Internet Engineering Task Force (IETF)
Request for Comments: 7165
Category: Informational
ISSN: 2070-1721

R. Barnes
Mozilla
April 2014

# Use Cases and Requirements for JSON Object Signing and Encryption (JOSE)

## Abstract

Many Internet applications have a need for object-based security mechanisms in addition to security mechanisms at the network layer or transport layer. For many years, the Cryptographic Message Syntax (CMS) has provided a binary secure object format based on ASN.1. Over time, binary object encodings such as ASN.1 have become less common than text-based encodings, such as the JavaScript Object Notation (JSON). This document defines a set of use cases and requirements for a secure object format encoded using JSON, drawn from a variety of application security mechanisms currently in development.

## Status of This Memo

## Copyright Notice

## Table of Contents

## 1. Introduction

Internet applications rest on the layered architecture of the Internet and take advantage of security mechanisms at all layers. Many applications rely primarily on channel-based security technologies such as IPsec and Transport Layer Security (TLS), which create a secure channel at the IP layer or transport layer over which application data can flow [RFC4301] [RFC5246]. These mechanisms, however, cannot provide end-to-end security in some cases. For example, in protocols with application-layer intermediaries, channel-based security protocols would protect messages from attackers between intermediaries, but not from the intermediaries themselves. These cases require object-based security technologies, which embed application data within a secure object that can be safely handled by untrusted entities.

The most well-known example of such a protocol today is the use of Secure/Multipurpose Internet Mail Extensions (S/MIME) protections within the email system [RFC5751] [RFC5322]. An email message typically passes through a series of intermediate Mail Transfer Agents (MTAs) en route to its destination. While these MTAs often apply channel-based security protections to their interactions (e.g., STARTTLS [RFC3207]), these protections do not prevent the MTAs from interfering with the message. In order to provide end-to-end security protections in the presence of untrusted MTAs, mail users can use S/MIME to embed message bodies in a secure object format that can provide confidentiality, integrity, and data origin authentication.

S/MIME is based on the Cryptographic Message Syntax (CMS) for secure objects [RFC5652]. CMS is defined using Abstract Syntax Notation 1 (ASN.1) and typically encoded using the ASN.1 Distinguished Encoding Rules (DER), which define a binary encoding of the protected message and associated parameters [ITU.X690.2002]. In recent years, usage of ASN.1 has decreased (along with other binary encodings for general objects), while more applications have come to rely on text-based formats such as the Extensible Markup Language (XML) [W3C.REC-xml] or the JavaScript Object Notation (JSON) [RFC7159].

Many current applications thus have much more robust support for processing objects in these text-based formats than ASN.1 objects; indeed, many lack the ability to process ASN.1 objects at all. To simplify the addition of object-based security features to these applications, the IETF JSON Object Signing and Encryption (JOSE) working group has been chartered to develop a secure object format based on JSON. While the basic requirements for this object format are straightforward -- namely, confidentiality and integrity mechanisms encoded in JSON -- discussions in the working group indicated that different applications hoping to use the formats defined by JOSE have different requirements. This document summarizes the use cases for JOSE envisioned by those potential applications and the resulting requirements for security mechanisms and object encodings.

Some systems that use XML have specified the use of XML-based security mechanisms for object security, namely XML Digital Signatures and XML Encryption [W3C.xmldsig-core] [W3C.xmlenc-core]. These mechanisms are used by several security token systems (e.g., Security Assertion Markup Language (SAML) [OASIS.saml-core-2.0-os], Web Services Federation [WS-Federation]), and the Common Alerting Protocol (CAP) emergency alerting format [CAP]. In practice, however, XML-based secure object formats introduce similar levels of complexity to ASN.1 (e.g., due to the need for XML canonicalization), so developers that lack the tools or motivation to handle ASN.1 aren't likely to use XML security either. This situation motivates the creation of a JSON-based secure object format that is simple enough to implement and deploy that it can be easily adopted by developers with minimal effort and tools.

## 2. Definitions

This document makes extensive use of standard security terminology [RFC4949]. In addition, because the use cases for JOSE and CMS are similar, we will sometimes make analogies to some CMS concepts [RFC5652].

The JOSE working group charter calls for the group to define three basic JSON object formats:

1. Integrity-protected object format

2. Confidentiality-protected object format

3. A format for expressing keys

In this document, we will refer to these as the "signed object format", the "encrypted object format", and the "key format", respectively. The JOSE working group items intended to describe these formats are JSON Web Signature [JWS], JSON Web Encryption [JWE], and JSON Web Key [JWK], respectively. Algorithms and algorithm identifiers used by JWS, JWE, and JWK are defined in JSON Web Algorithms [JWA].

