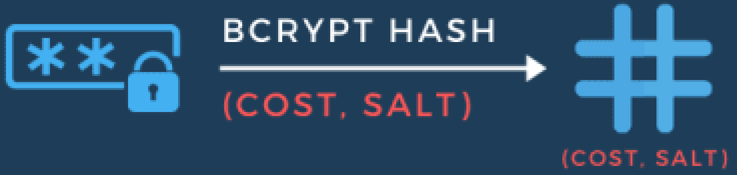
Securely store password?

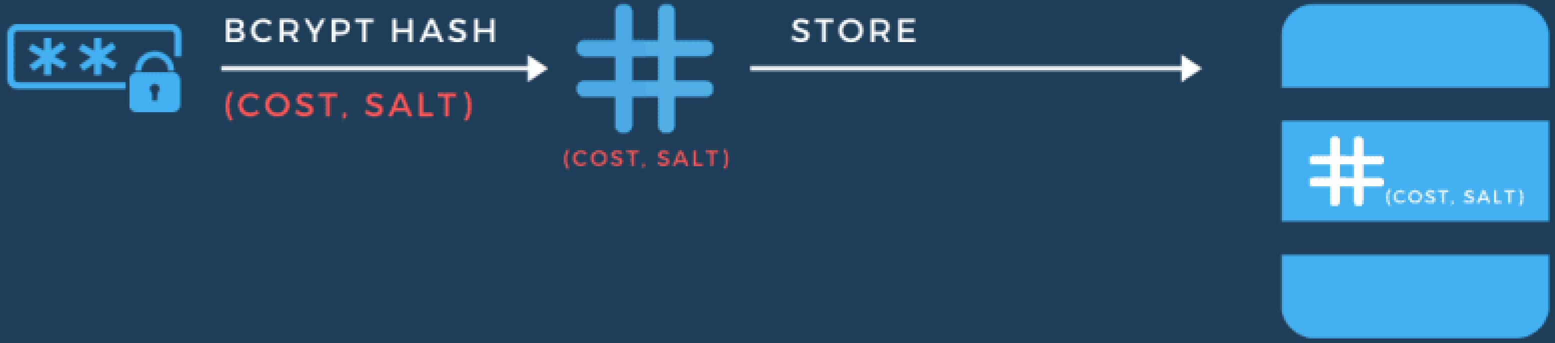
Hash it & store its hash value



```
        2^10 = 1024
        key expansion rounds
                    ↓
         $2A$10$N9QO8ULOICKGX2ZMRZOMYEIJZAGCFL7P92LDGXAD68LJZDL17LHWY
         \__/\/ \____________________/\_____________________________/
bcrypt ← ALG COST       SALT                       HASH
                         ↓                          ↓
              16 bytes = 128 bits         24 bytes = 192 bits
              22 characters (base64)      31 characters (base64)
```

