Workgroup: Network Working Group
Internet-Draft: draft-paragon-paseto-rfc-01
Published: 24 May 2022
Intended Status: Informational
Expires: 25 November 2022
R. Terjesen
Paragon Initiative Enterprises
S. Haussmann
Rensselaer Polytechnic Institute
S. Arciszewski
Paragon Initiative Enterprises

PASETO (Platform-Agnostic SEcurity TOkens)

Abstract

Platform-Agnostic SEcurity TOkens (PASETOs) provide a cryptographically secure, compact, and URL-safe representation of claims that may be transferred between two parties. The claims are encoded in JavaScript Object Notation (JSON), version-tagged, and either encrypted using shared-key cryptography or signed using public-key cryptography.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 25 November 2022.

Copyright Notice

Table of Contents

1. Introduction

A Platform-Agnostic SEcurity TOken (PASETO) is a cryptographically secure, compact, and URL-safe representation of claims intended for space-constrained environments such as HTTP Cookies, HTTP Authorization headers, and URI query parameters. A PASETO encodes claims to be transmitted (in a JSON [RFC8259] object by default), and is either encrypted symmetrically or signed using public-key cryptography.

1.1. Difference Between PASETO and JOSE

The key difference between PASETO and the JOSE family of standards (JWS [RFC7516], JWE [RFC7517], JWK [RFC7518], JWA [RFC7518], and JWT [RFC7519]) is that JOSE allows implementors and users to mix and match their own choice of cryptographic algorithms (specified by the "alg" header in JWT), while PASETO has clearly defined protocol versions to prevent unsafe configurations from being selected.

PASETO is defined in two pieces:

1. The PASETO Message Format, defined in Section 2

2. The PASETO Protocol Version, defined in Section 3

1.2. Why Not Update JOSE to Be Secure?

Backwards compatibility introduces the risk of downgrade attacks. Conversely, a totally separate standard can be designed from the ground up to be secure and misuse-resistant.

For that reason, PASETO does not aspire to update the JOSE family of standards. To do so would undermine the security benefits of a non-interoperable alternative.

1.3. Notation and Conventions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. PASETO Message Format

PASETOs consist of three or four segments, separated by a period (the ASCII character whose number, represented in hexadecimal, is 2E).

Without the Optional Footer:

```
version.purpose.payload
```

With the Optional Footer:

```
version.purpose.payload.footer
```

If no footer is provided, implementations **SHOULD NOT** append a trailing period to each payload.

2.1. PASETO Token Versions

The **version** is a string that represents the current version of the protocol. Currently, two versions are specified, which each possess their own ciphersuites. Accepted values: **v3**, **v4**.

(Earlier versions of the PASETO RFC specified **v1** and **v2**, but these are not proposed for IETF standardization.)

Future standardization efforts **MAY** optionally suffix an additional piece of information to the version to specify a non-JSON encoding for claims. The default encoding, when no suffix is applied, is JSON. This suffix does not change the cryptography protocol being used (except that the suffix is also authenticated).

2.2. PASETO Token Purposes

The **purpose** is a short string describing the purpose of the token. Accepted values: **local**, **public**.

- ***local**: shared-key authenticated encryption

- ***public**: public-key digital signatures; **not encrypted**

The **payload** is a string that contains the token's data. In a local token, this data is encrypted with a symmetric cipher. In a public token, this data is *unencrypted*.

Any optional data can be appended to the **footer**. This data is authenticated through inclusion in the calculation of the authentication tag along with the header and payload. The **footer MUST NOT** be encrypted.

2.3. Base64 Encoding

The payload and footer in a PASETO **MUST** be encoded using base64url as defined in [RFC4648], without = padding.

In this document. b64() refers to this unpadded variant of base64url.

2.4. Multi-Part Authentication

Multi-part messages (e.g. header, content, footer, implicit) are encoded in a specific manner before being passed to the appropriate cryptographic function, to prevent canonicalization attacks.

In local mode, this encoding is applied to the additional associated data (AAD). In public mode, which is not encrypted, this encoding is applied to the components of the token, with respect to the protocol version being followed.

We will refer to this process as **PAE** in this document (short for Pre-Authentication Encoding).

2.4.1. PAE Definition

PAE() accepts an array of strings.

LE64() encodes a 64-bit unsigned integer into a little-endian binary string. The most significant bit **MUST** be set to 0 for interoperability with programming languages that do not have unsigned integer support.

The first 8 bytes of the output will be the number of pieces. Currently, this will be 3 or 4. This is calculated by applying LE64() to the size of the array.

Next, for each piece provided, the length of the piece is encoded via LE64() and prefixed to each piece before concatenation.

```
function LE64(n) {
    var str = '';
    for (var i = 0; i < 8; ++i) {
        if (i === 7) {
            n &= 127;
        }
        str += String.fromCharCode(n & 255);
        n = n >>> 8;
    }
    return str;
}
function PAE(pieces) {
    if (!Array.isArray(pieces)) {
        throw TypeError('Expected an array.');
    }
    var count = pieces.length;
    var output = LE64(count);
    for (var i = 0; i < count; i++) {
        output += LE64(pieces[i].length);
        output += pieces[i];
    }
    return output;
}
```

Figure 1: JavaScript implementation of Pre-Authentication Encoding (PAE)

As a consequence:

- PAE([]) will always return \x00\x00\x00\x00\x00\x00\x00\x00
- PAE(['']) will always return \x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
- PAE(['test']) will always return \x01\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00test
- PAE('test') will throw a TypeError

As a result, partially controlled plaintext cannot be used to create a collision. Either the number of pieces will differ, or the length of one of the fields (which is prefixed to user-controlled input) will differ, or both.

Due to the length being expressed as an unsigned 64-bit integer, it is infeasible to encode enough data to create an integer overflow.

This is not used to encode data prior to decryption, and no decoding function is provided or specified. This merely exists to prevent canonicalization attacks.

3. Protocol Versions

This document defines two protocol versions for the PASETO standard.

Protocol versions (**Version 3**, **Version 4**) correspond to a specific message format version (**v3**, **v4**).

Each protocol version strictly defines the cryptographic primitives used. Changes to the primitives requires new protocol versions. Future RFCs **MAY** introduce new PASETO protocol versions by continuing the convention (e.g. **Version 5**, **Version 6**, ...).

Both **Version 3** and **Version 4** provide authentication of the entire PASETO message, including the **version**, **purpose**, **payload**, **footer**, and (optional) **implicit assertions**.

The initial recommendation is to use **Version 4**, allowing for upgrades to possible future versions **Version 5**, **Version 6**, etc. when they are defined in the future.

3.1. PASETO Protocol Guidelines

When defining future protocol versions, the following rules **SHOULD** or **MUST** be followed:

1. Everything in a token **MUST** be authenticated. Attackers should never be allowed the opportunity to alter messages freely.

    * If encryption is specified, unauthenticated modes (e.g. AES-CBC without a MAC) are forbidden.

    * The nonce or initialization vector must be covered by the authentication tag, not just the ciphertext.

2. Some degree of nonce-misuse resistance **SHOULD** be provided:

    * Supporting larger nonces (longer than 128-bit) is sufficient for satisfying this requirement, provided the nonce is generated by a cryptographically secure random number generator, such as **/dev/urandom** on Linux.

    * Key-splitting and including an additional HKDF salt as part of the nonce is sufficient for this requirement.

3. Public-key cryptography **MUST** be IND-CCA2 secure to be considered for inclusion.

    *This means that RSA with PKCS1v1.5 padding and unpadded RSA **MUST NOT** ever be used in a PASETO protocol.

4. PASETO Protocol Version 3

PASETO Version 3 is composed of NIST-approved algorithms, and will operate on tokens with the **v3** version header.

v3 messages **MUST** use a **purpose** value of either **local** or **public**.

4.1. v3.local

v3.local messages **SHALL** be encrypted and authenticated with **AES-256-CTR** (AES-CTR from [RFC3686] with a 256-bit key) and **HMAC-SHA-384** ([RFC4231]), using an **Encrypt-then-MAC** construction.

Encryption and authentication keys are split from the original key and 256-bit nonce, facilitated by HKDF [RFC5869] using SHA384.

Refer to the operations defined in **PASETO.v3.Encrypt** and **PASETO.v3.Decrypt** for a formal definition.

4.2. v3.public

v1.public messages **SHALL** be signed using ECDSA with NIST curve P-384 as defined in [RFC6687]. These messages provide authentication but do not prevent the contents from being read, including by those without either the **public key** or the **secret key**. Refer to the operations defined in **PASETO.v3.Sign** and **PASETO.v3.Verify** for a formal definition.

4.3. PASETO Version 3 Algorithms

4.3.1. PASETO.v3.Encrypt

Given a message m, key k, and optional footer f (which defaults to empty string), and an optional implicit assertion i (which defaults to empty string):

1. Before encrypting, first assert that the key being used is intended for use with v3.local tokens. If this assertion fails, abort encryption.

