A Token to Participate

Extending Proof-of-Possession Tokens to Vehicles.

Lab8

Lightweight An and Az: Extending JWT to ECUs

Remember

There is no 100% security

Security, like all engineering, involves tradeoffs

Know what you are trying to secure

The adversary…

State Sponsored

Cutting Edge

Today's topic (lightweight authorization, **Az**) is new to embedded

It is common in enterprise IT

AEF (Agricultural Industry Electronics Foundation) *may* explore it

AEF TIM (version 1)

AEF TIM ("Tractor Implement Management") provides industry-standard approach to automating common control tasks between and implement and a tractor.

The original idea was from John Deere

- A baler that could command a tractor to make the baling operation better

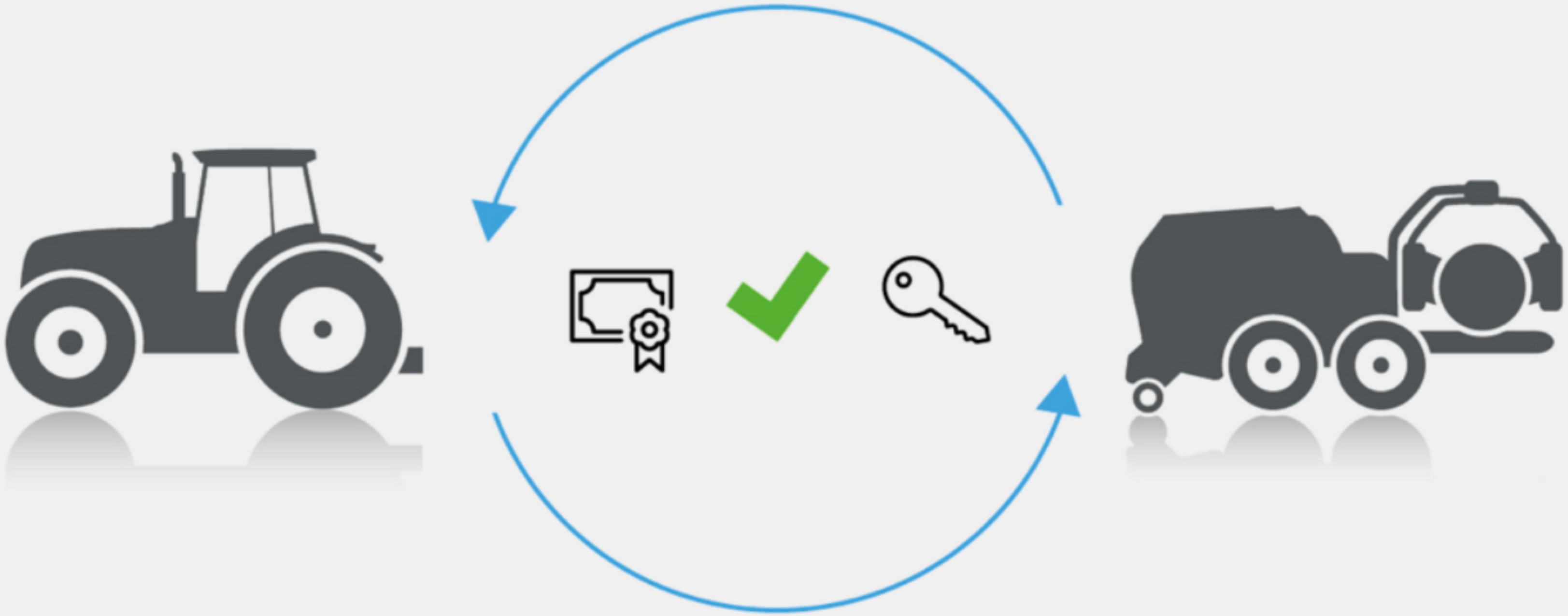
Other OEMs liked the idea, but knew that it wouldn't work in Europe for the wide array of implement manufactures and tractor manufacturers to each do their own approach

