In 2 Minutes

https://github.com/Cyphrme/Coze

HTTPS://CYPHR.ME/COZE

Zamicol

See also the "Information is Fundamental" presentation:
https://docs.google.com/presentation/d/1SNGBTsfqvY_pOcQQaus9SAfu3Bgmq8edhI

Example **Coze**


```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1623132000,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 },
 "sig":
"Jl8Kt4nznAf0LGgO5yn_9HkGdY3ulvjg-NyRGzlmJzhncbTkFFn9jrwIwGoRAQYhjc88wmwFNH5u_rO56USo_w"
}
```


Coze

Coze is a **cryptographic** JSON messaging specification.

Coze uses **digital signatures** and **cryptographic hashes** to ensure secure, human-readable, and interoperable communication.

Coze is a **cryptographic** abstraction layer,

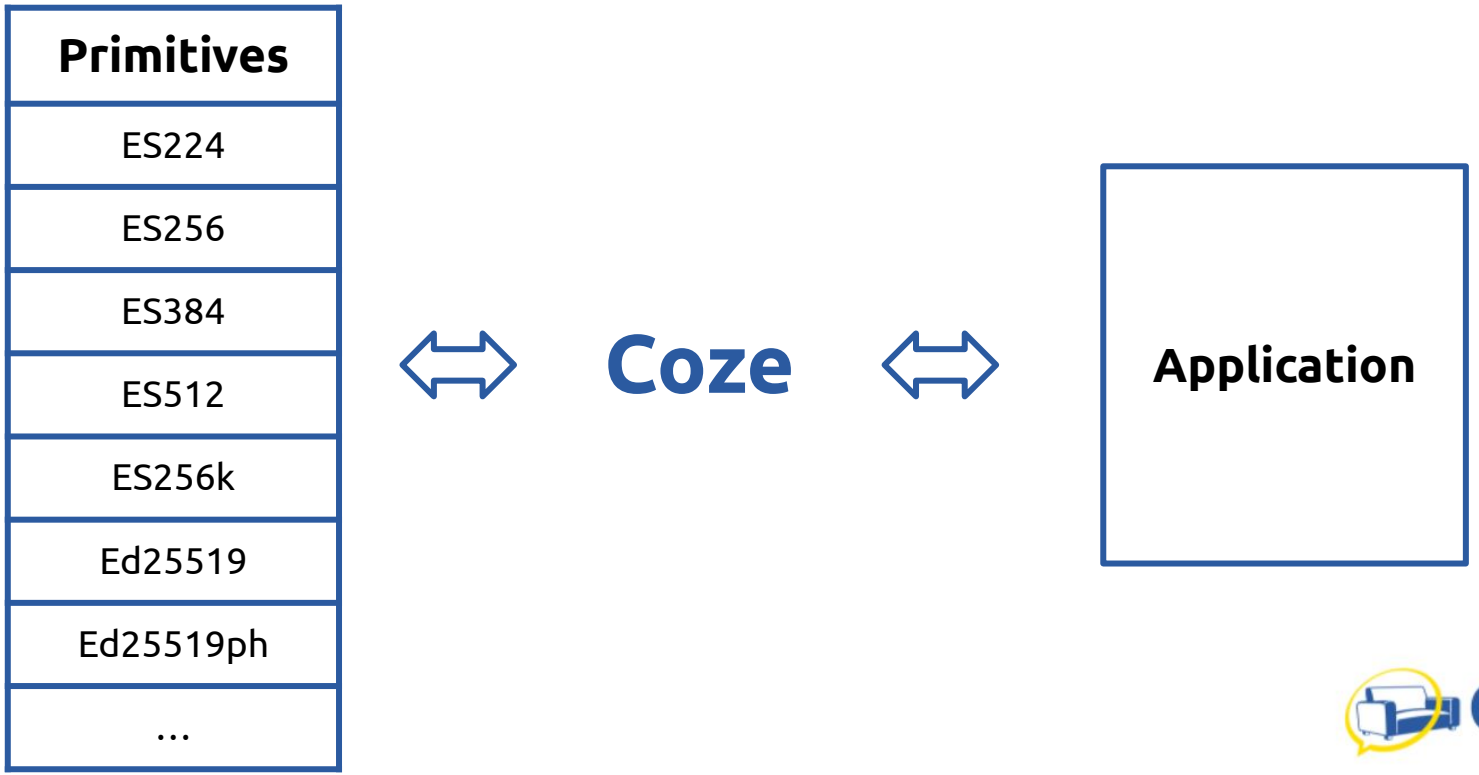
a "language" to speak cryptographic signing.

Coze is **open source** (3-clause BSD)

The English word **Coze** means "**a friendly chat**", "to converse in a friendly way."

Coze

Coze provides a common framework between cryptographic primitives and applications.



Coze

Coze Design Goals

1. Idiomatic JSON.
2. Human readable.
3. Small in scope.
4. Provide defined cipher suites.

Example **Coze**


```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1623132000,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 },
 "sig":
"Jl8Kt4nznAf0LGgO5yn_9HkGdY3ulvjg-NyRGzlmJzhncbTkFFn9jrwIwGoRAQYhjc88wmwFNH5u_rO56USo_w"
}
```


Coze

Coze is **human readable**, while remaining (relatively) **small in size**.

How to sign a **Coze**

0. Put data into JSON.
1. Add Coze JSON fields. (alg, iat, typ, tmb)
2. Remove unneeded spaces.
3. Hash.
4. Sign.

Coze

Coze Example

Let's send a verifiable message to a friend.
This is our data we need to put into JSON.

```
"Coze Rocks"
```

Coze

JSONify the message

```
{
 "msg": "Coze Rocks"
}
```

Coze

Add **Coze** fields


```
{
 "pay": {
    "msg": "Coze Rocks",
    "alg": "ES256",
    "iat": 1623132000,
    "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
    "typ": "cyphr.me/msg"
 },
 "sig":
"Jl8Kt4nznAf0LGgO5yn_9HkGdY3ulvjg-NyRGzlmJzhncbTkFFn9jrwIwGoRAQYhjc88wmwFNH5u_
rO56USo_w"
}
```

Let do that again, slower.

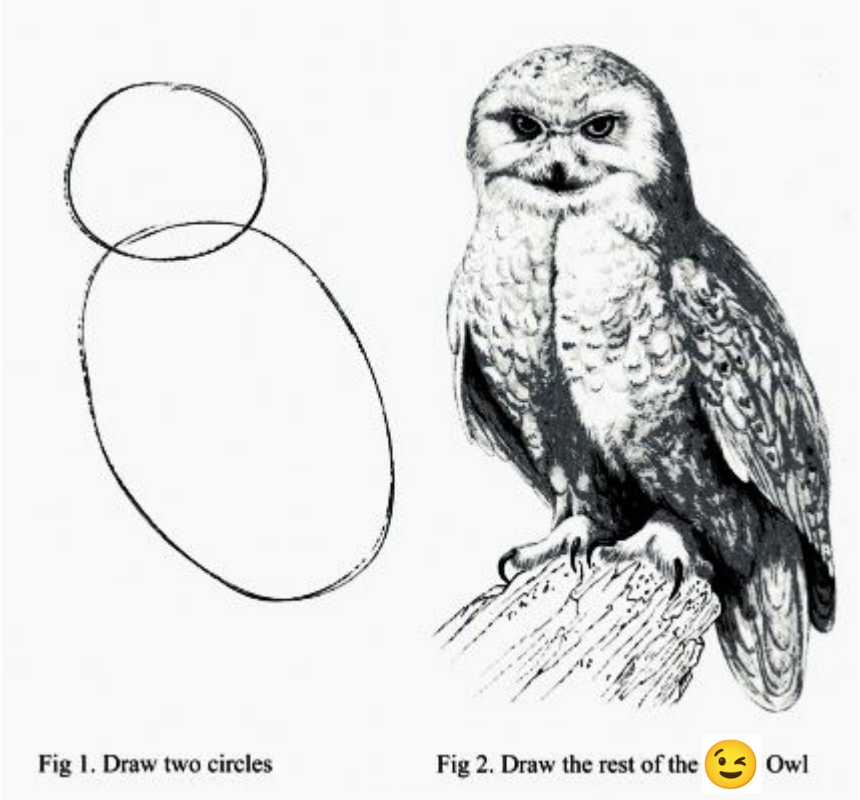


Fig 1. Draw two circles

Fig 2. Draw the rest of the 😉 Owl

Step 0: Put your data into JSON

```
{
    "msg": "Coze Rocks"
}
```

Coze

Step 1: Add **Coze** fields

```
{
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1623132000,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
}
```

Coze

Step 3: Remove Spaces


```
{"msg":"Coze
Rocks","alg":"ES256","iat":1623132000,"tmb":"cLj8vsYtMBwYkzoFVZH
BZo6SNL8wSdCIjCKAwXNuhOk","typ":"cyphr.me/msg"}
```


Coze

Step 4: Hash

SHA256(

```
{"msg":"Coze
Rocks","alg":"ES256","iat":1623132000,"tmb":"cLj8vsYtMBwYkzoFVZH
BZo6SNL8wSdCIjCKAwXNuhOk","typ":"cyphr.me/msg"}
```

) = **Ie3xL77AsiCcb4r0pbnZJqMcfSBqg5Lk0npNJyJ9BC4**

Coze

Step 5: Sign the digest

ES256(

Ie3xL77AsiCcb4r0pbnZJqMcfSBqg5Lk0npNJyJ9BC4

) =

`Jl8Kt4nznAf0LGgO5yn_9HkGdY3ulvjg-NyRGzlmJzhncbTkFFn9`

`jrwIwGoRAQYhjc88wmwFNH5u_rO56USo_w`

Coze

Add it together. All done!


```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1623132000,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 },
 "sig":
"Jl8Kt4nznAf0LGgO5yn_9HkGdY3ulvjg-NyRGzlmJzhncbTkFFn9jrwIwGoRAQYhjc88wmwFNH5u_rO56
USo_w"
}
```


Coze

Coze messages are signed from brace to brace


```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1623132000,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 },
 "sig":
"Jl8Kt4nznAf0LGgO5yn_9HkGdY3ulvjg-NyRGzlmJzhncbTkFFn9jrwIwGoRAQYhjc88wmwFNH5u_rO56USo_w"
}
```

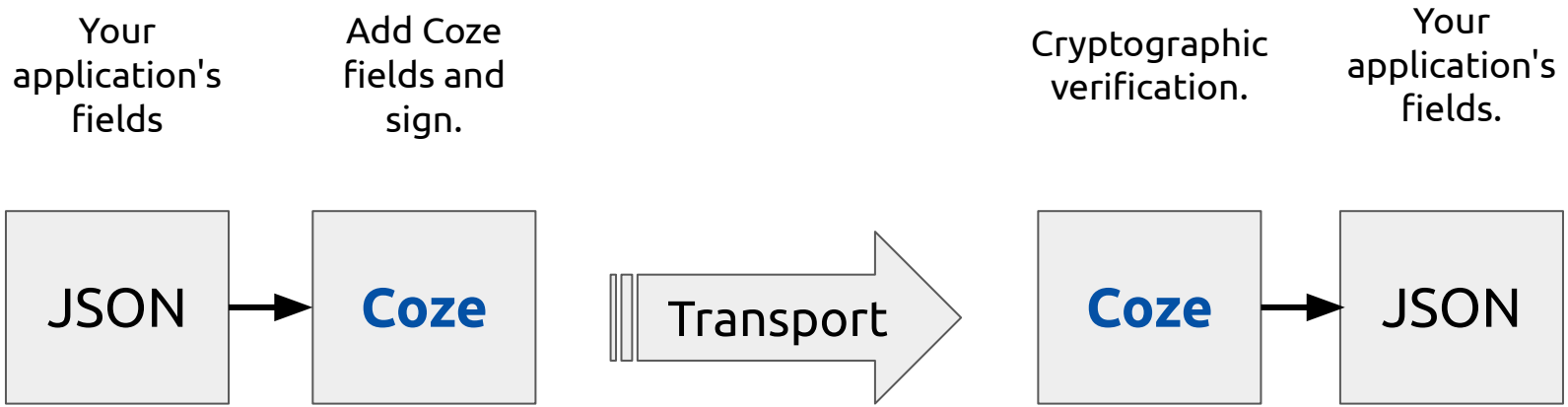

Coze

Anything not in `pay` is not signed.

