EXAMPLE 5: A more complex verifiable claim

This example is from the Verifiable Claims Data Model and Representations at https://www.w3.org/TR/verifiable-claims-data-model/

When looking at this, the question comes to mind: What exactly is signed? A digital signature is supposed to indicate that the content hasn't changed since signed. To recreate the hash, you need to know exactly which portion of this credential is signed:


```
{
 "@context": [
  "https://w3id.org/identity/v1",
  "https://w3id.org/security/v1"
 ],
 "id": "http://example.gov/credentials/3732",
 "type": ["Credential", "PassportCredential"],
 "name": "Passport",
 "issuer": "https://example.gov",
 "issued": "2010-01-01",
 "claim": {
  "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
  "name": "Alice Bobman",
  "birthDate": "1985-12-14",
  "gender": "female",
  "nationality": {
   "name": "United States"
  },
  "address": {
   "type": "PostalAddress",
   "addressStreet": "372 Sumter Lane",
   "addressLocality": "Blackrock",
   "addressRegion": "Nevada",
   "postalCode": "23784",
   "addressCountry": "US"
  },
  "passport": {
   "type": "Passport",
   "name": "United States Passport",
   "documentId": "123-45-6789",
   "issuer": "https://example.gov",
   "issued": "2010-01-07T01:02:03Z",
   "expires": "2020-01-07T01:02:03Z"
  }
 },
 "signature": {
  "type": "LinkedDataSignature2015",
  "created": "2016-06-21T03:40:19Z",
  "creator": "https://example.com/jdoe/keys/1",
  "domain": "json-ld.org",
  "nonce": "783b4dfa",
  "signatureValue":
    "Rxj7Kb/tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0D
    is5mUePIzII4+p/
    ewcOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
    2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ibsN
    k="
 }
}
```


EXAMPLE 2: A simple signed Linked Data document

This example is from the Draft Community Group Report 27 February 2018 at https://w3c-dvcg.github.io/ld-signatures/

Just looking at this, the same vagueness is present. However, this document is specific about how the signature hash is initially created then re-created in sections 7.1 and 7.2, a variation of ill-fated .XML signatures:


```
{
 "@context": "https://w3id.org/identity/v1",
 "title": "Hello World!",
 "proof": {
  "type": "RsaSignature2018",
  "creator": "https://example.com/i/pat/keys/5",
  "created": "2017-09-23T20:21:34Z",
  "domain": "example.org",
  "nonce": "2bbgh3dgjg2302d-d2b3gi423d42",
  "proofValue": "eyJ0eXAiOiJK...gFWFOEjXk"
 }
}
```


From section 7.1:

1. Create a copy of document, hereafter referred to as output.
2. Generate a canonicalized document by canonicalizing document according to a canonicalization algorithm (e.g. the GCA2015 [RDF-DATASET-NORMALIZATION] algorithm).
3. Create a value tbs that represents the data to be signed, and set it to the result of running the Create Verify Hash Algorithm, passing the information in options.
1. Digitally sign tbs using the privateKey and the the digital signature algorithm (e.g. JSON Web Signature using RSASSA-PKCS1-v1_5 algorithm). The resulting string is the signatureValue.
2. **Add a `signature` node to output** containing a linked data signature using the appropriate type and signatureValue values as well as all of the data in the signature options (e.g. creator, created, and if given, any additional signature options such as nonce and domain).
3. Return output as the signed linked data document.

From section 7.2:

1. Get the public key by dereferencing its URL identifier in the `signature` node of the default graph of signed document. Confirm that the linked data document that describes the public key specifies its owner and that its owner's URL identifier can be dereferenced to reveal a bi-directional link back to the key. Ensure that the key's owner is a trusted entity before proceeding to the next step.
2. Let document be a copy of signed document.
3. **Remove any `signature` nodes from the default graph in document** and save it as signature.
4. Generate a canonicalized document by canonicalizing document according to the canonicalization algorithm (e.g. the GCA2015 [RDF-DATASET-NORMALIZATION] algorithm).
5. Create a value tbv that represents the data to be verified, and set it to the result of running the Create Verify Hash Algorithm, passing the information in signature.
6. Pass the signatureValue, tbv, and the public key to the signature algorithm (e.g. JSON Web Signature using RSASSA-PKCS1-v1_5 algorithm). Return the resulting boolean value.