2. Set header h to v3.local.

3. Generate 32 random bytes from the OS's CSPRNG to get the nonce, n.

4. Split the key into an Encryption key (Ek) and Authentication key (Ak), using HKDF-HMAC-SHA384, with n appended to the info rather than the salt.

    *The output length **MUST** be 48 for both key derivations.

*The derived key will be the leftmost 32 bytes of the first HKDF derivation. The remaining 16 bytes of the first key derivation (from which Ek is derived) will be used as a counter nonce (n2):

5. Encrypt the message using AES-256-CTR, using Ek as the key and n2 as the nonce. We'll call the encrypted output of this step c.

6. Pack h, n, c, and f together (in that order) using PAE (see Section 2.4.1). We'll call this preAuth.

7. Calculate HMAC-SHA384 of the output of preAuth, using Ak as the authentication key. We'll call this t.

8. If f is:

*Empty: return h || b64(n || c || t)

*Non-empty: return h || b64(n || c || t) || . || b64(f)

*...where || means "concatenate"

Example code:

```
tmp = hkdf_sha384(
    len = 48,
    ikm = k,
    info = "paseto-encryption-key" || n,
    salt = NULL
);
Ek = tmp[0:32]
n2 = tmp[32:]
Ak = hkdf_sha384(
    len = 48,
    ikm = k,
    info = "paseto-auth-key-for-aead" || n,
    salt = NULL
);
```

Figure 2: Step 4: Key splitting with HKDF-SHA384 as per *[RFC5869]*.

```
c = aes256ctr_encrypt(
    plaintext = m,
    nonce = n2
    key = Ek
);
```

Figure 3: Step 5: PASETO Version 3 encryption (calculating c)

4.3.2. PASETO.v3.Decrypt

Given a message m, key k, and optional footer f (which defaults to empty string):

1. Before decrypting, first assert that the key being used is intended for use with v3.local tokens. If this assertion fails, abort decryption.

2. If f is not empty, implementations **MAY** verify that the value appended to the token matches some expected string f, provided they do so using a constant-time string compare function.

3. Verify that the message begins with v3.local., otherwise throw an exception. This constant will be referred to as h.

4. Decode the payload (m sans h, f, and the optional trailing period between m and f) from b64 to raw binary. Set:

   *n to the leftmost 32 bytes

   *t to the rightmost 48 bytes

   *c to the middle remainder of the payload, excluding n and t

5. Split the key (k) into an Encryption key (Ek) and an Authentication key (Ak), n appended to the HKDF info.

   *For encryption keys, the **info** parameter for HKDF **MUST** be set to **paseto-encryption-key**.

   *For authentication keys, the **info** parameter for HKDF **MUST** be set to **paseto-auth-key-for-aead**.

   *The output length **MUST** be 48 for both key derivations. The leftmost 32 bytes of the first key derivation will produce Ek, while the remaining 16 bytes will be the AES nonce n2.

6. Pack h, n, c, f, and i together (in that order) using PAE (see Section 2.4.1). We'll call this preAuth.

7. Recalculate HMAC-SHA-384 of preAuth using Ak as the key. We'll call this t2.

8. Compare t with t2 using a constant-time string compare function. If they are not identical, throw an exception.

9. Decrypt c using AES-256-CTR, using Ek as the key and the rightmost 16 bytes of n as the nonce, and return this value.

Example code:

```
tmp = hkdf_sha384(
    len = 48,
    ikm = k,
    info = "paseto-encryption-key" || n,
    salt = NULL
);
Ek = tmp[0:32]
n2 = tmp[32:]
Ak = hkdf_sha384(
    len = 48,
    ikm = k,
    info = "paseto-auth-key-for-aead" || n,
    salt = NULL
);
```

Figure 4: Step 4: Key splitting with HKDF-SHA384 as per *[RFC5869]*.

```
return aes256ctr_decrypt(
   cipherext = c,
   nonce = n2
   key = Ek
);
```

Figure 5: Step 8: PASETO Version 3 decryption

4.3.3. PASETO.v3.Sign

Given a message m, 384-bit ECDSA secret key sk, an optional footer f (which defaults to empty string), and an optional implicit assertion i (which defaults to empty string):

1. Before signing, first assert that the key being used is intended for use with v3.public tokens, and is a secret key (not a public key). If this assertion fails, abort signing.

2. Set cpk to the compressed point representation of the ECDSA public key (see point compression), using [#paseto-v3-compresspublickey].

3. Set h to v3.public.

4. Pack cpk, h, m, f, and i together (in that order) using PAE (see Section 2.4.1). We'll call this m2.

5. Sign m2 using ECDSA over P-384 and SHA-384 with the private key sk. We'll call this sig. The output of sig MUST be in the format r || s (where ||means concatenate), for a total length of 96 bytes.

   - Signatures **SHOULD** use deterministic k-values ([RFC6979]) if possible, to mitigate the risk of k-value reuse.

   - If possible, hedged signatures ([RFC6979] + additional randomness when generating k-values to provide resilience to fault attacks) are preferred over [RFC6979] alone.

   - If [RFC6979] is not available in your programming language, ECDSA **MUST** use a CSPRNG to generate the k-value.

6. If f is:

   - Empty: return h || b64(m || sig)

   - Non-empty: return h || b64(m || sig) || . || b64(f)

   - ...where || means "concatenate"

```
cpk = PASETO.v3.CompressPublicKey(sk.getPublicKey());
m2 = PASETO.PAE(cpk, h, m, f, i);
sig = crypto_sign_ecdsa_p384(
    message = m2,
    private_key = sk
);
```

Figure 6: Pseudocode: ECDSA signature algorithm used in PASETO v3

4.3.4. PASETO.v3.Verify

Given a signed message sm, ECDSA public key pk, and optional footer f (which defaults to empty string), and an optional implicit assertion i (which defaults to empty string):

1. Before verifying, first assert that the key being used is intended for use with v3.public tokens, and is a public key (not a secret key). If this assertion fails, abort verifying.

2. If f is not empty, implementations **MAY** verify that the value appended to the token matches some expected string f, provided they do so using a constant-time string compare function.

3. Set cpk to the compressed point representation of the ECDSA public key (see point compression), using [#paseto-v3-compresspublickey].

4. Verify that the message begins with v3.public., otherwise throw an exception. This constant will be referred to as h.

5. Decode the payload (sm sans h, f, and the optional trailing period between m and f) from base64url to raw binary. Set:

    *s to the rightmost 96 bytes

    *m to the leftmost remainder of the payload, excluding s

6. Pack h, m, f, and i together (in that order) using PAE (see Section 2.4.1). We'll call this m2.

7. Use RSA to verify that the signature is valid for the message. The padding mode **MUST** be RSASSA-PSS [RFC8017]; PKCS1v1.5 is explicitly forbidden. The public exponent e **MUST** be 65537. The mask generating function **MUST** be MGF1+SHA384. The hash function **MUST** be SHA384. (See below for pseudocode.)

8. If the signature is valid, return m. Otherwise, throw an exception.

```
cpk = PASETO.v3.CompressPublicKey(pk);
m2 = PASETO.PAE(cpk, h, m, f, i);
valid = crypto_sign_ecdsa_p384_verify(
    signature = s,
    message = m2,
    public_key = pk
);
```

Figure 7: Pseudocode: ECDSA signature validation for PASETO Version 3

4.3.5. PASETO.v3.CompressPublicKey

Given a public key consisting of two coordinates (X, Y):

1. Set the header to 0x02.

2. Take the least significant bit of Y and add it to the header.

3. Append the X coordinate (in big-endian byte order) to the header.

```
lsb(y):
   return y[y.length - 1] & 1

pubKeyCompress(x, y):
   header = [0x02 + lsb(y)]
   return header.concat(x)
```

Figure 8: Pseudocode: Point compression as used in PASETO Version 3.

5. PASETO Protocol Version v4

PASETO Version 4 is the recommended version of PASETO, and will operate on tokens with the **v4** version header.

v4 messages **MUST** use a **purpose** value of either **local** or **public**.

5.1. v4.local

v4.local messages **MUST** be encrypted with XChaCha20, a variant of ChaCha20 [RFC7539] defined in XChaCha20. Refer to the operations defined in **PASETO.v4.Encrypt** and **PASETO.v4.Decrypt** for a formal definition.

5.2. v4.public

v4.public messages **MUST** be signed using Ed25519 [RFC8032] public key signatures. These messages provide authentication but do not prevent the contents from being read, including by those without either the **public key** or the **private key**. Refer to the operations defined in **v4.Sign** and **v4.Verify** for a formal definition.

5.3. PASETO Version 4 Algorithms

5.3.1. PASETO.v4.Encrypt

Given a message m, key k, and optional footer f.

1. Before encrypting, first assert that the key being used is intended for use with v4.local tokens. If this assertion fails, abort encryption.

