Now come the decorations...

• 80% of what takes time in a spec, isn’t the spec.
  • Service Description
    • Version
    • Rate limiting?
    • Etc.
  • Service Management

• Going to JSON has big advantages
  • It’s a data serialization format, not a document description language
But we are among the first to come this way

- Lots of JSON specifications
- Very few designed to be mission critical

- We can’t follow an existing pattern
  - We should try to set one.
Encoding details matter

POST /acme/new-authorization HTTP/1.1
Host: example.com

{ "resource": "new-authz",
  "identifier": {
    "type": "dns",
    "value": "example.org" }
} /* Signed as JWS */

What exactly is signed?
What if...

• We decide to move away from HTTP?
• We decide to support a new encoding?
• The messages go through a proxy that rewrites URL?
A better approach...

POST <nobody cares now> HTTP/1.1
Host: <Irrelevant>

/* Start of signed data */
{
  "new-authorization": {
    "resource": "new-authz",
    "identifier": {
      "type": "dns",
      "value": "example.org"
    }
  }

/* End of JWS signed data */
But CA substitution!!!!

POST <nobody cares now> HTTP/1.1
Host: <Irrelevant>

/* Start of signed data */
{"new-authorization":
 { "CA": "example.com",
   "resource": "new-authz",
   "identifier": {
     "type": "dns",
     "value": "example.org" } }
/* End of JWS signed data */
We just corrected a bug

• In current spec, “example.com” is overloaded
  • HTTP end point
  • Identify CA to issue certificate

• In proposal, separate semantics have separate fields
Advantages

• Completely decouple from HTTP
  • HTTP in Web Services is a Presentation Layer
    • Layer separation is good design
    • A Web Service that reacts to HTTP fields is like an application protocol using TCP checksum.

• Simpler JWS approach
  • Just one signed blob, no additional protected headers
  • Can slot in CMS without difficulty

• Directory is no longer security sensitive
Nested vs Flat

"challenges": [
    { "type": "http-01",
      "uri": "https://example.com/authz/asdf/0",
      "token": "IlirfxKKXAsHtmzK29Pj8A"
    },
    { "type": "new-01",
      "uri": "https://example.com/authz/asdf/1",
      "param-x": "TBS"
    }
]
But this is equally valid

"challenges": [
  {
    "uri": "https://example.com/authz/asdf/0",
    "token": "IlirfxKKXAsHtmzK29Pj8A",
    "type": "http-01"
  },
  {
    "uri": "https://example.com/authz/asdf/1",
    "type": "new-01",
    "param-x": "TBS"
  }
]
Flat encoding assumes an implementation

• Parse JSON tree
  • Bind to tree elements in scripting language
  • We all write Web services in Perl, right?

• But Bobby Tables says the approach should be:
  • Parse input data
  • Validate against schema specification
  • Reject if invalid
  • Otherwise do stuff
Nested – actually shorter

"challenges": [
  { "http-01": {
    "uri": "https://example.com/authz/asdf/0",
    "token": "IlirfxKKXAsHtmzK29Pj8A"
  }},
  { "new-01": {
    "uri": "https://example.com/authz/asdf/1",
    "param-x": "TBS"
  }},
  ...]
Proposal

• Start every message with the ACME message type
• Eliminate all the ‘type” elements
  • Replace with nested encoding

• Advantages
  • Proper layer separation
  • Clearer examples (can elide HTTP entirely)
  • Allow for implementations in C, C#, Java
  • Allow others to use our pattern