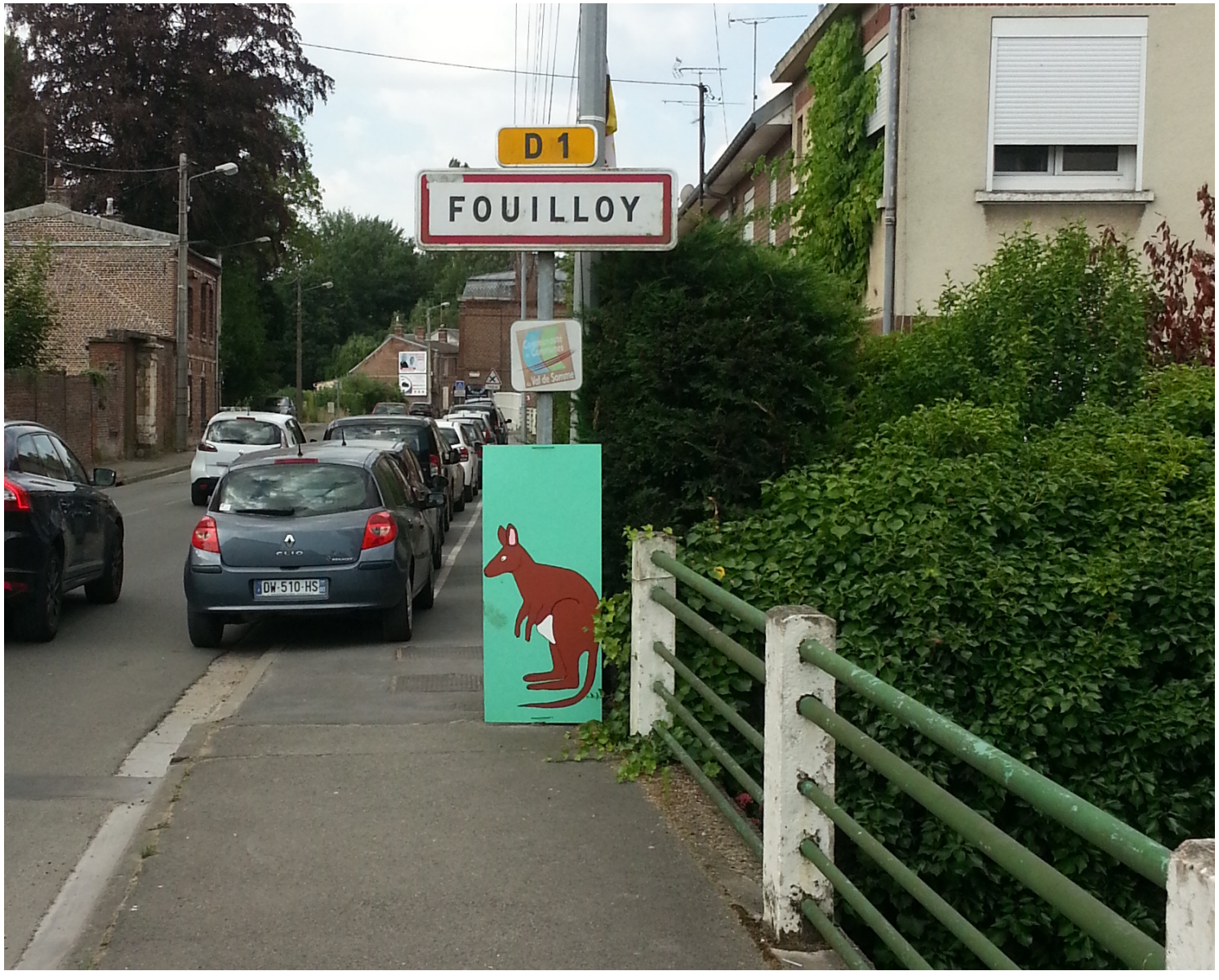
No way, JOSE!

