HTTP Message Signatures

HTTP Interim

Feb 1, 2022

Justin Richer and Annabelle Backman

What is it?

- Detached signature mechanism for generic HTTP messages
    - Request and response
    - Works* across HTTP versions
- Robust against expected changes
    - Proxy injection of header fields
    - Partial signature of stable aspects of message
- Allows multiple signatures
    - Including adding signatures over time
- Uses HTTP-native technologies
    - Structured Fields for encoding

```
POST /foo?param=value&pet=dog HTTP/1.1
Host: example.com
Date: Tue, 20 Apr 2021 02:07:55 GMT
Content-Type: application/json
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
Example-Dict: a=(1 2), b=3, c=4;aa=bb, d=(5 6);valid
Content-Length: 18

{"hello": "world"}
```

HTTP Message

```
"content-type": application/json
"digest": sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
"content-length": 18
"@target-uri": https://example.com/foo?param=value&pet=dog
"@signature-params": ("content-type" "digest" "content-length" "@target-uri");created=1643229457
```

Signature Input

```
Signature-Input: sig=("content-type" "digest" "content-length" "@target-uri");created=1643229457
Signature: sig=:DAE7QmnkzyM60Pddoz6M054wm3dvJKNsrqgJWgWz2t+/uR6zmczbCRFv8BYC+OGFmKk1pkrpOpLmj9z0kuLUbw==:
```

Signature Output

HTTP Signature Process

- Inputs:
  - HTTP Message
  - Key material
  - Required components
- Functions:
  - Cryptographic primitives: HTTP_SIGN (M, Ks) -> S
  - Key derivation (where needed)
  - Message hashing (where needed)
  - Binary encoding (where needed)
- Outputs:
  - Message signature
  - Signature parameters

HTTP Signature Verification Process

- Inputs:
  - HTTP Message
  - Key material
  - Signature parameters (includes covered components)
  - Message signature
- Functions:
  - Cryptographic primitives: HTTP_VERIFY (M, Kv, S) -> V
  - Key derivation (where needed)
  - Message hashing (where needed)
  - Binary encoding (where needed)
- Outputs:
  - Boolean verification status

Draft Status: -08

- Security and privacy considerations added to document
- Significant editorial clarifications
- Updated examples
- Updated cryptographic primitives
- Added ABNFs
- Renamed “Specialty” components to “Derived” components

Crypto Updates

- Ed25519
  - Uses PureEdDSA (no message hashing)
- ECDSA signature encoding
  - Uses raw signature encoding (no ASN.1)
- Non-deterministic algorithms (RSA PSS, ECDSA) are pointed out

Implementation Status

- Java implementation
  - XYZ GNAP implementation on Spring, Apache HTTP Components
- Python implementation (behind httpsig.org and in-doc examples)
- Scala library
- JavaScript (in-browser)
- Rust library (update of Cavage-draft implementation)
- Go library (from scratch)
- Should we add these to httpsig.org as they become usable?