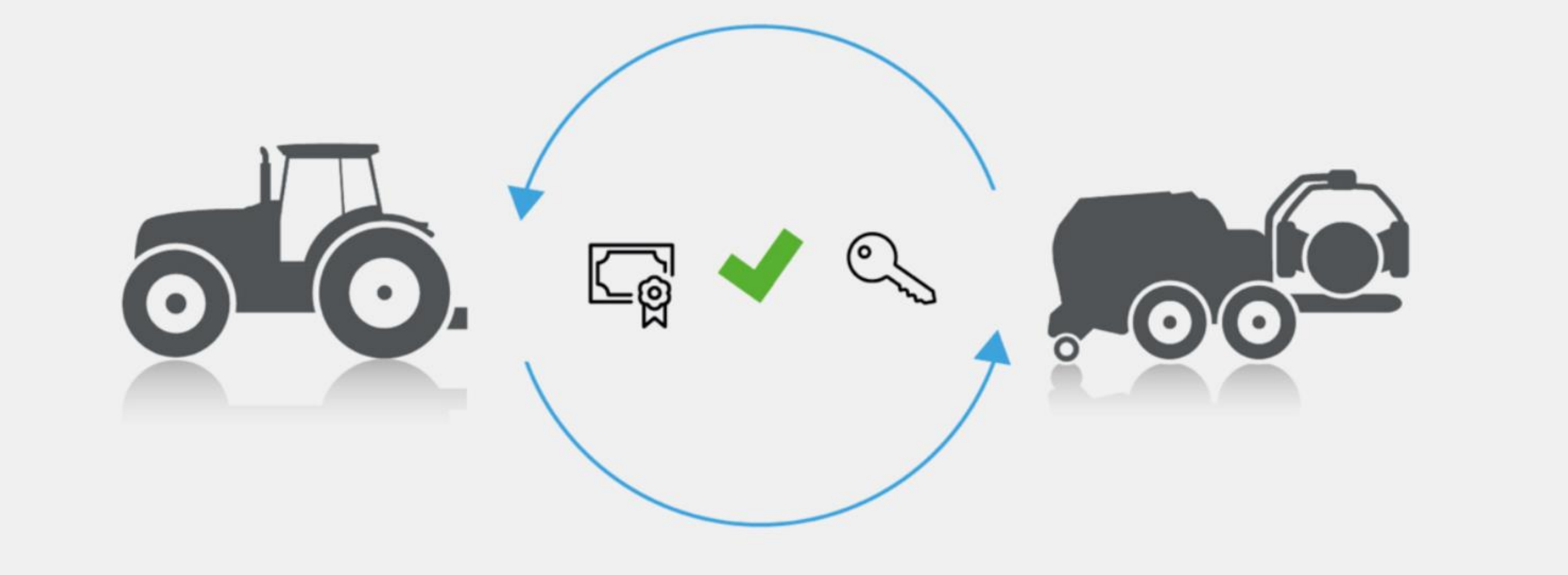
https://www.aef-online.org/about-us/about-the-aef.html
https://www.aef-online.org/about-us/activities/tractor-implement-management-tim.html

Tractor Implement Management (TIM)
The implement controls the tractor

TIM (tractor implement management system) is an ISOBUS-based solution for a barrier-free and cross-vendor agricultural technology system where the implement can control certain tractor functions. TIM's main concept is to use the intelligence of the entire combination – i.e. tractor and implement.

With other solutions, the tractor controls the implement, whereas with TIM experts speak of bidirectional communication, i.e. a transfer of control in both directions. That is to say: using TIM an implement is also capable of automatically controlling certain functions of a tractor – for example the forward speed or the remote valves. By requesting certain tractor functions the implement optimizes its operation themselves.





