black hat USA 2023

AUGUST 9-10, 2023

BRIEFINGS

Three New Attacks Against JSON Web Tokens

Tom Tervoort

Speaker intro

Secura

BUREAU VERITAS 1828

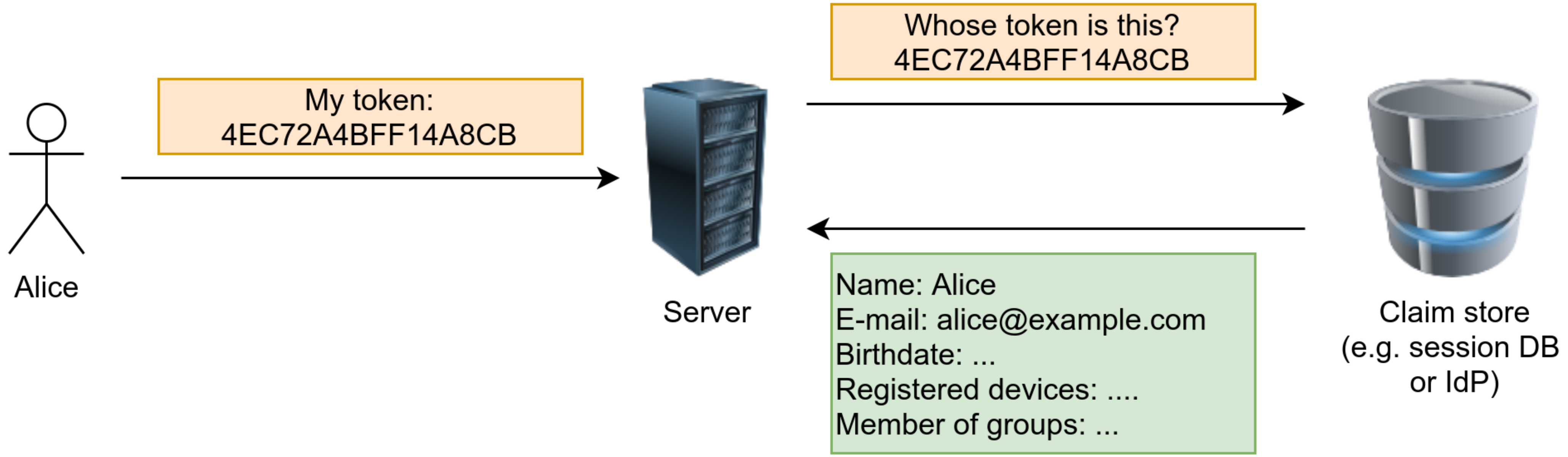
BUREAU VERITAS

Outline

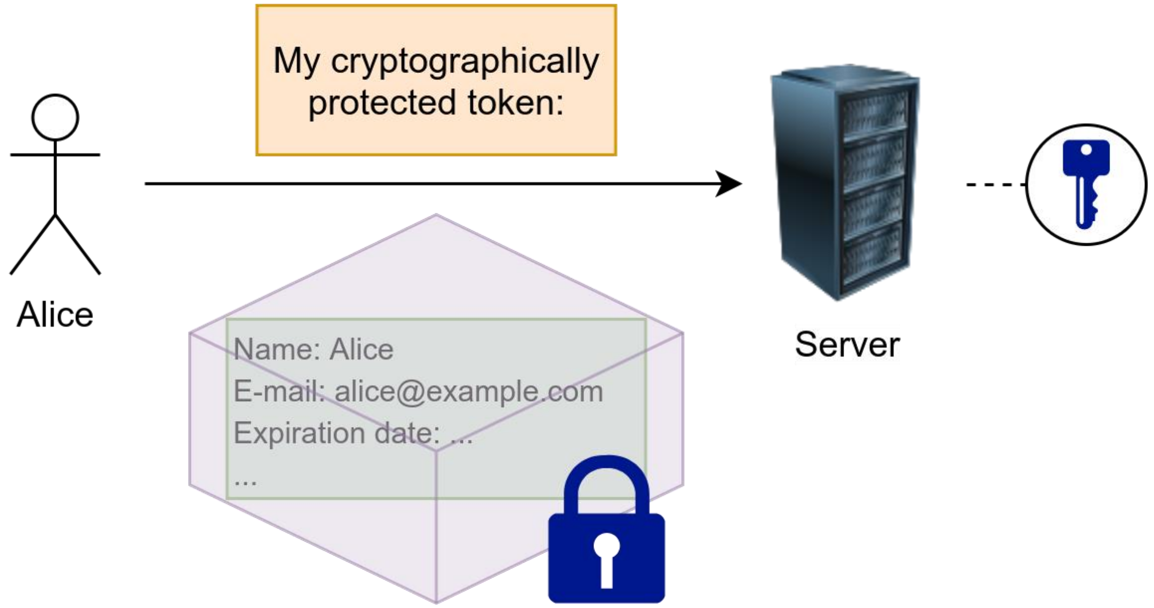
1. Background
   - Transferring identity claims
   - JSON Web Tokens
   - Prior attacks
   - Criticisms
2. New attacks
   - Sign/encrypt confusion
   - Polyglot token
   - Billion hash attack
3. Takeaways

