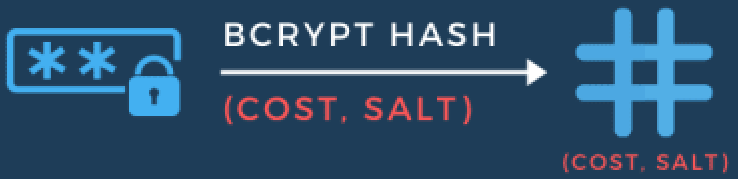
Securely store password?

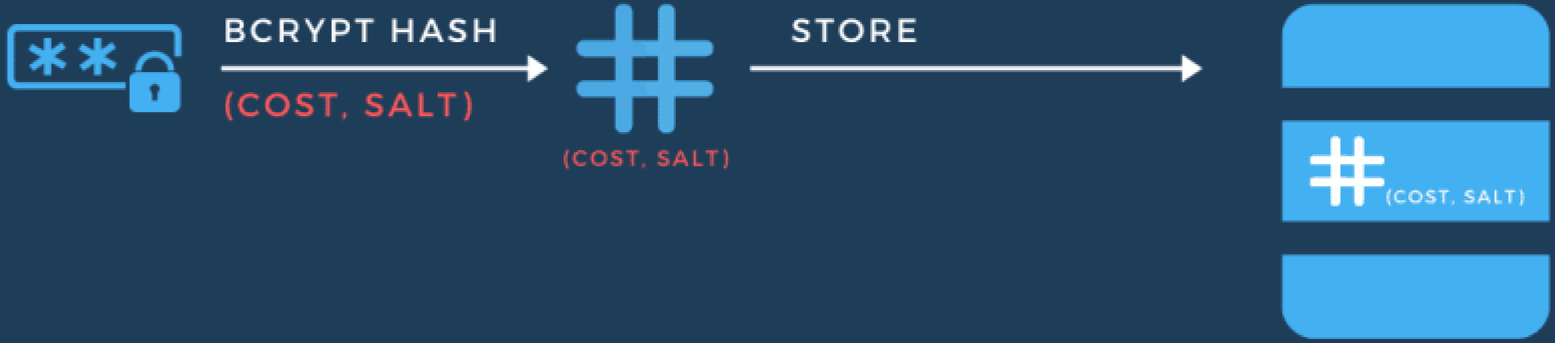
Hash it & store its hash value



```
   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)
```

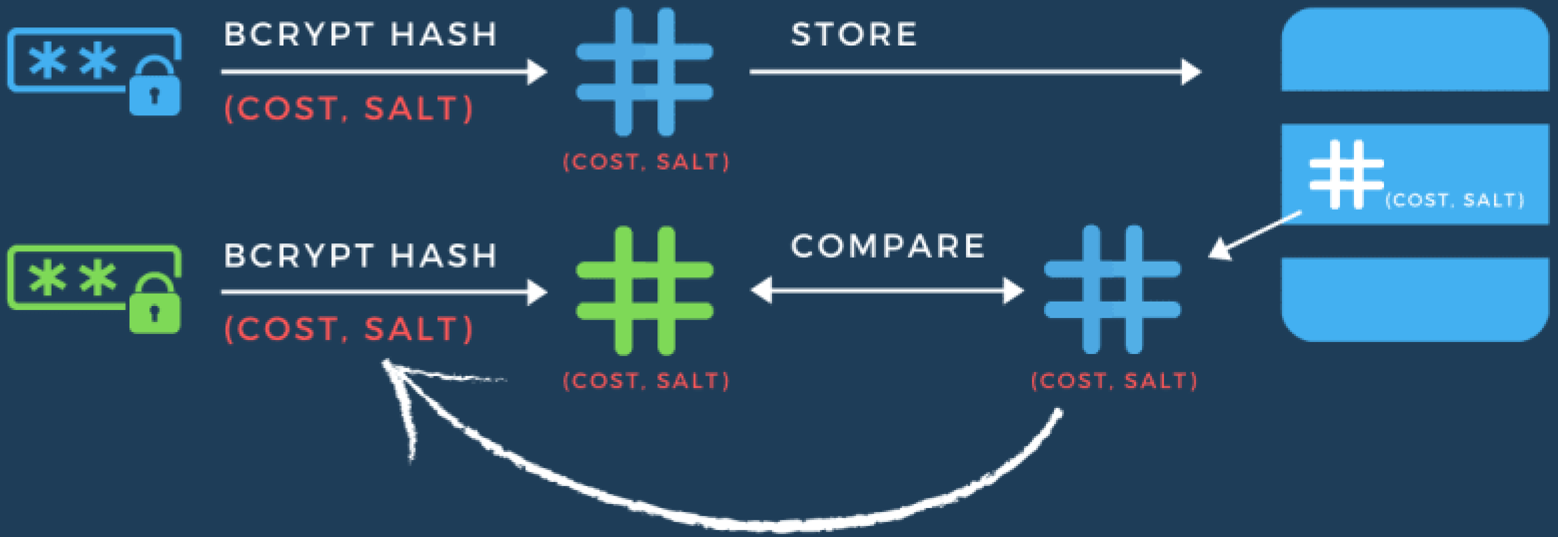
Securely store password?