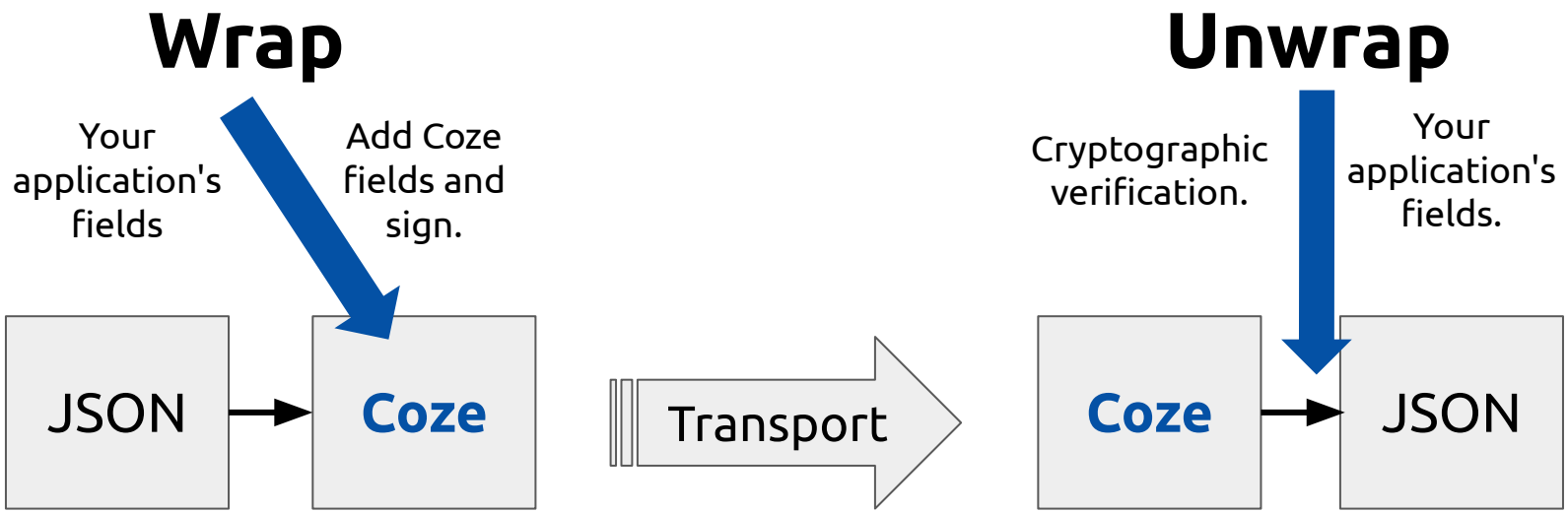

```
{
 "pay": {
    "msg": "Coze Rocks",
    "alg": "ES256",
    "iat": 1623132000,
    "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
    "typ": "cyphr.me/msg"
 },
 "sig": "Jl8Kt4nznAf0LGgO5yn_9HkGdY3ulvjg-NyRGzlmJzhncbTkFFn9jrwIwGoRAQYhjc88wmwFNH5u_rO56USo_w",
 "foo": "bar"
}
```


Coze

Typical Coze Workflow



Optionally, wrap and unwrap



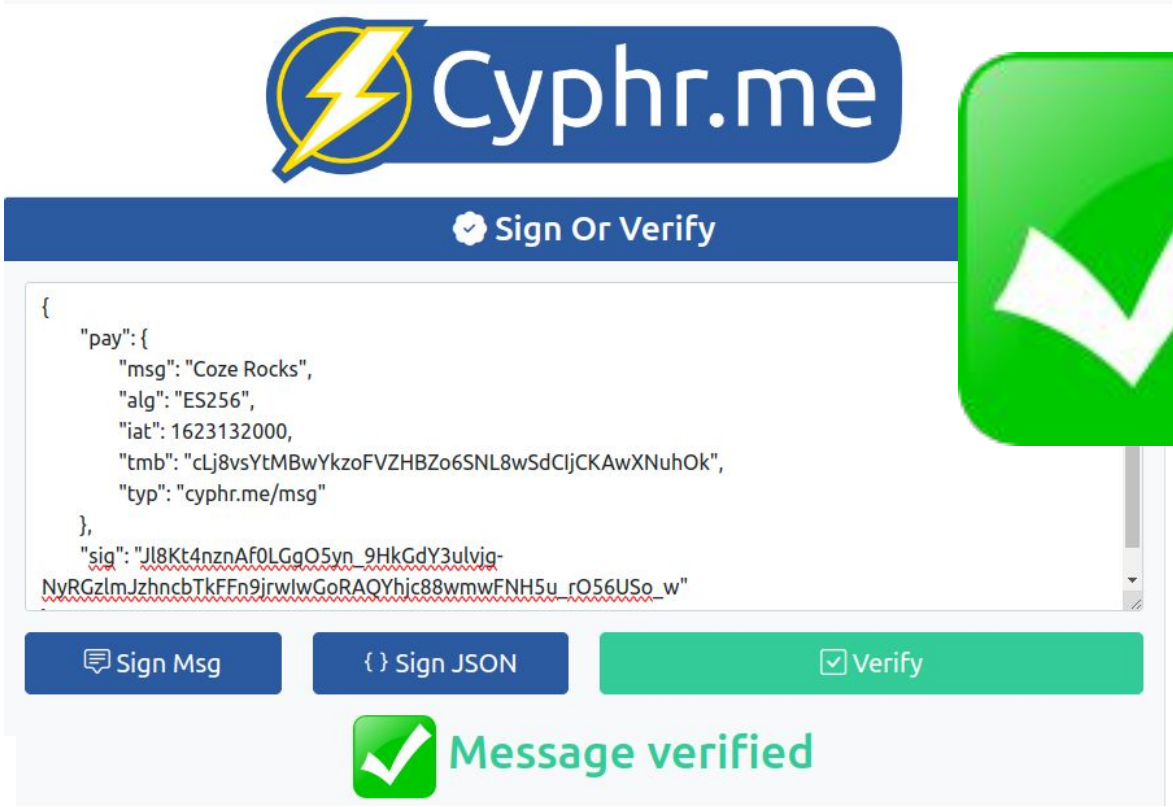
Coze

Create and **verify** **Coze** messages online tool.

Try it out!

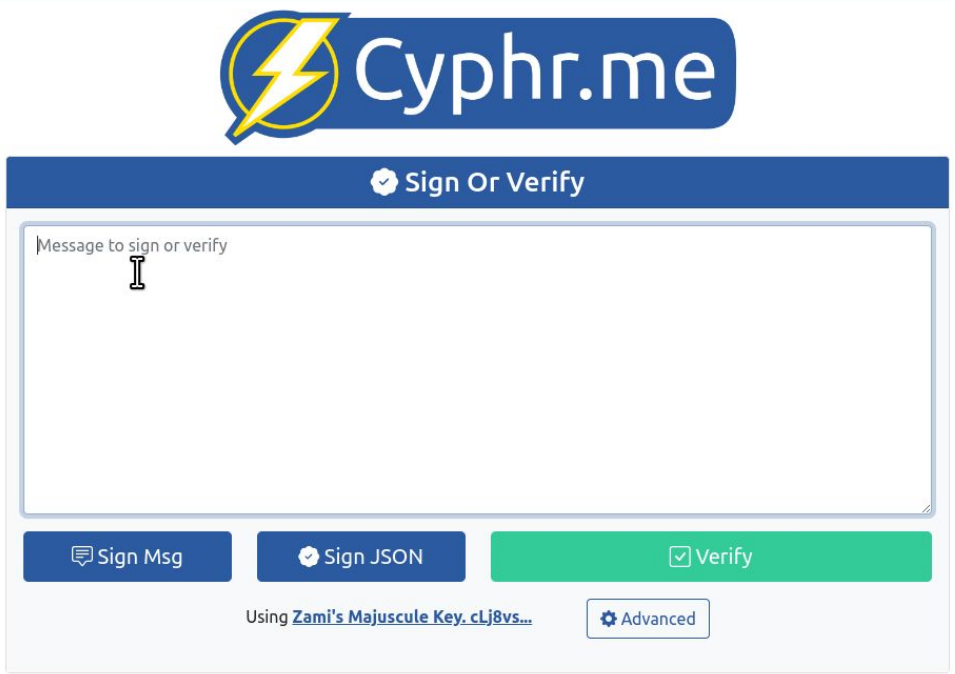
HTTPS://CYPHR.ME/COZE

Coze



HTTPS://CYPHR.ME/COZE

Coze



HTTPS://CYPHR.ME/COZE

Coze

Coze Key

Coze Key


```
{
 "alg":"ES256",
 "iat":1623132000,
 "kid":"Zami's Majuscule Key.",
 "d":"bNstg4_H3m3SlROufwRSEgibLrBuRq9114OvdapcpVA",
 "tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
 "x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq39Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"
}
```


Coze

Coze Key Fields

alg Specific algorithm. "**ES256**"

d Private component.

x Public component.

kid Human readable, **non-programmatic** identifier for the key. "My Coze Key"

tmb Thumbprint of the key. (Programmatic identifier)

typ Additional application information.

Coze

Coze Key - Public


```
{
 "alg":"ES256",
 "iat":1623132000,
 "kid":"Zami's Majuscule Key.",
 "tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
 "x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq39Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"
}
```


Any key **without** `d` and **with** `x`
is a public Coze key.

Coze

Coze Key - Private


```
{
 "alg":"ES256",
 "iat":1623132000,
 "kid":"Zami's Majuscule Key.",
 "d":"bNstg4_H3m3SlROufwRSEgibLrBuRq9114OvdapcpVA",
 "tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
 "x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq39Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"
}
```


Any key **with** field "d" is a private Coze key.

Coze

Coze needs your help!

Coze needs your help!

- We've implemented Coze in two languages
  - Coze Go
  - Coze JS
- We'd love to see **Coze** support in more languages. (Rust, Python, and more)
- Reach out on Github!

We're looking for **new auth** of **Coze** implementations.

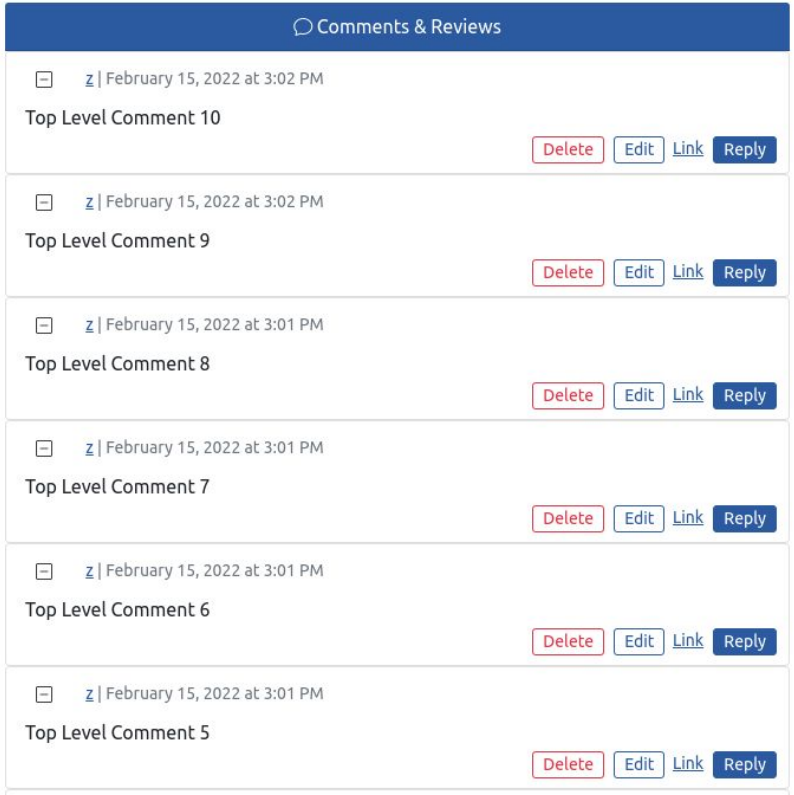
Coze

End of Section

What can **Coze** be used for?