In general, where there is no need to distinguish between asymmetric and symmetric operations, we will use the terms "signing", "signature", etc., to denote both true digital signatures involving asymmetric cryptography as well as Message Authentication Codes (MACs) using symmetric keys.

In the lifespan of a secure object, there are two basic roles, an entity that creates the object (e.g., encrypting or signing a payload) and an entity that uses the object (decrypting and verifying). We will refer to these roles as "sender" and "recipient", respectively. Note that while some requirements and use cases may refer to these as single entities, each object may have multiple entities in each role. For example, a message may be signed by multiple senders or decrypted by multiple recipients.

## 3. Basic Requirements

For the encrypted and signed object formats, the necessary protections will be created using appropriate cryptographic mechanisms: symmetric or asymmetric encryption for confidentiality and MACs or digital signatures for integrity protection. In both cases, it is necessary for the JOSE format to support both symmetric and asymmetric operations.

- The JOSE encrypted object format must support object encryption in the case where the sender and receiver share a symmetric key.

- The JOSE encrypted object format must support object encryption in the case where the sender has only a public key for the receiver.

- The JOSE signed object format must support integrity protection using MACs, for the case where the sender and receiver share only a symmetric key.

- The JOSE signed object format must support integrity protection using digital signatures, for the case where the receiver has only a public key for the sender.

In some applications, the key used to process a JOSE object is indicated by application context, instead of directly in the JOSE object. However, in order to avoid confusion, endpoints that lack the necessary context need to be able to recognize this and fail cleanly. Other than keys, JOSE objects do not support pre-negotiation; all cryptographic parameters must be expressed directly in the JOSE object.

- The JOSE signed and encrypted object formats must define the process by which an implementation recognizes whether it has the key required to process a given object, whether the key is specified by the object or by some out-of-band mechanism.

- Each algorithm used for JOSE must define which parameters are required to be present in a JOSE object using that algorithm.

In cases where two entities are going to be exchanging several JOSE objects, it might be helpful to pre-negotiate some parameters so that they do not have to be signaled in every JOSE object. However, so as not to confuse endpoints that do not support pre-negotiation, it is useful to signal when pre-negotiated parameters are in use in those cases.

- It should be possible to extend the base JOSE signed and encrypted object formats to indicate that pre-negotiated parameters are to be used to process the object. This extension should also provide a means of indicating which parameters are to be used.

The purpose of the key format is to provide the recipient with sufficient information to use the encoded key to process cryptographic messages. Thus, it is sometimes necessary to include additional parameters along with the bare key.

- The JOSE key format must enable inclusion of all algorithm parameters necessary to use the encoded key, including an identifier for the algorithm with which the key is used as well as any additional parameters required by the algorithm (e.g., elliptic curve parameters).

## 4. Requirements on Application Protocols

The JOSE secure object formats describe how cryptographic processing is done on secured content, ensuring that the recipient of an object is able to properly decrypt an encrypted object or verify a signature. In order to make use of JOSE, however, applications will need to specify several aspects of how JOSE is to be used:

- What application content is to be protected

- Which cryptographic algorithms are to be used

- How application protocol entities establish keys

- Whether keys are to be explicitly indicated in JOSE objects or associated by application context

- Which serialization(s) of JOSE objects are to be used

## 5. Use Cases

Several IETF working groups developing application-layer protocols have expressed a desire to use the JOSE data formats in their designs for end-to-end security features. In this section, we summarize the use cases proposed by these groups and discuss the requirements that they imply for the JOSE object formats.

### 5.1. Security Tokens

Security tokens are a common use case for object-based security, for example, SAML assertions [OASIS.saml-core-2.0-os]. Security tokens are used to convey information about a subject entity ("claims" or "assertions") from an issuer to a recipient. The security features of a token format enable the recipient to verify that the claims came from the issuer and, if the object is confidentiality protected, that they were not visible to other parties.

Security tokens are used in federation protocols such as SAML 2.0 [OASIS.saml-core-2.0-os], WS-Federation [WS-Federation], Mozilla Persona [Persona], and OpenID Connect [OpenID.Core], as well as in resource authorization protocols such as OAuth 2.0 [RFC6749], including for OAuth bearer tokens [RFC6750]. In some cases, security tokens are used for client authentication and for access control [JWT-BEARER] [SAML2].