Lessons for authors and implementers of open standards

Fraser Tweedale
@hackuador

July 3, 2018


D 1
FOUILLOY
Val de Somme
DW-510-HS
CLIO

A journey. . .

JOSE

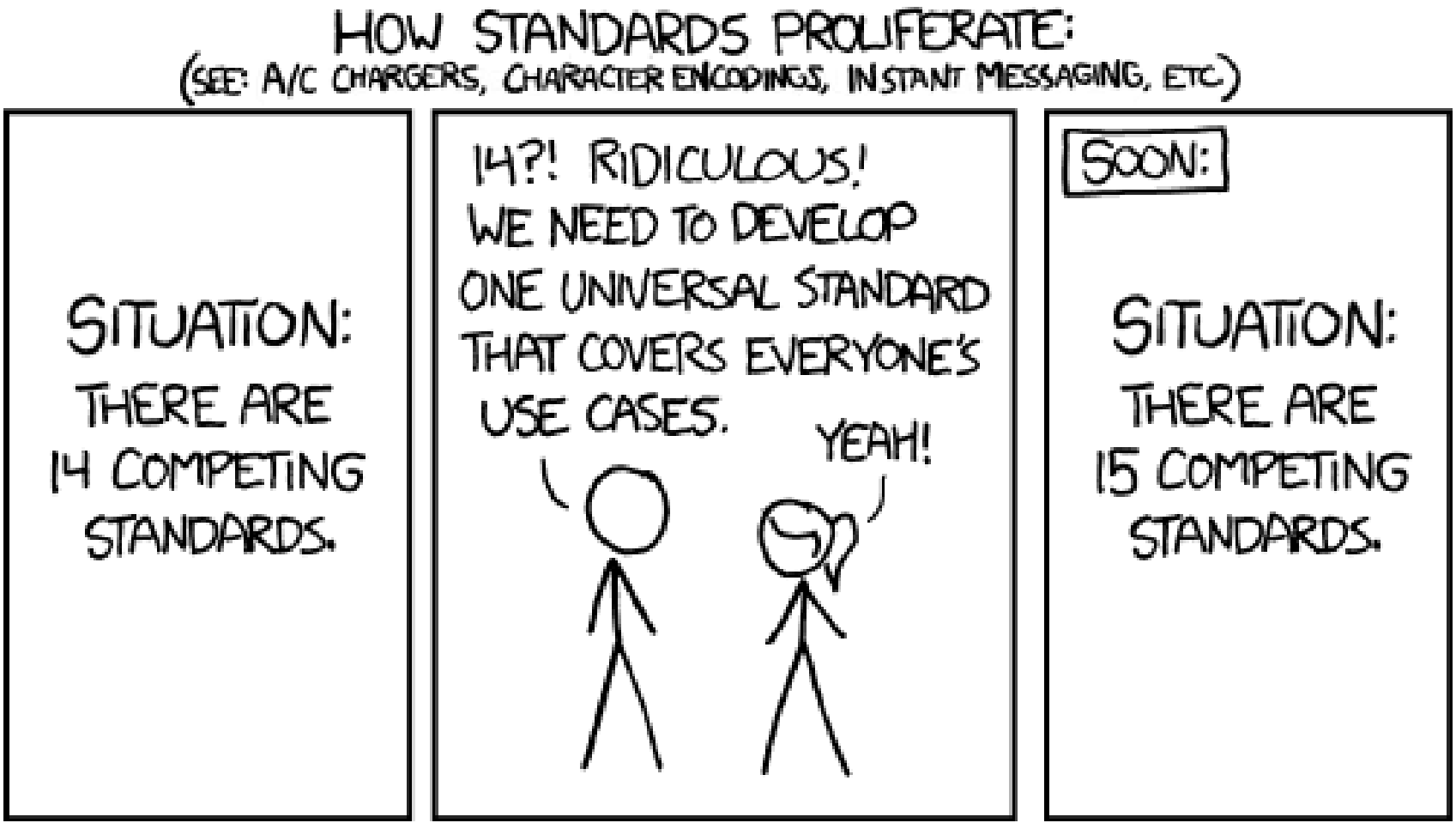
- **J**SON **O**bject **S**igning and **E**ncryption
- IETF WG formed 2011, RFCs 2015
- used in OpenID Connect, ACME

JOSE & me

- I wrote a JOSE library for Haskell
- I participated in IETF discussions
- JOSE has lots of problems (sorry...)