Background

Transferring identity claims
Classic (stateful) approach



Transferring identity claims
Cryptographic approach

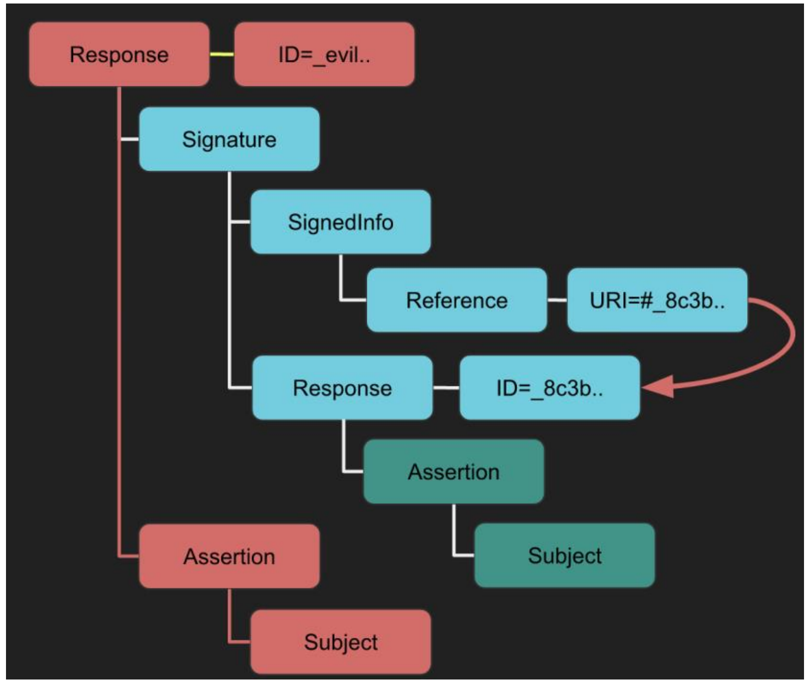


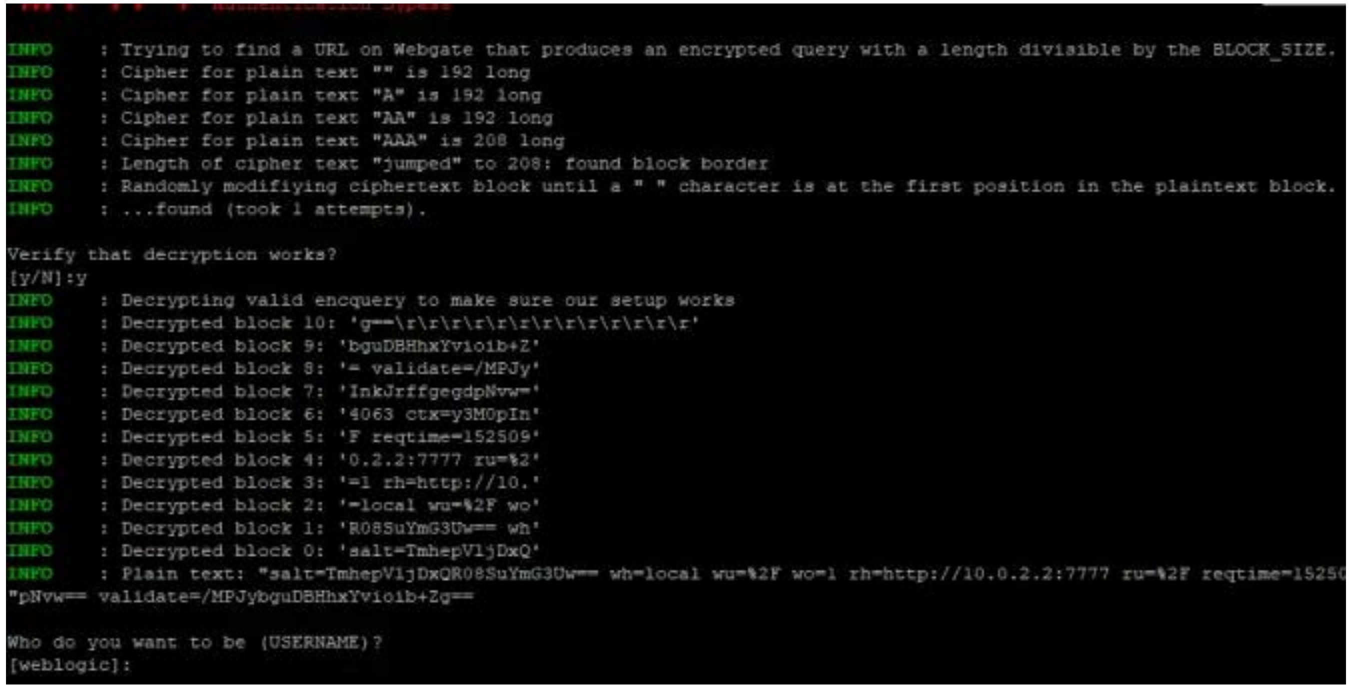
Comparison

Stateful tokens	Signed/encrypted claims
Many central DB lookups needed	Fast to verify and easy to scale
Mutable claims	Claims fixed until expiration
Trivially revocable	No revocation before expire date
Secrets are ephemeral	Requires key management
Token leak: compromise 1 user	Key leak: compromise all users
Easy to build, given secure RNG	Involves complex cryptography

Common hybrid approach: cryptographic access token and stateful “refresh token”