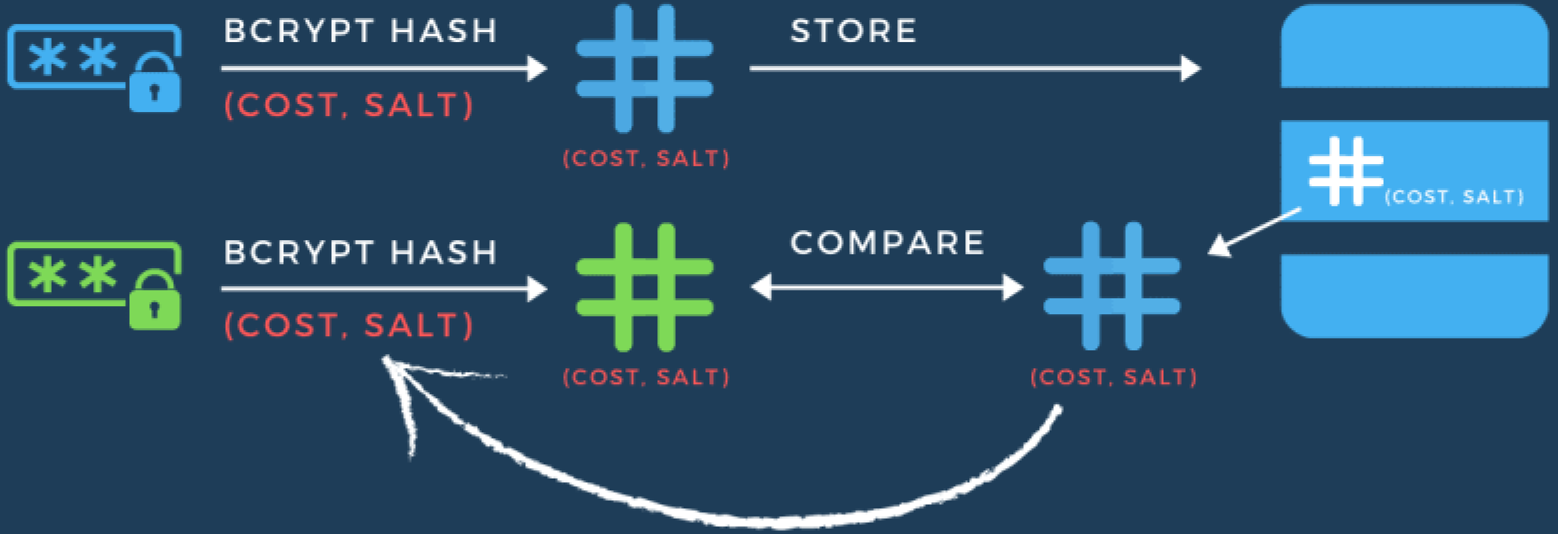
Securely store password?

Hash it & store its hash value

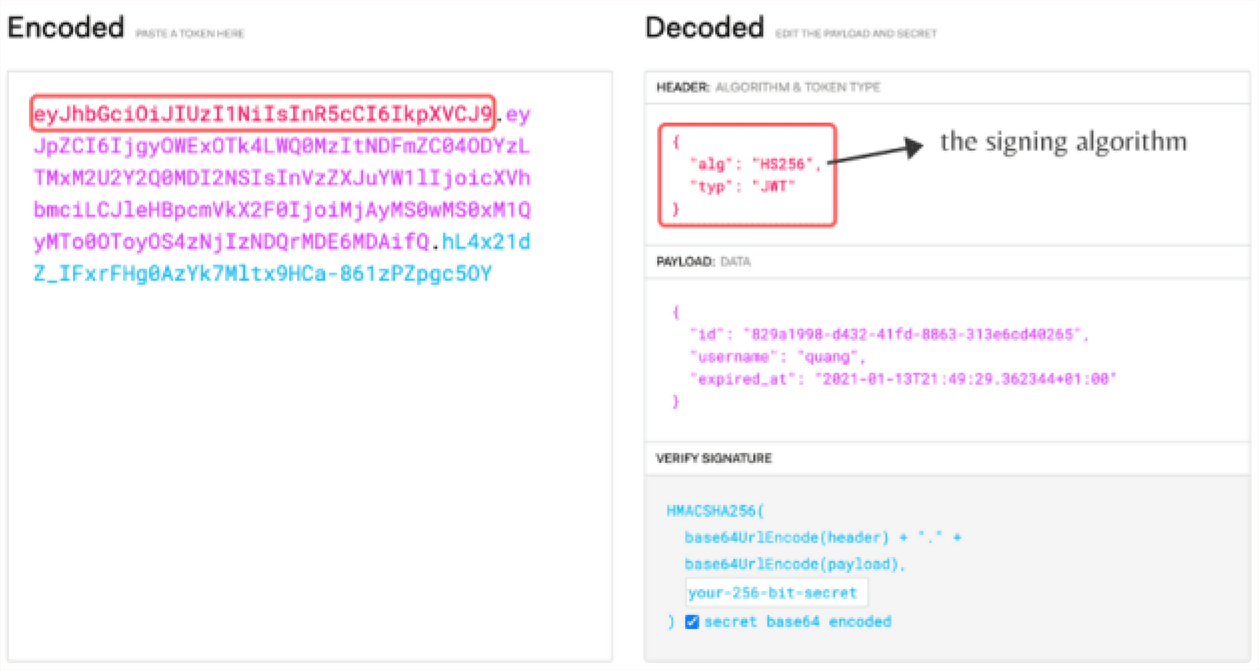


Securely store password?