Relationship to Digest

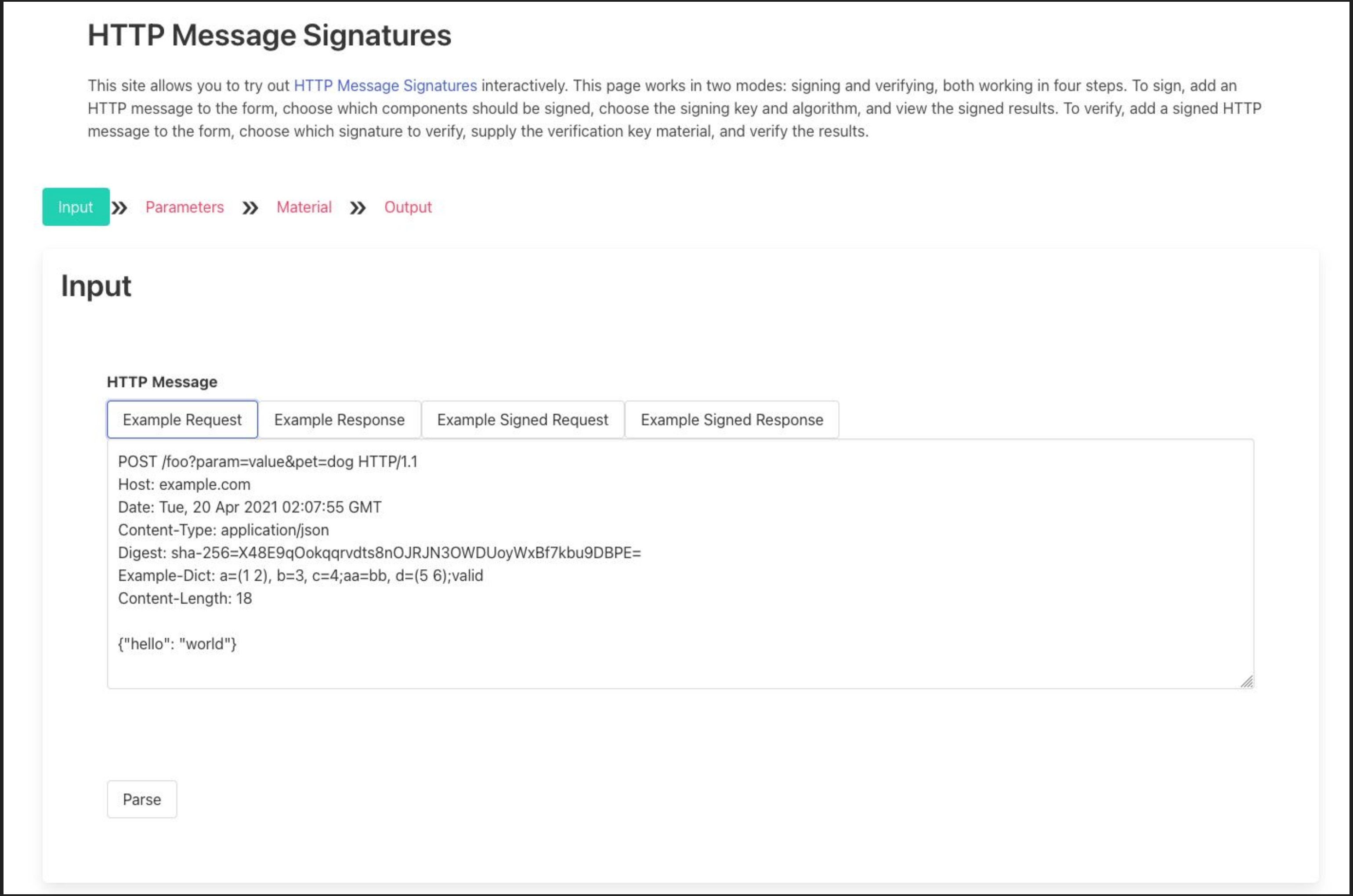
- Message signatures don’t protect message content
- Digest only protects message content (or representation)
  - No keys (body and hash can be swapped by attacker)
- But:
  - Digest encapsulates message content into a header field
  - We can sign header fields!
- Applications of Signatures are likely to need Digest too

Relationship to Signed HTTP Exchanges

- Signed HTTP Exchanges
  - Individual draft, targeted at WPACK
  - Expired
- Message signatures should be able to do everything that exchanges does
- Are there any next steps to make sure it can be used?

HTTP Signature Playground

https://httpsig.org/



HTTP Message Signatures

This site allows you to try out HTTP Message Signatures interactively. This page works in two modes: signing and verifying, both working in four steps. To sign, add an HTTP message to the form, choose which components should be signed, choose the signing key and algorithm, and view the signed results. To verify, add a signed HTTP message to the form, choose which signature to verify, supply the verification key material, and verify the results.