JSON Web Token [JWT] is a security token format based on JSON and JOSE. It is used with Mozilla Persona, OpenID Connect, and OAuth. Because JWTs are often used in contexts with limited space (e.g., HTTP query parameters), it is a core requirement for JWTs, and thus JOSE, to have a compact, URL-safe representation.

### 5.2. OAuth

The OAuth protocol defines a mechanism for distributing and using authorization tokens using HTTP [RFC6749]. A client that wishes to access a protected resource requests authorization from the resource owner. If the resource owner allows this access, he directs an authorization server to issue an access token to the client. When the client wishes to access the protected resource, he presents the token to the relevant resource server, which verifies the validity of the token before providing access to the protected resource.

```
                 +---------------+          +---------------+
                 |               |          |               |
                 |   Resource    |<........>| Authorization |
                 |    Server     |          |     Server    |
                 |               |          |               |
                 +---------------+          +---------------+
                              ^                |
                              |                |
                              |                |
                              |                |
                              |                |
                 +------------|--+          +--|------------+
                 |            +----------------+            |
                 |               |          |   Resource    |
                 |     Client    |          |     Owner     |
                 |               |          |               |
                 +---------------+          +---------------+
```

Figure 1: The OAuth Process

In effect, this process moves the token from the authorization server (as a sender of the object) to the resource server (recipient) via the client as well as the resource owner (the latter because of the HTTP mechanics underlying the protocol). As with email, we have a case where an application object is transported via untrusted intermediaries.

This application has two essential security requirements: integrity and data origin authentication. Integrity protection is required so that the resource owner and the client cannot modify the permission encoded in the token. Although the resource owner is ultimately the entity that grants authorization, it is not trusted to modify the authorization token, since this could, for example, grant access to resources not owned by the resource owner.

Data origin authentication is required so that the resource server can verify that the token was issued by a trusted authorization server.

Confidentiality protection may also be needed if the authorization server is concerned about the visibility of permissions information to the resource owner or client. For example, permissions related to social networking might be considered private information. Note, however, that OAuth already requires that the underlying HTTP transactions be protected by TLS, so tokens are already confidentiality protected from entities other than the resource owner and client.

The confidentiality and integrity needs are met by the basic requirements for signed and encrypted object formats, whether the signing and encryption are provided using asymmetric or symmetric cryptography. The choice of which mechanism is applied will depend on the relationship between the two servers, namely whether they share a symmetric key or only public keys.

Authentication requirements will also depend on deployment characteristics. Where there is a relatively strong binding between the resource server and the authorization server, it may suffice for the authorization server issuing a token to be identified by the key used to sign the token. This requires that the protocol carry either the public key of the authorization server or an identifier for the public or symmetric key. In OAuth, the "client_id" parameter (external to the token) identifies the key to be used.

There may also be more advanced cases where the authorization server's key is not known in advance to the resource server. This may happen, for instance, if an entity instantiated a collection of authorization servers (say for load balancing), each of which has an independent key pair. In these cases, it may be necessary to also include a certificate or certificate chain for the authorization server, so that the resource server can verify that the authorization server is an entity that it trusts.

The HTTP transport for OAuth imposes a particular constraint on the encoding. In the OAuth protocol, tokens frequently need to be passed as query parameters in HTTP URIs [RFC2616] after having been base64url encoded [RFC4648]. While there is no specified limit on the length of URIs (and thus of query parameters), in practice, URIs of more than 2,048 characters are rejected by some user agents. So this use case requires that JOSE objects be sufficiently small, even after being signed and possibly encrypted.

## 5.3. OpenID Connect

The OpenID Connect protocol [OpenID.Core] is a simple, REST/JSON-based identity federation protocol layered on OAuth 2.0. It uses the JWT and JOSE formats both to represent security tokens and to provide security for other protocol messages (performing signing and optionally encryption). OpenID Connect negotiates the algorithms to be used and distributes information about the keys to be used using protocol elements that are not part of the JWT and JOSE header parameters.

In the OpenID Connect context, it is possible for the recipient of a JWT to accept it without integrity protection in the JWT itself. In such cases, the recipient chooses to rely on transport security rather than object security. For example, if the payload is delivered over a TLS-protected channel, the recipient may regard the protections provided by TLS as sufficient, so JOSE protection would not be required.