Hash it & store its hash value



JSON Web Token - JWT



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JpZCI6IjgyOWExOTk4LWQ0MzItNDFmZC04ODYzL
TMxM2U2Y2Q0MDI2NSIsInVzZXJuYW1lIjoicXVh
bmciLCJleHBpcmVkX2F0IjoiMjAyMS0wMS0xM1Q
yMTo0OToyOS4zNjIzNDQrMDE6MDAifQ.hL4x21d
Z_IFxrFHg0AzYk7Mltx9HCa-861zPZpgc5OY
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

the signing algorithm

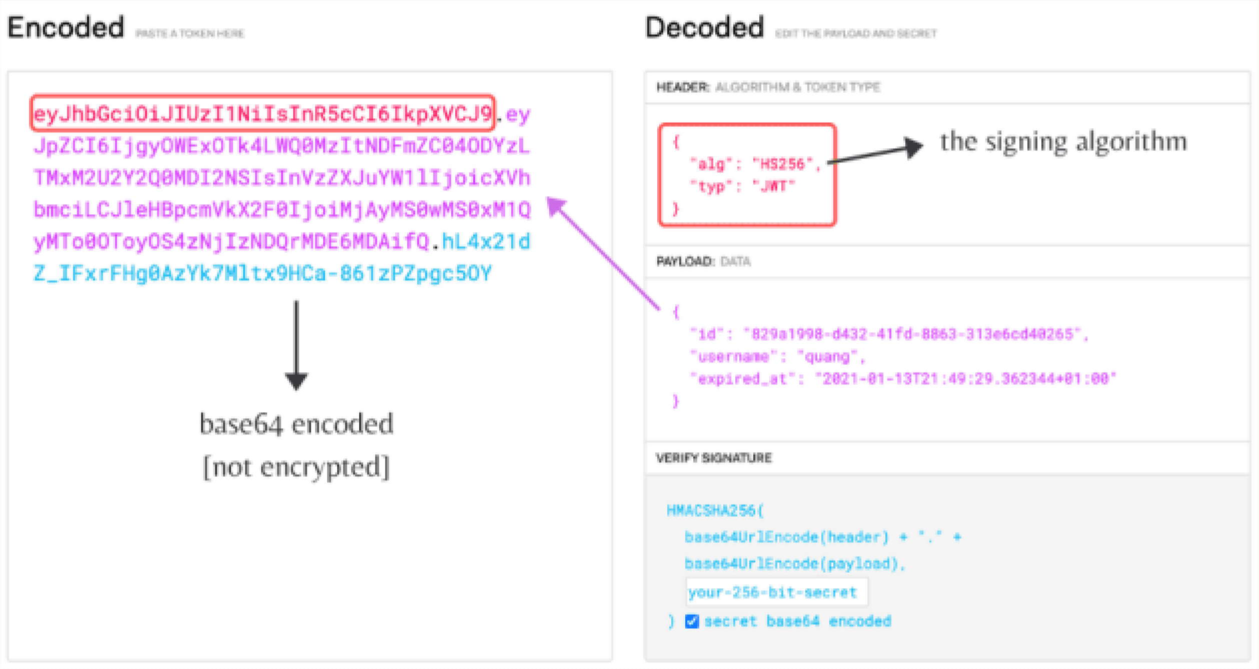
PAYLOAD: DATA

```
{
  "id": "829a1998-d432-41fd-8863-313e6cd40265",
  "username": "quang",
  "expired_at": "2021-01-13T21:49:29.362344+01:00"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☑ secret base64 encoded
```