What is a standard?

Do you need a new standard?



JOSE—rationale

*With the **increased usage of JSON** in protocols in the IETF and elsewhere, there is now a desire to offer security services, which use encryption, digital signatures, message authentication codes (MACs) algorithms, that carry their data in JSON format.*[1]

[1] https://tools.ietf.org/wg/jose/charters

JOSE—rationale

*Many current applications thus have **much more robust support** for processing objects in these text-based formats than ASN.1 objects; indeed, many **lack the ability to process ASN.1** objects at all. To **simplify** the addition of object-based security features to these applications, the working group has been chartered to develop a **secure** object format based on JSON.*[2]

[2] https://tools.ietf.org/html/rfc7165

JOSE—assumptions

- ASN.1 libraries don't exist
- It's better to define new standard than write a library
- JSON is suitable for security/cryptographic objects
- ASN.1 is bad

JOSE—irony

4.7. "x5c" (X.509 Certificate Chain) Parameter

The "x5c" (X.509 certificate chain) parameter contains a chain of one or more PKIX certificates [RFC5280]. The certificate chain is represented as a JSON array of certificate value strings. Each string in the array is a base64-encoded (Section 4 of [RFC4648] -- not base64url-encoded) DER [ITU.X690.1994] PKIX certificate value.

Takeaway: write libraries, not standards

Is JSON the right choice?

Falsehoods programmers believe about JSON...

JSON support is universal.

C

Rust

C++

Scala

Haskell

...

JSON is human readable.

{"signature":"M3oVLXrbeFRT9Ef9d3WzR-D7dGtIeYoPBYmiCdtYqus","protected":"eyJhbGciOiJIUzI1NiIsImtpZCI6ImthcmF0ZSJ9","payload":"eyJzdWJqZWN0IjoiZnJhc2VAZnJhc2UuaWQuYXUiLCJpc3MiOiJocy1qb3NlIiwiYXVkIjpbImFsaWNlIiwiYm9iIl19Cg"}

JSON is unambiguously specified.

JSON—ambiguities

- invalid code points
- data size limits

JSON objects are maps.

