by layers prior to the application layer. For example many API gateways will provide token validation services. As such, the following requirements are assumed to be self-evident in terms of standard Internet security practices and not unique to UFAA. Regardless, they will be noted as requirements of the system, but not discussed further.
    a. Access token is validly signed by recognized Authorization Server
    b. Access token contains the required claims
    c. Access token contains valid claims related to time
    d. Access token contains the appropriate scope for the endpoint
3. **Pass access token claims to the application layer.** Most gateway layers offer a facility for passing access token claims to the application layer. This is a necessary step to allow further security checks.
4. **Recreate JWS.** Using information from Section 5.12.6 and RFC 7515 use the `x-utm-message-signature` header and the HTTP JSON body to recreate the JWS as originally generated by the client server.
5. **Check JWS.** Using RFC 7515, check that the JWS is valid.
6. **Check names.** This is the critical step within UFAA. These checks are unique to UFAA as they require checking against elements in the data payload with other cryptographically secure elements in the data exchange.
    a. Check that uss_name == sub claim. This check is discussed in Section 6.3.1. The relaxation of this check is further explored in Section 6.3.1.1.
    b. Check that uss_name == (a subject-alternative name within the UFAA Certificate). This check is discussed in Section 6.3.1. The relaxation of this check is further explored in Section 6.3.1.2.

Appendices

A1. Open Issues

As discussed in the opening of this document, there were open issues to be discussed and developed further before this document could be formalized. In this section, a list and description is provided for the several known issues. This list should not be assumed to be complete. Only through further collaborative development with varied stakeholders will a reasonable maturity level be reached for this document. This current document should be taken as a starting point for discussions on the road to standardization of authentication and authorization with a future, operational UTM System.

A1.1. Message Signing

In the development of the Message Signing section, several concepts were explored, but no specific, established standard was easily applied to UTM. Several draft RFCs exist that point the way to potential solutions. The approach presented here is a synthesis of that publicly-available information, coupled with key discussions with security professionals and UTM domain experts. The separation of the signature from the payload can be problematic, but this is currently balanced by the ability to use off-the-shelf tooling for JSON payloads in the HTTP body and limiting the size of the HTTP headers.

If security were the only concern, the correct approach to exchanging data would be to use JWS-over-HTTP REST approach. In this hypothetical approach, USSs would exchange JWS in the HTTP body rather than JSON. This would easily allow for multiple signers of the same payload (as JWS allows for this) and would prevent splitting the signature from the signed payload (which has known issues related to reconstruction of the JWS). However this approach is not supported by RESTful development tools and libraries. In general the most accessible tools rely on the philosophy of exchanging JSON payloads via HTTP, which then facilitates consistent, reliable, easily-maintained software for serialization/deserialization, validation, persistence, and several other qualities/features. Losing all of those benefits currently outweigh the security gains of exchanging JWS in the HTTP body. If the security cost equation changes for any reason, this approach needs to be revisited.

A1.2. Authentication to the Authorization Server

The approach to authenticating a USS to the authorization server presented above was the result of detailed examination within NASA (see the HTTP signing discussion above, for example), however the approach was not implemented nor tested. This issue will need further discussion and testing before formalization. A change in approach based on broader stakeholder feedback would not be surprising, but should be well-justified.

OpenID Connect 1.0 has approaches for authenticating to an authorization server. However, this is mainly for retrieving an access token that can be used explicitly for authentication purposes with other systems. The approach to authenticating to the authorization server could still be leveraged, if for no other reason than it is well-defined, has implementations in the wild, and would serve the needs of UTM. A minor downside is additional claims in the access token that is provided by the authorization server that serve no real purpose within UTM. This approach could also levy additional requirements on the authorization server (like additional OAuth 2.0 flows), which may not be desired for simplicity's sake.

A1.3. Use of .well-known

It may not be necessary to self-host CA-anchored certs. If this is true, then the entire .well-known concept may have limited value within UFAA.