Input » Parameters » Material » Output

Input

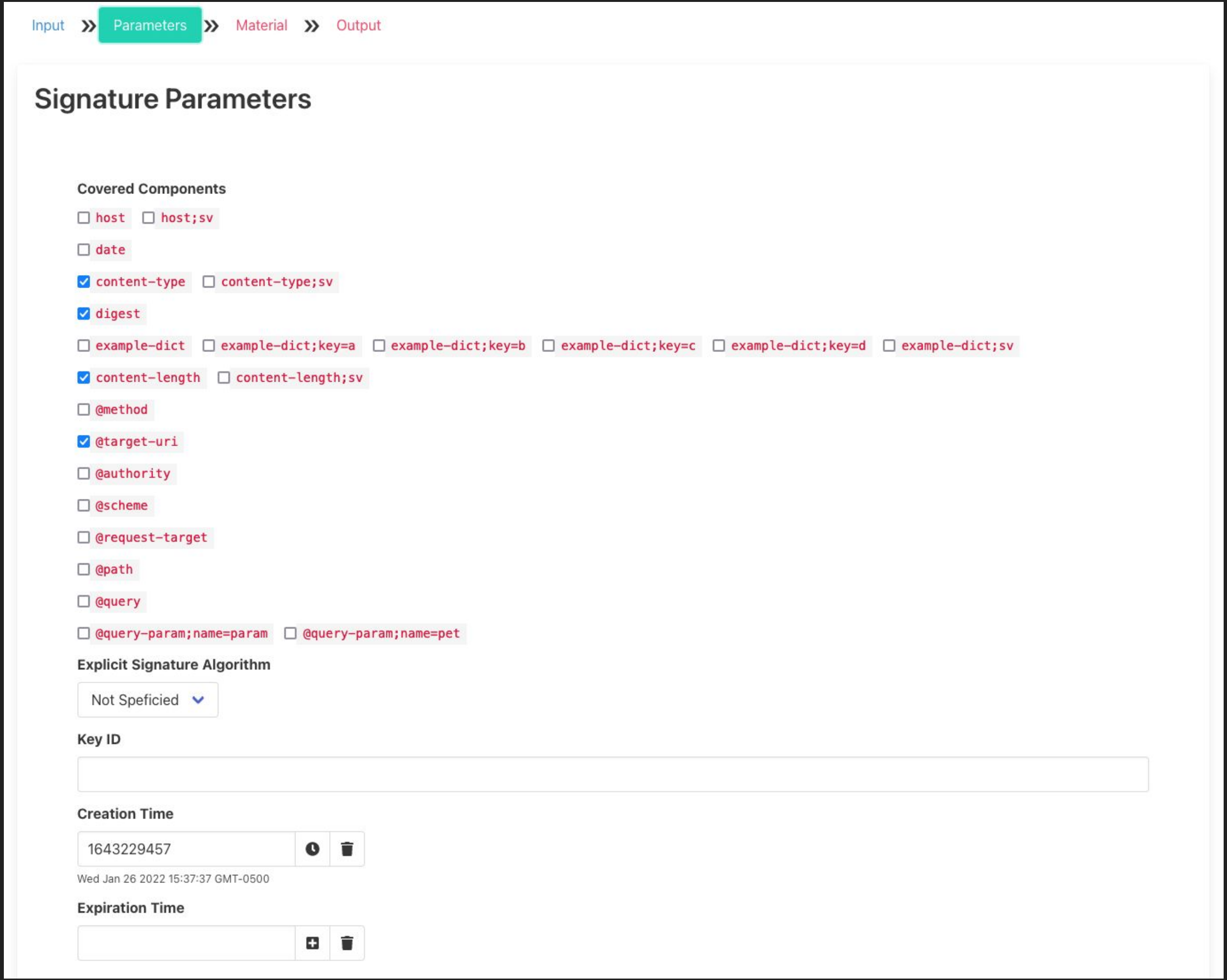
HTTP Message

Example Request | Example Response | Example Signed Request | Example Signed Response

```
POST /foo?param=value&pet=dog HTTP/1.1
Host: example.com
Date: Tue, 20 Apr 2021 02:07:55 GMT
Content-Type: application/json
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
Example-Dict: a=(1 2), b=3, c=4;aa=bb, d=(5 6);valid
Content-Length: 18

{"hello": "world"}
```

Parse



Input » Parameters » Material » Output

Signature Parameters

Covered Components

- [ ] `host` [ ] `host;sv`
- [ ] `date`
- [✓] `content-type` [ ] `content-type;sv`
- [✓] `digest`
- [ ] `example-dict` [ ] `example-dict;key=a` [ ] `example-dict;key=b` [ ] `example-dict;key=c` [ ] `example-dict;key=d` [ ] `example-dict;sv`
- [✓] `content-length` [ ] `content-length;sv`
- [ ] `@method`
- [✓] `@target-uri`
- [ ] `@authority`
- [ ] `@scheme`
- [ ] `@request-target`
- [ ] `@path`
- [ ] `@query`
- [ ] `@query-param;name=param` [ ] `@query-param;name=pet`