The problem with the instructions in sections 7.1 and 7.2 is that ***the signed content is modified by the signer. Then, the receiver needs to undo the modifications and canonicalize the content to verify the signature.*** In theory, this is doable as long as the instructions are very clear. However, this is ill-advised. It creates a situation where the signature may be impossible to verify, especially with signature sets..

Note, that in the Linked Data Signatures 1.0, issue 6 states:

> **ISSUE 6**
>
> **The signature parameters should be included as headers and values in the data to be signed.**

This requirement to include signature parameters within the signed content is the core of the problems in the previous examples: ***It requires that the signed content be modified after it is signed*** in order to re-create the signature hash.

Let's take a look at some more successful signature formats for guidance:

SUCCESSFULL SIGNATURE FORMATS:

OpenPGP/GPG signatures:

An example OpenPGP/GPG clearsigned text from https://www.gnupg.org/gph/en/manual/x135.html:

```
alice% gpg --clearsign doc

You need a passphrase to unlock the secret key for
user: "Alice (Judge) <alice@cyb.org>"
1024-bit DSA key, ID BB7576AC, created 1999-06-04

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

[...]
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v0.9.7 (GNU/Linux)
Comment: For info see http://www.gnupg.org

iEYEARECAAYFAjdYCQoACgkQJ9S6ULt1dqz6IwCfQ7wP6i/
    i8HhbcOSKF4ELyQB1
oCoAoOuqpRqEzr4kOkQqHRLE/b8/Rw2k
=y6kj
-----END PGP SIGNATURE-----
```

Note that the signature parameters are within the signature block and not within the signed message block. Though, the Hash algorithm is in the signed message block. It allows for a comment. This could be useful for stating the purpose of the signature, though GPG standards say that the comment is only to be used for the signer's name.

This type of signature has been used successfully for quite a while and can be the basis for digitally signed HTML, as illustrated here:

http://tjl73.altervista.org/HTML_sign_tutorial/en/en.html

JWS (signed JWT):

JWS has a lot of potential. It doesn't work for our needs as-is because:

- It isn't human readable
- It needs to be unpacked (from its BASE64URL encoding) before it's usable as linked data

But, it's based on a JSON structure and ***it doesn't require that signed content be modified after it has been signed except in the case of signature sets***. There's a good description of JSON serialization in 7.2.1 of the standards track RFC 7515 http://self-issued.info/docs/draft-ietf-jose-json-web-signature.html#rfc.section.7.2.1

A JWS without the BASE64URL encoding could work nicely for our needs. One interesting feature of JWS is the optional Protected Header, which is signed along with the Payload. This gives us the option of addressing "Issue 6" mentioned above. It also gives us the option of ignoring "Issue 6". And, it gives a well-defined protocol for allowing a different Protected Header for each signature on the Payload. Here's that general format without the BASE64URL encoding. Note that there will EITHER be a 'protected' OR a "header" for each signature, not both:

Flattened for a single signature:

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

For a signature set:

```
{
 "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>"}]
}
```

The "payload" + the optional "protected" is digitally signed, presumably in a format such as this:

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

PASETO:

As I prepare to pursue un-encoded JWS, I find these articles:

https://paragonie.com/blog/2017/03/jwt-json-web-tokens-is-bad-standard-that-everyone-should-avoid

https://paragonie.com/blog/2018/03/paseto-platform-agnostic-security-tokens-is-secure-alternative-jose-standards-jwt-etc

https://blogs.adobe.com/security/2017/03/critical-vulnerability-uncovered-in-json-encryption.html

To be clear, JOSE signatures can be fixed and some steps have been taken to remedy vulnerabilities. But, some issues remain.

Let's take a look at the PASETO structure:

```
Version: v2
Purpose: public [public-key digital signature]
Payload:
{
  "data": "this is a signed message",
  "exp": "2039-01-01T00:00:00+00:00"
}
Signature (hex-encoded):
d600bbfa3096b0dde6bf8b89699c59a746ed2c981cc95c0bfacbc90f
    b7f8207c
86b5e29edc74cb8c761318723532d0aa27e1120cb36813ba2d908cda
    985b2408
Public key (hex-encoded):
11324397f535562178d53ff538e49d5a162242970556b4edd950c87c
    7d86648a
```

There are some significant pros and cons to this structure relative to the JWS JSON serialization syntax:.