A1.4. Use of `aud` Claim

The use of the `aud` claim in a bearer token offers a great deal of additional security, protecting against several token re-use and other attacks. The cost is increased token management on the USSs within the UTM System since USSs would need to keep track of different tokens per endpoint per other USS. This may be a reasonable cost. Rather than introduce additional complexity, we decided against the use of the aud claim as we felt we had good coverage of our documented threats through other means. However, as this analysis is reviewed and formalized by others, it would not be surprising to have the `aud` claim highlighted as a key security feature.

A1.5. Implementation of JWK

The section on JWK requires significant further review. This was not implemented nor tested in NASA's UTM Project. There may be bugs in the descriptions and there may be significant changes required in its use, especially in light of potential changes in .well-known.

A1.6. PKI

The use of public, trusted CAs may have vulnerabilities (open to attacks on the Internet as a whole) and acceptability issues (some security-minded stakeholders may object to relying on general, non-governmental CAs). The upside is the ease of adoption and use. Further analysis of this approach is warranted. The outcome may point to the necessity of a different PKI approach.

In addition, there may be requirements for USSs to interact with networks rooted at a different trust anchor, such as those within the US Federal Government. If this were the case, then USSs would need to be evaluated to ensure they could make such exchanges since they are often limited to only trusting traditional Internet-only CAs. This issue should be tracked as UTM implementations mature and requirements become clearer.

A1.7. Varying Token Timeouts

The research implementation of the FIMS Authorization server currently uses a blanket value for defining the exp claim of an access token. For certain scopes, it may make sense to have different exp values, thus extending or shortening the valid time for those tokens. NASA did not perform any analysis to determine any of these varying timings, but it is likely that this approach would be a reasonable mitigation against token abuse for high-value access tokens that could cause problems within the USS Network, if they were compromised.

A1.8. Encrypted Messages

While TLS provides transport-level encryption of data exchanges, there may be future use cases wherein application-level encryption is also required. In such cases, the base standard for JWE should be considered a primary option. The main reasons for this recommendation include the fact that JWE is a recognized standard, there are several JWE library implementations that currently exist supporting several programming languages, and JWE security issues are monitored by the security community. Further analysis of the current recommended approach to message exchange detailed in this document should be performed to determine how or if the approach should change to support application-level encryption via JWE. It is the authors' perspective that exchange of JWE could be relatively easily incorporated into the approach documented herein, but details will not be provided. Some considerations include deciding which plaintext data to encrypt and how the JWE should be exchanged (as the body of the HTTP request or as an element in a new JSON schema, for example).

A1.9. Order of Checks

A lesson learned from NASA Technical Capability Level 4 testing was that the order of the various checks matters. At a minimum, the order of the checks has an impact on the HTTP responses that a server would return to an invalid data request. For example, a bad access token may result in an HTTP 403 status code response, but if that data request also had malformed JSON, it may elicit an HTTP 400 status code response. The order of these checks was not specified in the TCL4 version of the USS Specification, which led to issues with checking out the USS participants. For example, some tests of the USSs assumed a certain HTTP response, but when a different one was returned, that test would be marked as "failed" for that USS, although the USS had just implemented rule checking in a different order and was within the published USS Specification.

Thus, issues related to the order of security and business rule checks need to be specified, or USSs need to be robust to differing implementations for an operational UTM System.

A1.10. Additional Uses of Signatures

This document focuses on the use of certificates and signatures for USS-to-USS communications. It may be appropriate, depending on the operational architecture of the UTM System, to use the security elements elsewhere as well. A key example is within the discovery process for USSs. Discovery is the process by which USSs gather information about how and when to contact other USSs. For consistent and secure communications within UTM, signatures of certain data may be relevant in the discovery process. This does not preclude the use of these certificates and signatures in other parts of UTM, but discovery is a key example.

A1.11. JWT Best Current Practices

Since the work documented in this paper began, a new RFC draft Best Current Practices (BCP) document entitled “JSON Web Token Best Current Practices” has been published. In that document there are several vulnerabilities and mitigations that are relevant to JWT within UFAA. When the work herein is formalized into a standard or set of operational rules, the details of that document need to be incorporated.