Explicit Signature Algorithm

Not Speficied

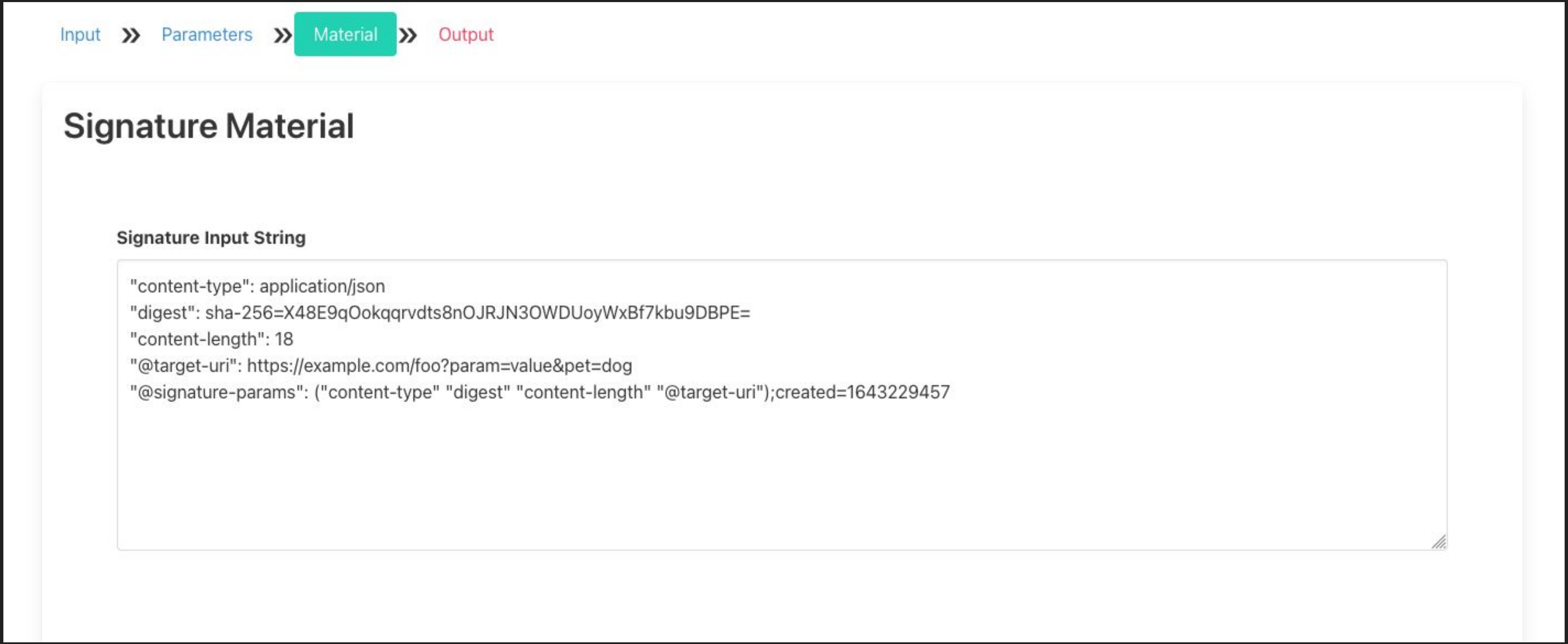
Key ID

Creation Time

1643229457

Wed Jan 26 2022 15:37:37 GMT-0500

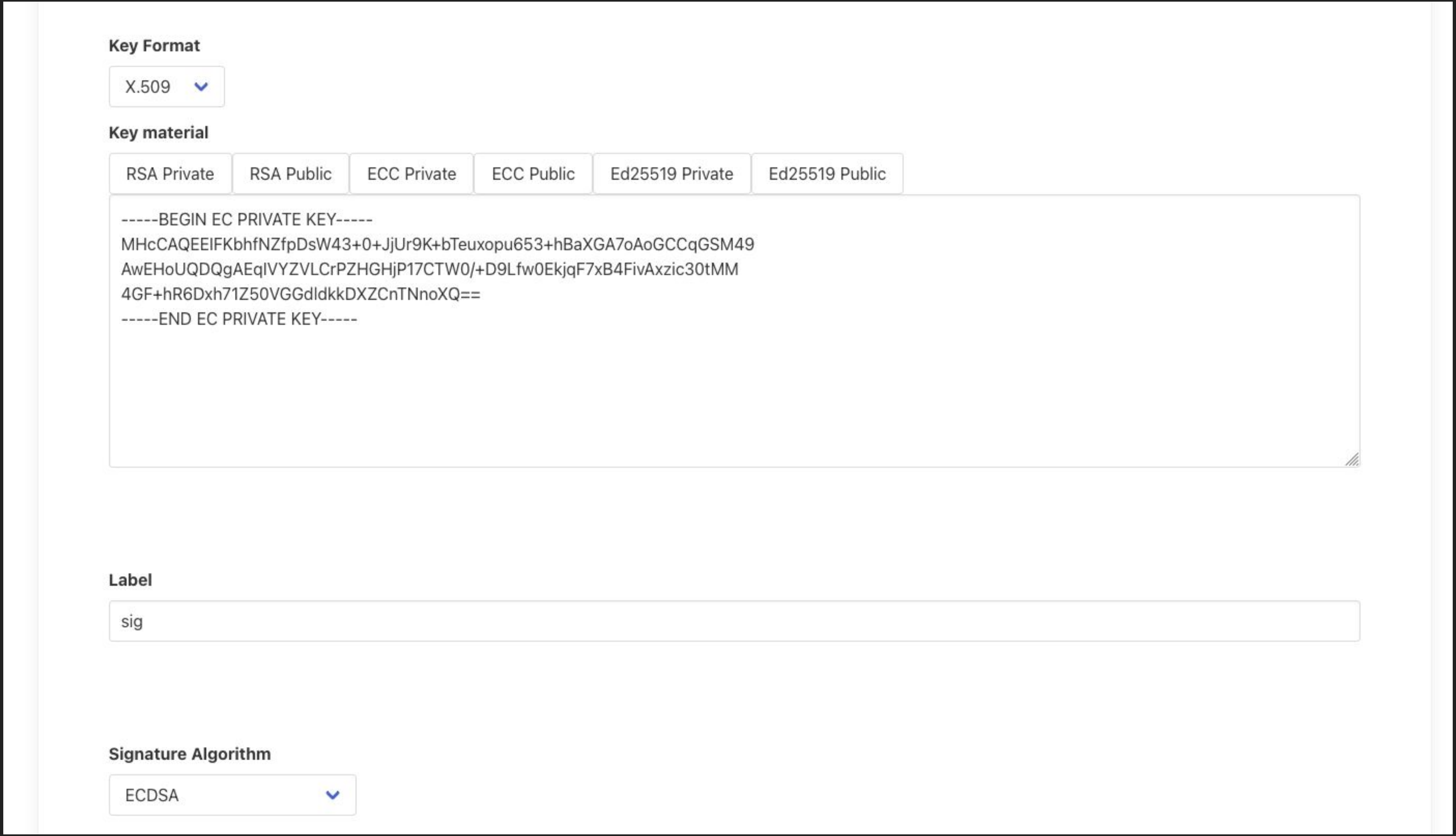
Expiration Time



Signature Material

Signature Input String

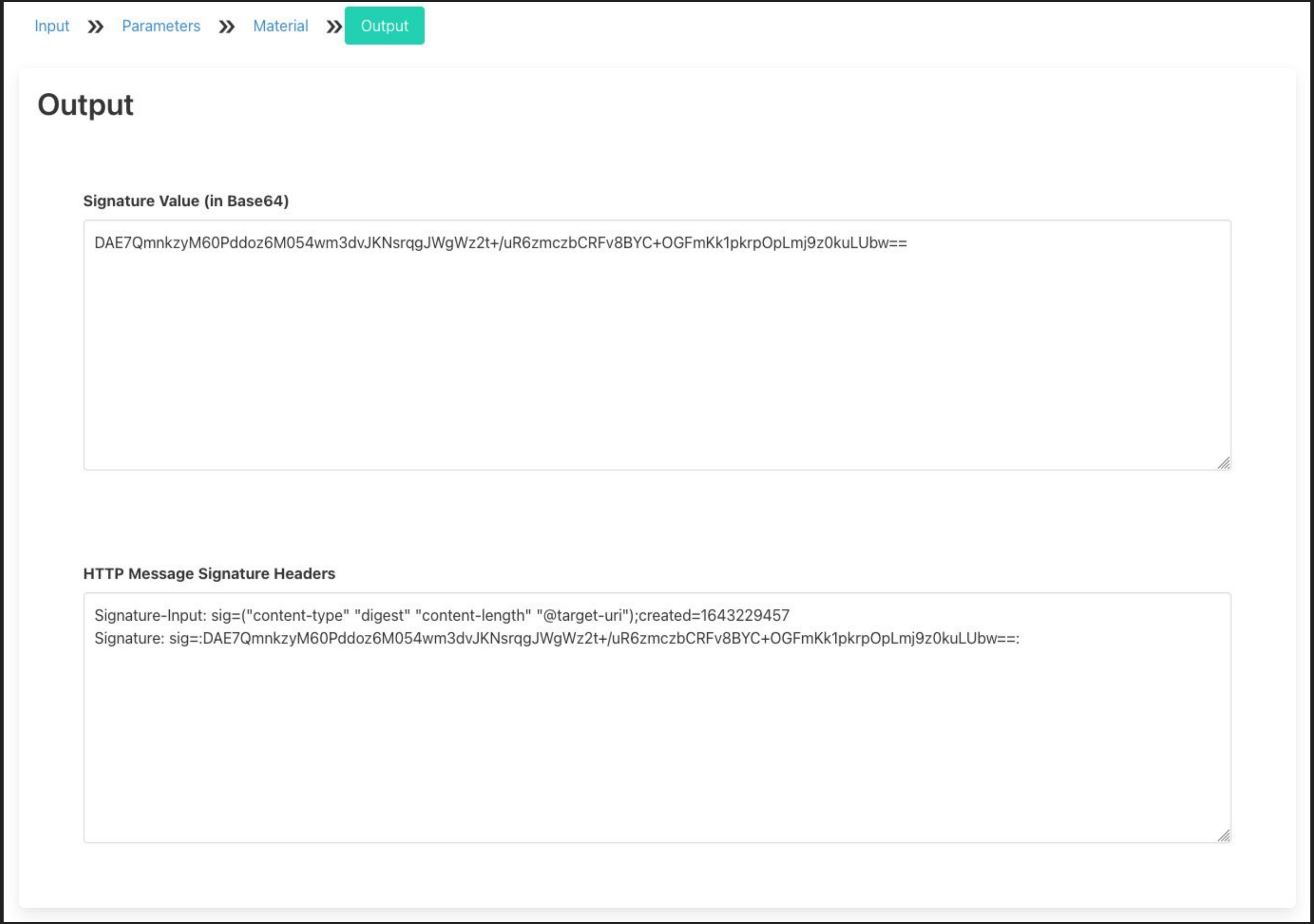
```
"content-type": application/json
"digest": sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
"content-length": 18