Cryptography is hard





Ciphertext submitted by attacker	Decryption result (internal to server)	HTTP response send to attacker
9870d401a7d4b9f4c7c5728c980bb6d5 c546ad79e8a198440929c3cf6f9ab793 7465878d11de5a8bee5555554efcdb07	expire:1645826339090$u:user\:arealm/bob% 1645826339090 %m2ZQz+j4D0LL+zW8EIEgtAxrcd6mOZZi[...]	200 OK
9870d401a7d4b9f4c7c5728c980bb6d5 00000000000000000000000000000000 c546ad79e8a198440929c3cf6f9ab793 7465878d11de5a8bee5555554efcdb07	expire:1645826339090$u:user\:arealm/bob% �!�1B8���W��[ẢW���I�s�8�nn9d XBa�Ɣ�fB�1�#g���1w7j��W��?xg�b %m2ZQz+j4D0LL+zW8EIEgtAxrcd6mOZZi[...]	200 OK
9870d401a7d4b9f4c7c5728c980bb6d5 00000000000000000000000000000001 c546ad79e8a198440929c3cf6f9ab793 7465878d11de5a8bee5555554efcdb07	expire:1645826339090$u:user\:arealm/bob% 9%�ఊ{�#x�Ђ�1(�]$�NH�µ�6'&!yAAq XBa�Ɣ�fB�1�#g���1w7j��W��?xg�c %m2ZQz+j4D0LL+zW8EIEgtAxrcd6mOZZi[...]	403 Access denied Set-Cookie: LtpaToken2=""

Co-incidental percent sign

Attacker-controlled bitflip

Image sources: HackTricks, SEC Consult blog

JSON Web Tokens

- Massive improvement over legacy standards
- Proper integrity protection
- Easy to read and debug
- Simple and concise claims
- > 100 implementations
- Used by OpenID Connect
- **They're everywhere**

Image source: jwt.io

Some JSON Web Acronyms

JWT (JSON Web Token): JSON-based claims format using JOSE for protection

JOSE (Javascript Object Signing and Encryption): set of open standards, including:

JWS (JSON Web Signature): JOSE standard for cryptographic authentication

JWE (JSON Web Encryption): JOSE standard for encryption

JWA (JSON Web Algorithms): cryptographic algorithms for use in JWS/JWE

JWK (JSON Web Keys): JSON-based format to represent JOSE keys

Prior JWT attacks

- Bypass signature validation by providing a token signed with the "**none**" algorithm
- Bypass blocklist filter with "**nOne**"…
- **Algorithm confusion**: using an RSA public key as an HMAC secret key
- **Key injection**/self-signed JWT: putting your own key in the "jwk" header
- Classic crypto attacks against primitives: RSA padding oracle; CurveSwap

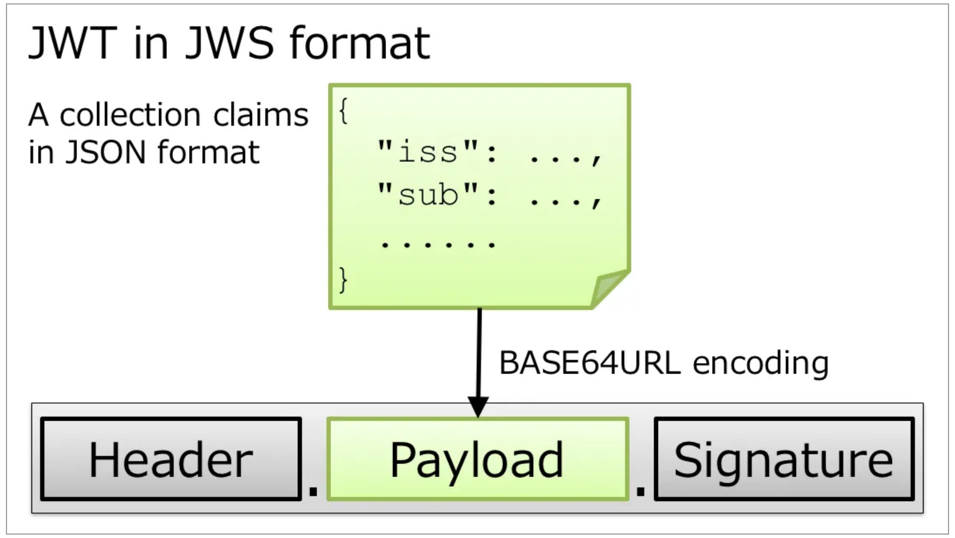
- Probably most common: **simple dictionary words** being used as cryptographic keys

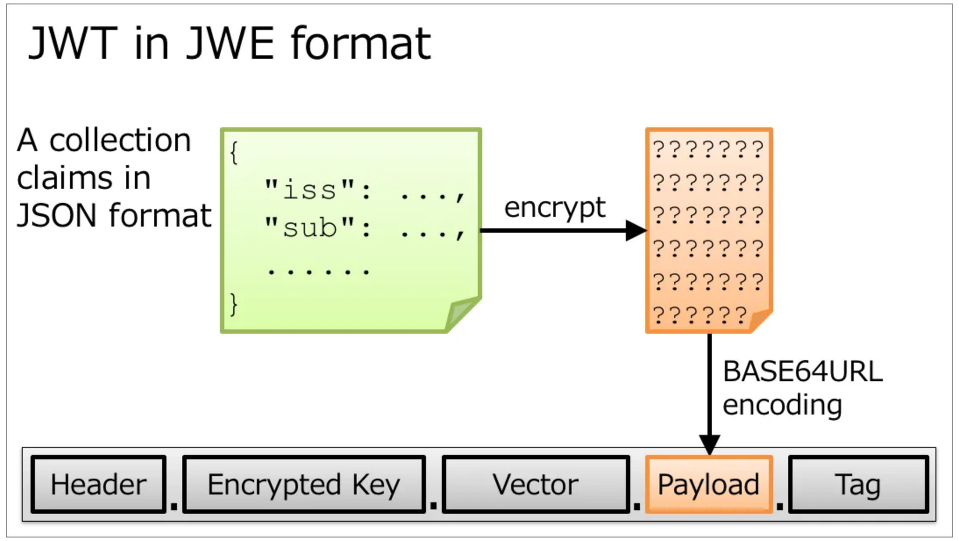
Important design flaws
(personal opinion)