As an example, the set of algorithms that are allowed in UFAA may need to be pruned and may need to differ from the algorithms required by JWA to meet the security needs of UFAA. Specifically, since Hashed Message Authentication Codes (HMAC) are not used in UFAA, all HMAC-based algorithms may need to be excluded from use in UFAA despite being required per the JWA specification. This is due to a known vulnerability wherein attackers can take advantage of the ‘alg’ claim in the JOSE header by indicating one approach to signing, while actually signing with another. This is only possible due to the acceptance of both HMAC and RSA-based signing algorithms. Another key mitigation recommended in the BCP document is the use of the ‘aud’ claim as discussed here in Section A1.4. In general, any published updates to JWT guidance (or any relevant security practices) must be recognized with this specification updated appropriately.

A2. Role-Based Access Control

A2.1. RBAC Overview

UTM implements Role-Based Access Control (RBAC). RBAC is a method for defining authorizations in a system. Roles are defined with certain permissions and then users are assigned to one or more roles. They inherit all the permissions for their roles. There are many intricacies that need to be addressed for successful implementation of an RBAC scheme, very few of which will be addressed in the current draft of this document. The definitions used for describing the implementation of RBAC are taken from the INCITS 359 standard.

RBAC is only one way in which authorization can be managed in a system. Another popular method is Attribute-based access control (ABAC). With ABAC, users obtain authorizations based on attributes of their identity and those of the resources. ABAC is argued as being better at fine-grained controls, but it takes much more effort to manage. RBAC is more straightforward to manage, but may take more upfront planning to be truly effective. Other methods will not be detailed here. NIST maintains documentation on RBAC.

There are five key definitions used in RBAC as recommended by INCITS 359. These are Operations, Objects, Permissions, Roles, and Users.

A2.2. Operations

The allowed operations within the UTM authorization framework are simplified to read, write, and all. The write operation is assumed to have all the capabilities of the read operation, plus the ability to write data. The write operation is assumed to allow creation, update, and deletion of objects (when allowed by the concept). The 'all' operation implies a superset of the write operation's capabilities, but usually these will be equivalent in capability level. It is noted here for potential future use cases.

A2.3. Objects

The objects in the UTM authorization framework map to the API resources that are protected. Essentially these are the endpoints.

A2.4. Permissions/Scopes

Permissions are a subset of the cross product between the allowed operations and the set of objects. The subset used defines all of the allowed actions on any object. These permissions are translated into scopes within the UTM authorization framework based on OAuth 2.0. In general, we will use the OAuth terminology.

The set of scopes used in the UTM OAuth 2.0 implementation is listed in the table below. Scopes take the form of <namespace>_<operation>.<object> where "operation" is the type of permission (read/write, etc.) and "object" is the type of thing the action is performed upon. A "write" action implicitly grants read access to the subject as well as writing/updating. The namespace may aid in deconfliction and clarity of scopes. Terms conform to INCITS 359.

Scopes are annotated directly to API endpoints for clarity on the access requirements of each endpoint. Scopes within UTM are listed in Table A1.

Table A1: Scopes

Scope	Description
`utm.nasa.gov_read.fimsadmin`	Subject can read data internal to FIMS.
`utm.nasa.gov_write.fimsadmin`	Subject can read, create, and update data internal to FIMS.
`utm.nasa.gov_read.operation`	Subject can read operational data such as nominal Operation plans and Positions.
`utm.nasa.gov_write.operation`	Subject can read, create, and update operational data such as Operation plans and Positions.
`utm.nasa.gov_read.message`	Subject can read message data such as UTM Message and NegotiationMessage.
`utm.nasa.gov_write.message`	Subject can read, create, and update message data such as UTM Message and NegotiationMessage.
`utm.nasa.gov_read.publicsafety`	Subject can read operations that are designated public safety operations.
`utm.nasa.gov_write.publicsafety`	Subject can read, create, and update operations that are designated public safety operations.
`utm.nasa.gov_read.constraint`	Subject can read UTM constraint data such as UVRs.
`utm.nasa.gov_write.constraint`	Subject can read, create, and update UTM constraint data such as UVRs. This means that the Subject can define areas that restrict other operations.
`utm.nasa.gov_read.conflictmanagement`	Subject can read conflict management data.
`utm.nasa.gov_write.conflictmanagement`	Subject can read and write conflict management data.