JSON—objects

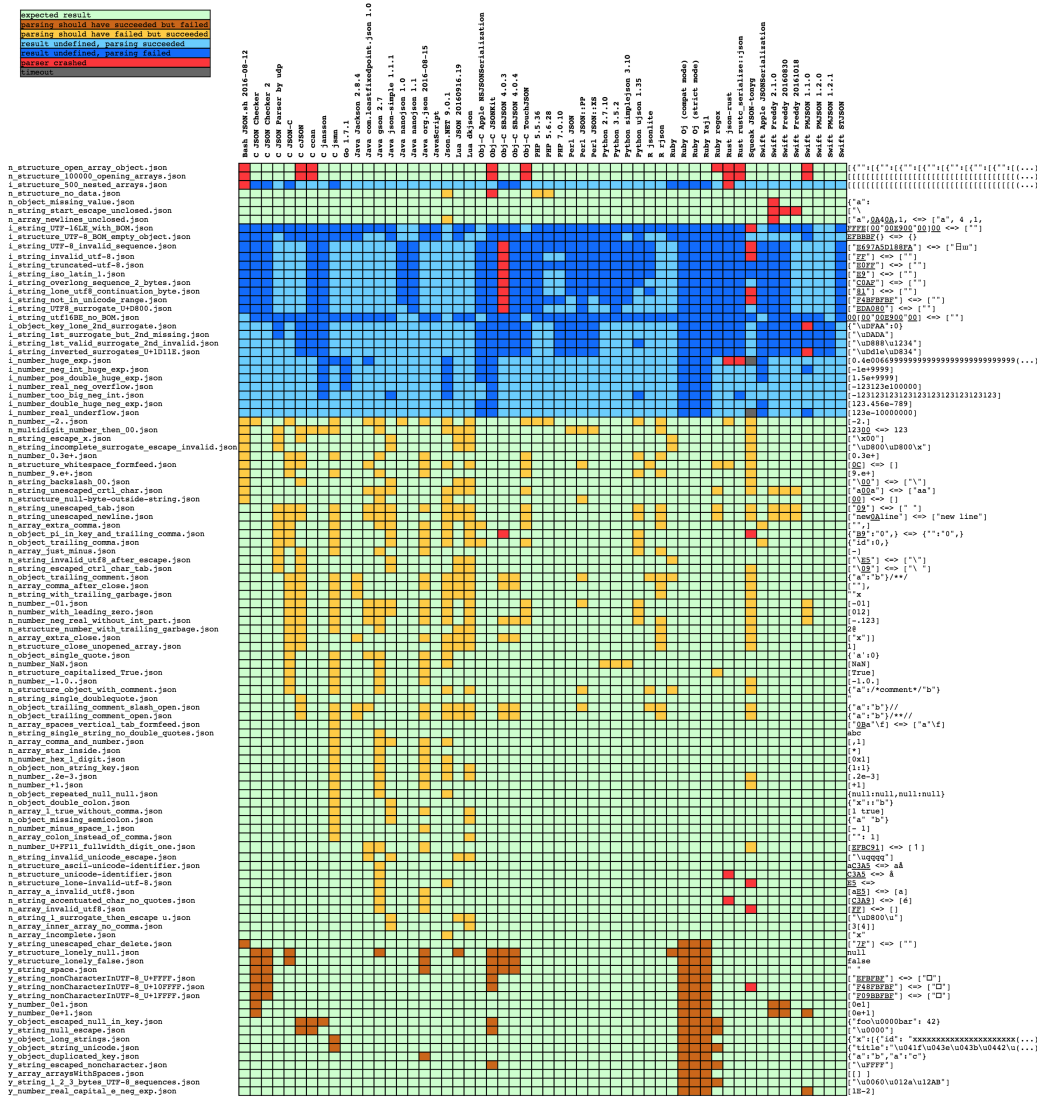
- `names within an object SHOULD be unique`—RFC 8259
- Is a JSON object a map?
- What *kind* of map?
- How should duplicate keys be treated?

JSON will be parsed the same way by different parsers.

JSON Parsing Tests, Pruned

Appendix to: seriot.ch [Parsing JSON is a Minefield](http://www.seriot.ch/parsing_json.php) http://www.seriot.ch/parsing_json.php