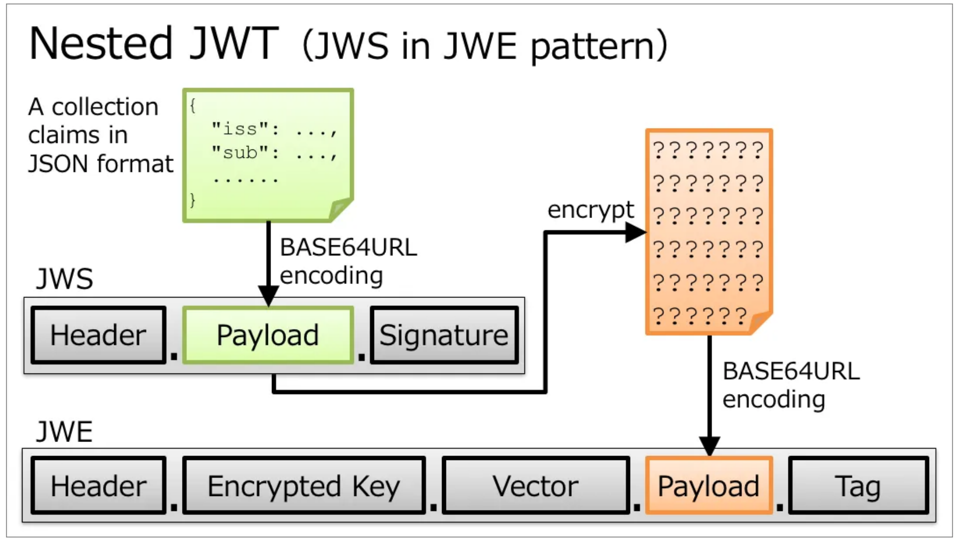
1. Deciding the decryption/validation algorithm based on untrusted ciphertext
2. Letting end users choose between cryptographic algorithms
3. … including one broken since 1998 (RSA PKCS#1 v1.5 encryption) and “none”
4. Some algorithms are interchangeable, some dramatically change security properties
5. Over-engineered: trying to support many (obscure) use cases at once

New attack: sign/encrypt confusion

JWT flavors



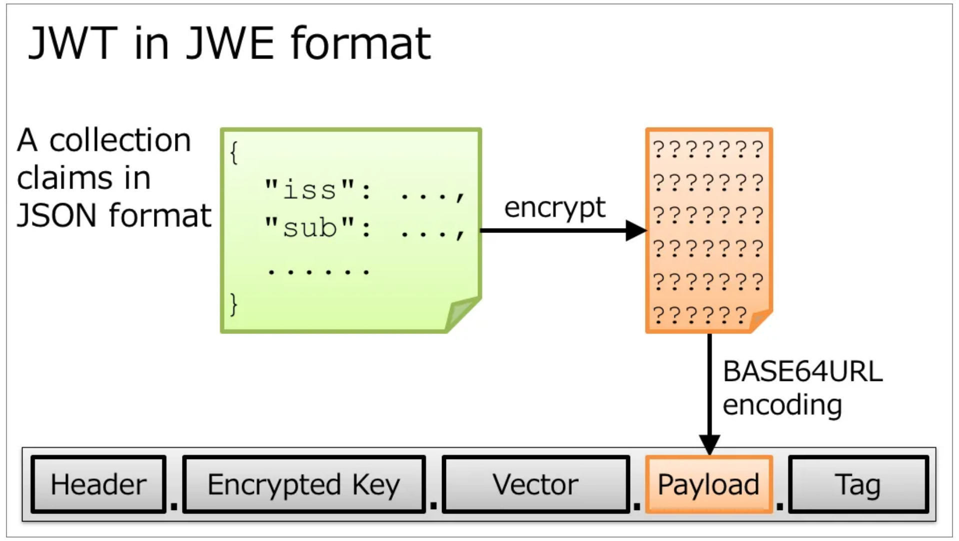




	Symmetric JWS	Asymmetric JWS	Symmetric JWE	Asymmetric JWE
Authenticity	✓	✓	✓	✗
Confidentiality	✗	✗	✓	✓

Image source: Takahiro Kawasaki

JWT flavors



	Symmetric JWS	Asymmetric JWS	Symmetric JWE	Asymmetric JWE
Authenticity	✓	✓	✓	✗
Confidentiality	✗	✗	✓	✓