There are six functions defined in TIMv1

- Lab approval required
- Issued certificate (X.509) with fields indicating which functions were approved

No notion of change

- Authorized for “life” (99 years?)
- No notion of “subscription”

AEF conflates

- Authentication, An
- Authorization, Az

Motivation – Philosophical / Technical

Split An and Az into more appropriate tech

An : Authentic / Authenticate / Authentication

- A big commitment
  - Unusual to change
  - Much more formal when serious about security
- Solution: X.509 certificate from Public Key Infrastructure

- *Expensive*
- *Rigid processes*
- *ASN.1 is complex*
- *Not Best Fit for Az*

Az : Authorized / Authorization

- Not a big commitment
  - Change is expected (or required?)
  - Self-corrects within X hours/days/months
- Solution: Proof-of-Possession Tokens (CWTs or JWTs)

- *Lightweight signing process*
- *CWT parsers are intentionally simple and small*

Where We Are Going

From

- X.509 certificate for An and Az

To

- X.509 certificate for An
- Proof-of-Possession CWT for Az
  - Lists security “claims”
  - The list is signed
  - Example: *I claim I’m authorized to use function 1*

Where CWT is compact and intended for constrained environments

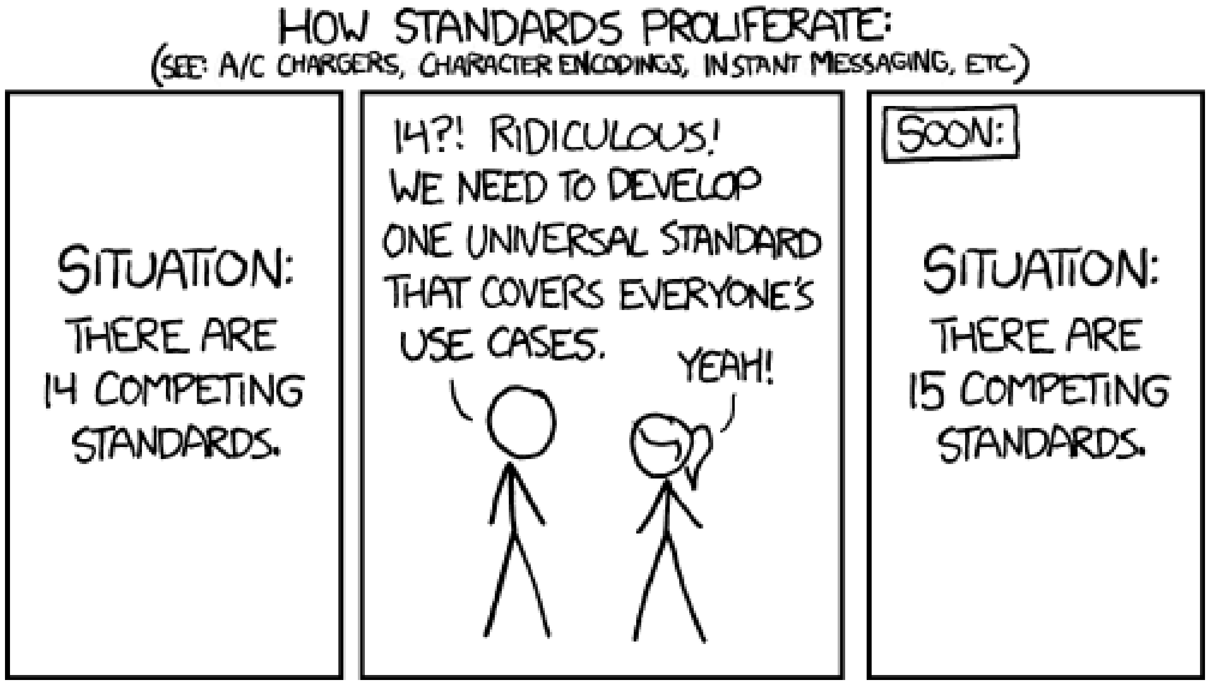
RFC 8949
Concise Binary Object Representation (CBOR)

- https://www.rfc-editor.org/rfc/rfc8949.html
- “The Concise Binary Object Representation (CBOR) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation. These design goals make it different from earlier binary serializations such as ASN.1 and MessagePack.”

- https://cbor.io/ and https://cbor.me/

RFC 8949 (cont.)