Hash it & store its hash value

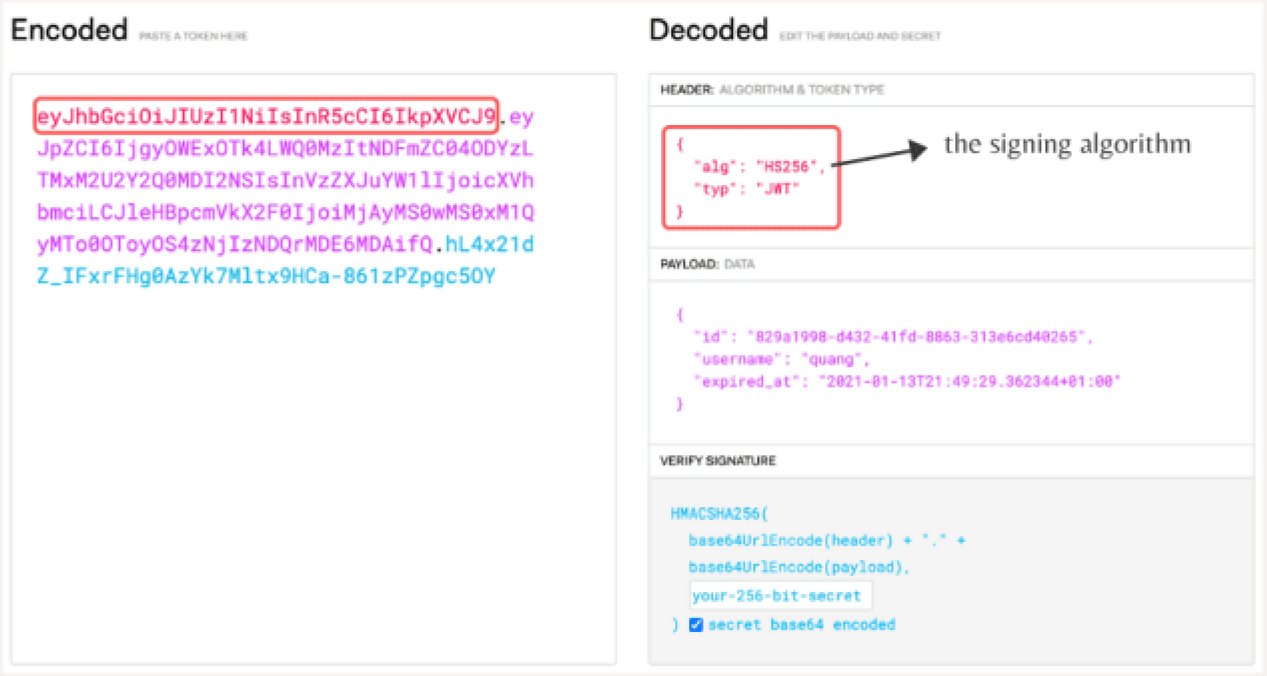


Securely store password?