2. Set header h to v4.local.

3. Generate 32 random bytes from the OS's CSPRNG, n.

4. Split the key into an Encryption key (Ek) and Authentication key (Ak), using keyed BLAKE2b, using the domain separation constants and n as the message, and the input key as the key. The first value will be 56 bytes, the second will be 32 bytes. The derived key will be the leftmost 32 bytes of the hash output. The remaining 24 bytes will be used as a counter nonce (n2).

5. Encrypt the message using XChaCha20, using n2 from step 3 as the nonce and Ek as the key.

6. Pack h, n, c, f, and i together (in that order) using PAE (see Section 2.4.1). We'll call this preAuth.

7. Calculate BLAKE2b-MAC of the output of preAuth, using Ak as the authentication key. We'll call this t.

8. If f is:

   *Empty: return h || b64(n || c)

   *Non-empty: return h || b64(n || c) || . || b64(f)

   *...where || means "concatenate"

```
tmp = crypto_generichash(
    msg = "paseto-encryption-key" || n,
    key = key,
    length = 56
);
Ek = tmp[0:32]
n2 = tmp[32:]
Ak = crypto_generichash(
    msg = "paseto-auth-key-for-aead" || n,
    key = key,
    length = 32
);
```

Figure 9: Step 4: Key splitting with BLAKE2b.

```
c = crypto_stream_xchacha20_xor(
    message = m
    nonce = n2
    key = Ek
);
preAuth = PASETO.PAE(h, n, c, f, i)
t = crypto_generichash(
    message = preAuth
    key = Ak,
    length = 32
);
```

Figure 10: Steps 5-7: PASETO Version 4 encryption

5.3.2. PASETO.v4.Decrypt

Given a message m, key k, and optional footer f.

1. Before decrypting, first assert that the key being used is intended for use with v4.local tokens. If this assertion fails, abort decryption.

2. If f is not empty, implementations **MAY** verify that the value appended to the token matches some expected string f, provided they do so using a constant-time string compare function.

3. Verify that the message begins with v4.local., otherwise throw an exception. This constant will be referred to as h.

4. Decode the payload (m sans h, f, and the optional trailing period between m and f) from base64url to raw binary. Set:

    *n to the leftmost 32 bytes

    *c to the middle remainder of the payload, excluding n.

5. Split the key into an Encryption key (Ek) and Authentication key (Ak), using keyed BLAKE2b, using the domain separation constants and n as the message, and the input key as the key. The first value will be 56 bytes, the second will be 32 bytes. The derived key will be the leftmost 32 bytes of the hash output. The remaining 24 bytes will be used as a counter nonce (n2)

6. Pack h, n, c, f, and i together (in that order) using PAE (see Section 2.4.1). We'll call this preAuth.

7. Re-calculate BLAKE2b-MAC of the output of preAuth, using Ak as the authentication key. We'll call this t2.

8. Compare t with t2 using a constant-time string compare function. If they are not identical, throw an exception.

    *You **MUST** use a constant-time string compare function to be compliant. If you do not have one available to you in your programming language/framework, you MUST use Double HMAC.

9. Decrypt c using XChaCha20, store the result in p.

10. If decryption failed, throw an exception. Otherwise, return p.

```
tmp = crypto_generichash(
    msg = "paseto-encryption-key" || n,
    key = key,
    length = 56
);
Ek = tmp[0:32]
n2 = tmp[32:]
Ak = crypto_generichash(
    msg = "paseto-auth-key-for-aead" || n,
    key = key,
    length = 32
);
```

Figure 11: Step 4: Key splitting with BLAKE2b.

```
preAuth = PASETO.PAE(h, n, c, f, i)
t2 = crypto_generichash(
    message = preAuth
    key = Ak,
    length = 32
);
if (not constant_time_compare(t2, t)) {
    throw new Exception("Invalid auth tag");
}
p = crypto_stream_xchacha20_xor(
   ciphertext = c
   nonce = n2
   key = Ek
);
```

Figure 12: Steps 5-8: PASETO v4 decryption

5.3.3. PASETO.v4.Sign

Given a message m, Ed25519 secret key sk, and optional footer f (which defaults to empty string):

1. Before signing, first assert that the key being used is intended for use with v4.public tokens, and is a secret key (not a public key). If this assertion fails, abort signing.

2. Set h to v4.public.

3. Pack h, m, f, and i together (in that order) using PAE (see Section 2.4.1). We'll call this m2.

4. Sign m2 using Ed25519 sk. We'll call this sig. (See below for pseudocode.)

5. If f is:

    *Empty: return h || b64(m || sig)

    *Non-empty: return h || b64(m || sig) || . || b64(f)

    *...where || means "concatenate"

```
m2 = PASETO.PAE(h, m, f, i);
sig = crypto_sign_detached(
    message = m2,
    private_key = sk
);
```

Figure 13: Step 4: Generating an Ed25519 with libsodium

5.3.4. PASETO.v4.Verify

Given a signed message sm, public key pk, and optional footer f (which defaults to empty string), and an optional implicit assertion i (which defaults to empty string):

1. Before verifying, first assert that the key being used is intended for use with v4.public tokens, and is a public key (not a secret key). If this assertion fails, abort verifying.

2. If f is not empty, implementations **MAY** verify that the value appended to the token matches some expected string f, provided they do so using a constant-time string compare function.

3. Verify that the message begins with v4.public., otherwise throw an exception. This constant will be referred to as h.

4. Decode the payload (sm sans h, f, and the optional trailing period between m and f) from base64url to raw binary. Set:

   *s to the rightmost 64 bytes

   *m to the leftmost remainder of the payload, excluding s

5. Pack h, m, f, and i together (in that order) using PAE (see Section 2.4.1). We'll call this m2.

6. Use Ed25519 to verify that the signature is valid for the message: (See below for pseudocode.)

7. If the signature is valid, return m. Otherwise, throw an exception.

```
m2 = PASETO.PAE(h, m, f, i);
valid = crypto_sign_verify_detached(
    signature = s,
    message = m2,
    public_key = pk
);
```

Figure 14: Steps 5-6: Validating the Ed25519 signature using libsodium.

6. Payload Processing

All PASETO payloads **MUST** be a JSON object [RFC8259].

If non-UTF-8 character sets are desired for some fields, implementors are encouraged to use Base64url encoding to preserve the original intended binary data, but still use UTF-8 for the actual payloads.

6.1. Type Safety with Cryptographic Keys

PASETO library implementations **MUST** implement some means of preventing type confusion bugs between different cryptography keys. For example:

- Prepending each key in memory with a magic byte to serve as a type indicator (distinct for every combination of version and purpose).
- In object-oriented programming languages, using separate classes for each cryptography key object that may share an interface or common base class.

Cryptographic keys **MUST** require the user to state a version and a purpose for which they will be used. Furthermore, given a cryptographic key, it **MUST NOT** be possible for a user to use this key for any version and purpose combination other than that which was specified during the creation of this key.

6.2. Registered Claims

6.2.1. Payload Claims

The following keys are reserved for use within PASETO payloads. Users **MUST NOT** write arbitrary/invalid data to any keys in a top-level PASETO in the list below:

Key	Name	Type	Example
iss	Issuer	string	{"iss":"paragonie.com"}
sub	Subject	string	{"sub":"test"}
aud	Audience	string	{"aud":"pie-hosted.com"}
exp	Expiration	DtTime	{"exp":"2039-01-01T00:00:00+00:00"}
nbf	Not Before	DtTime	{"nbf":"2038-04-01T00:00:00+00:00"}
iat	Issued At	DtTime	{"iat":"2038-03-17T00:00:00+00:00"}
jti	Token ID	string	{"jti":"87IFSGFgPNtQNNuw0AtuLttP"}

Table 1

In the table above, DtTime means an ISO 8601 compliant DateTime string.

Any other claims can be freely used. These keys are only reserved in the top-level JSON object.

The keys in the above table are case-sensitive.

Implementors (i.e. library designers) **SHOULD** provide some means to discourage setting invalid/arbitrary data to these reserved claims.

For example: Storing any string that isn't a valid ISO 8601 DateTime in the exp claim should result in an exception or error state (depending on the programming language in question).

6.2.2. Optional Footer Claims

The optional footer **MAY** contain an optional JSON object [RFC8259]. It does not have to be JSON, but if it is, implementations **MUST** implement the safety controls in [#json-handling]. If the optional footer does contain JSON, the following claims may be stored in the footer.

Users SHOULD NOT write arbitrary/invalid data to any keys in a top-level PASETO footer in the list below:

Key	Name	Type	Example
kid	Key ID	string	{"kid":"k4.lid.iVtYQDjr5gEijCSjJC3fQaJm7nCeQSeaty0Jixy8dbsk"}
wpk	Wrapped PASERK	string	{"wpk":"k4.local-wrap.pie.pu-fBxw... (truncated) ...0eo8iCS"}

Table 2

Any other claims can be freely used. These keys are only reserved in the top-level JSON object (if the footer contains a JSON object).

The keys in the above table are case-sensitive.