However, even in this case, it is desirable to associate some metadata with the JWT payload (claim set), such as the content type, or other application-specific metadata. In a signed or encrypted object, these metadata values could be carried in a header with other metadata required for signing or encryption. It would thus simplify the design of OpenID Connect if there could be a JOSE object format that does not apply cryptographic protections to its payload, but allows a header to be attached to the payload in the same way as a signed or encrypted object.

## 5.4. XMPP

The Extensible Messaging and Presence Protocol (XMPP) routes messages from one end client to another by way of XMPP servers [RFC6120]. There are typically two servers involved in delivering any given message: The first client (Alice) sends a message for another client (Bob) to her server (A). Server A uses Bob's identity and the DNS to locate the server for Bob's domain (B) and then delivers the message to that server. Server B then routes the message to Bob.

```
+-------+   +----------+   +----------+   +-----+
| Alice |-->| Server A |-->| Server B |-->| Bob |
+-------+   +----------+   +----------+   +-----+
```

Figure 2: Delivering an XMPP Message

The untrusted-intermediary problems are especially acute for XMPP because in many current deployments, the holder of an XMPP domain outsources the operation of the domain's servers to a different entity. In this environment, there is a clear risk of exposing the domain holder's private information to the domain operator. XMPP already has a defined mechanism for end-to-end security using S/MIME, but it has failed to gain widespread deployment [RFC3923], in part because of key management challenges and in part because of the difficulty of processing S/MIME objects.

The XMPP working group is in the process of developing a new end-to-end encryption system with an encoding based on JOSE and a clearer key management system [XMPP-E2E]. The process of sending an encrypted message in this system involves two steps: First, the sender generates a symmetric Session Master Key (SMK), encrypts the message content (including a per-message Content Master Key), and sends the encrypted message to the desired set of recipients.

Second, each recipient "dials back" to the sender, providing his public key. The sender then responds with the relevant SMK, wrapped with the recipient's public key.

```
                 +-------+   +----------+   +----------+   +-----+
                 | Alice |<->| Server A |<->| Server B |<->| Bob |
                 +-------+   +----------+   +----------+   +-----+
                     |              |              |           |
                     |------------Encrypted message---------->|
                     |              |              |           |
                     |<---------------Public key---------------|
                     |              |              |           |
                     |---------------Wrapped SMK------------->|
                     |              |              |           |
```

Figure 3: Delivering a Secure XMPP Message

The main thing that this system requires from the JOSE formats is confidentiality protection via content encryption, plus an integrity check via a MAC derived from the same symmetric key. The separation of the key exchange from the transmission of the encrypted content, however, requires that the JOSE encrypted object format allow wrapped symmetric keys to be carried separately from the encrypted payload. In addition, the encrypted object will need to have a tag for the key that was used to encrypt the content, so that the recipient (Bob) can present the tag to the sender (Alice) when requesting the wrapped key.

Another important feature of XMPP is that it allows for the simultaneous delivery of a message to multiple recipients. In the diagrams above, Server A could deliver the message not only to Server B (for Bob) but also to Servers C, D, E, etc., for other users. In such cases, to avoid the multiple "dial back" transactions implied by the above mechanism, XMPP systems will likely reuse a given SMK for multiple individual messages, refreshing the SMK on a periodic and/or event-driven basis (e.g., when the recipient's presence changes). They might also cache public keys for end recipients, so that wrapped keys can be sent along with content on future messages. This implies that the JOSE encrypted object format must support the provision of multiple versions of the same wrapped SMK (much as a CMS EnvelopedData structure can include multiple RecipientInfo structures).

In the current draft of the XMPP end-to-end security system, each party is authenticated by virtue of the other party's trust in the XMPP message routing system. The sender is authenticated to the receiver because he can receive messages for the identifier "Alice" (in particular, the request for wrapped keys) and can originate messages for that identifier (the wrapped key). Likewise, the receiver is authenticated to the sender because he received the original encrypted message and originated the request for a wrapped key. So, the authentication here requires not only that XMPP routing be done properly, but also that TLS be used on every hop. Moreover, it requires that the TLS channels have strong authentication, since a man in the middle on any of the three hops can masquerade as Bob and obtain the key material for an encrypted message.

Because this authentication is quite weak (depending on the use of TLS on three hops) and unverifiable by the endpoints, it is possible that the XMPP working group will integrate some sort of credentials for end recipients, in which case there would need to be a way to associate these credentials with JOSE objects.