Hash it & store its hash value



JSON Web Token - JWT



Encoded

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

Decoded

HEADER: ALGORITHM & TOKEN TYPE

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

the signing algorithm

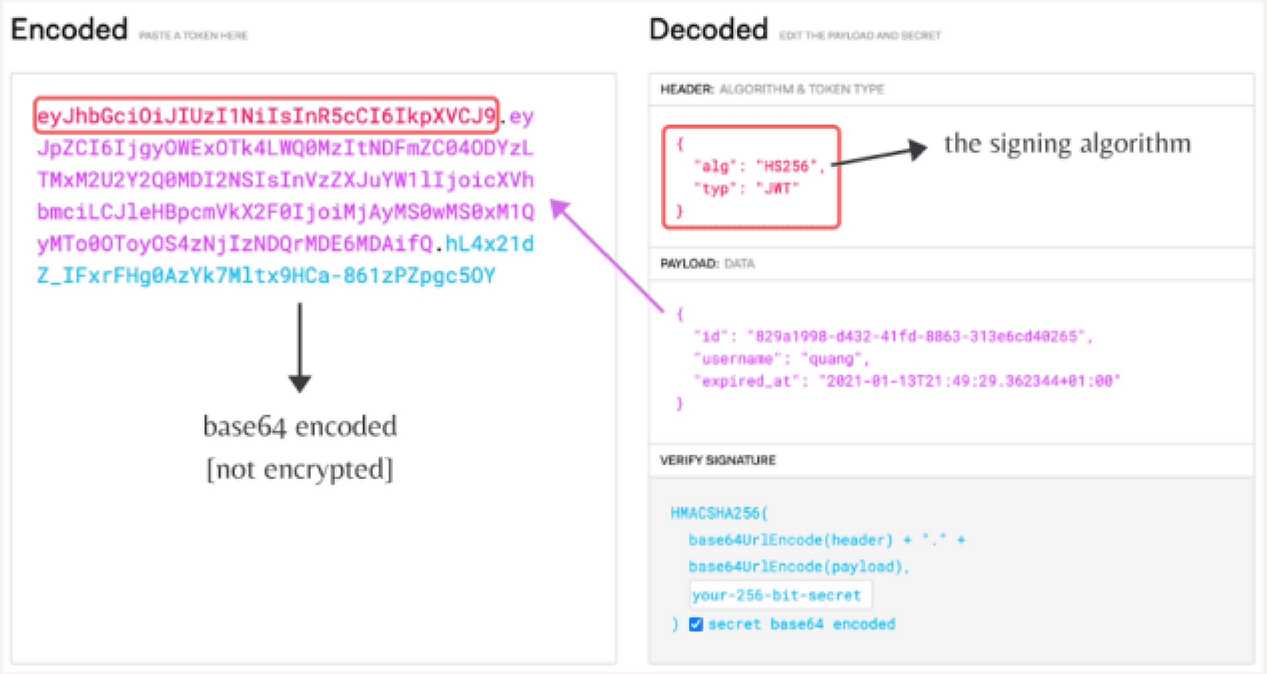
PAYLOAD: DATA

```
{
  "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

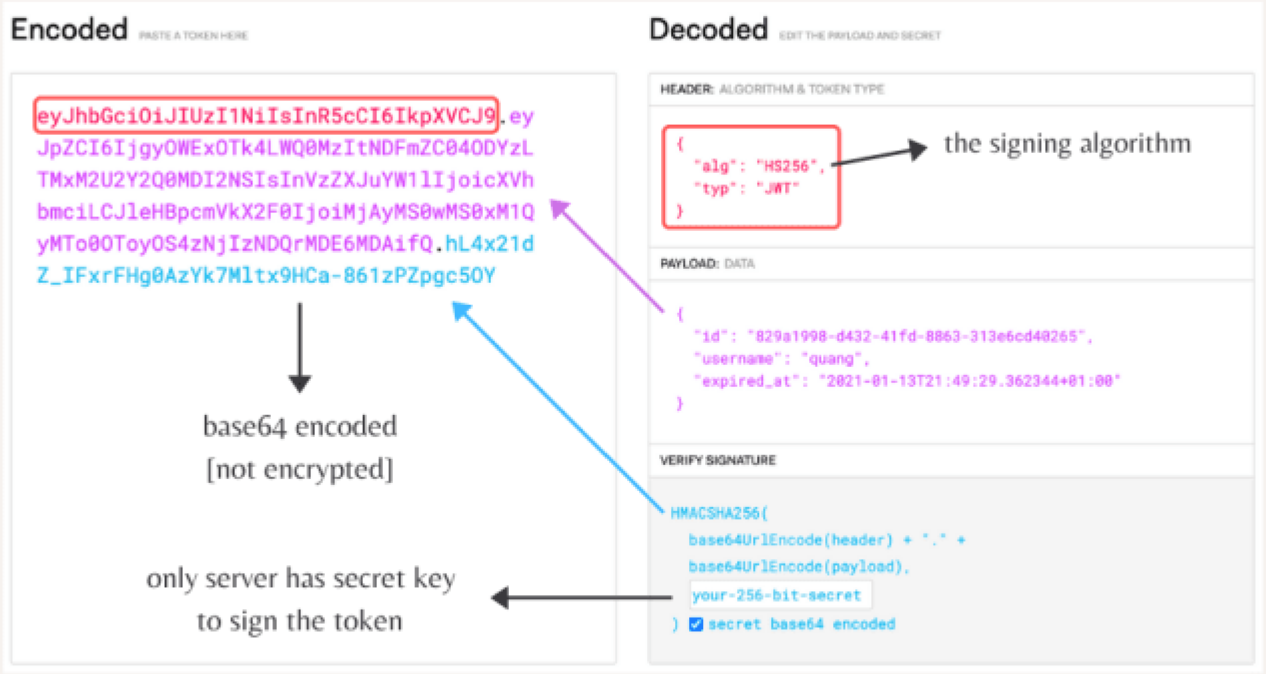
PAYLOAD: DATA

```
{
  "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
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE
the signing algorithm
base64 encoded
[not encrypted]
only server has secret key
to sign the token

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

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

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

```
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]