- “There are hundreds of standardized formats for binary representation of structured data (also known as binary serialization formats). ...In the IETF, probably the best-known formats... are ASN.1's BER and DER [ASN.1].”
- Extends JSON (RFC 8259)



https://xkcd.com/927/

RFC 8392
CBOR Web Token (CWT)

- https://www.rfc-editor.org/rfc/rfc8392.html
- “CBOR Web Token (CWT) is a compact means of representing claims to be transferred between two parties. The claims in a CWT are encoded in the Concise Binary Object Representation (CBOR), and CBOR Object Signing and Encryption (COSE) is used for added application-layer security protection. A claim is a piece of information asserted about a subject and is represented as a name/value pair consisting of a claim name and a claim value. CWT is derived from JSON Web Token (JWT) but uses CBOR rather than JSON.”

Claims

Registered

- iss (Issuer) Claim
- sub (Subject) Claim
- aud (Audience) Claim
- exp (Expiration Time) Claim
- nbf (Not Before) Claim
- iat (Issued At) Claim
- cti (CWT ID) Claim

Other Standard

- cnf (Confirmation) Claim

The public part of PoP key

Proprietary

- func1 (Function 1) Claim
- …

We create what we need

JWT & CWT

JSON Web Token & CBOR Web Token

Extension:

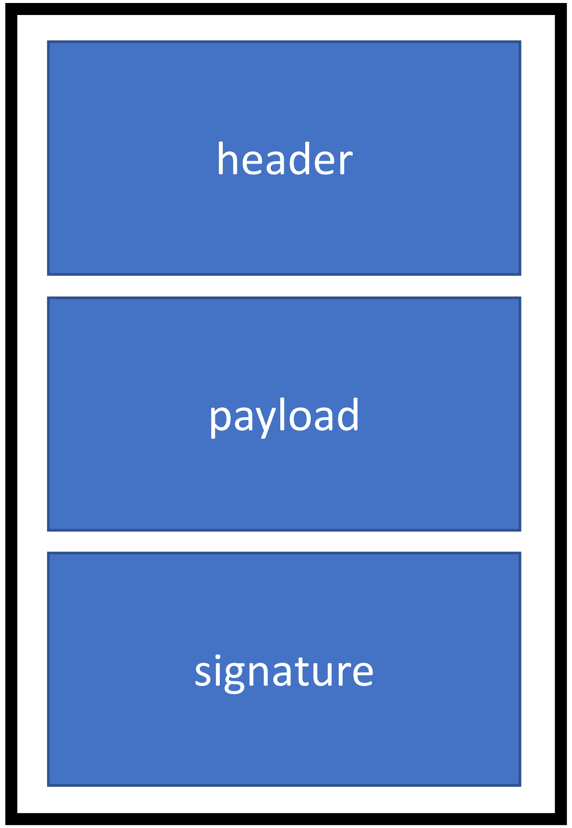
1. For ECU Az
2. For ECU “proof-of-possession”

Authorization (Az)

- Payload contains roles that this ECU is authorized to perform

Authentication (An)

- Payload contains:
    - “subject” (the ECU)
    - “issuer” (the OEM)
    - “audience” (the target, e.g., “AEF TIM Server” or “AEF TIM Client”)
    - “cnf” (confirmation key)
- ECU demonstrates “proof-of-possession”



https://jwt.io/introduction
https://www.rfc-editor.org/rfc/rfc7519

Proof-of-Possession (Enterprise IT)

```
+--------------+
|              |                         +--------------+
|              |--(3) Presentation of -->|              |
|              |      JWT w/ Public      |              |
|  Presenter   |      PoP Key            |              |
|              |                         |              |
|              |<-(4) Communication ---->|              |
|              |      Authenticated by   |              |
+--------------+      PoP Key            |              |
  |          ^                           |              |
  |          |                           |              |
 (1) Public (2) JWT w/                   |   Recipient  |
  |  PoP     |  Public                   |              |
  |  Key     |  PoP Key                  |              |
  v          |                           |              |
+--------------+                         |              |
|              |                         |              |
|              |                         |              |
|              |                         |              |
|    Issuer    |                         |              |
|              |                         |              |
|              |                         |              |
|              |                         +--------------+
+--------------+
```