Image source: Takahiro Kawasaki

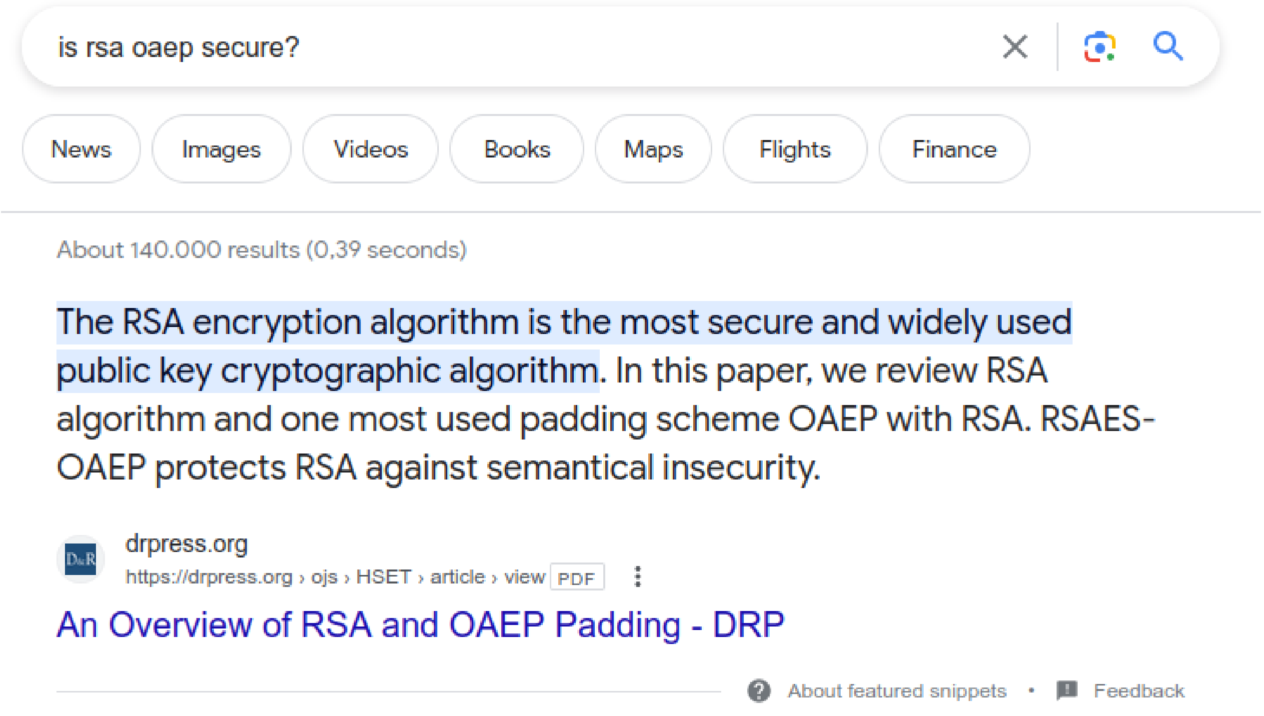
Should we expect developers to be crypto experts?

"alg" Param Value	Key Management Algorithm	More Header Params	Implementation Requirements
RSA1_5	RSAES-PKCS1-v1_5	(none)	Recommended-
RSA-OAEP	RSAES OAEP using default parameters	(none)	Recommended+
RSA-OAEP-256	RSAES OAEP using SHA-256 and MGF1 with SHA-256	(none)	Optional
A128KW	AES Key Wrap with default initial value using 128-bit key	(none)	Recommended
A192KW	AES Key Wrap with default initial value using 192-bit key	(none)	Optional
A256KW	AES Key Wrap with default initial value using 256-bit key	(none)	Recommended
dir	Direct use of a shared symmetric key as the CEK	(none)	Recommended
ECDH-ES	Elliptic Curve Diffie-Hellman Ephemeral Static key agreement using Concat KDF	"epk", "apu", "apv"	Recommended+
ECDH-ES+A128KW	ECDH-ES using Concat KDF and CEK wrapped with "A128KW"	"epk", "apu", "apv"	Recommended
ECDH-ES+A192KW	ECDH-ES using Concat KDF and CEK wrapped with "A192KW"	"epk", "apu", "apv"	Optional

Not suitable for JWTs!

Fine for JWTs

Not suitable for JWTs!



What if we just avoid encrypted JWTs?

Key file:

```
{
    "kty": "RSA",
    "n": "sEFRQzskiSOrUYiaWAPUMF66YOxWymrbf6PQqnCdnUla8PwI4KDVJ2XgNGg9XOdc-jRICmp
    "e": "AQAB",
    "d": "dsIr_P7WqUjNYEyIopFB4a2SK0hTWmQRrbk1GgJzUM1iZ0mKub_kn303SliKMBT8QuIDQHF
    "p": "2ubPBIRKrNgC8TOMaim0fJpGa4ZTUc0wntIX4Rzb2JZlThUfFeTq8OGFRgcMTn1W54cqjzM
    "q": "ziBDoJVUNK7s-WDXlkr_69rxwLI0r6I183jC2BxV3g2xY0oybPj7yvnXeMUDH8kfNTqPbZZ
    "dp": "NzgJ-MW2YKuM8nNidFVPUDdKlE0qL3RnU2kEBRFWk-g8XdoOIWPBsEnzaJrWi-YqSfVa0w
    "dq": "XOFm98YyImcsOxbrLjrvZPzMcLMcUIP8YZBp4-2ot51d8EqvvDDZbNX1xOKpjLoYyOhxVs
    "qi": "1QH5d-TiaZL_Q_-NalMj3rFL8VILo3lTr0Qz6c1lp6p0NoKOL7BCyosYSo0RvainM3i7nv
}
```

JWT signer:

```python
from authlib.jose import jwt, JsonWebKey
from time import time
import json

with open('rsa-key.jwk', 'r') as keyfile:
  key = JsonWebKey.import_key(json.load(keyfile))

header = {'alg': 'RS256'}
payload = {'iss': 'secure-issuer', 'sub': username, 'exp': round(time()) + 3600}
token = jwt.encode(header, payload, key).decode()
```

JWT validator:

```python
from authlib.jose import jwt, JsonWebKey
import sys, json

with open('rsa-key.jwk', 'r') as keyfile:
  key = JsonWebKey.import_key(json.load(keyfile))

claims = jwt.decode(token, key)
username = claims.validate()['sub']
```

What if we just avoid encrypted JWTs?