Coze

Coze example application: Comments



Coze

Each comment is cryptographically signed

Coze


```
{
 "pay": {
  "root": "2Nw_gaosyBHwWvSmIOyKzd3UOLdC-Koog8BAakYv3tI",
  "text": "Top Level Comment  9",
  "alg": "ES256",
  "iat": 1644962544,
  "tmb": "2tphTbl0F2vilXj-ZakHom1im_DtMGxmlafmuG8R21Q",
  "typ": "cyphr.me/comment/create"
 },
 "sig": "uGvEjShoFoIWSXFLZdmNi6uAHLO4xtKARyoqIGGYQYMqyeG_jdPk05Du_1YdQ6d-X0ML0MtV0FmqcnlW0hwyBw"
}
```


Verify

Coze

Each comment is cryptographically verifiable


```
{
  "pay": {
    "root": "2Nw_gaosyBHwWvSmIOyKzd3UOLdC-Koog8BAakYv3tI",
    "text": "Top Level Comment  9",
    "alg": "ES256",
    "iat": 1644962544,
    "tmb": "2tphTbl0F2vilXj-ZakHom1im_DtMGxmlafmuG8R21Q",
    "typ": "cyphr.me/comment/create"
  },
  "sig": "uGvEjShoFoIWSXFLZdmNi6uAHLO4xtKARyoqIGGYQYMqyeG_jdPk05Du_1YdQ6d-X0ML0MtV0FmqcnlW0hwyBw"
}
```

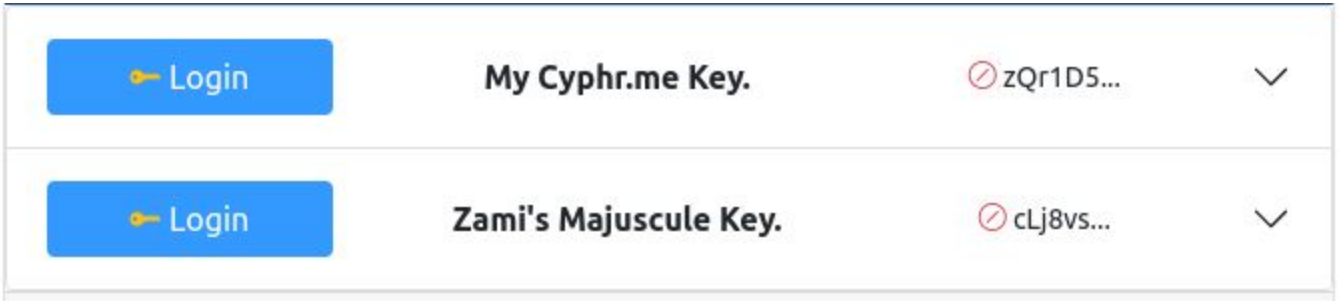

Sign Msg | Sign JSON | Verify

✅ Message verified.

Coze

https://developer.apple.com/documentation/authenticationservices/public-private_key_authentication/supporting_passkeys

Coze: Key Wallet.
No Passwords! No Emails!



Coze

Coze can be used to cryptographically associate data to users for ingest by **AI systems**.

This can help users quantify their contributions.

End of Section

Why **Coze**?

Post-quantum?

Systems are not ready!

Coze is ready!

Coze assumes current algorithms will need to be **deprecated** in the future.

Post-quantum? Systems are not ready!

Coze makes deprecating algorithms in the future easy for your systems.

Most systems give **primitives** first class priority.

They don't think generally about crypto.

Coze gives abstractions first class priority so primitives are easy to use or replace.

Most systems tightly couple to a **single primitive** making upgrades very hard or impossible. (git)

Coze has **first class abstractions**.

Primitives are easy to use or replace.

Concept:

Authenticated Atomic Action (AAA)

Concept: **Authenticated Atomic Action (AAA)**

"Each Coze is authenticated"

Sessions are unneeded since each action is verifiable.

Bearer tokens are not needed.

Passwords are not needed.

Coze

Coze Replaces Traditional JSON Authentication

- Bearer Token Authentication + TLS
  - "Session Tokens"
  - Both TLS and bearer tokens are needed.
- Transport Independent
  - TLS (SSL/HTTPS)
  - HTTP
  - …

Each Coze is authenticated (AAA Authenticated Atomic Action)

Coze

If you're working on a JSON API and the transport is unknown, you'll need TLS + bearer tokens.

Or just use **Coze**.

This is so important, it's worth saying again:

If you're working on a JSON API and the transport is unknown, you'll need TLS + bearer tokens.

Or just use **Coze**.

Coze Replaces Traditional JSON Authentication

- Bearer Token Authentication
- Transport Independent
  - TLS (SSL/HTTPS)
  - HTTP
  - …

Each Coze is authenticated by itself.

No bearer/login system is needed.

Coze

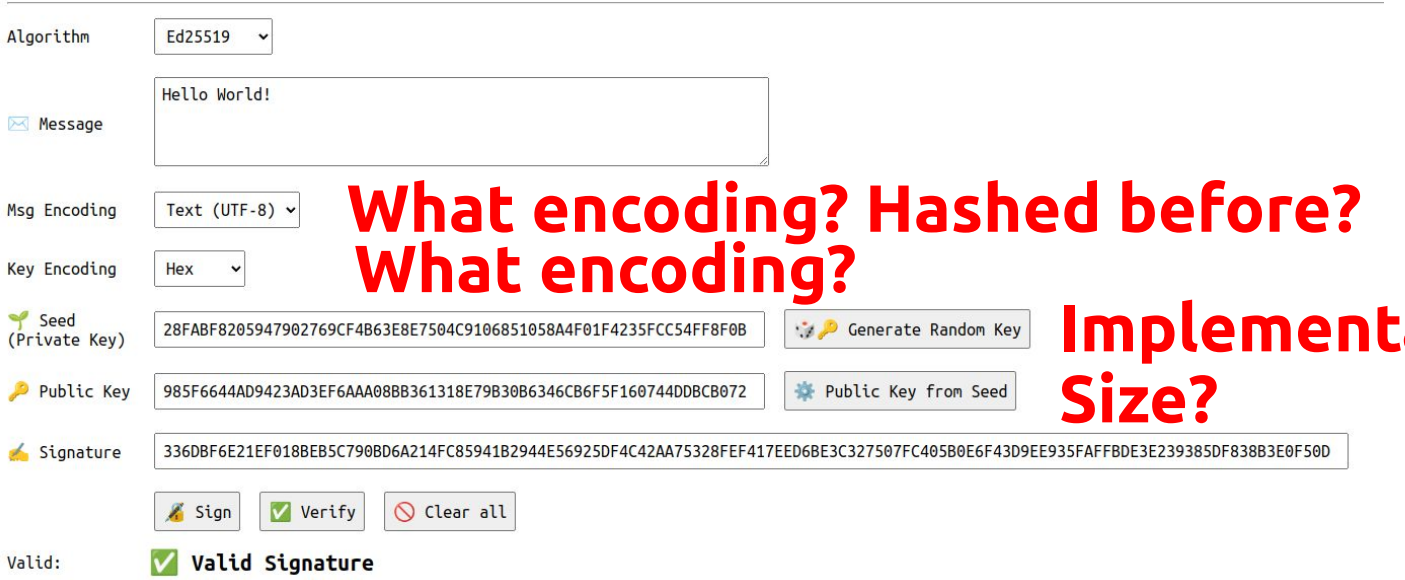
Coze standardizes usage across many primitives

Primitive Ed25519

- What's a private key?
  - Seed? `Secrete scalar s || prefix`? `Seed || Public`?
- Is the encoding Hex?
  - Base 64? Url or unsafe? Padded?
- Are "high s" signatures okay?

Coze

Sometimes there are **many ways** to implement a primitive.



What encoding? Hashed before?
What encoding?

Implementation?
Size?

Coze

Ed25519 Questions

- What's a private key for Ed25519?
  - `Secrete scalar s || prefix`? `Seed || Public`?
- How are keys (public/private) encoded?
- What encoding is used for messages?
  - Base64? B64ut? Hex? Other?
- Are "high s" signatures okay?

Coze

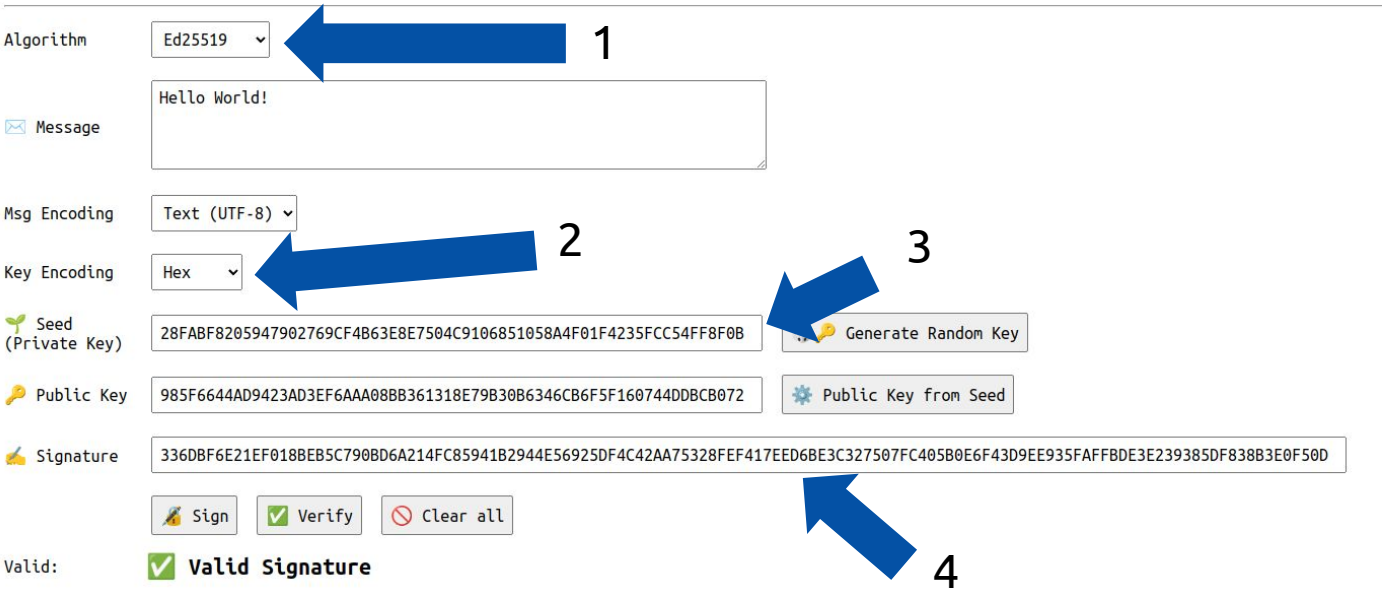
Coze answers!

- What's a private key for Ed25519?
  - **The seed.**
- Key encoding?
  - **Coze key**
- Encoding?
  - **b64ut.**
- Are "high s" signatures okay?
  - **No.**

Coze

Coze provides **generalization** and **standardization** for all applications.

There are four "hidden" parameters/variable various applications make decisions about



Standardization for all algorithms

Coze provides standardized abstractions.

✉️ Message
(JSON or text)

```
Hello World!
```

🔑 Key

```
{
 "alg": "ES256",
 "x": "OeKJKaDG45cPvXbWLCImABK7w68CnD8oCwAASLVzsS1zT2gHbjS1YJ1P6k_T4VV15RFspMH4arVzp4x3ukLKGQ",
 "d": "vYqQ3vFqcmJWPtgi5Az8FPGvQctHPTPeEY8RUZhzA7c",
 "tmb": "gDc5_WDO27WUs3LPfg7rcxqD8ZQDyYSy5A4VlMfR9YU",
 "iat": 1678062154,
 "kid": "My Cyphr.me Key."
}
```

✅ Verify | 🖋️ Sign | 🎲🔑 Generate Random key | ES256 | 🚫 Clear all

Coze

Coze

End of Section

Coze Advanced

Coze Canon

Canon

- A **canon** is a list of fields used for normalization.
  - ["alg","iat","tmb","typ"]
- Coze objects are canonicalized and hashed for creating digests, signing, and verification.
- The **canonical form** is generated by applying a given canon and removing unnecessary whitespace.
- `pay`'s default canon are the existing fields in order of appearance.
- A new canon applied to `pay` mutates its canon.