- The encryption options and restrictions are welcomed, addressing some of the vulnerabilities of JWS encryption options.
- The signature doesn't have a Protected Header option. Is this an indication that a Protected Header simply isn't needed? Or, is the header signed? I don't see anything clarifying whether just the payload is signed or if the payload+header are signed. There's ambiguity again.
- This doesn't have a provision for signature sets or signature chains. It only has an option of one signature
- It includes the public key (should have a provision for the certificate, certificate chain, DID identity anchor, SoLiD identity anchor or similar)
- It isn't a JSON structure

Maybe the answer is to use the PASETO encryption options with the JWS structure…. Therefore:

Proposed format for a single signature:

A signed JSON-LD flattened for one signature will take the following form, where there is EITHER a “protected” OR a “header” and there is EITHER a “public key” OR a “certificate” OR a “certificate chain” OR a public key packaged in a SoLiD or similar identity anchor. Note that “version” and “purpose” in headers refer to PASETO encryption options. “payload” + the optional “protected” is signed:

```
{
  "payload":
  {
    "iss": "joe",
    "exp": 1300819380,
    "http://example.com/is_root": true
  },
  "protected":
  {
    "version": "v2",
    "purpose": "public"
  },
  "header":
  {
    "version": "v2",
    "purpose": "public"
  },
  "signature":
    "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ew
    cOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
    2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ib
    sNk=",
  "public key":
    "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ew
    cOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
    2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ib
    sNk=",
  "certificate":
    "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ew
    cOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
    2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ib
    sNk=",
  "certificate chain":
  {
    "signer":
      "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5
      ewcOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0
      qoNq2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzP
      oCb9ibsNk=",
    "sub-ca":
      "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5
      ewcOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0
      qoNq2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzP
      oCb9ibsNk=",
    "ca":
      "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5
      ewcOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0
      qoNq2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzP
      oCb9ibsNk="

  }
}
```

For brevity, all following examples will use a “protected” and a “certificate”:

Proposed format for signature sets:

Signature Sets would take the following form. ***This requires some modification of signed content after it has been signed***, but the process is much clearer. It would be much easier to recreate the signed content (“payload” + “protected”):

```
{
  "payload":
  {
    "iss": "joe",
    "exp": 1300819380,
    "http://example.com/is_root": true
  },
  "signatures":
  [
    {
      "protected":
      {
        "version": "v2",
        "purpose": "public"
      },
      "signature":
        "Rxj7Kb/
        tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Di
        s5mUePIzII4+p/
        ewcOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNz
        ur0qoNq2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9o
        yqDMzPoCb9ibsNk=",
      "certificate":
        "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0D
        is5ewcOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5
        JNzur0qoNq2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5uf
        o9oyqDMzPoCb9ibsNk="
    },
    ...
    {
      "protected":
      {
        "version": "v2",
        "purpose": "public"
      },
      "signature":
        "Rxj7Kb/
        tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Di
        s5mUePIzII4+p/
        ewcOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNz
        ur0qoNq2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9o
        yqDMzPoCb9ibsNk=",
      "certificate":
        "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0D
        is5ewcOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5
        JNzur0qoNq2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5uf
        o9oyqDMzPoCb9ibsNk="
    }
  ]
}
```

Proposed format for signature chains:

Signature chains will take the following form, with each subsequent signature signing the preceding signature(s) (colors added to clearly show the nesting):

```
{
  "payload":
  {
    "payload":
    {
      "payload":
      {
        "iss": "joe",
        "exp": 1300819380,
        "http://example.com/is_root": true
      },
      "protected":
      {
        "version": "v2",
        "purpose": "public"
      },
      "signature":
        "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ew
        cOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
        2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ib
        sNk=",
      "certificate":
        "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ew
        cOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
        2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ib
        sNk="
    },
    "protected":
    {
      "version": "v2",
      "purpose": "public"
    },
    "signature":
      "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ew
      cOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
      2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ib
      sNk=",
    "certificate":
      "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ew
      cOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
      2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ib
      sNk="
  },
  "protected":
  {
    "version": "v2",
    "purpose": "public"
  },
  "signature":
    "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ew
    cOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
    2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ib
    sNk=",
  "certificate":
    "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ew
    cOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
    2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ib
    sNk="
}
```