A2.5. Roles

A role is defined by INCITS 359 as "a job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role." Roles within UTM are implemented as a set of permissions (scopes) and are summarized in Table A2. Some defined scopes in are not assigned to Roles since UTM did not yet have a need for the implementation.

Table A2: Roles

Scope	Description
`UTM_AUTHORITY`	A small set of trusted users who manage identities of UTM actors such as USSs. For example, the UTM Authority vets a USS and assigns it USS_BASIC role. Similarly, the UTM Authority vets a USS's capability for Public Safety, and adds the PUBLIC_SAFETY role to this USS in the FIMS Authorization server's data set.
`FIMS_AUTHZ`	This is the service role supporting the FIMS Authorization server. This role cannot create, modify or delete UTM Identity data, rather access is read-only.
`USS_BASIC`	A USS. The UTM Authority assigns role after out-of-band vetting. The Basic role has these scopes: utm.nasa.gov_write.operation, utm.nasa.gov_write.message, utm.nasa.gov_read.constraint, utm.nasa.gov_read.uvin, utm.nasa.gov_write.conflictmanagement
`USS_PUBLIC_SAFETY`	A USS with the capability to support public safety operations and queries. This role can be assigned only to a USS that has the USS_BASIC role. The UTM Authority assigns this role after out-of-band vetting. For example, a Public Safety USS supports a public safety query by fanning-out requests to other USSs.
`USS_PUBLIC_SAFETY_READ`	A USS with the capability to support public safety queries. This role can be assigned only to a USS that has the USS_BASIC role.

	For example a Public Safety USS queries other USSs to determine which USS is operating a particular vehicle.
`CONSTRAINT_MANAGER`	A USS with this role can publish airspace constraints such as UASVolumeReservations (UVRs) and other aeronautical data. The UTM Authority assigns this role after out-of-band vetting. For example because FIMS_OPS has the USS_CONSTRAINT_MANAGER role, thus is can publish UASVolumeReservations to the USS Network. Similarly a USS with this role publishes UVRs to the USS Network.

A2.6. Users/Subjects

A user is defined by INCITS 359 "as a human being. Although the concept of a user can be extended to include machines, networks, or intelligent autonomous agents, the definition is limited to a person in this document for simplicity reasons." Alternatively, the NIST SP 800-63 series defines a subject as "a person, organization, device, hardware, network, software, or service." Since "user" denotes more of a human-centric entity, UTM uses the term "subject" instead. This exchange of terms does not affect the INCITS 359 concept in any material way, and is intended to just clarify who/what can be assigned a role.

A2.7. Subject Assignments

Subject Assignments are the mapping from subjects to roles. This is a many-to-many mapping. This mapping is created upon onboarding of a new subject. That process will be documented elsewhere. For the present discussion, assume that this mapping exists.

A2.8. Permission Assignments

Permission Assignments are the mapping from roles to permissions. This is a many-to-many mapping. The following table includes the permission assignments for each role. Note that if two roles have the exactly same set of scopes assigned to them, then they should not be separate roles.

A2.9. Example

Figure A1 shows a mapping between three USSs (the subjects) to two roles within UTM. This mapping is the subject assignment. The roles are mapped via permission assignments to various permissions. The permissions have a one-to-one mapping to scopes as described in Table A1 above.