Implementors SHOULD provide some means to discourage setting invalid/arbitrary data to these reserved claims.

6.2.3. Key-ID Support

Some systems need to support key rotation, but since the payloads of a *local* token are always encrypted, it is impractical to store the key id in the payload.

Instead, users should store Key-ID claims (*kid*) in the unencrypted footer.

For example, a footer of {"kid":"gandalf0"} can be read without needing to first decrypt the token (which would in turn allow the user to know which key to use to decrypt the token).

Implementations **SHOULD** provide a means to extract the footer from a PASETO before authentication and decryption. This is possible for *local* tokens because the contents of the footer are *not* encrypted. However, the authenticity of the footer is only assured after the authentication tag is verified.

While a key identifier can generally be safely used for selecting the cryptographic key used to decrypt and/or verify payloads before verification, provided that the *kid* is a public number that is associated with a particular key which is not supplied by attackers, any other fields stored in the footer **MUST** be distrusted until the payload has been verified.

IMPORTANT: Key identifiers **MUST** be independent of the actual keys used by PASETO.

A fingerprint of the key is allowed as long as it is impractical for an attacker to recover the key from said fingerprint.

For example, the user **MUST NOT** store the public key in the footer for a **public** token and have the recipient use the provided public key. Doing so would allow an attacker to replace the public key with one of their own choosing, which will cause the recipient to accept any signature for any message as valid, therefore defeating the security goals of public-key cryptography.

Instead, it's recommended that implementors and users use a unique identifier for each key (independent of the cryptographic key's contents) that is used in a database or other key-value store to select the appropriate cryptographic key. These search operations **MUST** fail closed if no valid key is found for the given key identifier.

6.3. Optional Footer

PASETO places no restrictions on the contents of the authenticated footer. The footer's contents **MAY** be JSON-encoded (as is the payload), but it doesn't have to be.

The footer contents is intended to be free-form and application-specific.

6.3.1. Storing JSON in the Footer

Implementations that allow users to store JSON-encoded objects in the footer **MUST** give users some mechanism to validate the footer before decoding.

Some example parser rules include:

1. Enforcing a maximum length of the JSON-encoded string.

2. Enforcing a maximum depth of the decoded JSON object. (Recommended default: Only 1-dimensional objects.)

3. Enforcing the maximum number of named keys within an object.

The motivation for these additional rules is to mitigate the following security risks:

1. Stack overflows in JSON parsers caused by too much recursion.

2. Denial-of-Service attacks enabled by hash-table collisions.

6.3.1.1. Enforcing Maximum Depth Without Parsing the JSON String

Arbitrary-depth JSON strings can be a risk for stack overflows in some JSON parsing libraries. One mitigation to this is to enforce an upper limit on the maximum stack depth. Some JSON libraries do not allow you to configure this upper limit, so you're forced to take matters into your own hands.

A simple way of enforcing the maximum depth of a JSON string without having to parse it with your JSON library is to employ the following algorithm:

1. Create a copy of the JSON string with all \" sequences and whitespace characters removed. This will prevent weird edge cases in step 2.

2. Use a regular expression to remove all quoted strings and their contents. For example, replacing /"[^"]+?"([:,\}\]])/ with the first match will strip the contents of any quoted strings.

3. Remove all characters except [, {, }, and].

4. If you're left with an empty string, return 1.

5. Initialize a variable called depth to 1.

6. While the stripped variable is not empty **and** not equal to the output of the previous iteration, remove all {} and [] pairs, then increment depth.

7. If you end up with a non-empty string, you know you have invalid JSON: Either you have a [that isn't paired with a], or a { that isn't paired with a }. Throw an exception.

8. Return depth.

An example of this logic implemented below:

```
function getJsonDepth(data: string): number {
    // Step 1
    let stripped = data.replace(/\\"/g, '').replace(/\s+/g, '');

    // Step 2
    stripped = stripped.replace(/"[^"]+"([:,\}\]])/g, '$1');

    // Step 3
    stripped = stripped.replace(/[^\[\{\}\]]/g, '');

    // Step 4
    if (stripped.length === 0) {
        return 1;
    }
    // Step 5
    let previous = '';
    let depth = 1;

    // Step 6
    while (stripped.length > 0 && stripped !== previous) {
        previous = stripped;
        stripped = stripped.replace(/({}|\[\])/g, '');
        depth++;
    }

    // Step 7
    if (stripped.length > 0) {
        throw new Error(`Invalid JSON string`);
    }

    // Step 8
    return depth;
}
```

Figure 15: JSON Depth Calculation

6.3.1.2. Enforcing Maximum Key Count Without Parsing the JSON String

Hash-collision Denial of Service attacks (Hash-DoS) is made possible by creating a very large number of keys that will hash to the same value, with a given hash function (e.g., djb33).

One mitigation strategy is to limit the number of keys contained within an object (at any arbitrary depth).

The easiest way is to count the number of times you encounter a ": token that isn't followed by a backslash (to side-step corner-cases where JSON is encoded as a string inside a JSON value).

```
/**
 * Split the string based on the number of `":` pairs without a precedin
 * backslash, then return the number of pieces it was broken into.
 */
function countKeys(json: string): number {
    return json.split(/[^\\]":/).length;
}
```

Figure 16: Counting the number of keys in a JSON object

6.4. Implicit Assertions

The Optional Footer Section 6.3 provides a mechanism for Key IDs (and therefore key rotation), and thus qualifies as additional authenticated data when using encryption (local tokens).

Implicit Assertions are an additional layer of additional authenticated data for a PASETO token. Unlike the optional footer, Implicit Assertions are never stored in the token payload. They are, however, passed as an input to PAE() Section 2.4.1 when minting or consuming a PASETO token.

Implicit Assertions are useful for cryptographically binding a PASETO token to a specific domain or context without increasing the size of the payload.

Additionally, Implicit Assertions can be used to bind a token to data too sensitive to disclose in the payload.

7. Intended Use-Cases for PASETO

Like JWTs, PASETOs are intended to be single-use tokens, as there is no built-in mechanism to prevent replay attacks within the token lifetime.

- **local** tokens are intended for tamper-resistant encrypted cookies or HTTP request parameters. A resonable example would be long-term authentication cookies which re-establish a new session cookie if a user checked the "remember me on this computer" box when authenticating. To accomplish this, the server would look use the jti claim in a database lookup to find the appropriate user to associate this session with. After each new browsing session, the jti would be rotated in the database and a fresh cookie would be stored in tbe browser.

- **public** tokens are intended for one-time authentication claims from a third party. For example, **public** PASETO would be suitable for a protocol like OpenID Connect.

8. Security Considerations

PASETO was designed in part to address known deficits of the JOSE standards that directly caused insecure implementations.

PASETO uses versioned protocols, rather than in-band negotiation, to prevent insecure algorithms from being selected. Mix-and-match is not a robust strategy for usable security engineering, especially when implementations have insecure default settings.

Cryptography keys in PASETO are defined as a tuple of (version, purpose, bytes) rather than merely (bytes). This implies that cryptography keys **MUST NOT** be used for a different version of PASETO, or for a different purpose (local, public).

If a severe security vulnerability is ever discovered in one of the specified versions, a new version of the protocol that is not affected should be decided by a team of cryptography engineers familiar with the vulnerability in question. This prevents users from having to rewrite and/or reconfigure their implementations to side-step the vulnerability.

PASETO implementors should only support the two most recent protocol versions (currently **PASETO Version 3** and **PASETO Version 4**) at any given time.

PASETO users should beware that, although footers are authenticated, they are never encrypted. Therefore, sensitive information **MUST NOT** be stored in a footer.

Furthermore, PASETO users should beware that, if footers are employed to implement Key Identification (**kid**), the values stored in the footer **MUST** be unrelated to the actual cryptographic key used in verifying the token as discussed in Section 6.2.3.

PASETO has no built-in mechanism to resist replay attacks within the token's lifetime. Users **SHOULD NOT** attempt to use PASETO to obviate the need for server-side data storage when designing web applications.

PASETO's cryptography features requires the availability of a secure random number generator, such as the getrandom(2) syscall on newer Linux distributions, /dev/urandom on most Unix-like systems, and CryptGenRandom on Windows computers.

The use of userspace pseudo-random number generators, even if seeded by the operating system's cryptographically secure pseudo-random number generator, is discouraged.

Implementors **MUST NOT** skip steps, although they **MAY** implement multiple steps in a single code statement.

The "Implicit Assertions" feature Section 6.4 is intended to provide a mechanism for additional authenticated data (AAD) that isn't stored in the token payload.

Applications may leverage this feature to bind tokens to a specific domain or context, but as it is not stored in the PASETO token, the application is solely responsible for managing this data. Failure to manage this state will result in authentication failures and could become a Denial of Service risk.

9. IANA Considerations

The IANA should reserve a new "PASETO Headers" registry for the purpose of this document and superseding RFCs.