JSON Web Token - JWT



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JpZCI6IjgyOWExOTk4LWQ0MzItNDFmZC04ODYzL
TMxM2U2Y2Q0MDI2NSIsInVzZXJuYW1lIjoicXVh
bmciLCJleHBpcmVkX2F0IjoiMjAyMS0wMS0xM1Q
yMTo0OToyOS4zNjIzNDQrMDE6MDAifQ.hL4x21d
Z_IFxrFHg0AzYk7Mltx9HCa-861zPZpgc5OY
```

base64 encoded
[not encrypted]

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

the signing algorithm

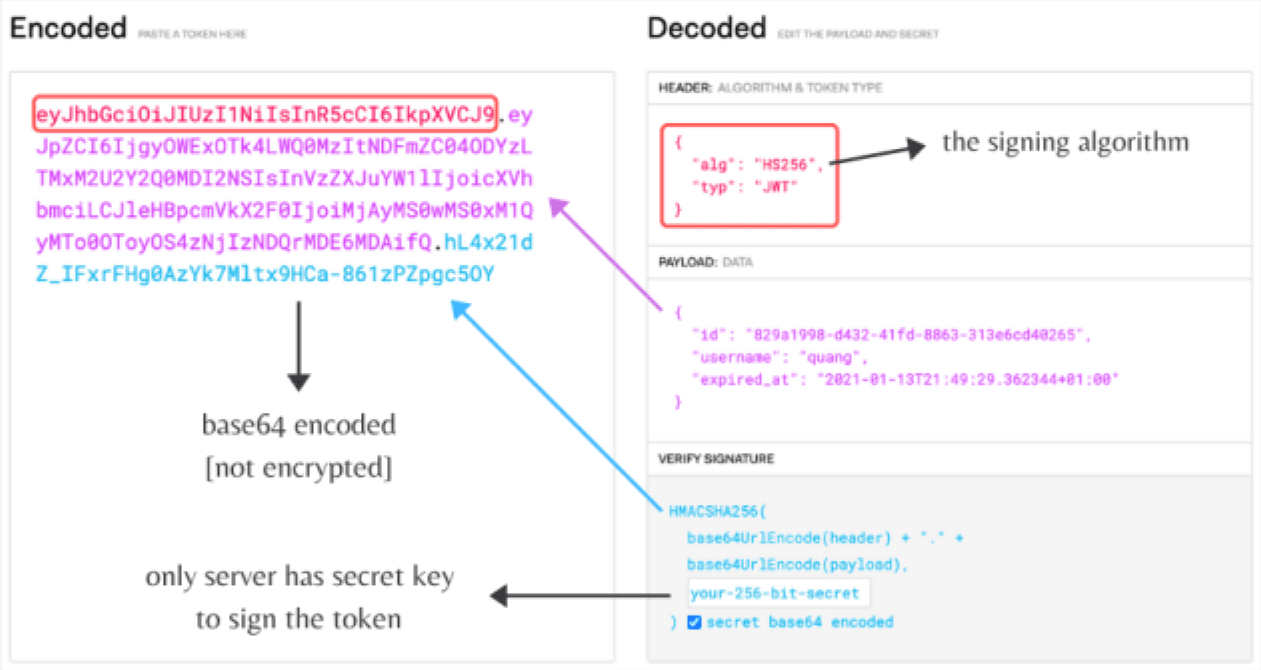
PAYLOAD: DATA

```
{
  "id": "829a1998-d432-41fd-8863-313e6cd40265",
  "username": "quang",
  "expired_at": "2021-01-13T21:49:29.362344+01:00"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☑ secret base64 encoded
```

JSON Web Token - JWT



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JpZCI6IjgyOWExOTk4LWQ0MzItNDFmZC04ODYzL
TMxM2U2Y2Q0MDI2NSIsInVzZXJuYW1lIjoicXVh
bmciLCJleHBpcmVkX2F0IjoiMjAyMS0wMS0xM1Q
yMTo0OToyOS4zNjIzNDQrMDE6MDAifQ.hL4x21d
Z_IFxrFHg0AzYk7Mltx9HCa-861zPZpgc5OY
```

base64 encoded
[not encrypted]

only server has secret key
to sign the token

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

the signing algorithm

PAYLOAD: DATA

```
{
  "id": "829a1998-d432-41fd-8863-313e6cd40265",
  "username": "quang",
  "expired_at": "2021-01-13T21:49:29.362344+01:00"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☑ secret base64 encoded
```

JWT SIGNING ALGORITHMS

Symmetric digital signature algorithm

- The same secret key is used to sign & verify token
- For local use: internal services, where the secret key can be shared
- HS256, HS384, HS512
  - HS256 = HMAC + SHA256
  - HMAC: Hash-based Message Authentication Code
  - SHA: Secure Hash Algorithm
  - 256/384/512: number of output bits