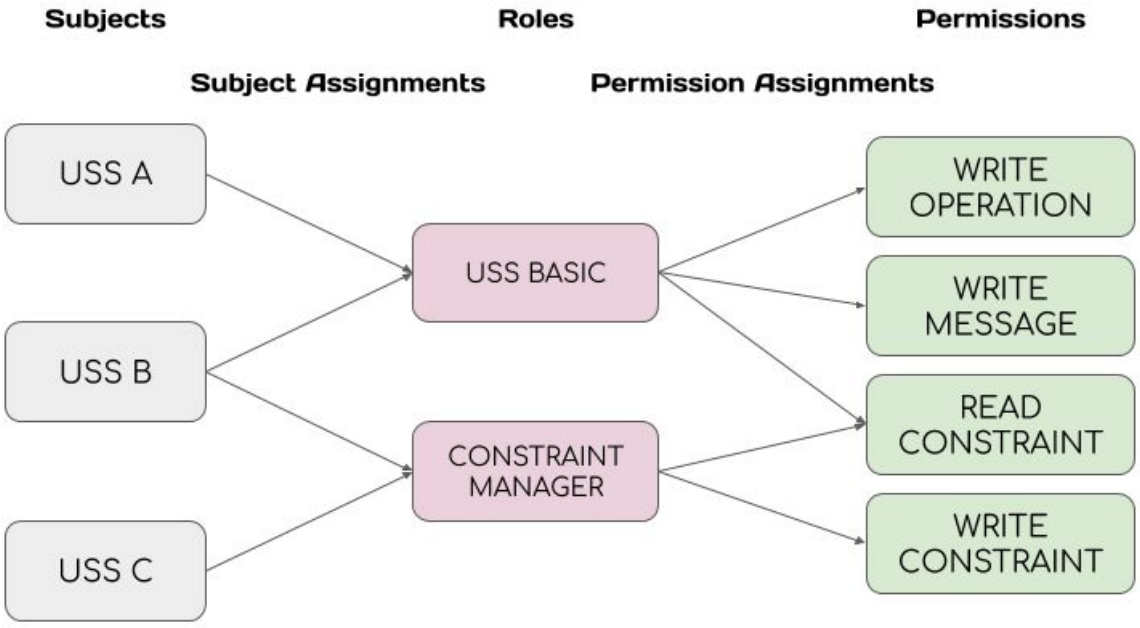


Figure A1. Example mapping of subjects to permissions in role-based access control.

In this notional example, USS A has the role “USS BASIC” which imparts the permissions WRITE OPERATION, WRITE MESSAGE, and READ CONSTRAINT. USS C has the role “CONSTRAINT MANAGER” which imparts the permissions READ CONSTRAINT and WRITE CONSTRAINT. USS B has both roles and thus has all of the listed permissions. In an operational system, there would be several more roles and permissions. Each role is awarded to a subject via some checkout or certification process.

A3. Design Decisions Considered

Several design decisions were considered for inclusion into this specification. In this section, some of these are captured for reference, completeness, and potential future discussion.

A3.1. HTTP Signing for Token Requests

HTTP Signing is an approach described in an expired RFC draft document entitled “A Method for Signing HTTP Requests for OAuth.” This approach creates a JWS that contains the access token as a data element along with other fields. The JWS may be signed using a symmetric or asymmetric approach, but in UFAA, an asymmetric approach is employed. A key pair supported by an X.509 certificate (specifically UFAA Certificate) described by a JWK that is stored in a .well-known location accessible only via a TLS 1.2+ connection.

A3.1.1. JWS Details in HTTP Signing

The JSON that would be signed in this approach is guided by Section 3 of the RFC draft document and contains the following elements:

- at: The complete access token provided by the authorization server. The value is null, but the field is still provided, when using HTTP Signing to authenticate to the authorization server for a token request.
- ts: A UNIX timestamp indicating the time of signing.
- m: The HTTP method used to make this request (GET, POST, PUT, etc.).
- u: The host component of the URL to where this request is sent.
- p: The path component of the URL to where this request is sent.
- b: base64URL encoded hash of the HTTP request body, calculated as the SHA256 of the byte array of the body. Note that this field needs further discussion on how to maintain some algorithmic flexibility.

