JWS Direct Signing

ISSUE-23

Goal

- Maximize the extent to which the inputs and outputs of JOSE cryptographic operations can be **controlled by applications**, as opposed to involving processing specific to JOSE. This allows JOSE the flexibility to address the needs of **many cryptographic protocols**.

Scenario

- JWS with no protected header (thus JSON)
- What should JWS Signing Input be?

```
{
  "unprotected": {
    "alg": "HS256",
    "kid": "1"
  },
  "payload": "U2lnbiBtZSE"
  "signature": "???"
}
```

JWS -13

OLD:
Compute the JWS Signature in the manner defined for the particular algorithm being used over the JWS Signing Input (**the concatenation of the Encoded JWS Header, a period ('.') character, and the Encoded JWS Payload**).

JWS-13

```
JWS_Signing_Input
  == '.' + base64(payload)
  == ".U2lnbiBtZSE"
```

Proposed

NEW:
Compute the JWS Signing Input. If the JWS Protected Header is present then the JWS Signing input is **the concatenation of the Encoded JWS Header, a period ('.') character, and the Encoded JWS Payload.** If there is no JWS Protected Header, then the JWS Signing Input is the **JWS Payload (unencoded).** Compute the JWS Signature in the manner defined for the particular algorithm being used.

Proposed

```
JWS_Signing_Input
== payload
== "Sign me!"
```

Side by Side

OLD:
Compute the JWS Signature in the manner defined for the particular algorithm being used over the JWS Signing Input (**the concatenation of the Encoded JWS Header, a period ('.') character, and the Encoded JWS Payload**).

NEW:
Compute the JWS Signing Input. If the JWS Protected Header is present then the JWS Signing input is **the concatenation of the Encoded JWS Header, a period ('.') character, and the Encoded JWS Payload**. If there is no JWS Protected Header, then the JWS Signing Input is **the JWS Payload (unencoded)**. Compute the JWS Signature in the manner defined for the particular algorithm being used.

Complexity

- Only for JSON implementations
- Only if you accept both protected and unprotected headers
- **No change for compact-only implementations**

Security

- Concern: Shifting data between protected header and content
- For example, the following are equivalent:
  - protected = “qwer”, payload = “asdf”
  - protected = “” , payload = base64(“qwer.asdf”)
- Current draft prevents by only using concatenation of encoded forms

Security

- Mostly no problem for the proposed scheme
- Within each case (protected/not), no shifting can occur
- So only need to care about switching cases
  - Content slicing: payload -> header+payload
  - Content fusion: header+payload -> payload
- These are fundamental to the requirements (they also exist in CMS SignedData)

Summary

- .base64(payload) vs payload
- Benefit: Support for many more use cases
- Cost:
  - Complexity for some JSON implementations (**no cost for compact-only**)
  - Slicing/fusion risks (as in CMS)