Figure 2: Proof of Possession with an Asymmetric Key

RFC 7800 Proof-of-Possession Key for JWTs April 2016

In the case illustrated in Figure 2, the presenter generates a public/private key pair and (1) sends the public key to the issuer, which creates a JWT that contains the public key (or an identifier for it) in the confirmation claim. The entire JWT is integrity protected using a digital signature to protect it against modifications. The JWT is then (2) sent to the presenter. When the presenter (3) presents the JWT to the recipient, it also needs to demonstrate possession of the private key. The presenter, for example, (4) uses the private key in a Transport Layer Security (TLS) exchange with the recipient or (4) signs a nonce with the private key. The recipient is able to verify that it is interacting with the genuine presenter by extracting the public key from the confirmation claim of the JWT (after verifying the digital signature of the JWT) and utilizing it with the private key in the TLS exchange or by checking the nonce signature.

In both cases, the JWT may contain other claims that are needed by the application.

https://www.rfc-editor.org/rfc/rfc7800.html

Proof-of-Possession (Embedded)

The “Proof” process is not standardized for embedded. This is a proposal…

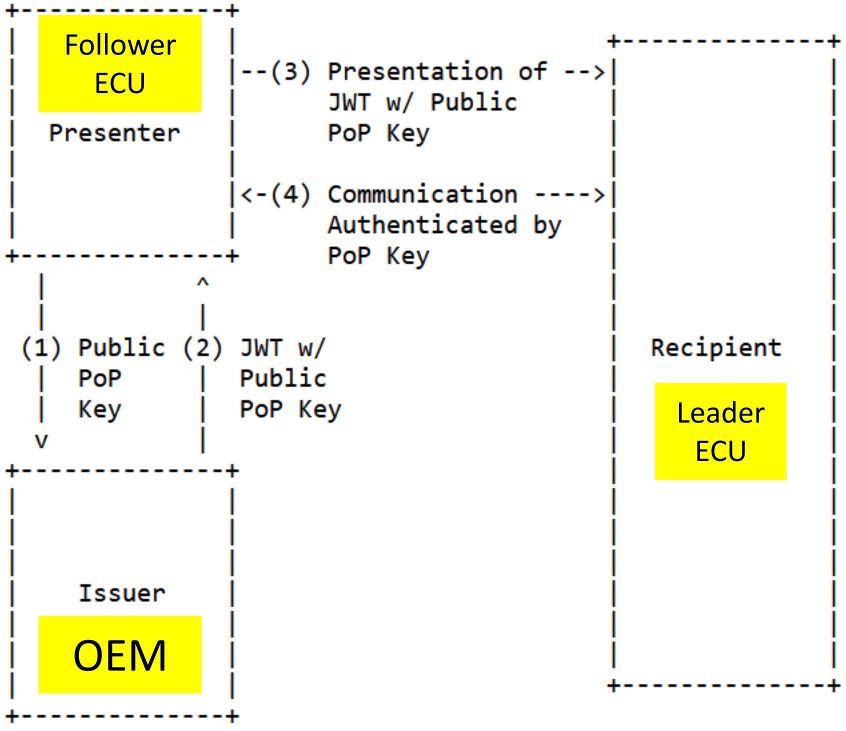
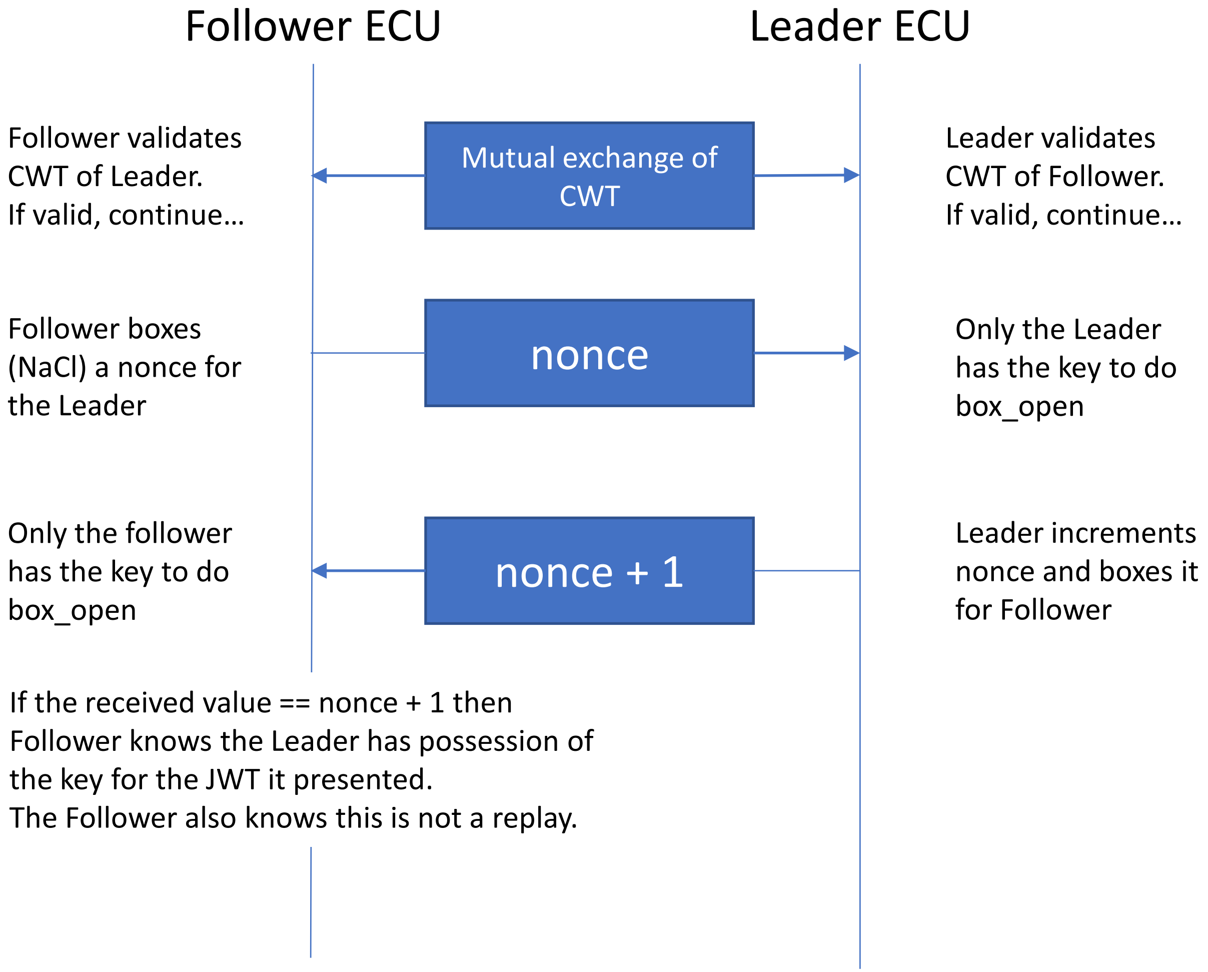


Figure 2: Proof of Possession with an Asymmetric Key



Repeat the proof-of-possession with roles reversed, so that the Leader knows the Follower has possession of the key and that this is not a replay

https://www.rfc-editor.org/rfc/rfc7800.html

A Library for CWT

- https://github.com/dajiaji/python-cwt#signed-cwt

Lab – Transport Protocol

- To send more than fits in a frame
- Tools provided by ‘can-utils’