2016-11-05 00:04:08



n_structure_open_array_object.json
n_structure_100000_opening_arrays.json
i_structure_500_nested_arrays.json
n_structure_no_data.json
n_object_missing_value.json
n_string_start_escape_unclosed.json
n_array_newlines_unclosed.json
i_string_UTF-16LE_with_BOM.json
i_structure_UTF-8_BOM_empty_object.json
i_string_UTF-8_invalid_sequence.json
i_string_invalid_utf-8.json
i_string_truncated-utf-8.json
i_string_iso_latin_1.json
i_string_overlong_sequence_2_bytes.json
i_string_lone_utf8_continuation_byte.json
i_string_not_in_unicode_range.json
i_string_UTF8_surrogate_U+D800.json
i_object_key_lone_2nd_surrogate.json
i_string_1st_surrogate_but_2nd_missing.json
i_string_1st_valid_surrogate_2nd_invalid.json
i_string_inverted_surrogates_U+1D11E.json
i_number_huge_exp.json
i_number_neg_int_huge_exp.json
i_number_pos_double_huge_exp.json
i_number_real_neg_overflow.json
i_number_too_big_neg_int.json
i_number_double_huge_neg_exp.json
i_number_real_underflow.json
n_number_-2..json
n_multidigit_number_then_00.json
n_string_escape_x.json
n_string_incomplete_surrogate_escape_invalid.json
n_number_0.3e+.json
n_structure_whitespace_formfeed.json
n_number_9.e+.json
n_string_backslash_00.json
n_string_unescaped_crtl_char.json
n_structure_null-byte-outside-string.json
n_string_unescaped_tab.json
n_string_unescaped_newline.json
n_array_extra_comma.json
n_object_pi_in_key_and_trailing_comma.json
n_object_trailing_comma.json
n_array_just_minus.json
n_string_invalid_utf8_after_escape.json
n_string_escaped_ctrl_char_tab.json
n_object_trailing_comment.json
n_array_comma_after_close.json
n_string_with_trailing_garbage.json
n_number_-01.json
n_number_with_leading_zero.json
n_number_neg_real_without_int_part.json
n_structure_number_with_trailing_garbage.json
n_array_extra_close.json
n_structure_close_unopened_array.json
n_object_single_quote.json
n_number_NaN.json
n_structure_capitalized_True.json
n_number_-1.0..json
n_structure_object_with_comment.json
n_string_single_doublequote.json
n_object_trailing_comment_slash_open.json
n_object_trailing_comment_open.json
n_array_spaces_vertical_tab_formfeed.json
n_string_single_string_no_double_quotes.json
n_array_comma_and_number.json
n_array_star_inside.json
n_number_hex_1_digit.json
n_object_non_string_key.json
n_number_.2e-3.json
n_number_+1.json
n_object_repeated_null_null.json
n_object_double_colon.json
n_array_1_true_without_comma.json
n_object_missing_semicolon.json
n_number_minus_space_1.json
n_array_colon_instead_of_comma.json
n_number_U+FF11_fullwidth_digit_one.json
n_string_invalid_unicode_escape.json
n_structure_ascii-unicode-identifier.json
n_structure_unicode-identifier.json
n_structure_lone-invalid-utf-8.json
n_array_a_invalid_utf8.json
n_string_accentuated_char_no_quotes.json
n_array_invalid_utf8.json
n_string_1_surrogate_then_escape_u.json
n_array_inner_array_no_comma.json
n_array_incomplete.json
y_string_unescaped_char_delete.json
y_structure_lonely_null.json
y_structure_lonely_false.json
y_string_space.json
y_string_nonCharacterInUTF-8_U+FFFF.json
y_string_nonCharacterInUTF-8_U+10FFFF.json
y_string_nonCharacterInUTF-8_U+1FFFF.json
y_number_0e1.json
y_number_0e+1.json
y_object_escaped_null_in_key.json
y_string_null_escape.json
y_object_long_strings.json
y_object_string_unicode.json
y_object_duplicated_key.json
y_string_escaped_noncharacter.json
y_array_arraysWithSpaces.json
y_string_1_2_3_bytes_UTF-8_sequences.json
y_number_real_capital_e_neg_exp.json