Key file:

```
{
    "kty": "RSA",
    "n": "sEFRQzskiSOrUYiaWAPUMF66YOxWymrbf6PQqnCdnUla8PwI4KDVJ2XgNGg9XOdc-jRICmp
    "e": "AQAB",
    "d": "dsIr_P7WqUjNYEyIopFB4a2SK0hTWmQRrbk1GgJzUM1iZ0mKub_kn303SliKMBT8QuIDQHF
    "p": "2ubPBIRKrNgC8TOMaim0fJpGa4ZTUc0wntIX4Rzb2JZlThUfFeTq8OGFRgcMTn1W54cqjzM
    "q": "ziBDoJVUNK7s-WDXlkr_69rxwLI0r6I183jC2BxV3g2xY0oybPj7yvnXeMUDH8kfNTqPbZZ
    "dp": "NzgJ-MW2YKuM8nNidFVPUDdKlE0qL3RnU2kEBRFWk-g8XdoOIWPBsEnzaJrWi-YqSfVa0w
    "dq": "XOFm98YyImcsOxbrLjrvZPzMcLMcUIP8YZBp4-2ot51d8EqvvDDZbNX1xOKpjLoYyOhxVs
    "qi": "1QH5d-TiaZL_Q_-NalMj3rFL8VILo3lTr0Qz6c1lp6p0NoKOL7BCyosYSo0RvainM3i7nv
}
```

RSA JWK file usable for:

- Signing
- Validation
- Encryption
- Decryption

JWT signer:

```python
from authlib.jose import jwt, JsonWebKey
from time import time
import json

with open('rsa-key.jwk', 'r') as keyfile:
  key = JsonWebKey.import_key(json.load(keyfile))

header = {'alg': 'RS256'}
payload = {'iss': 'secure-issuer', 'sub': username, 'exp': round(time()) + 3600}
token = jwt.encode(header, payload, key).decode()
```

JWT validator:

```python
from authlib.jose import jwt, JsonWebKey
import sys, json

with open('rsa-key.jwk', 'r') as keyfile:
  key = JsonWebKey.import_key(json.load(keyfile))

claims = jwt.decode(token, key)
username = claims.validate()['sub']
```

Decides algorithm based on JWT header. Accepts RSA-encrypted JWE!

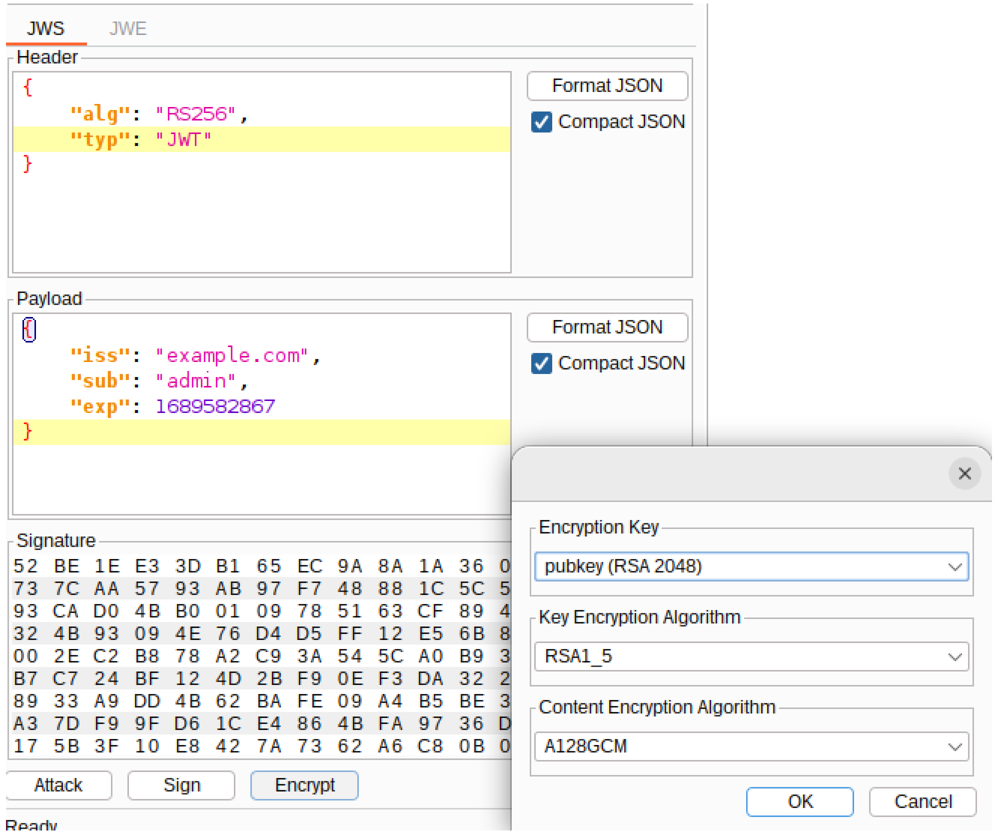
Sign/encrypt confusion attack

Preconditions:

1. Library supports asymmetric JWTs
2. App uses JWS tokens with RSA or ECDSA (RS*/PS*/ES*)
3. Private key accessible by validation function
4. No specific algorithm or JWT wrapper type is enforced
5. Attacker can determine public key. E.g. by:

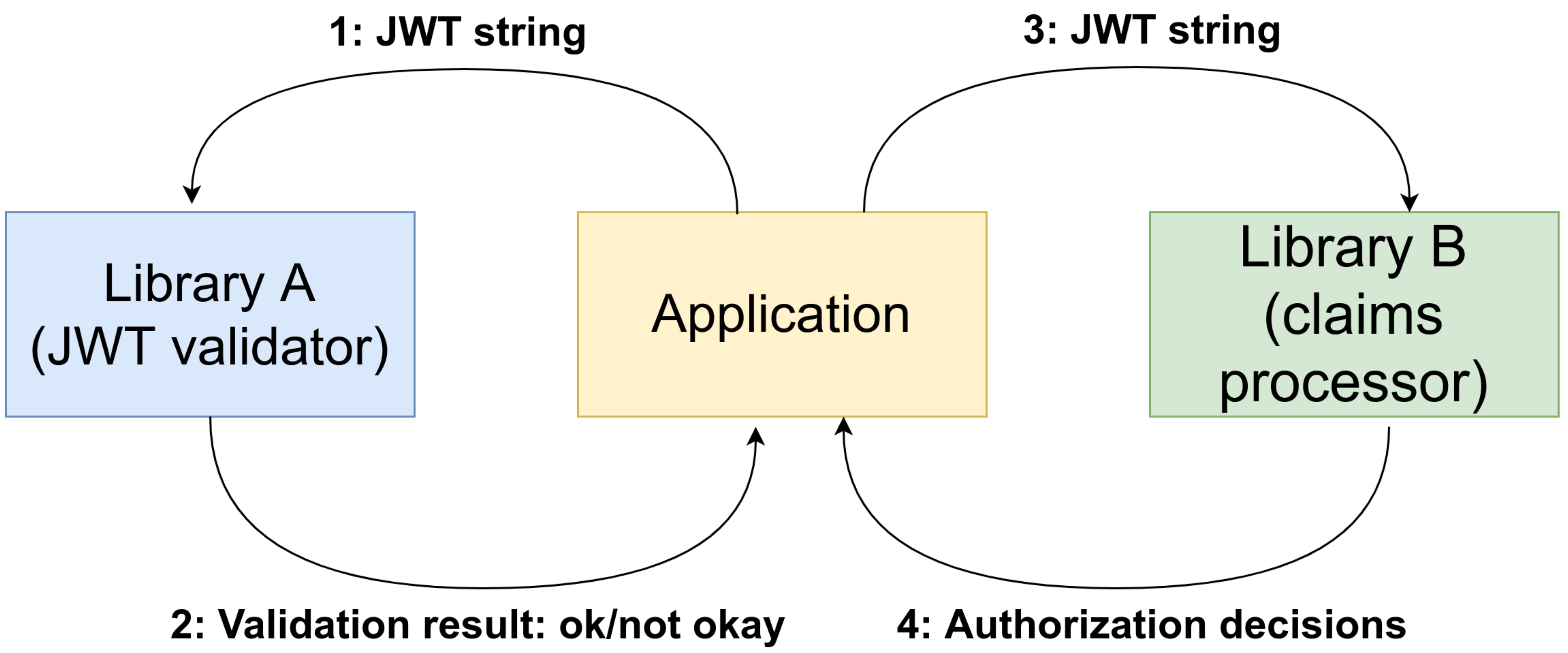
   - Reading it from OIDC endpoint **/jwks.json**

   - If alg is RS*, can **compute it from two tokens** (https://github.com/SecuraBV/jws2pubkey)



New attack: polyglot JWT

A dangerous pattern



What if library A and library B parse JWTs differently?

Maybe exploit JSON ambiguity?

```
{
    "name": "alice",
    "name": "bob"
}
```

See also: https://bishopfox.com/blog/json-interoperability-vulnerabilities

Or an alternative serialization format?

JWS Compact Serialization

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhbGljZSIsImlhdCI6MTUxNjIzOTAyMn0.rv61W60MY3WdNuyFrbDb31rcbBpfuYWoS4fOI6Mmjeg

JWS Flattened JSON Serialization


```
{
    "protected":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9",
    "payload":"eyJzdWIiOiJhbGljZSIsImlhdCI6MTUxNjIzOTAyMn0",
    "signature":"rv61W60MY3WdNuyFrbDb31rcbBpfuYWoS4fOI6Mmjeg"
}
```


JWT spec requires compact, but some libraries pass the JWT to a general JWS parser that accepts either type

Library mismatch

python-jwt JWT validator
(assumes compact)

```python
header, claims, _ = jwt.split('.')

parsed_header = json_decode(base64url_decode(header))

alg = parsed_header.get('alg')
if alg is None:
    raise _JWTError('alg header not present')
if alg not in allowed_algs:
    raise _JWTError('algorithm not allowed: ' + alg)

if not ignore_not_implemented:
    for k in parsed_header:
        if k not in JWSHeaderRegistry:
            raise _JWTError('unknown header: ' + k)
        if not JWSHeaderRegistry[k].supported:
            raise _JWTError('header not implemented: ' + k)

if pub_key:
    token = JWS()
    token.allowed_algs = allowed_algs
    token.deserialize(jwt, pub_key)
elif 'none' not in allowed_algs:
    raise _JWTError('no key but none alg not allowed')

parsed_claims = json_decode(base64url_decode(claims))
```

jwcrypto JWS validator
(first tries JSON; then compact)

```python
try:
    djws = json_decode(raw_jws)
    if 'signatures' in djws:
        o['signatures'] = []
        for s in djws['signatures']:
            os = self._deserialize_signature(s)
            o['signatures'].append(os)
            self._deserialize_b64(o, os.get('protected'))
    else:
        o = self._deserialize_signature(djws)
        self._deserialize_b64(o, o.get('protected'))

    if 'payload' in djws:
        if o.get('b64', True):
            o['payload'] = base64url_decode(str(djws['payload']))
        else:
            o['payload'] = djws['payload']

except ValueError:
    c = raw_jws.split('.')
    if len(c) != 3:
        raise InvalidJWSObject('Unrecognized'
                               ' representation') from None
    p = base64url_decode(str(c[0]))
    if len(p) > 0:
        o['protected'] = p.decode('utf-8')
        self._deserialize_b64(o, o['protected'])
    o['payload'] = base64url_decode(str(c[1]))
    o['signature'] = base64url_decode(str(c[2]))
```