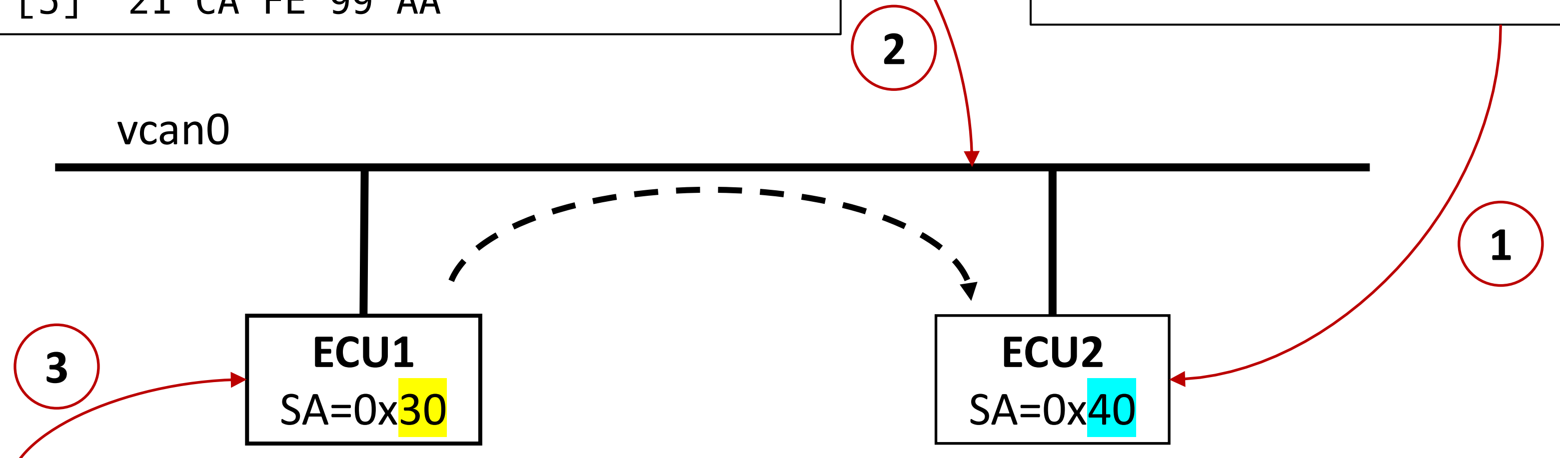
- In our tools: ISO 15765-2 (for automotive diagnostics)
- In vehicles: J1939/21 Transport Protocol (RTS/CTS)

Lab – Transport Protocol

1. Get ready to receive with ***isotprecv***
2. Get ready to dump traffic with ***candump***
3. Send using ***isotpsend***

```
$ candump vcan0

  vcan0  030   [8]  10 0A 11 22 33 44 55 66
  vcan0  040   [3]  30 00 00
  vcan0  030   [5]  21 CA FE 99 AA
```

```
$ isotprecv -s 40 -d 30 -l vcan0

11 22 33 44 55 66 CA FE 99 AA
```