CVE-2017-12635

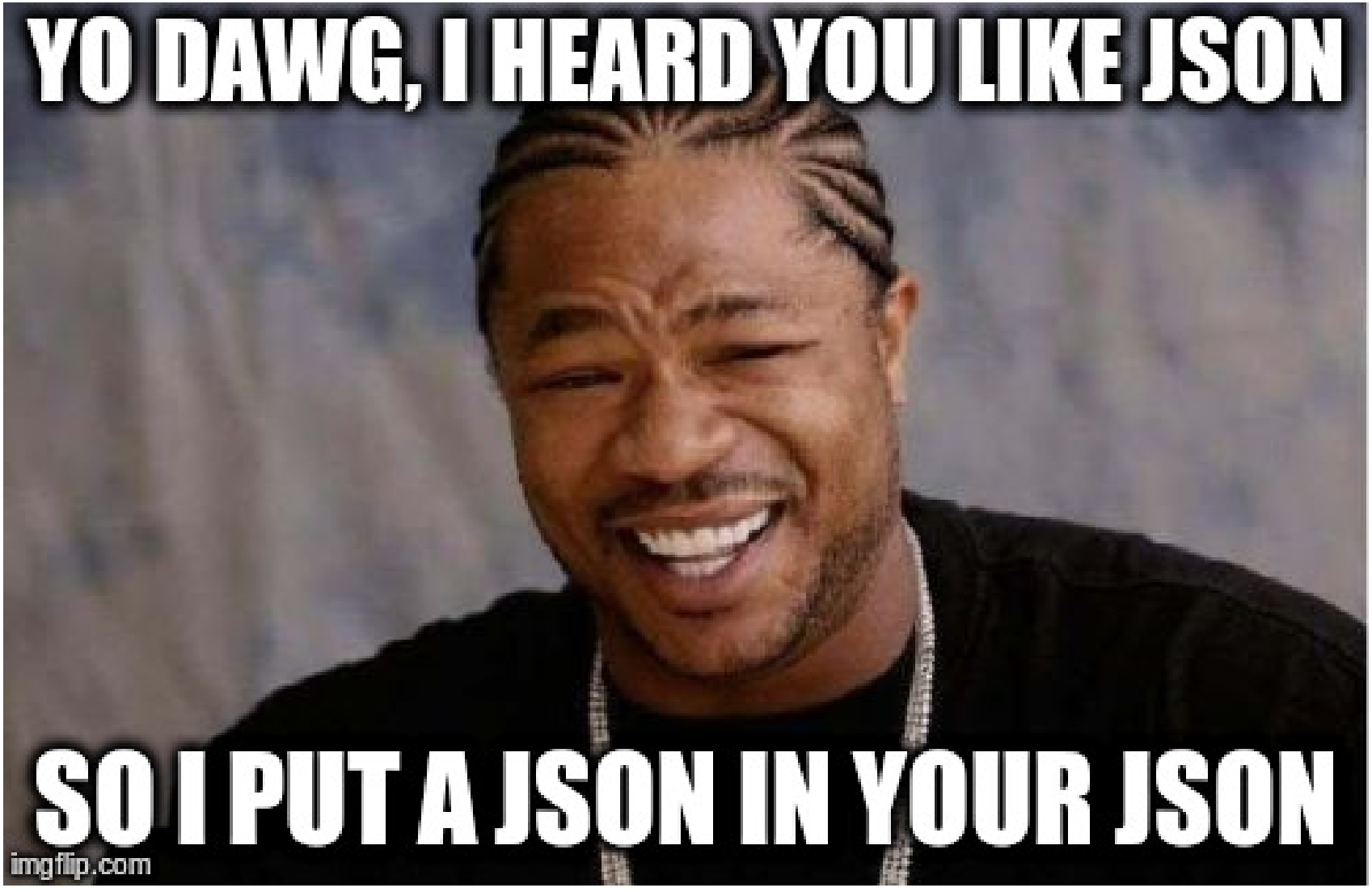
```
{
  "type": "user",
  "name": "alice",
  "roles": ["_admin"],
  "roles": []
}
```

JSON—other problems

- Numbers
- Binary data?
- No canonical serialisation

{"signature":"M3oVLXrbeFRT9Ef9d3WzR-D7dGtIeYoPBYmiCdtYqus","protected":"eyJhbGciOiJIUzI1NiIsImtpZCI6ImthcmF0ZSJ9","payload":"eyJzdWJqZWN0IjoiZnJhc2VAZnJhc2UuaWQuYXUiLCJpc3MiOiJocy1qb3NlIiwiYXVkIjpbImFsaWNlIiwiYm9iIl19Cg"}

```
{"subject":"frase@frase.id.au",
  "iss":"hs-jose",
  "aud":["alice","bob"]}
```