Coze

Default Canon is the fields by appearance. (UTF-8)

```
{
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1627518000,
   "tmb": "cLj8vs…",
   "typ": "cyphr.me/msg"
 }
```

has the canon

["msg","alg","iat","tmb","typ"]

Coze

Apply given canon **["msg","alg","iat","tmb","typ"]**

Fields are reordered and "foo" is dropped.

```
{
   "foo": "bar",
   "alg": "ES256",
   "msg": "Coze Rocks",
   "iat": 1627518000,
   "tmb": "cLj8vs…",
   "typ": "cyphr.me/msg"
 }
```

Apply Canon

["msg","alg","iat","tmb","typ"]

```
{
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1627518000,
   "tmb": "cLj8vs…",
   "typ": "cyphr.me/msg"
 }
```

Coze

Canonical Form generation steps

1. Apply a given canon.
2. Elide unnecessary whitespace.

Coze

Canonical Form

1. Apply a given canon. (Already done.)
2. Remove unnecessary whitespace.

```
{
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1627518000,
   "tmb": "cLj8vs…",
   "typ": "cyphr.me/msg"
 }
```

This is the canonical form

```
{"msg": "Coze Rocks","alg":
"ES256","iat": 1627518000,"tmb":
"cLj8vs…","typ": "cyphr.me/msg"}
```

Coze

Coze Key Canonical form

Pre-canonical form:

```
{
 "alg":"ES256",
 "iat":1623132000,
 "kid":"Zami's Majuscule Key.",
 "d":"bNstg4_H3m3SlROufwRSEgibLrBuRq9114OvdapcpVA",
 "tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
 "x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq39Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"
}
```

Applying **["alg","x"]** results in the canonical form:

```
{"alg":"ES256","x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq39Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"}
```

Coze

One of the key architectural decisions that allows **Coze** to be so small and simple is **canonicalization.**

Without canonicalization, the specification would be much more complicated.

Canonical Digest

Canonical digest

The canonical form:

```
{"alg":"ES256","x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq39Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"}
```

Has a canonical digest of:

```
"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk"
```

The canonical digest of a **Coze** key is `tmb`

Coze

`pay` Canonical digest

```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1627518000,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 }
}
```

Has a canonical digest of:

```
"LSgWE4vEfyxJZUTFaRaB2JdEclORdZcm4UVH9D8vVto"
```

The canonical digest of a `pay` is `**cad**`

Coze

`czd` Coze Canonical digest

```
{
 "cad": "LSgWE4vEfyxJZUTFaRaB2JdEclORdZcm4UVH9D8vVto",
 "sig":"ywctP6lEQ_HcYLhgpoecqhFrqNpBSyNPuAPOV94SThuztJek7x7H9mXFD0xTrlmQPg_WC7jwg70nzNoGn70JyA"
}
```

Has a canonical digest of:

```
"d0ygwQCGzuxqgUq1KsuAtJ8IBu0mkgAcKpUJzuX075M"
```

The canonical digest of ["cad","sig"] is `**czd**`

`czd` stands for **co**z**e** **d**igest, the digest of a coze.

Coze

Canonical Digest

Canonical digest of

- **key** is **tmb**
- **pay** is **cad**
- ["**cad**","**sig**"] is **czd**

Coze

"But JSON **objects** are unordered!"

Yes, but JSON **arrays** are ordered. In JSON, {} is unordered, [] is ordered.

UTF-8 is also ordered.

A **Coze** canon is an array. A **Coze** canon can be generated from UTF-8. **Coze** objects are serialized into UTF-8, which has order.

Coze

`coze` field names

Coze Fields

coze JSON name for Coze objects.

can Canon of `pay`.

cad Canonical digest of `pay`.

czd Coze digest over `["cad","sig"]`

pay Label for the signed payload.

sig Signature over `cad`.

Coze

A "full" **coze** is too much.

Simplify!

How to Simplify?

- `**key**` may be looked up using `**tmb**`.
- `**can**`, `**cad**`, and `**czd**` are recalculatable.
- The label `**coze**` may be inferred.
    - (An api endpoint should specify what kind of payloads it receives or generates. "This endpoint generates cozies with this canon.")

Coze

"Full" Verbose **Coze** - A coze with everything included


```
{
 "coze": {
   "pay": {
     "msg": "Coze Rocks",
     "alg": "ES256",
     "iat": 1627518000,
     "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
     "typ": "cyphr.me/msg"
   },
   "key": {
     "alg":"ES256",
     "iat":1623132000,
     "kid":"Zami's Majuscule Key.",
     "tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
     "x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq39Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"
   },
   "can": ["alg","iat","msg","tmb","typ"],
   "cad": "LSgWE4vEfyxJZUTFaRaB2JdEclORdZcm4UVH9D8vVto",
   "czd": "d0ygwQCGzuxqgUq1KsuAtJ8IBu0mkgAcKpUJzuX075M",
   "sig": "ywctP6lEQ_HcYLhgpoecqhFrqNpBSyNPuAPOV94SThuztJek7x7H9mXFD0xTrlmQPg_WC7jwg70nzNoGn70JyA"
 }
}
```

Elidable Parts


```
{
 "coze": {
   "pay": {
     "msg": "Coze Rocks",
     "alg": "ES256",
     "iat": 1627518000,
     "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
     "typ": "cyphr.me/msg"
   },
   "key": {
     "alg":"ES256",
     "iat":1623132000,
     "kid":"Zami's Majuscule Key.",
     "tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
     "x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq39Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"
   },
   "can": ["alg","iat","msg","tmb","typ"],
   "cad": "LSgWE4vEfyxJZUTFaRaB2JdEclORdZcm4UVH9D8vVto",
   "czd": "d0ygwQCGzuxqgUq1KsuAtJ8IBu0mkgAcKpUJzuX075M",
   "sig": "ywctP6lEQ_HcYLhgpoecqhFrqNpBSyNPuAPOV94SThuztJek7x7H9mXFD0xTrlmQPg_WC7jwg70nzNoGn70JyA"
 }
}
```

Simplified


```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1627518000,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 },
 "sig": "ywctP6lEQ_HcYLhgpoecqhFrqNpBSyNPuAPOV94SThuztJek7x7H9mXFD0xTrlmQPg_WC7jwg70nzNoGn70JyA"
}
```


Coze

99% use case. A **coze** should look like this:

```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1627518000,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 },
 "sig": "ywctP6lEQ_HcYLhgpoecqhFrqNpBSyNPuAPOV94SThuztJek7x7H9mXFD0xTrlmQPg_WC7jwg70nzNoGn70JyA"
}
```

Coze

But sometimes you may want to, or need to, include more **Coze** components.

Sometimes a verbose **coze** is needed

- By including `key`, the coze is **self-verifiable**. No key lookup is needed.
  - First time communication. What if the system doesn't have the key yet?
- By including the canon, API expectations can be explicitly set. Debugging.
- Including `czd`, `cad` serves as a checksum or explicit reference key.
- API endpoint may be flexible, so labels like `coze` might be required.


```
{
 "coze": {
   "pay": {
     "msg": "Coze Rocks",
     "alg": "ES256",
     "iat": 1627518000,
     "tmb":
"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
     "typ": "cyphr.me/msg"
   },
   "key": {
     "alg":"ES256",
     "iat":1623132000,
     "kid":"Zami's Majuscule Key.",

"tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",

"x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq3
9Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"
   },
   "can": ["alg","iat","msg","tmb","typ"],
   "cad":
"LSgWE4vEfyxJZUTFaRaB2JdEclORdZcm4UVH9D8vVto",
   "czd":
```

Coze Optional Fields

Optional **Coze** `pay` fields highlighted in yellow

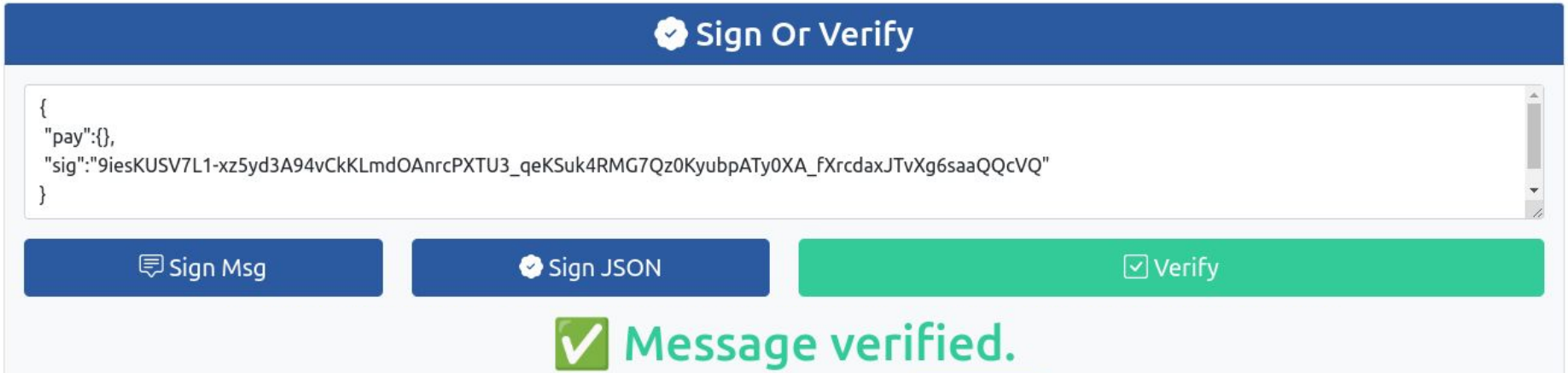
```
{
   "alg": "ES256",
   "iat": 1627518000,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 }
```

Coze

The Empty Coze (valid)


```
{
 "pay":{},
 "sig":"9iesKUSV7L1-xz5yd3A94vCkKLmdOAnrcPXTU3_qeKSuk4RMG7Qz0KyubpATy0XA_fXrcdaxJTvXg6saaQQcVQ"
}
```


Coze


Sign Or Verify
{
"pay":{},
"sig":"9iesKUSV7L1-xz5yd3A94vCkKLmdOAnrcPXTU3_qeKSuk4RMG7Qz0KyubpATy0XA_fXrcdaxJTvXg6saaQQcVQ"
}
Sign Msg
Sign JSON
Verify
Message verified.

Coze

All **Coze** `pay` fields are optional

`iat`, `typ` are not required.

`alg`, `tmb` may be implicit.

All **Coze** `pay` fields are optional

- It's not always a good practice to omit fields.
- **Use best practices**
  - Typically includes all **Coze** `pay` fields.
  - Typically excludes unneeded `coze` fields.

Coze

Coze key revoke

Coze has built-in key revokes

Coze Key Revoke

- A Coze key may be revoked by signing a **self-revoke coze**.
- A self-revoke coze has the field `**rvk**` with an integer value greater than `**0**`.
- Any non-zero integer value may be used for `rvk` to denote key revocation.
  - `1` is suitable to denote revocation.
  - `0` *does not* denote revocation.
- The **Unix timestamp** of now is the suggested value for `rvk`.

Coze

Coze Key Revoke

```
{
 "pay": {
   "alg": "ES256",
   "iat": 1655924566,
   "msg": "Posted my private key on github",
   "rvk": 1655924566,
   "tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/key/revoke"
 },
 "sig": "y3wpVXpBeaJNnUn8Q_3j9WOZH4gey78naDrP14TEToio0tloGP-6mNrXGQdWsvMvVYgg09EoxJYC9mE4PEuMXg"
}
```

- `rvk` - Unix timestamp of when the key was expired. (now)

Coze

Coze Key Revoke

- Key expiration policies, such as key rotation, are **outside the scope** of Coze.
- Third parties may revoke leaked keys.
  - Systems storing Coze keys should provide an interface permitting a given Coze key to be mark as revoked by receiving a self-revoke message.
  - Self-revokes with future times must immediately be considered as revoked.

Coze

Coze b64ut

URI, Canonical, Padding Truncated

Coze b64ut: **b**ase**64, U**RI, Canonical, Padding **T**runcated

- Binary values (digests, signatures) are encoded as **b64ut**.
- **There are many base 64's!**
  - There are 8 combinations of RFC base64.
    - (There's natural base 64's as well!)
  - "base 64" is too broad.
  - **Coze** uses the specific term **b64ut**.