A polyglot token

```
{
    "AAAA":".XXXX.",
    "protected": "AAAA",
    "payload": "BBBB",
    "signature": "CCCC"
}
```

A polyglot token

jwcrypto ignored unknown JSON fields:

```
{
    ~~"AAAA":".XXXX.",~~
    "protected": "AAAA",
    "payload": "BBBB",
    "signature": "CCCC"
}
```

A polyglot token

python-jwt split on periods, and ignored non-base64 characters:

header — payload

```
{
  "AAAA":".XXXX.",
  "protected": "AAAA",
  "payload": "BBBB",
  "signature": "CCCC"
}
```

Given a token with a legitimate payload, the attacker can replace it with any spoofed claims

New attack: billion hashes attack

Some interesting JWE “alg” values

PBES2-HS256+A128KW	PBES2 with HMAC SHA-256 and "A128KW" wrapping	"p2s", "p2c"	Optional
PBES2-HS384+A192KW	PBES2 with HMAC SHA-384 and "A192KW" wrapping	"p2s", "p2c"	Optional
PBES2-HS512+A256KW	PBES2 with HMAC SHA-512 and "A256KW" wrapping	"p2s", "p2c"	Optional

4.8. Key Encryption with PBES2

This section defines the specifics of performing password-based encryption of a JWE CEK, by first deriving a key encryption key from a user-supplied password using PBES2 schemes as specified in Section 6.2 of [RFC2898], then by encrypting the JWE CEK using the derived key.

What can go wrong?

- Standard designer wants versatility: includes useful PBES algorithms
- Library implementer wants feature-completeness: implements all JWE algorithms
- Library implementer wants simple and clean interface: same API for all algorithms
- User decodes token with default settings, assuming these must be secure

- Result: application will try to decrypt JWTs claiming to be encrypted with a password, even though that doesn't really make sense

- But if there's no token spoofing cross-protocol attack between PBES and other algorithms this should not be a problem, right?

A PBES header parameter

4.8.1.2. "p2c" (PBES2 Count) Header Parameter

The "p2c" (PBES2 count) Header Parameter contains the PBKDF2 iteration count, represented as a positive JSON integer. This Header Parameter MUST be present and MUST be understood and processed by implementations when these algorithms are used.

The iteration count adds computational expense, ideally compounded by the possible range of keys introduced by the salt. A minimum iteration count of 1000 is RECOMMENDED.

DoS with a token header

```
{
    "alg": "PBES2-HS512+A256KW",
    "p2s": "AAAAAAAAAAAAAAAAAAAAAA",
    "p2c": 2147483647,
    "enc": "A128CBC-HS256"
}
```

- Rest of the JWE can consist of bogus strings.
- The server needs to perform more than **4 billion SHA512 hashes** to derive the token encryption key in before it can determine that this JWT is invalid.
- **Unauthenticated**: attacker does not need to know what a valid token looks like.
- It has to do this for **every request** with a JWT!

Takeaways

JWT library research

- Focus on popular open source libraries. Could not cover all 100+ JWT libraries!
- Vulnerabilities mainly found in highly featured libraries.
- Responsible disclosure very pleasant: fast and excellent response in each case
- Vulnerabilities found and mitigations implemented in the following libraries:

Library	Language	Affected versions	Vulnerability	CVE
Authlib	Python	< v1.1.0	Sign/encrypt confusion	CVE-2022-39174
JWCrypto	Python	< v1.4	Sign/encrypt confusion	CVE-2022-3102
JWX	PHP	< 0.12.0	Sign/encrypt confusion	
Python-jwt	Python	< v3.3.4	Polyglot token	CVE-2022-39227
Jose	JavaScript	< v1.28.1, v2.0.5, v3.20.3, v4.9.1	Billion hashes	CVE-2022-36083
Jose-jwt	.NET	< v4.1	Billion hashes	

Recommendations for JWT library developers

- Less is more: don't implement features with rare use cases, or turn them off by default.
- Don't use the "alg" parameter in the token to decide the algorithm. Instead force users to make this explicit in their code or key file.
- Don't support JWTs using asymmetric or password-based encryption.
- Avoid validate-then-parse-again patterns.

Recommendations for the JOSE working group

- Specify security recommendations to avoid the issues discussed here.
- Explicitly list which JWS and JWE algorithms are allowed for JWTs. Exclude the likes of “none”, PBES and public key encryption.
- Encourage existing methods to enforce that a key is only used with a single algorithm.
- Ideally, remove “alg” from token headers altogether.

Recommendations for application developers using JWTs

- Reconsider if you really need encrypted claims. Boring old random tokens have many advantages!
- Consider JWT alternatives like PASETO, Macaroons or Biscuits.
- When using JWT, always explicitly configure the validation algorithm.
- A JWT validation library is a critical dependency. Don’t forget to patch them!

Thank you!