YO DAWG, I HEARD YOU LIKE JSON
SO I PUT A JSON IN YOUR JSON
imgflip.com

JSON—alternatives

- ASN.1
- CBOR

Takeaway: don't automatically reach for JSON

Cryptography in JOSE

JOSE cryptography—issues

- PKCS #1 v1.5 padding
- Weierstrass curves
- `"none"` signature algorithm
- AES Key Wrap

Algorithmic agility

- more complex protocol
- more ways to mess up
- end up using insecure crypto anyway

JOSE cryptography—common vulnerabilities

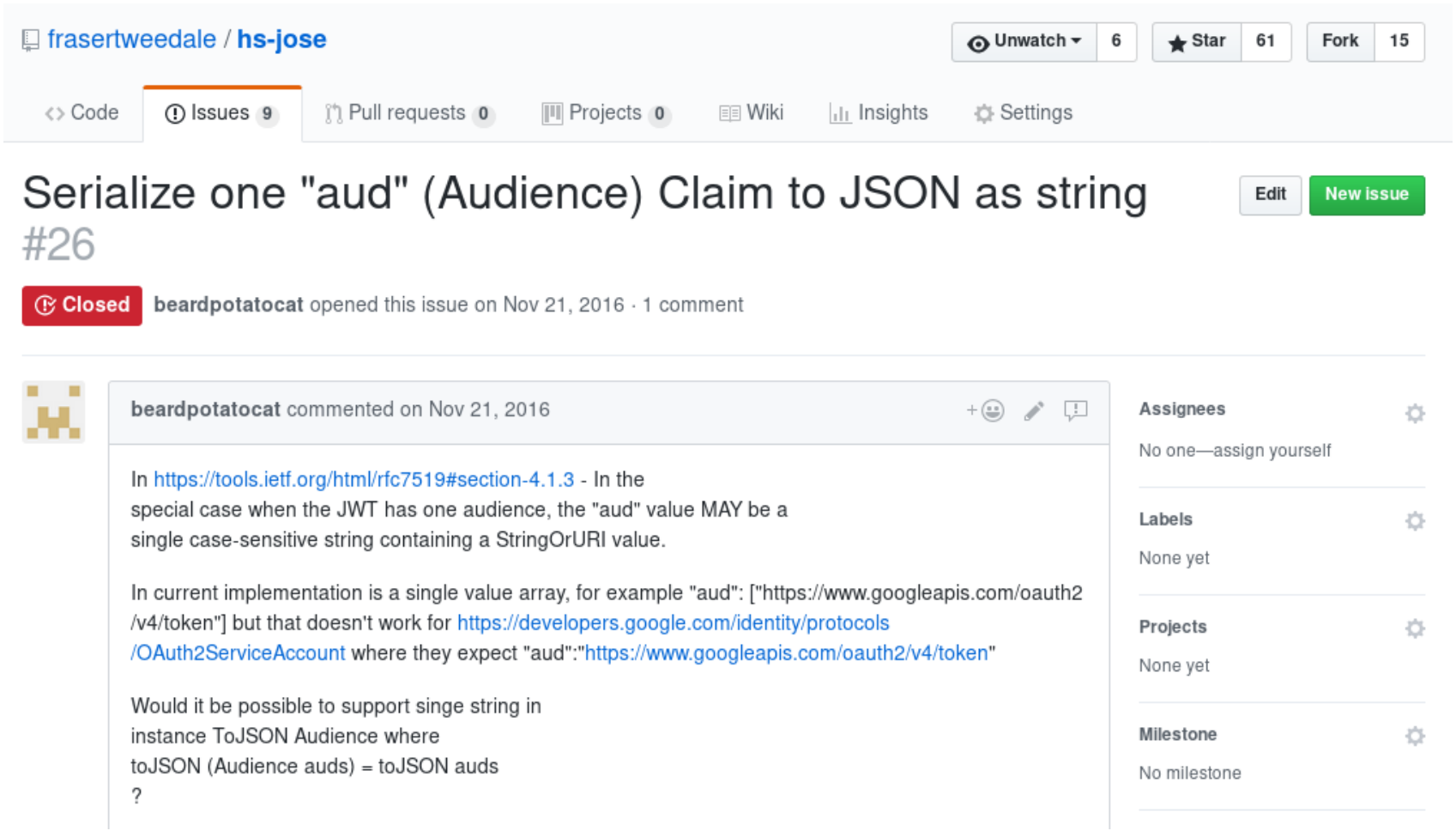
- `"none"` downgrade attack
- invalid curve attack
- algorithm substitution attack