Coze

b64ut: RFC 4648 **b**ase**64, U**RI, Canonical, Padding **T**runcated

- **RFC base64**
- **URI** - URI alphabet
  - Not the "URI unsafe" alphabet.
  - URI unsafe is popular, but not web friendly.
- **Canonical** - only one valid encoding
  - "hOk" and "hOl" decode to the same bytes.
  - Non-canonical base 64 is prohibited.
- **Padding Truncated** - No padding.
  - "hOk=" Coze omits padding, "hOk".
  - Typically the algorithm always includes padding and then padding is removed as an additional step, thus "truncated".

Coze

End of Section

For Go

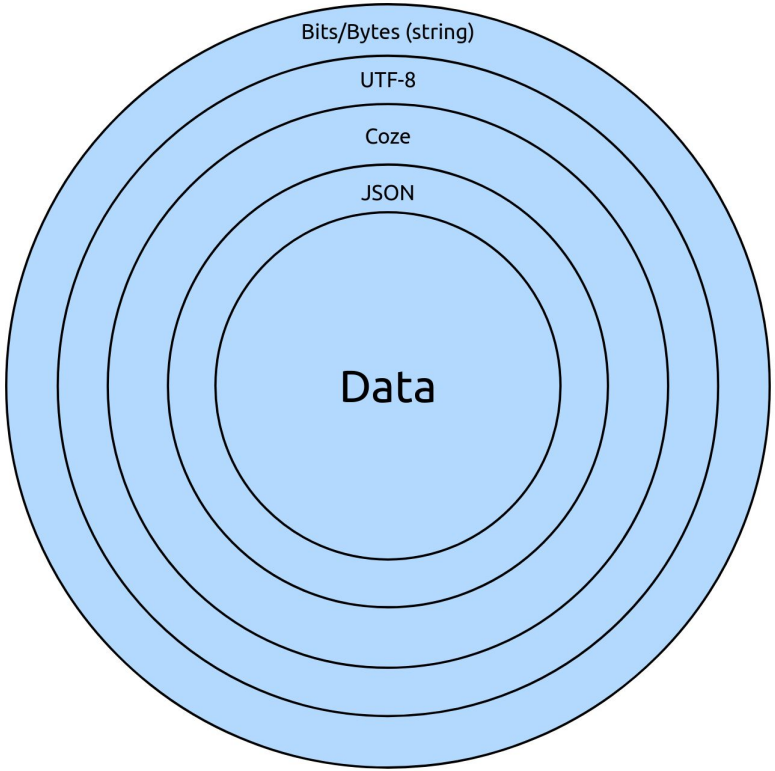
https://github.com/Cyphrme/Coze

For JSON APIs

https://github.com/Cyphrme/Coze

End of Section

The Coze Onion



Coze

https://github.com/Cyphrme/Coze

Disclaimer!

- We **respect** the various projects in the space.
- Other projects have **noble goals** and we're thankful they exists.
- It's not cool to trash someone else's work.
    - Authors work hard to bring value, frequently for free, to everyone.

That being said, it's important to give specific reason **why Coze's design is different** and why **Coze** was needed.

Coze

Coze Design Goals

1. Idiomatic JSON.
2. Human readable.
3. Small in scope.
4. Provide defined cipher suites.

Coze is simple

Simple	Complex
JSON	XML
UTF-8	UTF-16
Markdown	HTML
Coze	PGP/PEM/JWT/JOSE/Ect…

If you prefer XML over JSON, you may not like Coze's simple design.

Coze

Why **Coze**? Others were:

- **Not human readable.**
- **Not JSON.**
  - **Some specs claims to be JSON but then are not valid JSON.**
- **Not designed for future algorithms.**
- **Not small in scope.**
- Hard to use.
- Required specific libraries in specific languages.
- No online tools.
- No reference implementation.
- No longer maintained.

Coze

Coze  PGP

Coze

Coze @CozeJSON · 10h

I was born on 2021/06/08 (1623132000) which is 30 years and one day after the initial release of PGP 1.0.

Coze

PGP Key

Please don't use PGP.

https://blog.cryptographyengineering.com/2014/08/13/whats-matter-with-pgp/

Coze

Elliptic curve produces surprisingly large PGP keys

(1)

my miniLock ID 26r9AUYfVgAvJoof52mHhFjRRFfiUWC4HTt1cQT9cjs63k

(2)

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.1.0-ecc (GNU/Linux)

mFIETJPQrRMIKoZIzj0DAQcCAwQLx6e669XwjHTHe3HuROe7CloYMXuZbaU5PjOs
xSkyxtL2D00e/jWgufuNN4ftS+6XygEtB7jlglvnCTVF1TLmtCRlY19kc2FfZGhf
MjU2IDxvcGVucGdwQGJyYWluaHViLm9yZz6IegQTEwgAIgUCTJPQrQIbAwYLCQgH
AwIGFQgCCQoLBBYCAwECHgECF4AACgkQC6Ut8LqlnZzmXQEAiKgiSzPSpUOJcX9d
JtLJ5As98Alit2oFwzhxG7mSVmQA/RP67yOeoUtdsK6bwmRA95cwf9lBIusNjehx
XDfpHj+/uFYETJPQrRIIKoZIzj0DAQcCAwR/cMCoGEzcrqXbILqP7Rfke977dElX
XsRJEwrzftreZYrn7jXSDoiXkRyfVkvjPZqUvB5cknsaoH/3UNLRHClxAwEIB4hh
BBgTCAAJBQJMk9CtAhsMAAoJEAulLfC6pZ2clyYBAOSUmaQ8rkgihnepbnpK7tNz
3QEocsLEtsTCDUBGNYGyAQDclifYqsUChXlWKaw3md+yHJPcWZXzHt37c4q/MhIm
oQ==
=hMzp
-----END PGP PUBLIC KEY BLOCK-----
```

(3a)

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG/MacGPG2 v2.0.20 (Darwin)
Comment: GPGTools - http://gpgtools.org

mQGNBFPo5fABDACgB1jG3AoJAMY8JxZBcqhe6EL7afojr2xkL47YH767GDFW/tTk
o5tJrHSHuDQBYDK6ZAs79hSI2ycckBobHdexOWs70QYqrYv0YGdiRYFY/hkl8rmv
Nktb/20JeX6trLSrTkfvXyoHegYnJxt4wgUV3KkRDqH4bTw/0MoUbQVW+oGmWUXQ
73Yej3As85WeiCw0HcOkOteB7zaFukOL910PawsZtwqBr4dRrPU6FJd9V0SrKIwG
zd2loP5sNKkJGM4K0RVCzq/vzFyESik1t7yeetuEreJXnacXckErIVWbFtxnaJ0S
Q8neOrr9tjbdrs2SeCtkWxRiqaR8+SzA6SpgdZOv2JS/f65D9Wlpo02cdwBXHY8V
JjGPd8TD698O2BShz6e9Ofq6Uv4aJBMJ/5YbQ0d3EB4BqAE5heOWc1qXDPfJhNak
PxBqVa0PGa7Kqi0hGxqlakvNrZjx4UTxt/AjMn3BPEGgIRqB7E51HVsMCafup+9s
NsBDcnqEXIjVsakAEQEAAbQeVGVzdCBLZXkgPHRlc3RrZXlAdGVzdGtleS5uZXQ+
iQG9BBMBCgAnBQJT6OXwAhsDBQkHhh+ABQsJCAcDBRUKCQgLBRYCAwEAAh4BAheA
AAoJEPgPqBHec9M7mMgMAIBgniZCwUTFbgJdd/DzcShFpf2rz2MhcQ/eeis2Vuc7
xpQERR78EMvbA8Ori3CJ1V6HSkuehS0mKMD6TaIcRPKah7Dr7BhKCG2Ivt76LEAF
bhYpeWX+9IWfPLawNKEWo4FQMIWZfa/WErxmvHVyQ5zYAQe+MB6hjTlrP3q95vxH
JOzik/DnhmT5GTt3lu4GaINk2wiQcIbt63tiJ3ExIKerAP8hq7kS/pFItwoGxWNS
WZGkA4X/bABiF12LVuNnlVyKyYLKphQWIDPXgZxgITChPExBkEP50Zs1L6UqI8zu
gOgsip6Udb637gcB0goNInytI797U7+Dbxx30dR5DTq5CjsvEUJ7lxMGjcSFy6fJ
LyyIjvPvLjErOYeJ3hxa8Uk+Uvnh/d+OEt+Q1U7Bzb8mXBwCO7XCiFuP6X2SYTk1
PoBqcdk8CoisisoKQ7obSmvTDamrsxVIpDsW9rw6wiGa2OcIh/VFsHf6zEVb247q
TgJo7y4lJJX1SllmIXfdSrkBjQRT6OXwAQwAObTTyszTCjvz/oJ/unIpM9VuuVI2
pq+cvhTbMzTanqRvnfb+povYv+VmjAU1khvGdK3oyYOj5CWByRL2WGMGLXwWziRC
UDPnpvzxgxd2lUfCiAI+6IVI+Lc1ZNgigKJDWLB6RVRGEpX8FV19mFGgS7XL0Gxv
iJrXhYL3yxNdMkmFHCmDYrzqgcvEaddjFghCeKCjyQQEdykrMdnCHmBWvAZ2CtiW
lpREf/DHXVHua411+mX3hzDpdolA4PNElATtNbg2/IjQaxOaNnDJofAOs08TEBzM
hglBlY69LgQHCQJDye2zlFAiqtmo5qvx9Z5PHmmG8ZP5jaTzVQeS5yyrlJxYB6g7
SwdTihiRy1Ud8rQIN+cyWwG8IBOvO2m2fP3ezk48XR69RWqGmT88IyFamXKE1Zs+
ygs2A4kVp42uiaQo/ACS2Zfe6F9Ai2Z/JqgEu0E3scFApZyAlgxNT/Y9sauw3k38
ipF0koPiuGWyvUJlHZdPSfthbCJVcKLHZcfBABEBAAGJAaUEGAEKAA8FAlPo5fAC
GwwFCQeGH4AACgkQ+A+oEd5z0zsJ+Qv+OXHJ3eav7WRbWSRHjyzlGCjG85x6cW27
/ORj4ieqxcN+GHF6Il9DPhSjMBdoFr81QYcSjq+zeGhFuwynaKf0gz7Bf9JynINc
F90anF6pFZkyrlZn3Jzuvj7NEUbLb0lzM9aW77G+H0AALCFFnXslbEnDg8ghUe9C
zYYOTKzt7J1D0/zNqDAlam6SZoLOE/HueHqFz3bSyDu74R/JeycGzquxp/LJUuFt
xmwErtYOL3kZQ4nn00Qud1Yl1P9R9YDWct+8gCkM/ra2xJJnVqjVklch+NugGG2H
gISsZgK4LONXCZUS23OKrePXw8KJWASvgjCBnecMiUwl1fAzMbtmJ0HUrB9L8KGD
xVORAiQ0fFUDp6J7yY9npfysl3PsIV0qALEZS+KIl8Ykzlhga8Cgp5GDBRX9g8pL
Q0GWgI7evReAgE/PsmirlVIBj+TodK+4IYQAtPZjmBS6IUpWuMxtSuaTuNAcFeCr
DkO9RlQ1jELh/1gZV7qpxlWGWbSw8C4v
=3ND2
-----END PGP PUBLIC KEY BLOCK-----
```

(3b)

B268 0152 E274 EDE5 53C3 7C80 F80F A811 DE73 D33B

From section "PGP Keys Suck"
https://blog.cryptographyengineering.com/2014/08/13/whats-matter-with-pgp

Coze

"[There is] a **fundamental issue** with the PGP design.

PGP assumes keys are too big and complicated to be **managed by mortals**, but then in practice it practically **begs users** to handle them anyway."

From section "PGP Keys Suck"
https://blog.cryptographyengineering.com/2014/08/13/whats-matter-with-pgp/

Coze

Joseph Bonneau
@josephbonneau

Email from Phil Zimmerman: "Sorry, but I cannot decrypt this message. I don't have a version of PGP that runs on any of my devices"

11:55 AM · Sep 1, 2015 · Twitter Web Client

Ask me anything, 7 years later in 2022

[–] **okeefe** [+1] **309 points** 23 hours ago

Is it weird that I expected proof to be a PGP-signed message?

permalink source embed save save-RES report give award reply hide child comments

[–] **prz1954** [+4] Verified [F] **448 points** 22 hours ago

LOL! Not weird at all. Let me tell you something even more weird. I have not used PGP for many years, because it does not run on my iPhone, where I process nearly all my email. Yup. Weird indeed.

permalink source embed save save-RES parent report give award reply hide child comments

Reply by Phil Zimmermann, author of PGP

Coze

https://old.reddit.com/r/IAmA/comments/x3b3gj/im_phil_zimmermann_and_i_created_pgp_the_most/imoexbh/

PGP: Too hard for mere mortals and mere gods?