Finally, it's worth noting that XMPP is based on XML, not JSON. So by using JOSE, XMPP will be carrying JSON objects within XML. It is thus a desirable property for JOSE objects to be encoded in such a way as to be safe for inclusion in XML. Otherwise, an explicit CDATA indication must be given to the parser to indicate that it is not to be parsed as XML. One way to meet this requirement would be to apply base64url encoding, but for XMPP messages of medium-to-large size, this could impose a fair degree of overhead.

## 5.5. ALTO

Application-Layer Traffic Optimization (ALTO) is a system for distributing network topology information to end devices, so that those devices can modify their behavior to have a lower impact on the network [RFC6708]. The ALTO protocol distributes topology information in the form of JSON objects carried in HTTP [RFC2616] [ALTO]. The basic version of ALTO is simply a client-server protocol, so simple use of HTTPS suffices for this case [RFC2818]. However, there is beginning to be some discussion of use cases for ALTO in which these JSON objects will be distributed through a collection of intermediate servers before reaching the client, while still preserving the ability of the client to authenticate the original source of the object. Even the base ALTO protocol notes that "ALTO Clients obtaining ALTO information through redistribution must be able to validate the received ALTO information" to ensure that it was generated by an appropriate ALTO server.

In this case, the security requirements are straightforward. JOSE objects carrying ALTO payloads will need to bear digital signatures from the originating servers, which will be bound to certificates attesting to the identities of the servers. There is no requirement for confidentiality in this case, since ALTO information is generally public.

The more interesting questions are encoding questions. ALTO objects are likely to be much larger than payloads in the two cases above, with sizes of up to several megabytes. Processing of such large objects can be done more quickly if it can be done in a single pass, which may be possible if JOSE objects require specific orderings of fields within the JSON structure.

In addition, because ALTO objects are also encoded as JSON, they are already safe for inclusion in a JOSE object. Signed JOSE objects will likely carry the signed data in a string alongside the signature. JSON objects have the property that they can be safely encoded in JSON strings. All they require is that unnecessary white space be removed, a much simpler transformation than, say, base64url encoding. This raises the question of whether it might be possible to optimize the JOSE encoding for certain "JSON-safe" cases.

Finally, it may be desirable for ALTO to have a "detached signature" mechanism, that is, a way to encode signature information separate from the protected content. This would allow the ALTO protocol to include the signature in an HTTPS header, with the signed content as the HTTPS entity body.

### 5.6. Emergency Alerting

Emergency alerting is an emerging use case for IP networks [ALERT-REQ]. Alerting systems allow authorities to warn users of impending danger by sending alert messages to connected devices. For example, in the event of a hurricane or tornado, alerts might be sent to all devices in the path of the storm.

The most critical security requirement for alerting systems is that it must not be possible for an attacker to send false alerts to devices. Such a capability would potentially allow an attacker to create wide-spread panic. In practice, alert systems prevent these attacks both by controls on sending messages at points where alerts are originated, and by having recipients of alerts verify that the alert was sent by an authorized source. The former type of control is implemented with local security on hosts from which alerts can be originated. The latter type is implemented by digital signatures on alert messages (using channel-based or object-based mechanisms). With an object-based mechanism, the signature value is encoded in a secure object. With a channel-based mechanism, the alert is "signed" by virtue of being sent over an authenticated, integrity-protected channel.

Alerts typically reach end recipients via a series of intermediaries. For example, while a national weather service might originate a hurricane alert, it might first be delivered to a national gateway and then to network operators, who broadcast it to end subscribers.

```
           +------------+    +------------+    +------------+
           | Originator |    | Originator |    | Originator |
           +------------+    +------------+    +------------+
                 |                 .                 .
                 +-----------------+..................
                                   |
                                   V
                              +---------+
                              | Gateway |
                              +---------+
                                   |
                      +------------+------------+
                      |                         |
                      V                         V
                 +---------+               +---------+
                 | Network |               | Network |
                 +---------+               +---------+
                      |                         |
               +------+-----+            +------+-----+
               |            |            |            |
               V            V            V            V
           +--------+   +--------+   +--------+   +--------+
           | Device |   | Device |   | Device |   | Device |
           +--------+   +--------+   +--------+   +--------+
```

Figure 4: Delivering an Emergency Alert

In order to verify alert signatures, recipients must be provisioned with the proper public keys for trusted alert authorities. This trust may be "piece-wise" along the path the alert takes. For example, the alert relays operated by networks might have a full set of certificates for all alert originators, while end devices may only trust their local alert relay. Or, devices might require that a device be signed by an authorized originator and by its local network's relay.