Takeaway: don't cut corners with crypto

Ambiguities and Interoperability

```
{
 "payload":"<payload contents>",
 "signatures":[
  {"protected":"<integrity-protected header 1 contents>",
   "header":<non-integrity-protected header 1 contents>,
   "signature":"<signature 1 contents>"},
  ...
  {"protected":"<integrity-protected header N contents>",
   "header":<non-integrity-protected header N contents>,
   "signature":"<signature N contents>"}]
}
```

```
{
 "payload":"<payload contents>",
 "protected":"<integrity-protected header contents>",
 "header":<non-integrity-protected header contents>,
 "signature":"<signature contents>"
}
```



http://github.com/frasertweedale/hs-jose/issues/26

JOSE flattened serialisation—drawbacks

- more work for library authors
- incompatible libraries and programs
- more work for *downstream* standard authors

JOSE flattened serialisation—benefits

- saved a few bytes

Takeaway: use case “optimisations” belong in libraries, not standards.

hs-jose—dealing with ambiguity

```
data List a = Nil
            | a : (List a)

data Identity a = Identity a
```

hs-jose—dealing with ambiguity

```
data JWS t = JWS ByteString (t Signature)

type GeneralJWS = JWS List Protection

type FlattenedJWS = JWS Identity Protection
```

Dealing with ambiguity—abstraction

- abstract over ambiguities
- let the user decide what they want
- provide simple API for the common use cases

Takeaway: use abstraction to deal with ambiguities in standards

Writing safe APIs

```
verifyJWSWithPayload
  :: ( MonadError Error m
     , VerificationKeyStore m payload k
     , Foldable t
     )
  => ValidationSettings
  -> (ByteString -> m payload) -- ^ decoder
  -> k                         -- ^ key store
  -> JWS t
  -> m payload
```

Static type systems—benefits

- abstraction
- avoid type confusion attacks
- readability & maintainability
- enable advanced techniques for security[3,4,5]

[3] Two Can Keep a Secret, If One of Them Uses Haskell

[4] FaCT: A Flexible, Constant-Time Programming Language

[5] HOWTO: Static access control using phantom types

Takeaway: static type systems enable safe, ergonomic APIs

So you're going to write a new standard...

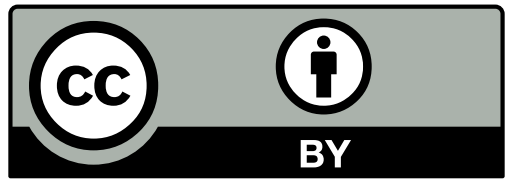
Advice for standards authors

- avoid ambiguity & special cases
- exclude esoteric use cases
- get cryptographers to review
- **write multiple implementations**

Recap

- write libraries, not standards
- don't automatically reach for JSON
- don't cut corners with crypto
- special cases belong in libraries
- abstract over ambiguities
- use statically typed languages
- write multiple implementations

Questions?



https://speakerdeck.com/frasertweedale

@hackuador

hackage.haskell.org/package/jose