Using tools shouldn't be so hard that the authors themselves don’t use it.

Coze

JOSE

Coze

What's good about JOSE?

- Updates old standards that are hard to use or require dependencies.
- Defines cryptographic key representation in JSON.
- Key has a thumbprint (applies to JWK).
    - Like a PGP/SSH fingerprints or an Ethereum address.
    - Thumbprints universally address specific keys.
- Defines algorithm suites.
- Uses some JSON.
- Some parts are human readable.

Coze

Coze vs. JWT
The Spec

Coze spec

- A markdown document on Github.
- A reference implementation written in Go.

JOSE spec

Lots: JWS/JWE/JWA/JWK/JWT

- JSON Web Signature (JWS) (RFC 7515)
- JSON Web Encryption (JWE) (RFC 7516)
- JSON Web Key (JWK) (RFC 7517)
- JSON Web Algorithms (JWA) (RFC 7518)
- JSON Web Token (JWT)(RFC 7519)
- Examples of Protecting Content Using JSON Object Signing and Encryption (JOSE) (RFC 7520)
- JSON Web Key (JWK) Thumbprint (RFC 7638)
- JSON Web Signature (JWS) Unencoded Payload Option (RFC 7797)
- Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs) (RFC 7800)
- FRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE) (RFC 8037)
- JSON Web Token Best Current Practices (RFC 8725)
- …

Coze

Coze vs. JOSE

The Spec

JOSE has no reference implementations.

A reference implementation would have

1. Informed specification design decisions.
2. Demonstrated best practices and resolved any ambiguities.
3. There have been some critical errors (like the no alg bug) in industry standard implementations.

Coze vs. JWT
Signature Malleability

Coze vs. JOSE: Replay attack prevention

Coze prohibits signature malleability.

Replay prevention using `czd`.

JOSE allows signature malleability.

JOSE requires application defined identifiers.
https://www.rfc-editor.org/rfc/rfc7515#section-10.10

Various systems are not out-of-the-box compatible.

Coze

Jose considers signature malleability to be **out of scope**. Does not provide best practices

Coze considers this in scope. "Provide defined cipher suites".

Coze vs. JOSE

JSON

A JSON Web Token (JWT)

is not JSON

Yes, JWT has "JSON" in the name,

but it's not JSON.

This dog's name is "tiger"

It looks kinda like a tiger, its name is "tiger",
but it's not a tiger.

Coze

Even after JWT's are decoded they are still not JSON

JOSE JWT

JWT base64 Encoded:

```
eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE2Mjc1MTgwMDAsIm1zZyI6IkNvemUgUm9ja3MiLCJ0bWIiOiJyYkxYM1NzV0xXQkpvSXFNUENOVUZ1VURScFZVX28tMFNERms4WWxURXU4IiwidHlwIjoiY3lwaHIub WUvbXNnL2NyZWF0ZSJ9.xjBKxqf9JQDgK_HMunPMQDwmREBCKNMqypffpRrKbqqRd2djh-jDg1Rwzpfv9YaMO1-QNS_Q-iLE5eg5iZnZzw
```

Decoded:

```
{
  "alg": "ES256",
  "typ": "JWT"
}.{
  "iat": 1627518000,
  "msg": "Coze Rocks",
  "tmb": "rbLX3SsWLWBJoIqMPCNUFuUDRpVU_o-0SDFk8YlTEu8",
  "typ": "cyphr.me/msg/create"
}
.quQsSXbY3RAQeNgN4GArL1pQCqKgn9MFqao124KXgQfRFPEhs5WMP-NbaAiSnHyQPC0NMcveafE12TRYe8j_-Q
```

Not JSON

Still not JSON

Coze

This problem, all by itself, is enough for another standard to be warranted.

Coze vs. JWT

Encoding

Coze vs. JWT
Re-encode ballooning

Coze

JOSE Re-encode ballooning

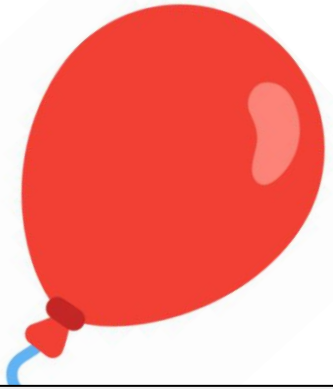
JOSE **encodes** binary values into base64

Then it **re-encodes** the already encoded values into base64, again.

Coze

JOSE Re-encode ballooning

54 characters to 72 characters



```
{"tmb": "cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk"}
```

```
eyJ0bWIiOiAiY0xqOHZzWXRNQndZa3pvRlZaSEJabzZTTkw4d1NkQ0lqQ0tBd1hOdWhPayJ9
```

Re-encoding results in needlessly large messages.
If JWT remained in JSON, this would not be an issue.

Coze

https://convert.zamicol.com/#?inAlph=text&in=%257B%2522tmb%2522%253A%2520%2522cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk%2522%257D&outAlph=b64ut

Encoded form is **larger** than the unencoded form

58 characters Vs. 42 Characters.

```
eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19
```

```
{"alg":"HS256","b64":false,"crit":["b64"]}
```

Coze

Smaller is better: Coze vs. JWT

227 bytes


```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1627518000,
"tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8w
SdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 },
"sig":"ywctP6lEQ_HcYLhgpoecqhFrqNpB
SyNPuAPOV94SThuztJek7x7H9mXFD0xTrlm
QPg_WC7jwg70nzNoGn70JyA"
}
```


280 bytes


```
eyJhbGciOiJFUzI1NiIsInR5cCI6Ik
pXVCJ9.eyJtc2ciOiJDb3plIFJvY2t
zIiwiaWF0IjoxNjI3NTE4MDAwLCJ0b
WIiOiJyYkxYM1NzV0xXQkpvSXFNUEN
OVUZ1VURScFZVX28tMFNERms4WWxUR
XU4IiwidHlwIjoiY3lwaHIubWUvbXN
nL2NyZWF0ZSJ9.7uLr31zS5_I-UeJW
j4Olrufu9C7sr2-2DB4dDyKY4yf3g6
Jr30JSLS3wfyMEWUbW1OVAzsB1wYha
WbUz0VWtGA
```


See other slide. Using key:

{"kty":"EC","d":"MAeyJgK2kB5Ysou4tLbaiowPqGCQj15hG3B5f65lG9w","crv":"P-256","x":"FD7r1byJlBcvsBXTi47ltG4JRbMVHgxp91Ds2eFiM_g","y":"Sfec2TDH4QJ1F
BmlogTgHB6y00H9LiVQ7MoqB6Tidg"}

Coze is 19% smaller

Coze

Coze Human readable and smaller?
Yep!

(Re-encode ballooning is a tragic design flaw.)

227 bytes


```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1627518000,
"tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8w
SdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 },
"sig":"ywctP6lEQ_HcYLhgpoecqhFrqNpB
SyNPuAPOV94SThuztJek7x7H9mXFD0xTrlm
QPg_WC7jwg70nzNoGn70JyA"
}
```


Binary values are encoded just once

See other slide. Using key:
{"kty":"EC","d":"MAqyJgK2kB5YsouHtLbaiowPqGCQj15hG3B5f65lG9w","crv":"P-256","x":"FD7r1byJlBcvsBXTi47ltG4JRbMVHgxp91Ds2eFiM_g","y":"5fec2TDH4QJ1F
BmlogTgHB6y00H9LiVQ7MoqB6Tidg"}

Coze

All binary values in the payload are encoded twice

280 bytes

```
eyJhbGciOiJFUzI1NiIsInR5cCI6Ik
pXVCJ9.eyJtc2ciOiJDb3plIFJvY2t
zIiwiaWF0IjoxNjI3NTE4MDAwLCJ0b
WIiOiJyYkxYM1NzV0xXQkpvSXFNUEN
OVUZ1VURScFZVX28tMFNERms4WWxUR
XU4IiwidHlwIjoiY3lwaHIubWUvbXN
nL2NyZWF0ZSJ9.7uLr31zS5_I-UeJW
j4Olrufu9C7sr2-2DB4dDyKY4yf3g6
Jr30JSLS3wfyMEWUbW1OVAzsB1wYha
WbUz0VWtGA
```

See other slide. Using key:
{"kty":"EC","d":"MAeyJgK2kB5Ysou4tLbaiowPqGCQj15hG3B5f65lG9w","crv":"P-256","x":"FD7r1byJlBcvsBXTi47ltG4JRbMVHgxp91Ds2eFiM_g","y":"5fec2TDH4QJ1F
BmlogTgHB6y00H9LJVQ7MoqB6Tidg"}

Coze

Coze vs. JWT
Human Readability

Coze vs. JWT

Human Readable


```
{
 "pay": {
    "msg": "Coze Rocks",
    "alg": "ES256",
    "iat": 1627518000,
"tmb":"cLj8vsYtMBwYkzoFVZHBZo6SN
L8wSdCIjCKAwXNuhOk",
    "typ": "cyphr.me/msg"
 },
"sig":"ywctP6lEQ_HcYLhgpoecqhFrq
NpBSyNPuAPOV94SThuztJek7x7H9mXFD
0xTrlmQPg_WC7jwg70nzNoGn70JyA"
}
```


Hieroglyphics


```
eyJhbGciOiJFUzI1NiIsInR5cCI6Ik
pXVCJ9.eyJtc2ciOiJDb3plIFJvY2t
zIiwiaWF0IjoxNjI3NTE4MDAwLCJ0b
WIiOiJyYkxYM1NzV0xXQkpvSXFNUEN
OVUZ1VURScFZVX28tMFNERms4WWxUR
XU4IiwidHlwIjoiY3lwaHIubWUvbXN
nL2NyZWF0ZSJ9.7uLr31zS5_I-UeJW
j4Olrufu9C7sr2-2DB4dDyKY4yf3g6
Jr30JSLS3wfyMEWUbW1OVAzsB1wYha
WbUz0VWtGA
```


See other slide. Using key:

{"kty":"EC","d":"MAeyJgK2kB5Ysou4tLbaiowPqGCQj15hG3B5f65lG9w","crv":"P-256","x":"FD7r1byJlBcvsBXTi47ltG4JRbMVHgxp91Ds2eFiM_g","y":"Sfec2TDH4QJ1F
BmlogTgHB6y00H9LJVQ7MoqB6Tidg"}

Coze

Coze vs. JWT encoding

Coze

- **JSON**
- 227 bytes.
- Human readable.
- UTF-8

JWT

- **Not JSON**.
- 280 bytes.
- Not human readable.
- UTF-8 -> base64

Coze is 19% smaller

Coze

What if JOSE looked more like **Coze**?

Let's tweak JOSE to be like **Coze**.

(Maybe we're not being fair, so let's apply the same standard to JWT.)

Coze vs. Hypothetical "Really Unencoded" JWS

227 bytes


```
{
 "pay": {
   "msg": "Coze Rocks",
   "alg": "ES256",
   "iat": 1627518000,
"tmb":"cLj8vsYtMBwYkzoFVZHBZo6SN
L8wSdCIjCKAwXNuhOk",
   "typ": "cyphr.me/msg"
 },
"sig":"ywctP6lEQ_HcYLhgpoecqhFrq
NpBSyNPuAPOV94SThuztJek7x7H9mXFD
0xTrlmQPg_WC7jwg70nzNoGn70JyA"
}
```


251 bytes


```
{
"Protected":
{"alg":"ES256"},
"Payload":{
"iat":1627518000,"msg":"Coze
Rocks","tmb":"rbLX3SsWLWBJoIqM
PCNUFuUDRpVU_o-0SDFk8YlTEu8","
typ":"cyphr.me/msg"},
"signature":"quQsSXbY3RAQeNgN4
GArL1pQCqKgn9MFqao124KXgQfRFPE
hs5WMP-NbaAiSnHyQPC0NMcveafE12
TRYe8j_-Q"
}
```


(Not valid JWS)

See other slide. Using key:
{"kty":"EC","d":"MAeyJgK2kB5Ysou4tLbaiowPqGCQj15hG3B5f65lG9w","crv":"P-256","x":"FD7r1byJlBcvsBXTi47ltG4JRbMVHgxp91Ds2eFiM_g","y":"5fec2TDH4QJ1F
BmlogTgHB6y00H9LJVQ7MoqB6Tidg"}

Coze

JOSE RFC

Valid unencoded JWS

```
{"protected":"eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19","payload":"$.02","signature":"A5dxf2s96_n5FLueVuW1Z_vh161FwXZC4YLPff6dmDY"}
```

147 characters

JOSE Hypothetical

JWS with true unencoding (invalid)

```
{"protected":{"alg":"HS256","b64":false,"crit":["b64"]},"payload":"$.02","signature":"A5dxf2s96_n5FLueVuW1Z_vh161FwXZC4YLPff6dmDY"}
```

131 characters

This form is incompatible/invalid JOSE

Coze

Coze is still smaller.