This document defines a suite of string prefixes for PASETO tokens, called "PASETO Headers" (see Section 2), which consists of two parts:

- **version**, with values **v3**, **v4** defined above

*__purpose__, with the values of __local__ or __public__

These two values are concatenated with a single character separator, the ASCII period character ..

Initial values for the "PASETO Headers" registry are given below; future assignments are to be made through Expert Review [RFC8126], such as the CFRG.

Value	PASETO Header Meaning	Definition
v3.local	Version 3, local	Section 4.1
v3.public	Version 3, public	Section 4.2
v4.local	Version 4, local	Section 5.1
v4.public	Version 4, public	Section 5.2

Table 3: PASETO Headers and their respective meanings

Additionally, the IANA should reserve a new "PASETO Claims" registry.

Value	PASETO Claim Meaning
iss	Issuer
sub	Subject
aud	Audience
exp	Expiration
nbf	Not Before
iat	Issued At
jti	Token ID
kid	Key ID
wpk	Wrapped PASERK

Table 4

10. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3686] Housley, R., "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)", RFC 3686, DOI 10.17487/RFC3686, January 2004, <https://www.rfc-editor.org/info/rfc3686>.

[RFC4231] Nystrom, M., "Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512", RFC 4231, DOI 10.17487/RFC4231, December 2005, <https://www.rfc-editor.org/info/rfc4231>.

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

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6687] Tripathi, J., Ed., de Oliveira, J., Ed., and JP. Vasseur, Ed., "Performance Evaluation of the Routing Protocol for Low-Power and Lossy Networks (RPL)", RFC 6687, DOI 10.17487/RFC6687, October 2012, <https://www.rfc-editor.org/info/rfc6687>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

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

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

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

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

[RFC7539] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 7539, DOI 10.17487/RFC7539, May 2015, <https://www.rfc-editor.org/info/rfc7539>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8259] 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>.

Appendix A. PASETO Test Vectors

A.1. PASETO v3 Test Vectors

A.1.1. v3.local (Shared-Key Encryption) Test Vectors

A.1.1.1. Test Vector v3-E-1

```
Token:      v3.local.AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADbfcIUR
            X_0pVZVU1mAESUzrKZAsRm2EsD6yBoZYn6cpVZNzSJOhSDN-sRaWjfLU-y
            n9OJH1J_B8GKtOQ9gSQlb8yk9Iza7teRdkiR89ZFyvPPsVjjFiepFUVcMa
            -LP18zV77f_crJrVXWa5PDNRkCSeHfBBeg
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      00000000 00000000 00000000 00000000
            00000000 00000000 00000000 00000000
Payload:    {"data":"this is a secret message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail: no
```

A.1.1.2. Test Vector v3-E-2

Same as v3-E-1, but with a slightly different message.

```
Token:      v3.local.AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADbfcIUR
            X_0pVZVU1mAESUzrKZAqhWxBMDgyBoZYn6cpVZNzSJOhSDN-sRaWjfLU-y
            n9OJH1J_B8GKtOQ9gSQlb8yk9IzZfaZpReVpHlDSwfuygx1riVXYVs-Ujc
            rG_apl9oz3jCVmmJbRuKn5ZfD8mHz2db0A
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      00000000 00000000 00000000 00000000
            00000000 00000000 00000000 00000000
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail: no
```

A.1.1.3. Test Vector v3-E-3

```
Token:      v3.local.JvdVM1RIKh2R1HhGJ4VLjaa4BCp5ZlI8K0BOjbvn9_LwY78vQ
            nDait-Q-sjhF88dG2B0ROIIykcrGHn8wzPbTrqObHhyoKpjy3cwZQzLdiw
            RsdEK5SDvl02_HjWKJW2oqGMOQJlxnt5xyhQjFJomwnt7WW_7r2VT0G704
            ifult011-TgLCyQ2X8imQhniG_hAQ4BydM
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      26f75533 54482a1d 91d47846 27854b8d
            a6b8042a 7966523c 2b404e8d bbe7f7f2
Payload:    {"data":"this is a secret message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail: no
```

A.1.1.4. Test Vector v3-E-4

Same as v3-E-3, but with a slightly different message.

```
Token:      v3.local.JvdVM1RIKh2R1HhGJ4VLjaa4BCp5ZlI8K0BOjbvn9_LwY78vQ
            nDait-Q-sjhF88dG2B0X-4P3EcxGHn8wzPbTrqObHhyoKpjy3cwZQzLdiw
            RsdEK5SDvl02_HjWKJW2oqGMOQJlBZa_gOpVj4gv0M9lV6Pwjp8JS_MmaZ
            aTA1LLTULXybOBZ2S4xMbYqYmDRhh3IgEk
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      26f75533 54482a1d 91d47846 27854b8d
            a6b8042a 7966523c 2b404e8d bbe7f7f2
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail: no
```

A.1.1.5. Test Vector v3-E-5

```
Token:      v3.local.JvdVM1RIKh2R1HhGJ4VLjaa4BCp5ZlI8K0BOjbvn9_LwY78vQ
            nDait-Q-sjhF88dG2B0ROIIykcrGHn8wzPbTrqObHhyoKpjy3cwZQzLdiw
            RsdEK5SDvl02_HjWKJW2oqGMOQJlkYSIbXOgVuIQL65UMdW9WcjOpmqvjq
            D40NNzed-XPqn1T3w-bJvitYpUJL_rmihc.eyJraWQiOiJVYmtLOFk2aXY
            0R1poRnA2VHgzSVdMV0xmTlhTRXZKY2RUM3pkUjY1WVp4byJ9
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      26f75533 54482a1d 91d47846 27854b8d
            a6b8042a 7966523c 2b404e8d bbe7f7f2
Payload:    {"data":"this is a secret message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"UbkK8Y6iv4GZhFp6Tx3IWLWLfNXSEvJcdT3zdR65YZxo"}
Implicit:
ExpectFail: no
```

A.1.1.6. Test Vector v3-E-6

Same as v3-E-5, but with a slightly different message.

```
Token:      v3.local.JvdVM1RIKh2R1HhGJ4VLjaa4BCp5ZlI8K0BOjbvn9_LwY78vQ
            nDait-Q-sjhF88dG2B0X-4P3EcxGHn8wzPbTrqObHhyoKpjy3cwZQzLdiw
            RsdEK5SDvl02_HjWKJW2oqGMOQJmSeEMphEWHiwtDKJftg41O1F8Hat-8k
            Q82ZIAMFqkx9q5VkWlxZke9ZzMBbb3Znfo.eyJraWQiOiJVYmtLOFk2aXY
            0R1poRnA2VHgzSVdMV0xmTlhTRXZKY2RUM3pkUjY1WVp4byJ9
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      26f75533 54482a1d 91d47846 27854b8d
            a6b8042a 7966523c 2b404e8d bbe7f7f2
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"UbkK8Y6iv4GZhFp6Tx3IWLWLfNXSEvJcdT3zdR65YZxo"}
Implicit:
ExpectFail: no
```

A.1.1.7. Test Vector v3-E-7

```
Token:      v3.local.JvdVM1RIKh2R1HhGJ4VLjaa4BCp5ZlI8K0BOjbvn9_LwY78vQ
            nDait-Q-sjhF88dG2B0ROIIykcrGHn8wzPbTrqObHhyoKpjy3cwZQzLdiw
            RsdEK5SDvl02_HjWKJW2oqGMOQJkzWACWAIoVa0bz7EWSBoTEnS8MvGBYH
            Ho6t6mJunPrFR9JKXFCc0obwz5N-pxFLOc.eyJraWQiOiJVYmtLOFk2aXY
            0R1poRnA2VHgzSVdMV0xmTlhTRXZKY2RUM3pkUjY1WVp4byJ9
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      26f75533 54482a1d 91d47846 27854b8d
            a6b8042a 7966523c 2b404e8d bbe7f7f2
Payload:    {"data":"this is a secret message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"UbkK8Y6iv4GZhFp6Tx3IWLWLfNXSEvJcdT3zdR65YZxo"}
Implicit:   {"test-vector":"3-E-7"}
ExpectFail: no
```

A.1.1.8. Test Vector v3-E-8

Same as v3-E-7, but with a slightly different message and implicit assertion.

```
Token:      v3.local.JvdVM1RIKh2R1HhGJ4VLjaa4BCp5ZlI8K0BOjbvn9_LwY78vQ
            nDait-Q-sjhF88dG2B0X-4P3EcxGHn8wzPbTrqObHhyoKpjy3cwZQzLdiw
            RsdEK5SDvl02_HjWKJW2oqGMOQJmZHSSKYR6AnPYJV6gpHtx6dLakIG_AO
            Phu8vKexNyrv5_1qoom6_NaPGecoiz6fR8.eyJraWQiOiJVYmtLOFk2aXY
            0R1poRnA2VHgzSVdMV0xmTlhTRXZKY2RUM3pkUjY1WVp4byJ9
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      26f75533 54482a1d 91d47846 27854b8d
            a6b8042a 7966523c 2b404e8d bbe7f7f2
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"UbkK8Y6iv4GZhFp6Tx3IWLWLfNXSEvJcdT3zdR65YZxo"}
Implicit:   {"test-vector":"3-E-8"}
ExpectFail: no
```