"@target-uri": https://example.com/foo?param=value&pet=dog
"@signature-params": ("content-type" "digest" "content-length" "@target-uri");created=1643229457
```



```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIFKbhfNZfpDsW43+0+JjUr9K+bTeuxopu653+hBaXGA7oAoGCCqGSM49
AwEHoUQDQgAEqIVYZVLCrPZHGHjP17CTW0/+D9Lfw0EkjqF7xB4FivAxzic30tMM
4GF+hR6Dxh71Z50VGGdIdkkDXZCnTNnoXQ==
-----END EC PRIVATE KEY-----
```

Label: sig

Signature Algorithm: ECDSA



Output

Signature Value (in Base64)

DAE7QmnkzyM60Pddoz6M054wm3dvJKNsrqgJWgWz2t+/uR6zmczbCRFv8BYC+OGFmKk1pkrpOpLmj9z0kuLUbw==

HTTP Message Signature Headers

```
Signature-Input: sig=("content-type" "digest" "content-length" "@target-uri");created=1643229457
Signature: sig=:DAE7QmnkzyM60Pddoz6M054wm3dvJKNsrqgJWgWz2t+/uR6zmczbCRFv8BYC+OGFmKk1pkrpOpLmj9z0kuLUbw==:
```

IETF 113

- GNAP Hackathon
  - Using HTTP Signatures as baseline key proofing method
- Approaching WGLC?