JOSE does provide a mode outside of JWT, JWS

How does it compare to **Coze**?

JWS (not a JWT) - JOSE JSON serialization

```
{
  "payload":
   "eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGF
    tcGxlLmNvbS9pc19yb290Ijp0cnVlfQ",
  "signatures":[
   {"protected":"eyJhbGciOiJSUzI1NiJ9",
    "header":
     {"kid":"2010-12-29"},
    "signature":
     "cC4hiUPoj9Eetdgtv3hF80EGrhuB__dzERat0XF9g2VtQgr9PJbu3XOiZj5RZ
      mh7AAuHIm4Bh-0Qc_lF5YKt_O8W2Fp5jujGbds9uJdbF9CUAr7t1dnZcAcQjb
      KBYNX4BAynRFdiuB--f_nZLgrnbyTyWzO75vRK5h6xBArLIARNPvkSjtQBMHl
      b1L07Qe7K0GarZRmB_eSN9383LcOLn6_dO--xi12jzDwusC-eOkHWEsqtFZES
      c6BfI7noOPqvhJ1phCnvWh6IeYI2w9QOYEUipUTI8np6LbgGY9Fs98rqVt5AX
      LIhWkWywlVmtVrBp0igcN_IoypGlUPQGe77Rw"},
   {"protected":"eyJhbGciOiJFUzI1NiJ9",
    "header":
     {"kid":"e9bc097a-ce51-4036-9562-d2ade882db0d"},
    "signature":
     "DtEhU3ljbEg8L38VWAfUAqOyKAM6-Xx-F4GawxaepmXFCgfTjDxw5djxLa8IS
      lSApmWQxfKTUJqPP3-Kg6NU1Q"}]
}
```

Technically JSON? Yes.

Is it **good** JSON?

Does good JSON have encoded JSON blobs inside more JSON?

Example from **RFC 7515** A.6.4

Even JWS base64 encodes JSON blobs inside more JSON.

This is true of JWT too.

Others noticed this problem, so there was an extension

"Unencoded JWTs" (RFC 7797)

They are still encoded despite the name

(Still base64's JSON into JSON)

JOSE JSON serialization unencoded (RFC 7797)

This is the header from an "unencoded" JWS (not a mistake)

```
eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19
```

Needs another unencoding step:

```
{"alg":"HS256","b64":false,"crit":["b64"]}
```

42 vs 58 characters

https://datatracker.ietf.org/doc/html/rfc7797#section-4.2

Coze

An "**unencoded**" JWT (RFC 7797)

is still **not unencoded**.

Coze

JOSE JSON serialization "unencoded" (RFC 7797)

```
{
  "protected":
  "eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19",
  "payload":
  "$.02",
  "signature":
  "A5dxf2s96_n5FLueVuW1Z_vh161FwXZC4YLPff6dmDY"
}
```

"Unencoded"

```
{"alg":"HS256","b64":false,"crit":["b64"]}
```

Finally decoded

https://datatracker.ietf.org/doc/html/rfc7797#section-4.2

Coze

Observation: JWT is not JOSE.

Observation: JWT is not JOSE.

- Many libraries don't care about JOSE.
- Further, they don't care about JWS.
- Further, they only care about the signing portion of JWT.
  - Most libraries don't support JSON encoded JWS.
  - We can't find a "JWT" library that does JWE (2022).
    - JWE is half of JWT.

This hints that JOSE is oversized for the niche.

(As a side complaint, most people online say "JWT" meaning a compatified **JWS**. JWT can be a **JWS** or a **JWE**, but most libraries don't implement JWE, meaning most libraries provide only partial JWT support despite their name. JWT itself is overused relative to JWS and in many cases is worse than the "unencoded" JWS option.)

Coze vs. JWT
Duplicate Fields

Coze

Coze vs. JOSE duplicates

Coze

`Errors on duplicate`

JOSE

🚨**Allows duplicates with last-value-wins**🚨

See other slide. Using key:
{"kty":"EC","d":"MAeyJgK2kB5Ysou4tLbaiowPqGCQj15hG3B5f65lG9w","crv":"P-256","x":"FD7r1byJlBcvsBXTi47ltG4JRbMVHgxp91Ds2eFiM_g","y":"5fec2TDH4QJ1FBmlogTgHB6y00H9LiVQ7MoqB6Tidg"}

Coze

JOSE permits duplicate claims.

This is a significant **security** issue.

"JWT parsers MUST either reject JWTs with duplicate
Claim Names or use a JSON parser that returns only the lexically last
duplicate member name" - RFC 7519

https://datatracker.ietf.org/doc/html/rfc7519#section-4

Coze

Duplicate claims are a security problem

```
{
        "give_money_to": "Mom",
        …
        …
        …
        …
        "give_money_to": "Evil hacker"
}
```

JOSE allows **last-value-wins**, so "Evil hacker" may win, depending on the JWT parser.

Coze

Error on duplicate is the only correct behavior.

Coze vs. JOSE
Canonicalization

JOSE does not canonicalize
(other than thumbprints)

Message agreement is not specified by JSON. Applications make their own.

JOSE has no equivalent to `cad` and `czd`

Coze

JOSE doesn't canonicalize

From the RFC:

> The JWT MUST conform to either the [JWS] or [JWE] specification. Note that whitespace is explicitly allowed in the representation and no canonicalization need be performed before encoding.

... Applications may need to define a convention for the canonical case [...] if more than one party might need to produce the same value so that they can be compared.

In JOSE, you're on your own to canonicalize.

Canonicalization is built into **Coze**.

Canonicalization allows Coze to be simple.

Coze vs. JWK
Keys

Coze tmb, kid vs. JWK kid

JOSE's kid is like **Coze**'s tmb.

JOSE does not have equivalent to **Coze**'s kid.

Coze tmb and kid vs. JOSE kid

tmb

- Always there.
- Used programmatically.
- Defined.
- All systems agree, recalulatable by everyone.

JOSE kid

- Key may not have a kid.
- Defined for programmatic use.
- Defined as anything, no default value.
  - May be application defined.
- Systems may not agree. May have multiple/different `kid`s
- (Why even have this in the spec?)

kid

- Human readable label.
- Not programmatic.

No equivalent

See other slide. Using key:
{"kty":"EC","d":"MAqyJgK2kBSYsouHtLbaiowPqGCQj15hG3B5f65IG9w","crv":"P-256","x":"FD7r1byJIBcvsBXTi47ltG4JRbMVHgxp91Ds2efiM_g","y":"Sfec2TDH4QJ1fBmIogTgHB6y00H9LiVQ7MoqB6TIdg"}

Coze key vs. JWK (5 vs. 9)


```
{
"alg":"ES256",
"d":"bNstg4_H3m3SlROufwRSEgibLrBuRq9114OvdapcpVA",
"iat":1624472390,
"kid":"Zami's Majuscule Key.",
"tmb":"cLj8vsYtMBwYkzoFVZHBZo6SNL8wSdCIjCKAwXNuhOk",
"x":"2nTOaFVm2QLxmUO_SjgyscVHBtvHEfo2rq65MvgNRjORojq39Haq9rXNxvXxwba_Xj0F5vZibJR3isBdOWbo5g"
}
```



```
{
 "kty": "EC",
 "crv": "P-256",
 "iat": 1624472390,
 "kid": "A JWK",
 "tmb":
"AUj0zZCTycvj6L940+bJuCTxHdLynisaYw3Rzh4XbN0",
 "d":
"MAqyJgK2kB5YsouHtLbaiowPqGCQj15hG3B5f65IG9w",
 "x":
"FD7r1byJlBcvsBXTi47ltG4JRbMVHgxp91Ds2efiM_g",
 "y":
"5fec2TDH4QJ1fBmIogTgHB6y00H9lJiVQ7MoqB6Tidg",
"use":"sig"
}
```


Both are private keys

Coze: `alg` is all you need.

"alg":"ES256"

Jose: Verbose

"kty": "EC",
"crv": "P-256",
"use":"sig"

Psudocode:

```
If kty == EC &&
crv == p-256, alg
= ES256; if use
!= sig, throw
error
```

Coze

Coze: `alg` is all you need.
"alg":"ES256"

Family	ec
Use	sig
Curve	P-256
Hash	SHA-256
HashSize	32
SigSize	64
XSize	64
Etc…	

Coze

Coze Thumbprint vs JWK Thumbprint

JWT:
Key (UTF-8) -> base64 -> ASCII -> digest (bytes) -> thumbprint

Coze:
Key (UTF-8) -> digest (bytes) -> thumbprint

- Fewer steps
- **There's no need to convert UTF-8 to base64 to ASCII before hashing.**
- **JWK currently always uses SHA-256 regardless of alg.**
    - Not cryptographically consistent.
    - Algorithms that don't use SHA-256 still need SHA-256 because of this arbitrary requirement.

Coze

JOSE: tmb's hash isn't known

- JWK has no explicit denotation for the thumbprint's hashing algorithm.
  - Everything is currently required to use SHA-256.
- JWK: although Ed25519 uses SHA-512, its thumbprint is made using SHA-256.

  https://datatracker.ietf.org/doc/html/rfc8037#appendix-A.3

Coze

Including a thumbprint in a message

JWT:

Key (UTF-8)-> base64 -> digest (bytes) -> thumbprint -> UTF-8 (message with thumbprint)-> base64 -> digest -> signature

Coze:

Key (UTF-8) -> digest (bytes) -> UTF-8 (message with thumbprint) -> digest -> signature

Yellow is Extra steps.

Coze

Coze vs. JWS
Decoding

Adventure: Decode RFC's example JWS

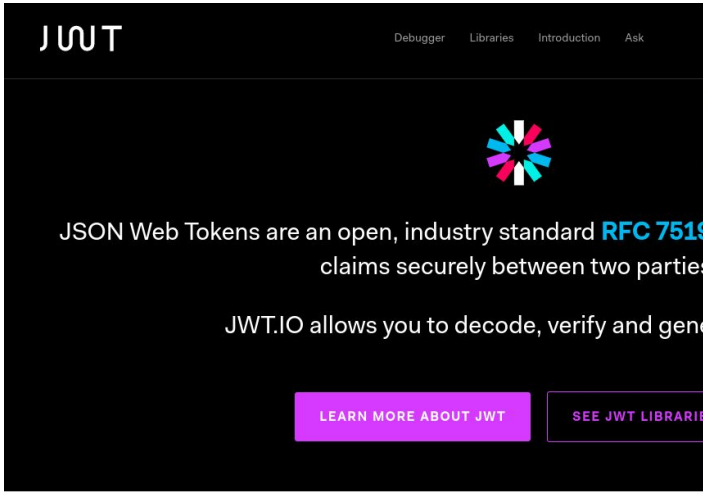
Adventure: Decode the specification example JWS


```
{
    "payload":
     "eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGF
      tcGxlLmNvbS9pc19yb290Ijp0cnVlfQ",
    "signatures":[
     {"protected":"eyJhbGciOiJSUzI1NiJ9",
      "header":
       {"kid":"2010-12-29"},
      "signature":
       "cC4hiUPoj9Eetdgtv3hF80EGrhuB__dzERat0XF9g2VtQgr9PJbu3XOiZj5RZ
        mh7AAuHIm4Bh-0Qc_lF5YKt_O8W2Fp5jujGbds9uJdbF9CUAr7t1dnZcAcQjb
        KBYNX4BAynRFdiuB--f_nZLgrnbyTyWzO75vRK5h6xBArLIARNPvkSjtQBMHl
        b1L07Qe7K0GarZRmB_eSN9383LcOLn6_dO--xi12jzDwusC-eOkHWEsqtFZES
        c6BfI7noOPqvhJ1phCnvWh6IeYI2w9QOYEUipUTI8np6LbgGY9Fs98rqVt5AX
        LIhWkWywlVmtVrBp0igcN_IoypGlUPQGe77Rw"},
     {"protected":"eyJhbGciOiJFUzI1NiJ9",
      "header":
       {"kid":"e9bc097a-ce51-4036-9562-d2ade882db0d"},
      "signature":
       "DtEhU3ljbEg8L38VWAfUAqOyKAM6-Xx-F4GawxaepmXFCgfTjDxw5djxLa8IS
        lSApmWQxfKTUJqPP3-Kg6NU1Q"}]
   }
```