A.1.1.9. Test Vector v3-E-9

```
Token:      v3.local.JvdVM1RIKh2R1HhGJ4VLjaa4BCp5ZlI8K0BOjbvn9_LwY78vQ
            nDait-Q-sjhF88dG2B0X-4P3EcxGHn8wzPbTrqObHhyoKpjy3cwZQzLdiw
            RsdEK5SDvl02_HjWKJW2oqGMOQJlk1nli0_wijTH_vCuRwckEDc82QWK8-
            lG2fT9wQF271sgbVRVPjm0LwMQZkvvamqU.YXJiaXRyYXJ5LXN0cmluZy1
            0aGF0LWlzbid0LWpzb24
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      26f75533 54482a1d 91d47846 27854b8d
            a6b8042a 7966523c 2b404e8d bbe7f7f2
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     arbitrary-string-that-isn't-json
Implicit:   {"test-vector":"3-E-9"}
ExpectFail: no
```

A.1.2. v3.public (Public-Key Authentication) Test Vectors

A.1.2.1. Test Vector v3-S-1

```
Token:       v3.public.eyJkYXRhIjoidGhpcyBpcyBhIHNpZ25lZCBtZXNzYWdlIiwi
             ZXhwIjoiMjAyMi0wMS0wMVQwMDowMDowMCswMDowMCJ9qqEwwrKHKi5lJ7
             b9MBKc0G4MGZy0ptUiMv3lAUAaz-JY_zjoqBSIxMxhfAoeNYiSyvfUErj7
             6KOPWm1OeNnBPkTSespeSXDGaDfxeIrl3bRrPEIy7tLwLAIsRzsXkfph
Secret key:  -----BEGIN EC PRIVATE KEY-----
             MIGkAgEBBDAgNHYJYHR3rKj7+8XmIYRV8xmWaXku+LRm+qh73Gd5gUTISN
             0DZh7tWsYkYTQM6pagBwYFK4EEACKhZANiAAT7y3xp7hxgV5vnozQTSHjZ
             xcW/NdVS2rY8AUA5ftFM72N9dyCSXERpnqMOcodMcvt8kgcrB8KcKee0HU
             23E79/s4CvEs8hBfnjSUd/gcAm08EjSIz06iWjrNy4NakxR3I=
             -----END EC PRIVATE KEY-----
Public Key:  -----BEGIN PUBLIC KEY-----
             MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE+8t8ae4cYFeb56M0E0h42cXFvz
             XVUtq2PAFAOX7RTO9jfXcgklxEaZ6jDnKHTHL7fJIHKwfCnCnntB1NtxO/
             f7OArxLPIQX540lHf4HAJtPBI0iM9Oolo6zcuDWpMUdy
             -----END PUBLIC KEY-----
Payload:     {"data":"this is a signed message",
             "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail:  no
```

A.1.2.2. Test Vector v3-S-2

```
Token:       v3.public.eyJkYXRhIjoidGhpcyBpcyBhIHNpZ25lZCBtZXNzYWdlIiwi
             ZXhwIjoiMjAyMi0wMS0wMVQwMDowMDowMCswMDowMCJ9ZWrbGZ6L0MDK72
             skosUaS0Dz7wJ_2bMcM6tOxFuCasO9GhwHrvvchqgXQNLQQyWzGC2wkr-V
             KII71AvkLpC8tJOrzJV1cap9NRwoFzbcXjzMZyxQ0wkshxZxx8ImmNWP.e
             yJraWQiOiJkWWtJU3lseFFlZWNFY0hFTGZ6Rjg4VVpyd2JMb2xOaUNkcHp
             VSEd3OVVxbiJ9
Secret key:  -----BEGIN EC PRIVATE KEY-----
             MIGkAgEBBDAgNHYJYHR3rKj7+8XmIYRV8xmWaXku+LRm+qh73Gd5gUTISN
             0DZh7tWsYkYTQM6pagBwYFK4EEACKhZANiAAT7y3xp7hxgV5vnozQTSHjZ
             xcW/NdVS2rY8AUA5ftFM72N9dyCSXERpnqMOcodMcvt8kgcrB8KcKee0HU
             23E79/s4CvEs8hBfnjSUd/gcAm08EjSIz06iWjrNy4NakxR3I=
             -----END EC PRIVATE KEY-----
Public Key:  -----BEGIN PUBLIC KEY-----
             MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE+8t8ae4cYFeb56M0E0h42cXFvz
             XVUtq2PAFAOX7RTO9jfXcgklxEaZ6jDnKHTHL7fJIHKwfCnCnntB1NtxO/
             f7OArxLPIQX540lHf4HAJtPBI0iM9Oolo6zcuDWpMUdy
             -----END PUBLIC KEY-----
Payload:     {"data":"this is a signed message",
             "exp":"2022-01-01T00:00:00+00:00"}
Footer:      {"kid":"dYkISylxQeecEcHELfzF88UZrwbLolNiCdpzUHGw9Uqn"}
Implicit:
ExpectFail:  no
```

A.1.2.3. Test Vector v3-S-3

```
Token:       v3.public.eyJkYXRhIjoidGhpcyBpcyBhIHNpZ25lZCBtZXNzYWdlIiwi
             ZXhwIjoiMjAyMi0wMS0wMVQwMDowMDowMCswMDowMCJ94SjWIbjmS7715G
             jLSnHnpJrC9Z-cnwK45dmvnVvCRQDCCKAXaKEopTajX0DKYx1Xqr6gcTdf
             qscLCAbiB4eOW9jlt-oNqdG8TjsYEi6aloBfTzF1DXff_45tFlnBukEX.e
             yJraWQiOiJkWWtJU3lseFFlZWNFY0hFTGZ6Rjg4VVpyd2JMb2xOaUNkcHp
             VSEd3OVVxbiJ9
Secret key:  -----BEGIN EC PRIVATE KEY-----
             MIGkAgEBBDAgNHYJYHR3rKj7+8XmIYRV8xmWaXku+LRm+qh73Gd5gUTISN
             0DZh7tWsYkYTQM6pagBwYFK4EEACKhZANiAAT7y3xp7hxgV5vnozQTSHjZ
             xcW/NdVS2rY8AUA5ftFM72N9dyCSXERpnqMOcodMcvt8kgcrB8KcKee0HU
             23E79/s4CvEs8hBfnjSUd/gcAm08EjSIz06iWjrNy4NakxR3I=
             -----END EC PRIVATE KEY-----
Public Key:  -----BEGIN PUBLIC KEY-----
             MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE+8t8ae4cYFeb56M0E0h42cXFvz
             XVUtq2PAFAOX7RTO9jfXcgklxEaZ6jDnKHTHL7fJIHKwfCnCnntB1NtxO/
             f7OArxLPIQX540lHf4HAJtPBI0iM9Oolo6zcuDWpMUdy
             -----END PUBLIC KEY-----
Payload:     {"data":"this is a signed message",
             "exp":"2022-01-01T00:00:00+00:00"}
Footer:      {"kid":"dYkISylxQeecEcHELfzF88UZrwbLolNiCdpzUHGw9Uqn"}
Implicit:    {"test-vector":"3-S-3"}
ExpectFail:  no
```

A.1.2.4. Test Vector v3-F-1

This test vector **MUST** fail, because the keys are not meant for local tokens.

```
Token:       v3.local.tthw-G1Da_BzYeMu_GEDp-IyQ7jzUCQHxCHRdDY6hQjKg6Cux
             ECXfjOzlmNgNJ-WELjN61gMDnldG9OLkr3wpxuqdZksCzH9Ul16t3pXCLG
             PoHQ9_l51NOqVmMLbFVZOPhsmdhef9RxJwmqvzQ_Mo_JkYRlrNA.YXJiaX
             RyYXJ5LXN0cmluZy10aGF0LWlzbid0LWpzb24
Secret key:  -----BEGIN EC PRIVATE KEY-----
             MIGkAgEBBDAgNHYJYHR3rKj7+8XmIYRV8xmWaXku+LRm+qh73Gd5gUTISN
             0DZh7tWsYkYTQM6pagBwYFK4EEACKhZANiAAT7y3xp7hxgV5vnozQTSHjZ
             xcW/NdVS2rY8AUA5ftFM72N9dyCSXERpnqMOcodMcvt8kgcrB8KcKee0HU
             23E79/s4CvEs8hBfnjSUd/gcAm08EjSIz06iWjrNy4NakxR3I=
             -----END EC PRIVATE KEY-----
Public Key:  -----BEGIN PUBLIC KEY-----
             MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE+8t8ae4cYFeb56M0E0h42cXFvz
             XVUtq2PAFAOX7RTO9jfXcgklxEaZ6jDnKHTHL7fJIHKwfCnCnntB1NtxO/
             f7OArxLPIQX540lHf4HAJtPBI0iM9Oolo6zcuDWpMUdy
             -----END PUBLIC KEY-----
Payload:
Footer:      arbitrary-string-that-isn't-json
Implicit:    {"test-vector":"3-F-1"}
ExpectFail:  YES
```