JWT SIGNING ALGORITHMS

Symmetric digital signature algorithm

- The same secret key is used to sign & verify token
- For local use: internal services, where the secret key can be shared
- HS256, HS384, HS512
    - HS256 = HMAC + SHA256
    - HMAC: Hash-based Message Authentication Code
    - SHA: Secure Hash Algorithm
    - 256/384/512: number of output bits

Asymmetric digital signature algorithm

- The private key is used to sign token
- The public key is used to verify token
- For public use: internal service signs token, but external service needs to verify it

JWT SIGNING ALGORITHMS

Symmetric digital signature algorithm

- The same secret key is used to sign & verify token
- For local use: internal services, where the secret key can be shared
- HS256, HS384, HS512
  - HS256 = HMAC + SHA256
  - HMAC: Hash-based Message Authentication Code
  - SHA: Secure Hash Algorithm
  - 256/384/512: number of output bits

Asymmetric digital signature algorithm

- The private key is used to sign token
- The public key is used to verify token
- For public use: internal service signs token, but external service needs to verify it
- RS256, RS384, RS512 || PS256, PS384, PS512 || ES256, ES384, ES512
  - RS256 = RSA PKCSv1.5 + SHA256 [PKCS: Public-Key Cryptography Standards]
  - PS256 = RSA PSS + SHA256 [PSS: Probabilistic Signature Scheme]
  - ES256 = ECDSA + SHA256 [ECDSA: Elliptic Curve Digital Signature Algorithm]

What's the problem of JWT?

Weak algorithms

- Give developers too many algorithms to choose
- Some algorithms are known to be vulnerable:
  - RSA PKCSv1.5: padding oracle attack
  - ECDSA: invalid-curve attack

Trivial Forgery

- Set "alg" header to "none"
- Set "alg" header to "HS256" while the server normally verifies token with a RSA public key

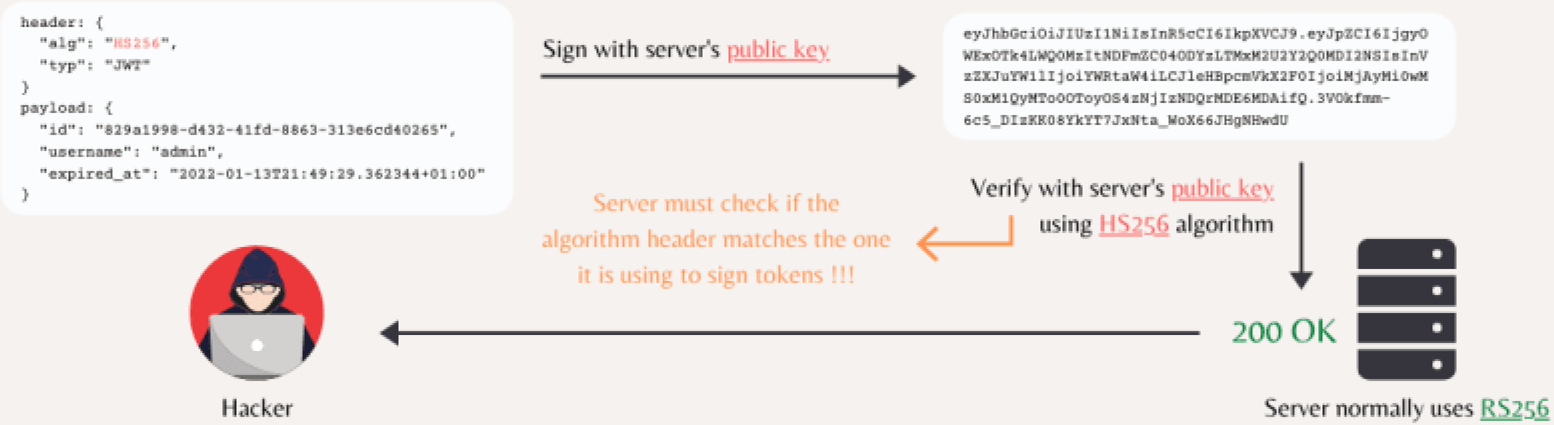
What's the problem of JWT?