This scenario creates a need for multiple signatures on alert documents, so that an alert can bear signatures from any or all of the entities that processed it along the path. In order to minimize complexity, these signatures should be "modular" in the sense that a new signature can be added without a need to alter or recompute previous signatures.

## 5.7. Web Cryptography

The W3C Web Cryptography API defines a standard cryptographic API for the Web [WebCrypto]. If a browser exposes this API, then JavaScript provided as part of a Web page can ask the browser to perform cryptographic operations, such as digest, MAC, encryption, or digital signing.

One of the key reasons to have the browser perform cryptographic operations is to avoid allowing JavaScript code to access the keying material used for these operations. For example, this separation would prevent code injected through a cross-site scripting (XSS) attack from reading and exfiltrating keys stored within a browser. While the malicious code could still use the key while running in the browser, this vulnerability can only be exercised while the malicious code is active in a user's browser.

However, the Web Cryptography API also provides a key export functionality, which can allow JavaScript to extract a key from the API in wrapped form. For example, JavaScript code might provide a public key for which the corresponding private key is held by another device. The wrapped key provided by the API could then be used to safely transport the key to the new device. While this could potentially allow malicious code to export a key, the need for an explicit export operation provides a control point, allowing for user notification or consent verification.

The Web Cryptography API also allows browsers to impose limitations on the usage of the keys it handles. For example, a symmetric key might be marked as usable only for encryption, and not for MAC. When a key is exported in wrapped form, these attributes should be carried along with it.

The Web Cryptography API thus requires formats to express several forms of keys. Obviously, the public key from an asymmetric key pair can be freely imported to and exported from the browser, so there needs to be a format for public keys. There is also a need for a format to express private keys and symmetric keys. For non-public keys, the primary need is for a wrapped form, where the confidentiality and integrity of the key is assured cryptographically; these protections should also apply to any attributes of the key. It may also be useful to define a direct, unwrapped format for use within a security boundary.

## 5.8. Constrained Devices

This section describes use cases for constrained devices as defined in [CONSTRAINED]. Typical issues with this type of device are limited memory, limited power supply, low processing power, and severe message size limitations for the communication protocols.

### 5.8.1. Example: MAC Based on ECDH-Derived Key

Suppose a small, low power device maker has decided on using the output of the JOSE working group as their encryption and authentication framework. The device maker has a limited budget for both gates and power. For this reason there are a number of short cuts and design decisions that have been made in order to minimize these needs.

The design team has determined that the use of MACs is going to be sufficient to provide the necessary authentication. However, although a MAC is going to be used, they do not want to use a single long-term shared secret. Instead, they have adopted the following proposal for computing a shared secret that can be validated:

o An Elliptic-Curve Diffie-Hellman (ECDH) key pair is generated for the device at the time of manufacturing. (Or, as part of the configuration process during installation.)

o An ECDH public key for the controller is configured at the time of configuration.

o The configuration system performs the ECDH computation and configures the device with the resulting shared secret. This process eliminates the need for the device to be able to perform the required ECDH processing. The security requirements on protecting this computed shared secret are the same as the requirements on protecting the private ECDH key.

o A counter and an increment value are configured onto the device.

o When a message is to be sent by the device, the counter is incremented and a new MAC key is computed from the ECDH secret and the counter value. A custom Key Derivation Function (KDF) based on AES-CBC is used to derive the required MAC key. The MAC key is then used to compute the MAC value for the message.

In a similar manner, the KDF function can be used to compute an Authenticated Encryption with Associated Data (AEAD) algorithm key when the system needs to provide confidentiality for the message. The controller, being a larger device, will perform the ECDH step and use a random number generator to generate the sender nonce value.

#### 5.8.2. Object Security for CoAP

This use case deals with constrained devices of class C0/C1 (see [CONSTRAINED]). These devices communicate using RESTful requests and responses transferred using the Constrained Application Protocol [CoAP]. To simplify matters, all communication is assumed to be unicast; i.e., these security measures don't cover multicast or broadcast.

In this type of setting, it may be too costly to use session-based security (e.g., to run a 4-pass authentication protocol) since receiving and in particular sending consumes a lot of power, especially for wireless devices. Therefore, to just secure the CoAP payload by replacing a plaintext payload of a request or response with a JWE object is an important alternative solution, which allows a trade-off between protection (the CoAP headers are not protected) and performance.

In a simple setting, consider the payload of a CoAP GET response from a sensor type device. The information in a sensor reading may be privacy or business sensitive and needs both integrity protection and encryption.