A.1.2.5. Test Vector v3-F-2

This test vector **MUST** fail, because the key is not meant for public tokens.

```
Token:       v3.public.eyJpbnZhbGlkIjoidGhpcyBzaG91bGQgbmV2ZXIgZGVjb2Rl
             In1hbzIBD_EU54TYDTvsN9bbCU1QPo7FDeIhijkkcB9BrVH73XyM3Wwvu1
             pJaGCOEc0R5DVe9hb1ka1cYBd0goqVHt0NQ2NhPtILz4W36eCCqyU4uV6x
             DMeLI8ni6r3GnaY.eyJraWQiOiJ6VmhNaVBCUDlmUmYyc25FY1Q3Z0ZUaW
             9lQTlDT2NOeTlEZmdMMVc2MGhhTiJ9
Key:         70717273 74757677 78797a7b 7c7d7e7f
             80818283 84858687 88898a8b 8c8d8e8f
Nonce:       df654812 bac49266 3825520b a2f6e67c
             f5ca5bdc 13d4e750 7a98cc4c 2fcc3ad8
Payload:
Footer:      {"kid":"zVhMiPBP9fRf2snEcT7gFTioeA9COcNy9DfgL1W60haN"}
Implicit:    {"test-vector":"3-F-2"}
ExpectFail:  YES
```

A.1.2.6. Test Vector v3-F-3

This test vector **MUST** fail, because token is Version 4 while we're operating in Version 3.

```
Token:      v4.local.1JgN1UG8TFAYS49qsx8rxlwh-9E4ONUm3slJXYi5EibmzxpF0
            Q-du6gakjuyKCBX8TvnSLOKqCPu8Yh3WSa5yJWigPy33z9XZTJF2HQ9wlL
            DPtVn_Mu1pPxkTU50ZaBKblJBufRA.YXJiaXRyYXJ5LXN0cmluZy10aGF0
            LWlzbid0LWpzb24
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      26f75533 54482a1d 91d47846 27854b8d
            a6b8042a 7966523c 2b404e8d bbe7f7f2
Payload:
Footer:     arbitrary-string-that-isn't-json
Implicit:   {"test-vector":"3-F-3"}
ExpectFail: YES
```

A.2. PASETO v4 Test Vectors

A.2.1. v4.local (Shared-Key Encryption) Test Vectors

A.2.1.1. Test Vector v4-E-1

```
Token:      v4.local.AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAr68P
            S4AXe7If_ZgesdkUMvSwscFlAl1pk5HC0e8kApeaqMfGo_7OpBnwJOAbY9
            V7WU6abu74MmcUE8YWAiaArVI8XJ5hOb_4v9RmDkneN0S92dx0OW4pgy7o
            mxgf3S8c3LlQg
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      00000000 00000000 00000000 00000000
            00000000 00000000 00000000 00000000
Payload:    {"data":"this is a secret message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail: no
```

A.2.1.2. Test Vector v4-E-2

Same as v4-E-1, but with a slightly different message.

```
Token:      v4.local.AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAr68P
            S4AXe7If_ZgesdkUMvS2csCgglvpk5HC0e8kApeaqMfGo_7OpBnwJOAbY9
            V7WU6abu74MmcUE8YWAiaArVI8XIemu9chy3WVKvRBfg6t8wwYHK0ArLxx
            fZP73W_vfwt5A
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      00000000 00000000 00000000 00000000
            00000000 00000000 00000000 00000000
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail: no
```

A.2.1.3. Test Vector v4-E-3

```
Token:      v4.local.32VIErrEkmY4JVILovbmfPXKW9wT1OdQepjMTC_MOtjA4kiqw
            7_tcaOM5GNEcnTxl60WkwMsYXw6FSNb_UdJPXjpzm0KW9ojM5f4O2mRvE2
            IcweP-PRdoHjd5-RHCiExR1IK6t6-tyebyWG6Ov7kKvBdkrrAJ837lKP3i
            Dag2hzUPHuMKA
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      df654812 bac49266 3825520b a2f6e67c
            f5ca5bdc 13d4e750 7a98cc4c 2fcc3ad8
Payload:    {"data":"this is a secret message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail: no
```

A.2.1.4. Test Vector v4-E-4

Same as v4-E-3, but with a slightly different message.

```
Token:      v4.local.32VIErrEkmY4JVILovbmfPXKW9wT1OdQepjMTC_MOtjA4kiqw
            7_tcaOM5GNEcnTxl60WiA8rd3wgFSNb_UdJPXjpzm0KW9ojM5f4O2mRvE2
            IcweP-PRdoHjd5-RHCiExR1IK6t4gt6TiLm55vIH8c_lGxxZpE3AWlH4WT
            R0v45nsWoU3gQ
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      df654812 bac49266 3825520b a2f6e67c
            f5ca5bdc 13d4e750 7a98cc4c 2fcc3ad8
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail: no
```

A.2.1.5. Test Vector v4-E-5

```
Token:      v4.local.32VIErrEkmY4JVILovbmfPXKW9wT1OdQepjMTC_MOtjA4kiqw
            7_tcaOM5GNEcnTxl60WkwMsYXw6FSNb_UdJPXjpzm0KW9ojM5f4O2mRvE2
            IcweP-PRdoHjd5-RHCiExR1IK6t4x-RMNXtQNbz7FvFZ_G-lFpk5RG3EOr
            wDL6CgDqcerSQ.eyJraWQiOiJ6VmhNaVBCUDlmUmYyc25FY1Q3Z0ZUaW9l
            QTlDT2NOeTlEZmdMMVc2MGhhTiJ9
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      df654812 bac49266 3825520b a2f6e67c
            f5ca5bdc 13d4e750 7a98cc4c 2fcc3ad8
Payload:    {"data":"this is a secret message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"zVhMiPBP9fRf2snEcT7gFTioeA9COcNy9DfgL1W60haN"}
Implicit:
ExpectFail: no
```

A.2.1.6. Test Vector v4-E-6

Same as v4-E-5, but with a slightly different message.

```
Token:      v4.local.32VIErrEkmY4JVILovbmfPXKW9wT1OdQepjMTC_MOtjA4kiqw
            7_tcaOM5GNEcnTxl60WiA8rd3wgFSNb_UdJPXjpzm0KW9ojM5f4O2mRvE2
            IcweP-PRdoHjd5-RHCiExR1IK6t6pWSA5HX2wjb3P-xLQg5K5feUCX4P2f
            pVK3ZLWFbMSxQ.eyJraWQiOiJ6VmhNaVBCUDlmUmYyc25FY1Q3Z0ZUaW9l
            QTlDT2NOeTlEZmdMMVc2MGhhTiJ9
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      df654812 bac49266 3825520b a2f6e67c
            f5ca5bdc 13d4e750 7a98cc4c 2fcc3ad8
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"zVhMiPBP9fRf2snEcT7gFTioeA9COcNy9DfgL1W60haN"}
Implicit:
ExpectFail: no
```

A.2.1.7. Test Vector v4-E-7

```
Token:      v4.local.32VIErrEkmY4JVILovbmfPXKW9wT1OdQepjMTC_MOtjA4kiqw
            7_tcaOM5GNEcnTxl60WkwMsYXw6FSNb_UdJPXjpzm0KW9ojM5f4O2mRvE2
            IcweP-PRdoHjd5-RHCiExR1IK6t40KCCWLA7GYL9KFHzKlwY9_RnIfRrMQ
            pueydLEAZGGcA.eyJraWQiOiJ6VmhNaVBCUDlmUmYyc25FY1Q3Z0ZUaW9l
            QTlDT2NOeTlEZmdMMVc2MGhhTiJ9
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      df654812 bac49266 3825520b a2f6e67c
            f5ca5bdc 13d4e750 7a98cc4c 2fcc3ad8
Payload:    {"data":"this is a secret message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"zVhMiPBP9fRf2snEcT7gFTioeA9COcNy9DfgL1W60haN"}
Implicit:   {"test-vector":"4-E-7"}
ExpectFail: no
```