Weak algorithms

- Give developers too many algorithms to choose
- Some algorithms are known to be vulnerable:
    - RSA PKCSv1.5: padding oracle attack
    - ECDSA: invalid-curve attack

Trivial Forgery

- Set "alg" header to "none"
- Set "alg" header to "HS256" while the server normally verifies token with a RSA public key



```
header: {
  "alg": "HS256",
  "typ": "JWT"
}
payload: {
  "id": "829a1998-d432-41fd-8863-313e6cd40265",
  "username": "admin",
  "expired_at": "2022-01-13T21:49:29.362344+01:00"
}
```

Sign with server's public key

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjgyO
WExOTk4LWQ0MzItNDFmZC04ODYzLTMxM2U2Y2Q0MDI2NSIsInV
zZXJuYW1lIjoiYWRtaW4iLCJleHBpcmVkX2F0IjoiMjAyMi0wM
S0xM1QyMTo0OToyOS4zNjIzNDQrMDE6MDAifQ.3VOkfmm-
6c5_DIzKK08YkYT7JxNta_WoX66JHgNHwdU
```

Verify with server's public key using HS256 algorithm

Server must check if the algorithm header matches the one it is using to sign tokens !!!

Server normally uses RS256

Platform-Agnostic SEcurity TOkens [PASETO]

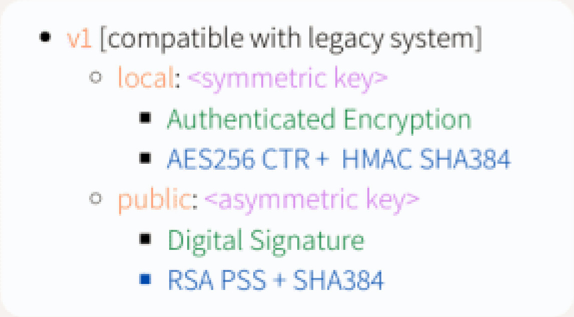
Stronger algorithms

- Developers don't have to choose the algorithm
- Only need to select the version of PASETO
- Each version has 1 strong cipher suite
- Only 2 most recent PASTO versions are accepted

Platform-Agnostic SEcurity TOkens [PASETO]

Stronger algorithms

- Developers don't have to choose the algorithm
- Only need to select the version of PASETO
- Each version has 1 strong cipher suite
- Only 2 most recent PASTO versions are accepted

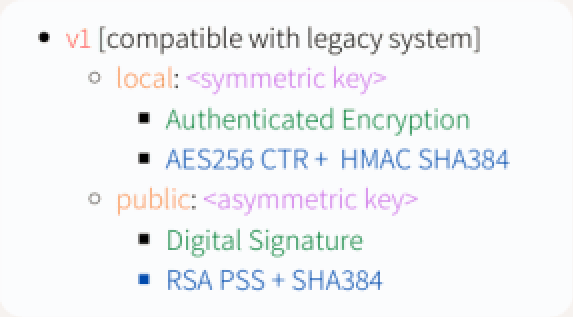


- v1 [compatible with legacy system]
  - local: <symmetric key>
    - Authenticated Encryption
    - AES256 CTR + HMAC SHA384
  - public: <asymmetric key>
    - Digital Signature
    - RSA PSS + SHA384

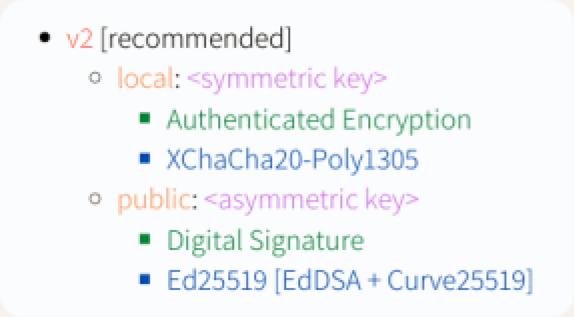
Platform-Agnostic SEcurity TOkens [PASETO]

Stronger algorithms

- Developers don't have to choose the algorithm
- Only need to select the version of PASETO
- Each version has 1 strong cipher suite
- Only 2 most recent PASTO versions are accepted



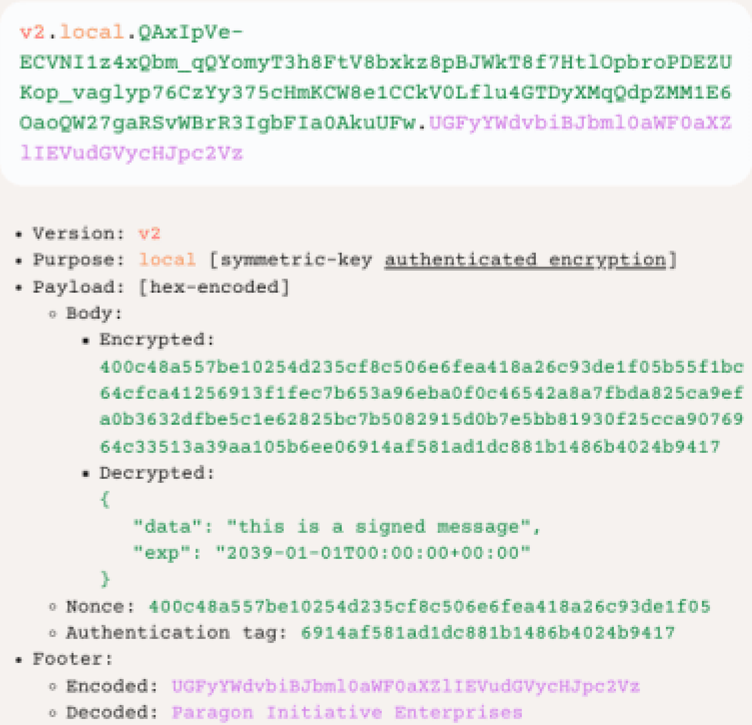
- v1 [compatible with legacy system]
  - local: <symmetric key>
    - Authenticated Encryption
    - AES256 CTR + HMAC SHA384
  - public: <asymmetric key>
    - Digital Signature
    - RSA PSS + SHA384



- v2 [recommended]
  - local: <symmetric key>
    - Authenticated Encryption
    - XChaCha20-Poly1305
  - public: <asymmetric key>
    - Digital Signature
    - Ed25519 [EdDSA + Curve25519]

PASETO STRUCTURE

Local