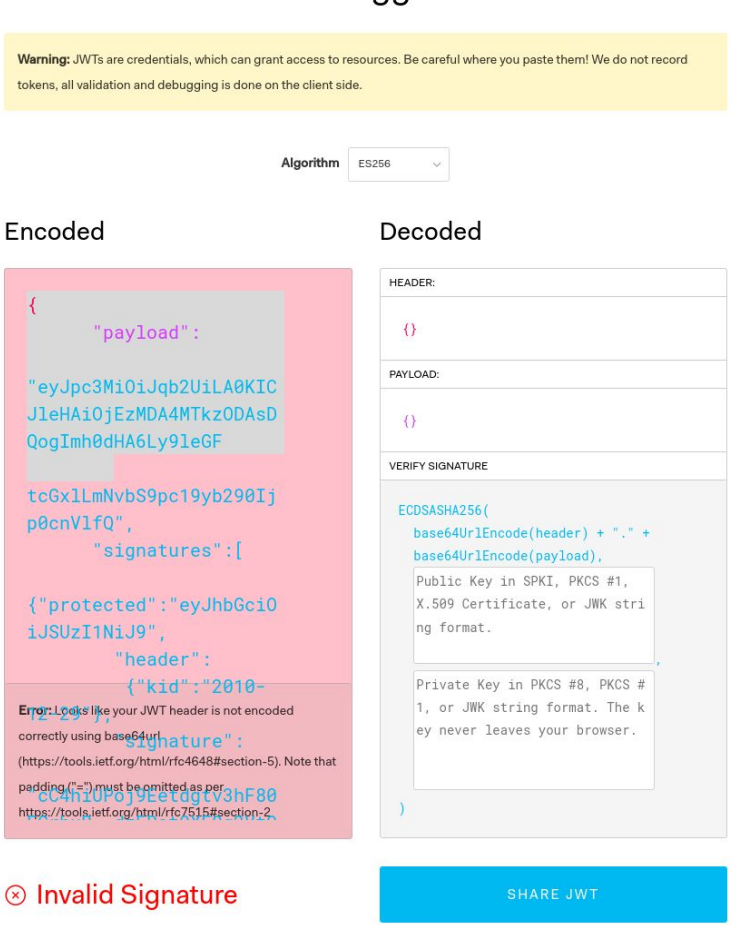

I can't read base64 so I want to know what's in this.

https://datatracker.ietf.org/doc/html/rfc7515#appendix-A.6.4

Coze

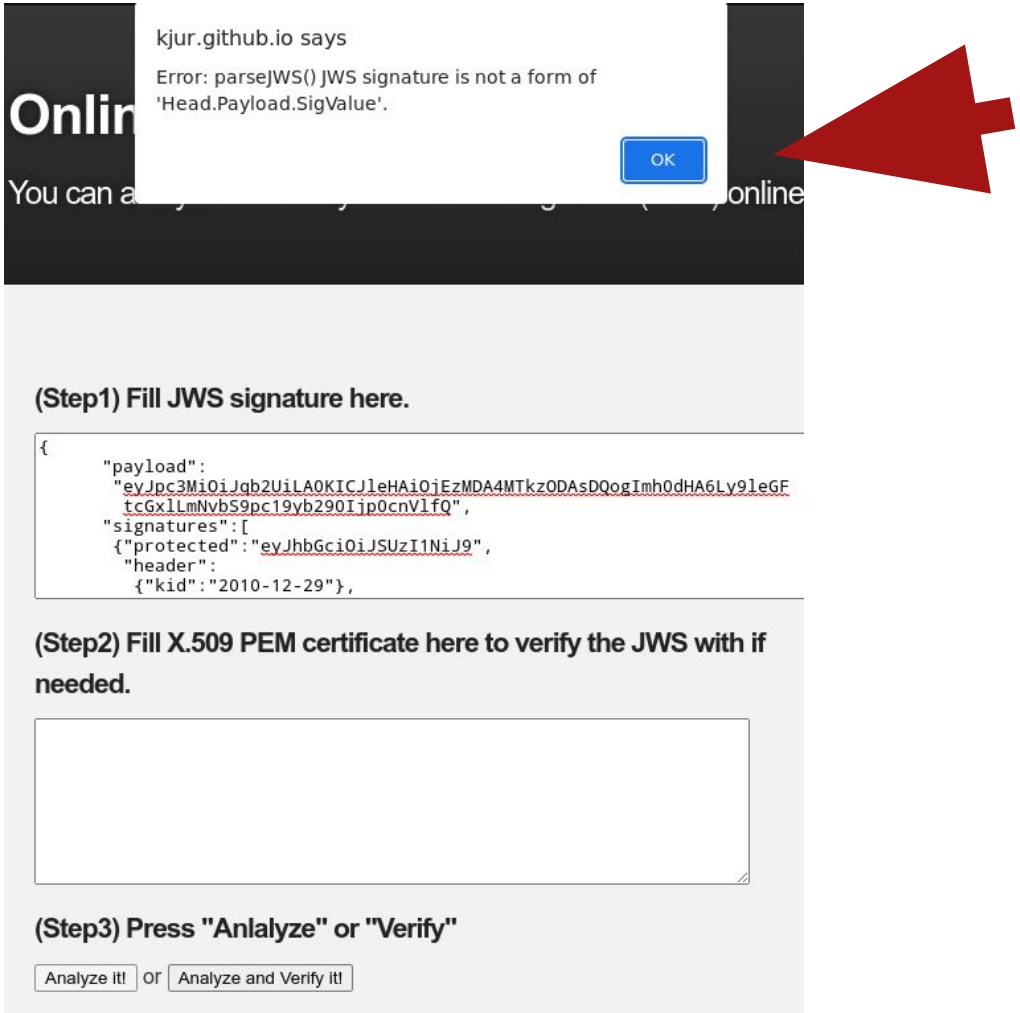


To be fair, this is a JWT decoder instead of a JWS decoder.

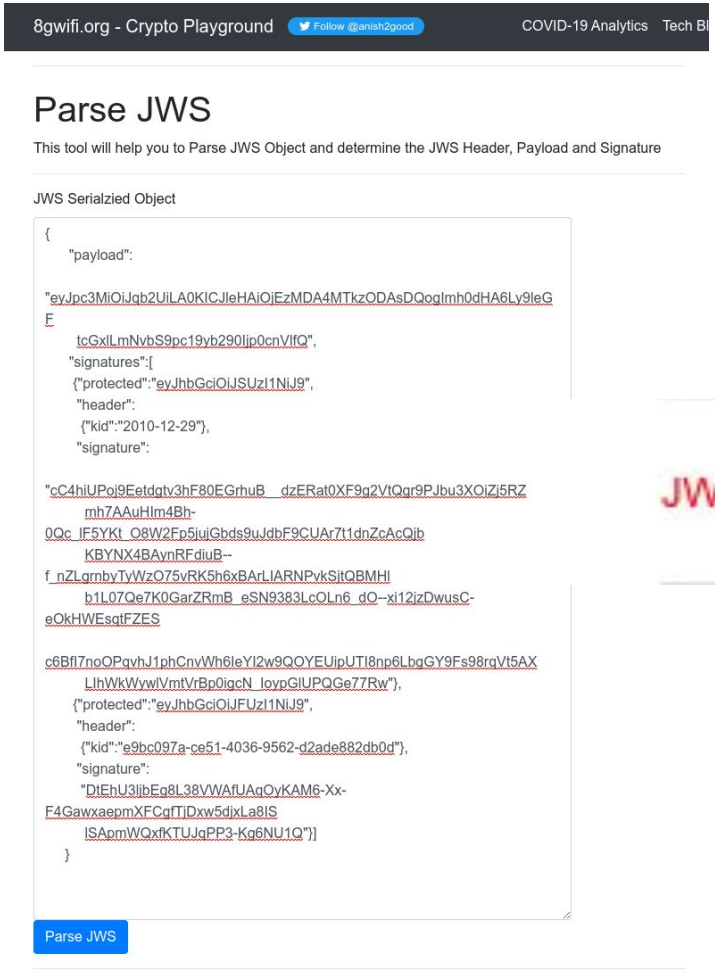


Coze

This is a "JWS" decoder.



Coze



Another "JWS" decoder.

JWS Seriazed Object is Not Valid....

Coze

Conclusion: There is **no** online tool that can decode the RFC Example JWS.

Why did JOSE use base64?

Streaming, JOSE supports binaries, and URI safety.

Coze design stance is that binaries should stay binaries, JSON should be JSON.

JSON should remain human readable where possible. If needed, use URI escaping (industry standard) or base64 once done.

It's not that much more overhead, but allows flexibility for any digest and reference.

End of presentation

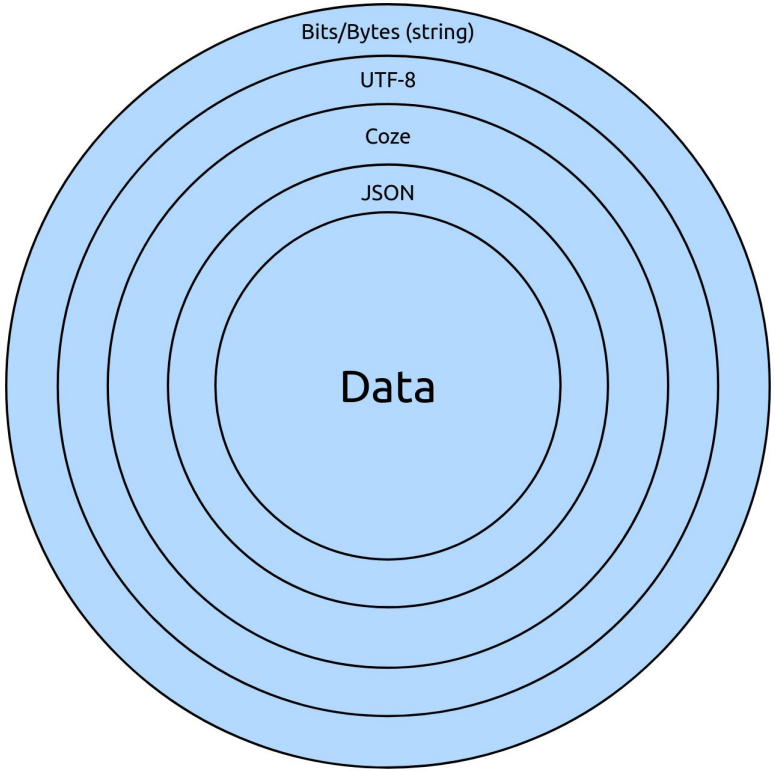
"JSON objects are unordered. Therefore Coze/JWT/Others bad."

We think this is an obviously silly argument, but let's state why:

- UTF-8 wraps JSON. Coze is a layer between UTF-8 and JSON.
- Order is transmitted by UTF-8, which is obviously ordered.
  - If UTF-8 didn't have order, the letters in this sentence would be out of order.
- Coze `can` is an array and arrays are ordered in JSON.
  - It's not true that all parts of JSON are unordered. JSON arrays are ordered, and we can take advantage of that for any seeming weakness of object ordering.
- Coze is JSON, but JSON is not necessarily valid Coze. Coze wraps JSON.
- Coze operations are over firstly digests, and secondly over UTF-8 not the "abstract JSON". JSON must first be serialized before being processed by Coze. JSON defines UTF-8 as its serialization method. UTF-8 is the specified, thus valid, JSON serialization method.

Coze

The Coze Onion



Coze