However, some sensor readings are very short, say, a few bytes, and in this case, default encryption and integrity protection algorithms (such as 128-bit AES-CBC with HMAC_SHA256) may cause a dramatic expansion of the payload, even disregarding JWE headers.

Also, the value of certain sensor readings may decline rapidly, e.g., traffic or environmental measurements, so it must be possible to reduce the security overhead.

This leads to the following requirements that could be covered by specific JWE/JWS profiles:

- The size of the secure object shall be as small as possible. Receiving an object may cost orders of magnitude more in terms of power than performing, say, public key cryptography on the object, in particular in a wireless setting.

- Integrity protection: The object shall be able to support integrity protection, i.e., have a field containing a digital signature, both public key signatures and keyed MACs shall be supported.

- Encryption: The object shall be able to support encryption as an optional addition to integrity protection. It shall be possible to exclude certain fields from encryption, which are needed before verifying integrity or decrypting the object.

- Cipher suites: It should be possible to support a variety of cipher suites to support the constrained devices' use cases. For example:

  * Block ciphers with block sizes of, e.g., 96 bits, in addition to the standard 128 bits.

  * Modes of operation for block ciphers that do not expand the message size to a block boundary, such as AES-GCM.

  * Cipher suites that support combined encryption and MAC calculation (i.e., AEAD modes for block ciphers).

## 6. Requirements

This section summarizes the requirements from the above use cases and lists further requirements not directly derived from the above use cases. There are also some constraints that are not hard requirements but that are still desirable properties for the JOSE system to have.

### 6.1. Functional Requirements

F1 Define formats for secure objects that provide the following security properties:

  * Digital signature (integrity/authentication under an asymmetric key pair)

  * Message authentication (integrity/authentication under a symmetric key)

  * Authenticated encryption

F2 Define a format for public keys and private keys for asymmetric cryptographic algorithms, with associated attributes, including a wrapped form for private keys.

F3 Define a format for symmetric keys with associated attributes, allowing for both wrapped and unwrapped keys.

F4 Define a JSON serialization for each of the above objects. An object in this encoding must be valid according to the JSON ABNF syntax [RFC7159].

F5 Define a compact, URL-safe text serialization for the encrypted and signed object formats.

F6 Allow for attributes associated to wrapped keys to be bound to them cryptographically.

F7 Allow for wrapped keys to be separated from a secure object that uses a symmetric key. In such cases, cryptographic components of the secure object other than the wrapped key (e.g., ciphertext, MAC values) must be independent of the wrapped form of the key. For example, if an encrypted object is prepared for multiple recipients, then only the wrapped key may vary, not the ciphertext.

F8 Do not impose more overhead than is required to meet the requirements in this document, especially when a large amount of application content is being protected.

## 6.2. Security Requirements

S1 Provide mechanisms to avoid repeated use of the same symmetric key for encryption or MAC computation. Instead, long-lived keys should be used only for key wrapping, not for direct encryption/MAC. It should be possible to use any of the key management techniques provided in CMS [RFC5652]:

* Key transport (wrapping for a public key)

* Key encipherment (wrapping for a symmetric key)

* Key agreement (wrapping for a Diffie-Hellman (DH) public key)

* Password-based encryption (wrapping under a key derived from a password)

S2 Where long-lived symmetric keys are used directly for cryptographic operations (i.e., where requirement S1 is not met), provide deployment guidance on key management practices, such as the need to limit key lifetimes.

S3 Use cryptographic algorithms in a manner compatible with major validation processes. For example, if typical validation standards allow algorithm A to be used for purpose X but not purpose Y, then JOSE should not recommend using algorithm A for purpose Y.

S4 Support operation with or without pre-negotiation. It must be possible to create or process secure objects without any configuration beyond key provisioning. If it is possible for keys to be derived from application context, it must be possible for a recipient to recognize when it does not have the appropriate key.

## 6.3. Desiderata

D1 Maximize compatibility with the W3C Web Crypto specifications, e.g., by coordinating with the Web Crypto working group to encourage alignment of algorithms and algorithm identifiers.

D2 Avoid JSON canonicalization to the extent possible. That is, all other things being equal, techniques that rely on fixing a serialization of an object (e.g., by encoding it with base64url) are preferred over those that require converting an object to a canonical form.

D3 Maximize the extent to which the inputs and outputs of JOSE cryptographic operations can be controlled by the applications, as opposed to involving processing specific to JOSE. This allows JOSE the flexibility to address the needs of many cryptographic protocols. For example, in some cases, it might allow JOSE objects to be translated to legacy formats such as CMS without the need for re-encryption or re-signing.