The header of the JWS used in HTTP Signing will have the following required fields (per JWS spec and UTM needs):

- alg: The algorithm used to sign the JWS, which is required to be a registered JSON Web Algorithm (JWA). See RFC 7518 for details on JWA.
- jku: The JWK Set URL is a URI that indicates where the UFAA Certificate JWKS is located. This must be the .well-known location discussed above.
- kid: The Key ID of the JWK used to sign. Note that the jku reference will provide a set of keys with unique key ids. Those key ids will be searched to match against this kid.
- x5t#S256: A thumbprint of the X.509 cert used to sign.
- crit: The critical field will not be used initially, but implementers should be aware that it may be needed as we work through use cases and security concerns. This field allows for defining additional required fields in this header that MUST be understood and processed that are not defined in the JWS specification.

A3.1.2. Supplying the Signature

The signature is sent via the HTTP Authorization header with the string “PoP” prepended and separated by a space. The example from the specification is as follows:

```
GET /resource/foo
Authorization: PoP eyJ....omitted for brevity...
```

A3.1.3. Reasons for Exclusion from UFAA

UFAA moved away from the HTTP Signing approach for a couple of key reasons:

- HTTP Signing is based on a proposed standard that was never finalized or fielded in a known operational system.
- In operational systems it is easy for the original hash as supplied to be disconnected from the source that was hashed. This can be a problem for auditing and other use cases.

Thus, JWS was chosen to provide this functionality, which is an industry-accepted standard and which may be able to better leverage the infrastructure of each stakeholder that will need to be in place to handle access tokens. In addition, JWS provides well-documented mechanisms to allow multiple entities to sign the same data, which may be a useful feature for some UTM data exchanges.

A3.2. UTM-Specific Certificate Authority

As a research activity, NASA investigated the potential of a UTM-specific CA. Creating a domain-specific CA has too many security implications for practical implementation and maintenance. There are documented practices for becoming and maintaining a CA (see RFC 3647, RFC 6484, RFC 7382, and "Standards and Industry Regulations Applicable to Certification Authorities," for example) and it is unclear which entity or collection of entities related to UTM would be willing to follow those practices for the exclusive use of the UTM System. In addition, there are implications for Internet communication when certificates are rooted at a non-traditional CA. This line of inquiry ended for NASA in 2017.

A3.3. Mutual TLS

Default TLS is used to confidently identify the server which a client is contacting. The identity of the client contacting that server is not checked at the Transport Layer in default TLS. Mutual TLS is a standardized approach (see RFC 5246) to having both servers identify each other in a data exchange. While there are implementations of Mutual TLS in the wild, it is not regular practice on the Internet as a whole. The NASA UTM Project decided that it would be difficult to assure that all USSs were capable of supporting Mutual TLS, so the concept was not implemented. If Mutual TLS were deemed feasible for USSs, its use would have a major positive impact on many security aspects within UTM. However, if implemented, many of the recommended requirements within this document would need to be re-evaluated.

A3.4. Use of Access Tokens Outside USS Network.

It is useful to note that the access tokens provided by the recognized authorization server within UFAA can be used by other services outside of the USS Network. This is a positive side effect of the UFAA. A key use case is the use of Supplementary Data Service Providers (SDSPs) by USSs. It is likely that certain SDSPs would limit access to valid USSs. By accepting access tokens from the USS Network, an SDSP would have confidence that the client requesting access is a valid USS. This benefit would be gained without the SDSP needing to validate USSs or to continually check that the USS stays valid.

A3.5. Multiple Authorization Server Providers

The OAuth 2.0 specification allows for resource servers to accept access tokens from more than one authorization server. This approach, if applied to UTM, would introduce many interoperability complexities as those Authorization Servers would all need to either meet the

same specification or the resource servers would need to be configured to accept the formatting, content, and security features of each individual authorization server. In addition, by distributing the creation of access tokens in this way, there are additional complexities related to the assignment and maintenance of roles within the UTM System. The attack surface of the UTM System grows greatly by increasing the number of authorization servers beyond one. Thus, this design decision was rejected early on in UTM development.