```
v2.local.QAxIpVe-
ECVNI1z4xQbm_qQYomyT3h8FtV8bxkz8pBJWkT8f7Htl0pbroPDEZU
Kop_vaglyp76CzYy375cHmKCW8e1CCkV0Lflu4GTDyXMqQdpZMM1E6
OaoQW27gaRSvWBrR3IgbFIa0AkuUFw.UGFyYWdvbiBJbml0aWF0aXZ
lIEVudGVycHJpc2Vz
```

- Version: v2
- Purpose: local [symmetric-key authenticated encryption]
- Payload: [hex-encoded]
  - Body:
    - Encrypted:
      400c48a557be10254d235cf8c506e6fea418a26c93de1f05b55f1bc64cfca41256913f1fec7b653a96eba0f0c46542a8a7fbda825ca9efa0b3632dfbe5c1e62825bc7b5082915d0b7e5bb81930f25cca9076964c33513a39aa105b6ee06914af581ad1dc881b1486b4024b9417
    - Decrypted:
      ```
      {
         "data": "this is a signed message",
         "exp": "2039-01-01T00:00:00+00:00"
      }
      ```
  - Nonce: 400c48a557be10254d235cf8c506e6fea418a26c93de1f05
  - Authentication tag: 6914af581ad1dc881b1486b4024b9417
- Footer:
  - Encoded: UGFyYWdvbiBJbml0aWF0aXZlIEVudGVycHJpc2Vz
  - Decoded: Paragon Initiative Enterprises

PASETO STRUCTURE

Local