EXAMPLE 5: A more complex verifiable claim - existing

Let's revisit the existing more complex verifiable claim from: https://www.w3.org/TR/verifiable-claims-data-model/

```
{
 "@context": [
  "https://w3id.org/identity/v1",
  "https://w3id.org/security/v1"
 ],
 "id": "http://example.gov/credentials/3732",
 "type": ["Credential", "PassportCredential"],
 "name": "Passport",
 "issuer": "https://example.gov",
 "issued": "2010-01-01",
 "claim": {
  "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
  "name": "Alice Bobman",
  "birthDate": "1985-12-14",
  "gender": "female",
  "nationality": {
   "name": "United States"
  },
  "address": {
   "type": "PostalAddress",
   "addressStreet": "372 Sumter Lane",
   "addressLocality": "Blackrock",
   "addressRegion": "Nevada",
   "postalCode": "23784",
   "addressCountry": "US"
  },
  "passport": {
   "type": "Passport",
   "name": "United States Passport",
   "documentId": "123-45-6789",
   "issuer": "https://example.gov",
   "issued": "2010-01-07T01:02:03Z",
   "expires": "2020-01-07T01:02:03Z"
  }
 },
 "signature": {
  "type": "LinkedDataSignature2015",
  "created": "2016-06-21T03:40:19Z",
  "creator": "https://example.com/jdoe/keys/1",
  "domain": "json-ld.org",
  "nonce": "783b4dfa",
  "signatureValue":
    "Rxj7Kb/tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0D
    is5mUePIzII4+p/
    ewcOTjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq
    2yXl+TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ibsN
    k="
 }
}
```

EXAMPLE 5: A more complex verifiable claim - proposed

Compare that side by side with the proposed revision to the more complex verifiable claim. Note that because a "creator" anchor is specified where we can find the public key, there's no need for a "certificate" or "public key". The "payload"+"protected" is digitally signed in this case:

```
{
  "payload":
  {
    "@context": [
      "https://w3id.org/identity/v1",
      "https://w3id.org/security/v1"
    ],
    "id": "http://example.gov/credentials/3732",
    "type": ["Credential", "PassportCredential"],
    "name": "Passport",
    "issuer": "https://example.gov",
    "issued": "2010-01-01",
    "claim": {
      "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
      "name": "Alice Bobman",
      "birthDate": "1985-12-14",
      "gender": "female",
      "nationality": {
        "name": "United States"
      },
      "address": {
        "type": "PostalAddress",
        "addressStreet": "372 Sumter Lane",
        "addressLocality": "Blackrock",
        "addressRegion": "Nevada",
        "postalCode": "23784",
        "addressCountry": "US"
      },
      "passport": {
        "type": "Passport",
        "name": "United States Passport",
        "documentId": "123-45-6789",
        "issuer": "https://example.gov",
        "issued": "2010-01-07T01:02:03Z",
        "expires": "2020-01-07T01:02:03Z"
      }
    }
  },
  "protected":
  {
    "version": "v2",
    "purpose": "public",
    "created": "2016-06-21T03:40:19Z",
    "creator": "https://example.com/jdoe/keys/1",
    "domain": "json-ld.org",
    "nonce": "783b4dfa"
  },
  "signature":
    "tDbGHFAs6ddHjVLsHDiNyYzxs2MPmNG8G47oS06N8i0Dis5ewcO
    TjvH7aJxnKEePCO9IrlqaHnO1TfmTut2rvXxE5JNzur0qoNq2yXl
    +TqUWmDXoHZF+jQ7gCsmYqTWhhsG5ufo9oyqDMzPoCb9ibsNk="
}
```

After finding significant issues with the Signed JSON-LD and VCWG formats, a revision was originally proposed by Kevin Poulsen on August 3rd, 2018. This 7th version, authored by Kevin Poulsen, is proposed on December 30, 2018.