A3.6 Token Checking and Revocation

As noted in Sections 5.10 and 5.11, there is a specification for token checking and token revocation, which an authorization server may implement. This is explicitly excluded from the UFAA due the increased attack surface this service exposes versus the security benefit gained. Most of the mitigations gained through the use of token checking and revocation by the authorization server are already realized through other mechanisms within UFAA. Further mitigations, such as the use of an ‘aud’ claim within the access_token, can further mitigate risks without exposing more issues through the implementation of a checking/revocation service.

Acronyms

ABAC	Attribute-based Access Control
ANSP	Air Navigation Service Provider
API	Application Programming Interface
ASN.1	Abstract Syntax Notation-1
CA	Certificate Authority
DER	Distinguished Encoding Rules
DNS	Domain Name System
DREAD	Damage, Reproducibility, Exploitability, Affected users, Discoverability
FIMS	Flight Information Management System
FIMS_Authz	FIMS Authorization Server
HMAC	Hashed Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IANA	Internet Assigned Number Authority
IETF	Internet Engineering Task Force
INCITS	InterNational Committee for Information Technology Standards
ISO	International Organization for Standardization
JOSE	Javascript Object Signing and Encryption
JSON	Javascript Object Notation
JWA	JSON Web Algorithms
JWE	JSON Web Encryption
JWK	JSON Web Key
JWKS	JSON Web Key Set
JWT	JSON Web Token
NAS	National Airspace System
NASA	National Aeronautics and Space Administration
NIST	National Institute of Standards and Technology
OAuth	Open Authorization
PKI	Public Key Infrastructure
PKIX	PKI (X.509)

PoP	Proof of Possession
RBAC	Role-based Access Control
REST	Representational State Transfer
RFC	Request for Comments
RSA	Rivest–Shamir–Adleman
SDSP	Supplemental Data Service Provider
SHA-256	Secure Hash Algorithm, 256-bit
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
sUAS	small UAS
TCL	Technical Capability Level
TLS	Transport Layer Security
UAS	Unmanned Aircraft System
UFAA	UTM Framework for Authentication and Authorization
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USS	UAS Service Supplier
UTF-8	Unicode Transformation Format - 8bit
UTM	UAS Traffic Management
UUID	Universally Unique Identifier
UUIDv4	UUID, version 4

References

IETF Request for Comment (RFC) Documents (in order of RFC #)

Chokhani, S., Ford, W., Sabett, R., Merrill, C., and S. Wu, "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework", **RFC 3647**, DOI 10.17487/RFC3647, November 2003, <https://www.rfc-editor.org/info/rfc3647>.

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, **RFC 3986**, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

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

Shirey, R., "Internet Security Glossary, Version 2", FYI 36, **RFC 4949**, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

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

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/info/rfc5280>.

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

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**, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

Kent, S., Kong, D., Seo, K., and R. Watro, "Certificate Policy (CP) for the Resource Public Key Infrastructure (RPKI)", BCP 173, **RFC 6484**, DOI 10.17487/RFC6484, February 2012, <https://www.rfc-editor.org/info/rfc6484>.

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

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

Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", **RFC 7230**, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

Kent, S., Kong, D., and K. Seo, "Template for a Certification Practice Statement (CPS) for the Resource PKI (RPKI)", BCP 173, **RFC 7382**, DOI 10.17487/RFC7382, April 2015, <https://www.rfc-editor.org/info/rfc7382>.

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

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

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

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

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

Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, **RFC 8259**, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

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/info/rfc8414>.

Richer, J., Ed., Bradley, J., and H. Tschofenig, “A Method for Signing HTTP Requests for OAuth”, **draft-ietf-oauth-signed-http-request-03**, Expired February 9, 2017, <https://tools.ietf.org/html/draft-ietf-oauth-signed-http-request-03>.