## 7. Security Considerations

The primary focus of this document is the requirements for a JSON-based secure object format. At the level of general security considerations for object-based security technologies, the security considerations for this format are the same as for CMS [RFC5652]. The primary difference between the JOSE format and CMS is that JOSE is based on JSON, which does not have a canonical representation. The lack of a canonical form means that it is difficult to determine whether two JSON objects represent the same information, which could lead to vulnerabilities in some usages of JOSE.

## 8. References

### 8.1. Normative References

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6708] Kiesel, S., Previdi, S., Stiemerling, M., Woundy, R., and Y. Yang, "Application-Layer Traffic Optimization (ALTO) Requirements", RFC 6708, September 2012.

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

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[W3C.REC-xml] Bray, T., Maler, E., Paoli, J., and C. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126/>.

[WebCrypto] Dahl, D. and R. Sleevi, "Web Cryptography API", W3C Working Draft, January 2013, <http://www.w3.org/TR/2013/WD-WebCryptoAPI-20130108/>.

### 8.2. Informative References

[ALERT-REQ] Schulzrinne, H., Norreys, S., Rosen, B., and H. Tschofenig, "Requirements, Terminology and Framework for Exigent Communications", Work in Progress, March 2012.

[ALTO] Alimi, R., Ed., Penno, R., Ed., and Y. Yang, Ed., "ALTO Protocol", Work in Progress, March 2014.

[CAP] Botterell, A. and E. Jones, "Common Alerting Protocol, v1.1", OASIS Standard CAP-V1.1, October 2005, <http://www.oasis-open.org/committees/download.php/15135/emergency-CAPv1.1-Corrected_DOM.pdf>.

[CONSTRAINED]
Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", Work in Progress, March 2014.

[CoAP] Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", Work in Progress, June 2013.

[ITU.X690.2002]
International Telecommunications Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, July 2002.

[JWA] Jones, M., "JSON Web Algorithms (JWA)", Work in Progress, March 2014.

[JWE] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", Work in Progress, March 2014.

[JWK] Jones, M., "JSON Web Key (JWK)", Work in Progress, March 2014.

[JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", Work in Progress, March 2014.

[JWT-BEARER]
Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", Work in Progress, March 2014.

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", Work in Progress, March 2014.

[OASIS.saml-core-2.0-os]
Cantor, S., Kemp, J., Maler, E., and R. Philpott, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", Oasis Standard, March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>.

[OpenID.Core]
Bradley, J., de Medeiros, B., Jones, M., Mortimore, C., and N. Sakimura, "OpenID Connect Core 1.0", December 2013, <http://openid.net/specs/openid-connect-core-1_0.html>.

[Persona] Mozilla Developer Network, "Mozilla Persona", April 2013, <https://developer.mozilla.org/en-US/docs/Persona>.

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

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3207] Hoffman, P., "SMTP Service Extension for Secure SMTP over Transport Layer Security", RFC 3207, February 2002.

[RFC3923] Saint-Andre, P., "End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)", RFC 3923, October 2004.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

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

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

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

[SAML2] Campbell, B., Mortimore, C., and M. Jones, "SAML 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants", Work in Progress, March 2014.

[W3C.xmldsig-core]
Eastlake, D., Reagle, J., and D. Solo, "XML-Signature Syntax and Processing", W3C Recommendation, June 2008, <http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/>.

[W3C.xmlenc-core]
Eastlake, D. and J. Reagle, "XML Encryption Syntax and Processing", W3C Candidate Recommendation, December 2002, <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>.

[WS-Federation]
Goodner, M., Kaler, C., McIntosh, M., and A. Nadalin, "Web Services Federation Language (WS-Federation) Version 1.2", Oasis Standard, May 2009, <http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html>.

[XMPP-E2E] Miller, M., "End-to-End Object Encryption and Signatures for the Extensible Messaging and Presence Protocol (XMPP)", Work in Progress, June 2013.

## Appendix A.  Acknowledgements

Thanks to Matt Miller for discussions related to the XMPP end-to-end security model and to Mike Jones for considerations related to security tokens and XML security.  Thanks to Mark Watson for raising the need for representing symmetric keys and binding attributes to them.  Thanks to Ludwig Seitz for contributing the constrained device use case.

## Author's Address

Richard Barnes
Mozilla
331 E. Evelyn Ave.
Mountain View, CA  94041
US

EMail: rlb@ipv.sx