A.2.1.8. Test Vector v4-E-8

Same as v4-E-7, but with a slightly different message and implicit assertion.

```
Token:      v4.local.32VIErrEkmY4JVILovbmfPXKW9wT1OdQepjMTC_MOtjA4kiqw
            7_tcaOM5GNEcnTxl60WiA8rd3wgFSNb_UdJPXjpzm0KW9ojM5f4O2mRvE2
            IcweP-PRdoHjd5-RHCiExR1IK6t5uvqQbMGlLLNYBc7A6_x7oqnpUK5WLv
            j24eE4DVPDZjw.eyJraWQiOiJ6VmhNaVBCUDlmUmYyc25FY1Q3Z0ZUaW9l
            QTlDT2NOeTlEZmdMMVc2MGhhTiJ9
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      df654812 bac49266 3825520b a2f6e67c
            f5ca5bdc 13d4e750 7a98cc4c 2fcc3ad8
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"zVhMiPBP9fRf2snEcT7gFTioeA9COcNy9DfgL1W60haN"}
Implicit:   {"test-vector":"4-E-8"}
ExpectFail: no
```

A.2.1.9. Test Vector v4-E-9

```
Token:      v4.local.32VIErrEkmY4JVILovbmfPXKW9wT1OdQepjMTC_MOtjA4kiqw
            7_tcaOM5GNEcnTxl60WiA8rd3wgFSNb_UdJPXjpzm0KW9ojM5f4O2mRvE2
            IcweP-PRdoHjd5-RHCiExR1IK6t6tybdlmnMwcDMw0YxA_gFSE_IUWl78a
            MtOepFYSWYfQA.YXJiaXRyYXJ5LXN0cmluZy10aGF0LWlzbid0LWpzb24
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      df654812 bac49266 3825520b a2f6e67c
            f5ca5bdc 13d4e750 7a98cc4c 2fcc3ad8
Payload:    {"data":"this is a hidden message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     arbitrary-string-that-isn't-json
Implicit:   {"test-vector":"4-E-9"}
ExpectFail: no
```

A.2.2. v4.public (Public-Key Authentication) Test Vectors

A.2.2.1. Test Vector v4-S-1

```
Token:      v4.public.eyJkYXRhIjoidGhpcyBpcyBhIHNpZ25lZCBtZXNzYWdlIiwi
            ZXhwIjoiMjAyMi0wMS0wMVQwMDowMDowMCswMDowMCJ9bg_XBBzds8lTZS
            hVlwwKSgeKpLT3yukTw6JUz3W4h_ExsQV-P0V54zemZDcAxFaSeef1QlXE
            FtkqxT1ciiQEDA
Secret Key: b4cbfb43 df4ce210 727d953e 4a713307
            fa19bb7d 9f850414 38d9e11b 942a3774
            1eb9dbbb bc047c03 fd70604e 0071f098
            7e16b28b 757225c1 1f00415d 0e20b1a2
Public Key: 1eb9dbbb bc047c03 fd70604e 0071f098
            7e16b28b 757225c1 1f00415d 0e20b1a2
Payload:    {"data":"this is a signed message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:
Implicit:
ExpectFail: no
```

A.2.2.2. Test Vector v4-S-2

```
Token:      v4.public.eyJkYXRhIjoidGhpcyBpcyBhIHNpZ25lZCBtZXNzYWdlIiwi
            ZXhwIjoiMjAyMi0wMS0wMVQwMDowMDowMCswMDowMCJ9v3Jt8mx_TdM2ce
            TGoqwrh4yDFn0XsHvvV_D0DtwQxVrJEBMl0F2caAdgnpKlt4p7xBnx1HcO
            -SPo8FPp214HDw.eyJraWQiOiJ6VmhNaVBCUDlmUmYyc25FY1Q3Z0ZUaW9
            lQTlDT2NOeTlEZmdMMVc2MGhhTiJ9
Secret Key: b4cbfb43 df4ce210 727d953e 4a713307
            fa19bb7d 9f850414 38d9e11b 942a3774
            1eb9dbbb bc047c03 fd70604e 0071f098
            7e16b28b 757225c1 1f00415d 0e20b1a2
Public Key: 1eb9dbbb bc047c03 fd70604e 0071f098
            7e16b28b 757225c1 1f00415d 0e20b1a2
Payload:    {"data":"this is a signed message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"zVhMiPBP9fRf2snEcT7gFTioeA9COcNy9DfgL1W60haN"}
Implicit:
ExpectFail: no
```

A.2.2.3. Test Vector v4-S-3

```
Token:      v4.public.eyJkYXRhIjoidGhpcyBpcyBhIHNpZ25lZCBtZXNzYWdlIiwi
            ZXhwIjoiMjAyMi0wMS0wMVQwMDowMDowMCswMDowMCJ9NPWciuD3d0o5eX
            JXG5pJy-DiVEoyPYWs1YSTwWHNJq6DZD3je5gf-0M4JR9ipdUSJbIovzmB
            ECeaWmaqcaP0DQ.eyJraWQiOiJ6VmhNaVBCUDlmUmYyc25FY1Q3Z0ZUaW9
            lQTlDT2NOeTlEZmdMMVc2MGhhTiJ9
Secret Key: b4cbfb43 df4ce210 727d953e 4a713307
            fa19bb7d 9f850414 38d9e11b 942a3774
            1eb9dbbb bc047c03 fd70604e 0071f098
            7e16b28b 757225c1 1f00415d 0e20b1a2
Public Key: 1eb9dbbb bc047c03 fd70604e 0071f098
            7e16b28b 757225c1 1f00415d 0e20b1a2
Payload:    {"data":"this is a signed message",
            "exp":"2022-01-01T00:00:00+00:00"}
Footer:     {"kid":"zVhMiPBP9fRf2snEcT7gFTioeA9COcNy9DfgL1W60haN"}
Implicit:   {"test-vector":"4-S-3"}
ExpectFail: no
```

A.2.2.4. Test Vector 4-F-1

This test vector **MUST** fail, because the keys are not meant for local tokens.

```
Token:      v4.local.vngXfCISbnKgiP6VWGuOSlYrFYU300fy9ijW33rznDYgxHNPw
            WluAY2Bgb0z54CUs6aYYkIJ-bOOOmJHPuX_34Agt_IPlNdGDpRdGNnBz2M
            pWJvB3cttheEc1uyCEYltj7wBQQYX.YXJiaXRyYXJ5LXN0cmluZy10aGF0
            LWlzbid0LWpzb24
Secret Key: b4cbfb43 df4ce210 727d953e 4a713307
            fa19bb7d 9f850414 38d9e11b 942a3774
            1eb9dbbb bc047c03 fd70604e 0071f098
            7e16b28b 757225c1 1f00415d 0e20b1a2
Public Key: 1eb9dbbb bc047c03 fd70604e 0071f098
            7e16b28b 757225c1 1f00415d 0e20b1a2
Payload:
Footer:     arbitrary-string-that-isn't-json
Implicit:   {"test-vector":"4-F-1"}
ExpectFail: YES
```

A.2.2.5. Test Vector 4-F-2

This test vector **MUST** fail, because the key is not meant for public tokens.

```
Token:      v4.public.eyJpbnZhbGlkIjoidGhpcyBzaG91bGQgbmV2ZXIgZGVjb2Rl
            In22Sp4gjCaUw0c7EH84ZSm_jN_Qr41MrgLNu5LIBCzUr1pn3Z-Wukg9h3
            ceplWigpoHaTLcwxj0NsI1vjTh67YB.eyJraWQiOiJ6VmhNaVBCUDlmUmY
            yc25FY1Q3Z0ZUaW9lQTlDT2NOeTlEZmdMMVc2MGhhTiJ9
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      df654812 bac49266 3825520b a2f6e67c
            f5ca5bdc 13d4e750 7a98cc4c 2fcc3ad8
Payload:
Footer:     {"kid":"zVhMiPBP9fRf2snEcT7gFTioeA9COcNy9DfgL1W60haN"}
Implicit:   {"test-vector":"4-F-2"}
ExpectFail: YES
```

A.2.2.6. Test Vector 4-F-3

This test vector **MUST** fail, because token is Version 3 while we're operating in Version 4.

```
Token:      v3.local.23e_2PiqpQBPvRFKzB0zHhjmxK3sKo2grFZRRLM-U7L0a8uHx
            uF9RlVz3Ic6WmdUUWTxCaYycwWV1yM8gKbZB2JhygDMKvHQ7eBf8GtF0r3
            K0Q_gF1PXOxcOgztak1eD1dPe9rLVMSgR0nHJXeIGYVuVrVoLWQ.YXJiaX
            RyYXJ5LXN0cmluZy10aGF0LWlzbid0LWpzb24
Key:        70717273 74757677 78797a7b 7c7d7e7f
            80818283 84858687 88898a8b 8c8d8e8f
Nonce:      26f75533 54482a1d 91d47846 27854b8d
            a6b8042a 7966523c 2b404e8d bbe7f7f2
Payload:
Footer:     arbitrary-string-that-isn't-json
Implicit:   {"test-vector":"4-F-3"}
ExpectFail: YES
```

Authors' Addresses

Robyn Terjesen
Paragon Initiative Enterprises
United States

Email: robyn@paragonie.com

Steven Haussmann
Rensselaer Polytechnic Institute
United States

Email: hausss@rpi.edu

Scott Arciszewski
Paragon Initiative Enterprises
United States

Email: security@paragonie.com