```
v2.local.QAxIpVe-
ECVNI1z4xQbm_qQYomyT3h8FtV8bxkz8pBJWkT8f7Htl0pbroPDEZU
Kop_vaglyp76CzYy375cHmKCW8e1CCkV0Lflu4GTDyXMqQdpZMM1E6
OaoQW27gaRSvWBrR3IgbFIa0AkuUFw.UGFyYWdvbiBJbml0aWF0aXZ
lIEVudGVycHJpc2Vz
```

- Version: v2
- Purpose: local [symmetric-key authenticated encryption]
- Payload: [hex-encoded]
    - Body:
        - Encrypted:
          400c48a557be10254d235cf8c506e6fea418a26c93de1f05b55f1bc
          64cfca41256913f1fec7b653a96eba0f0c46542a8a7fbda825ca9ef
          a0b3632dfbe5c1e62825bc7b5082915d0b7e5bb81930f25cca90769
          64c33513a39aa105b6ee06914af581ad1dc881b1486b4024b9417
        - Decrypted:
          ```
          {
            "data": "this is a signed message",
            "exp": "2039-01-01T00:00:00+00:00"
          }
          ```
    - Nonce: 400c48a557be10254d235cf8c506e6fea418a26c93de1f05
    - Authentication tag: 6914af581ad1dc881b1486b4024b9417
- Footer:
    - Encoded: UGFyYWdvbiBJbml0aWF0aXZlIEVudGVycHJpc2Vz
    - Decoded: Paragon Initiative Enterprises

Public

```
v2.public.eyJleHAiOiIyMDM5LTAxLTAxVDAwOjAwOjAwKzAwOjAw
IiwiZGF0YSI6InRoaXMgaXMgYSBzaWduZWQgbWVzc2FnZSJ91gC7-
jCWsN3mv4uJaZxZp0btLJgcyVwL-
svJD7f4IHyGteKe3HTLjHYTGHI1MtCqJ-ESDLNoE7otkIzamFskCA
```

- Version: v2
- Purpose: public [asymmetric-key digital signature]

PASETO STRUCTURE

Local

```
v2.local.QAxIpVe-
ECVNI1z4xQbm_qQYomyT3h8FtV8bxkz8pBJWkT8f7Htl0pbroPDEZU
Kop_vaglyp76CzYy375cHmKCW8e1CCkV0Lflu4GTDyXMqQdpZMM1E6
OaoQW27gaRSvWBrR3IgbFIa0AkuUFw.UGFyYWdvbiBJbml0aWF0aXZ
lIEVudGVycHJpc2Vz
```

- Version: v2
- Purpose: local [symmetric-key authenticated encryption]
- Payload: [hex-encoded]
  - Body:
    - Encrypted:
      400c48a557be10254d235cf8c506e6fea418a26c93de1f05b55f1bc
      64cfca41256913f1fec7b653a96eba0f0c46542a8a7fbda825ca9ef
      a0b3632dfbe5c1e62825bc7b5082915d0b7e5bb81930f25cca90769
      64c33513a39aa105b6ee06914af581ad1dc881b1486b4024b9417
    - Decrypted:
      ```
      {
         "data": "this is a signed message",
         "exp": "2039-01-01T00:00:00+00:00"
      }
      ```
  - Nonce: 400c48a557be10254d235cf8c506e6fea418a26c93de1f05
  - Authentication tag: 6914af581ad1dc881b1486b4024b9417
- Footer:
  - Encoded: UGFyYWdvbiBJbml0aWF0aXZlIEVudGVycHJpc2Vz
  - Decoded: Paragon Initiative Enterprises

Public

```
v2.public.eyJleHAiOiIyMDM5LTAxLTAxVDAwOjAwOjAwKzAwOjAw
IiwiZGF0YSI6InRoaXMgaXMgYSBzaWduZWQgbWVzc2FnZSJ91gC7-
jCWsN3mv4uJaZxZp0btLJgcyVwL-
svJD7f4IHyGteKe3HTLjHYTGHI1MtCqJ-ESDLNoE7otkIzamFskCA
```

- Version: v2
- Purpose: public [asymmetric-key digital signature]
- Payload:
  - Body:
    - Encoded: [base64]
      eyJleHAiOiIyMDM5LTAxLTAxVDAwOjAwOjAwKzAwOjAwIiwiZGF0YSI
      6InRoaXMgaXMgYSBzaWduZWQgbWVzc2FnZSJ91g
    - Decoded:
      ```
      {
         "data": "this is a signed message",
         "exp": "2039-01-01T00:00:00+00:00"
      }
      ```
  - Signature: [hex-encoded]
    d600bbfa3096b0dde6bf8b89699c59a746ed2c981cc95c0bfacbc90fb7
    f8207c86b5e29edc74cb8c761318723532d0aa27e1120cb36813ba2d90
    8cda985b2408