Sheffer, Y., Hardt, D., and M. Jones, “JSON Web Token Best Current Practices”, **draft-ietf-oauth-jwt-bcp-06**, Expires December 9, 2019, <https://tools.ietf.org/html/draft-ietf-oauth-jwt-bcp-06>.

National Institute of Standards and Technology (NIST) Documents

McKay, K., Cooper, D., “NIST Special Publication 800-52 Revision 2: Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations”, DOI 10.6028/NIST.SP.800-52r2, August 2019, <https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/final>.

Grassi, P., Garcia, M., and Fenton, J., “NIST Special Publication 800-63 Revision 3: Digital Identity Guidelines”, DOI 10.6028/NIST.SP.800-63-3, June 2017, <https://pages.nist.gov/800-63-3/sp800-63-3.html>.

Stouffer, K., Lightman, S., Pillitteri V., Abrams, M., Hahn, A., “NIST Special Publication 800-82 Revision 2: Guide to Industrial Control Systems (ICS) Security”, DOI 10.6028/NIST.SP.800-82r2, May 2015, <https://csrc.nist.gov/publications/detail/sp/800-82/rev-2/final>.

UTM Documents

Aweiss, A., et al., “Technical Capability Level 3 Unmanned Aircraft Systems (UAS) Traffic Management (UTM) Flight Demonstration”. 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC), September 2019.

Federal Aviation Administration, “Unmanned Aircraft System (UAS) Traffic Management (UTM): Concept of Operations v1.0: Foundational Principles, Roles and Responsibilities Use Cases and Operational Threads”, May 2018, <https://utm.arc.nasa.gov/docs/2018-UTM-ConOps-v1.0.pdf>.

Kopardekar, P., Rios, J., Prevot, T., Johnson, M., Jung, J., Robinson III, J., “Unmanned Aircraft System Traffic Management (UTM) Concept of Operations”, 16th AIAA Aviation Technology, Integration, and Operations Conference, AIAA, June 2016, <https://utm.arc.nasa.gov/docs/Kopardekar_2016-3292_ATIO.pdf>.

Rios, J., Smith I., Venkatesen P., Smith, D., Baskaran, V., Jurcak, S., Strauss, R., Iyer, S., Verma, P., “UTM UAS Service Supplier Development: Sprint 1 Toward Technical Capability Level 4”, NASA Technical Memorandum, NASA/TM-2018-220024, November 2018, <https://utm.arc.nasa.gov/docs/UTM_UAS_TCL4_Sprint1_Report.pdf>.

Rios, J., Smith I., Venkatesen P., Smith, D., Baskaran, V., Jurcak, S., Iyer, S., Verma, P., “UTM UAS Service Supplier Development: Sprint 2 Toward Technical Capability Level 4”, NASA

Technical Memorandum, NASA/TM-2018-220050, December 2018, <https://utm.arc.nasa.gov/docs/2018-UTM_UAS_TCL4_Sprint2_Report_v2.pdf>.

Other Documents

InterNational Committee for Information Technology Standards, “INCITS 359-2012 (R2017) Information Technology - Role Based Access Control”, December 2017, <https://standards.incits.org/apps/group_public/project/details.php?project_id=1906>.

International Telecommunication Union, “Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)”, ITU-T X.690, August 2015, <http://handle.itu.int/11.1002/1000/12483>.

Microsoft Corporation, “The STRIDE Threat Model”, Nov 2009, https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20).

N. Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, “OpenID Connect Core 1.0 incorporating errata set 1”, November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

OpenStack, “Security/OSSA-Metrics”, Accessed 7 Sept 2019, https://wiki.openstack.org/wiki/Security/OSSA-Metrics.

Shostack, A., “Experiences Threat Modeling at Microsoft”, 2008, <https://adam.shostack.org/modsec08/Shostack-ModSec08-Experiences-Threat-Modeling-At-Microsoft.pdf>.