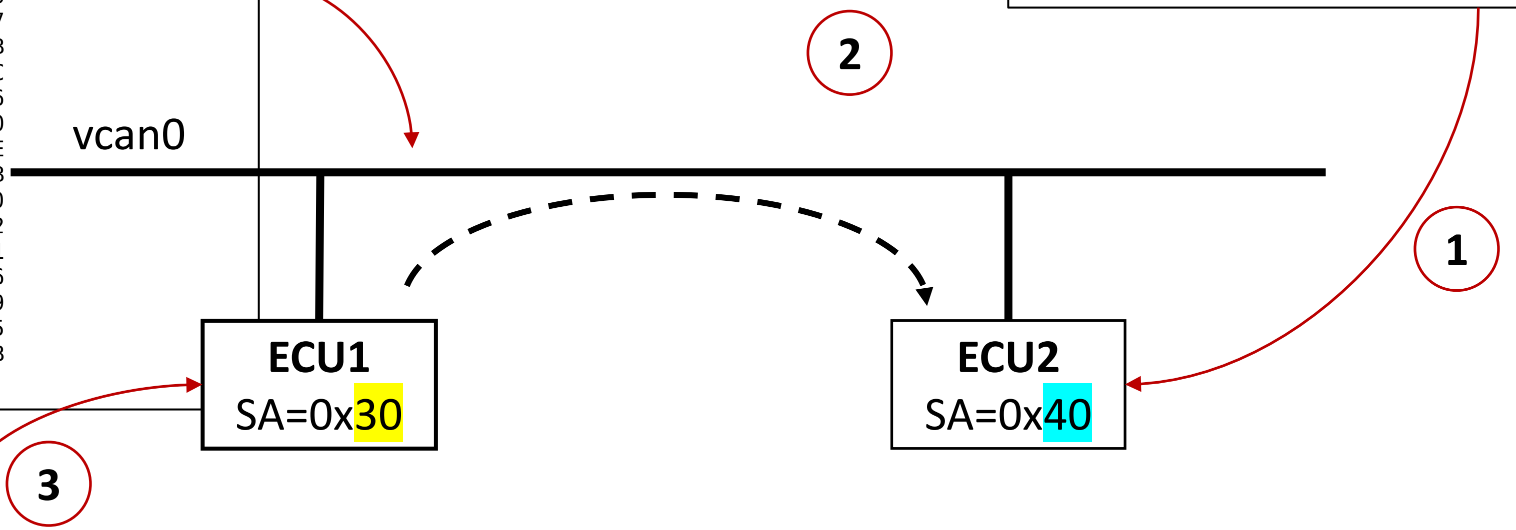

```
$ echo 11 22 33 44 55 66 CA FE 99 AA | isotpsend -s 30 -d 40 vcan0
```

Lab – TP of pop.cwt

1. Get ready to receive with ***isotprecv***
2. Get ready to dump traffic with ***candump***
3. Send using ***isotpsend***

```
$ candump vcan0
  vcan0  030   [8]  10 C4 D2 84 43 A1 01 27
  vcan0  040   [3]  30 00 00
  vcan0  030   [8]  21 A1 04 49 69 73 73 75
  vcan0  030   [8]  22 65 72 2D 30 31 58 6E
  vcan0  030   [8]  23 A7 01 67 6F 65 6D 2E
  vcan0  030   [8]  24 63 6F 6D 02 69 62 61
  vcan0  030   [8]  25 6C 65 72 31 30 30 31
  vcan0  030   [8]  26 03 67 74 72 61 63 74
  vcan0  030   [8]  27 6F 72 08 A1 01 A6 01
  vcan0  030   [8]  28 01 02 4C 70 72 65 73
  vcan0  030   [8]  29 65 6E 74 65 72 2D 30
  vcan0  030   [8]  2A 31 03 27 04 81 02 20
  vcan0  030   [8]  2B 06 21 58 20 A3 49 A1
  vcan0  030   [8]  2C 1D 7F 2B 89 3C 6E B6
  vcan0  030   [8]  2D 36 AF 6C F7 E0 A9 66
  vcan0  030   [8]  2E D2 5C F3 98 60 20 F7
  vcan0  030   [8]  2F 7A 0E 96 55 45 9C 5B
  vcan0  030   [8]  20 74 04 1A 64 2B 3A EC
  vcan0  030   [8]  21 05 1A 64 2B 2C DC 06
  vcan0  030   [8]  22 1A 64 2B 2C DC 58 40
  vcan0  030   [8]  23 E0 23 E3 86 91 F9 7E
  vcan0  030   [8]  24 E2 C7 E0 3E 33 B8 DB
  vcan0  030   [8]  25 C3 09 1E 54 DD 7B D0
  vcan0  030   [8]  26 28 B2 4C 7F 3C 22 82
  vcan0  030   [8]  27 C4 3D 8C 30 D1 46 71
  vcan0  030   [8]  28 5E C6 E4 6E D9 75 46
  vcan0  030   [8]  29 A2 C7 9A E9 D9 6A A9
  vcan0  030   [8]  2A D5 A1 26 6D 1B 5E 75
  vcan0  030   [8]  2B 0A 20 B0 69 7B CF 8B
  vcan0  030   [2]  2C 06
```

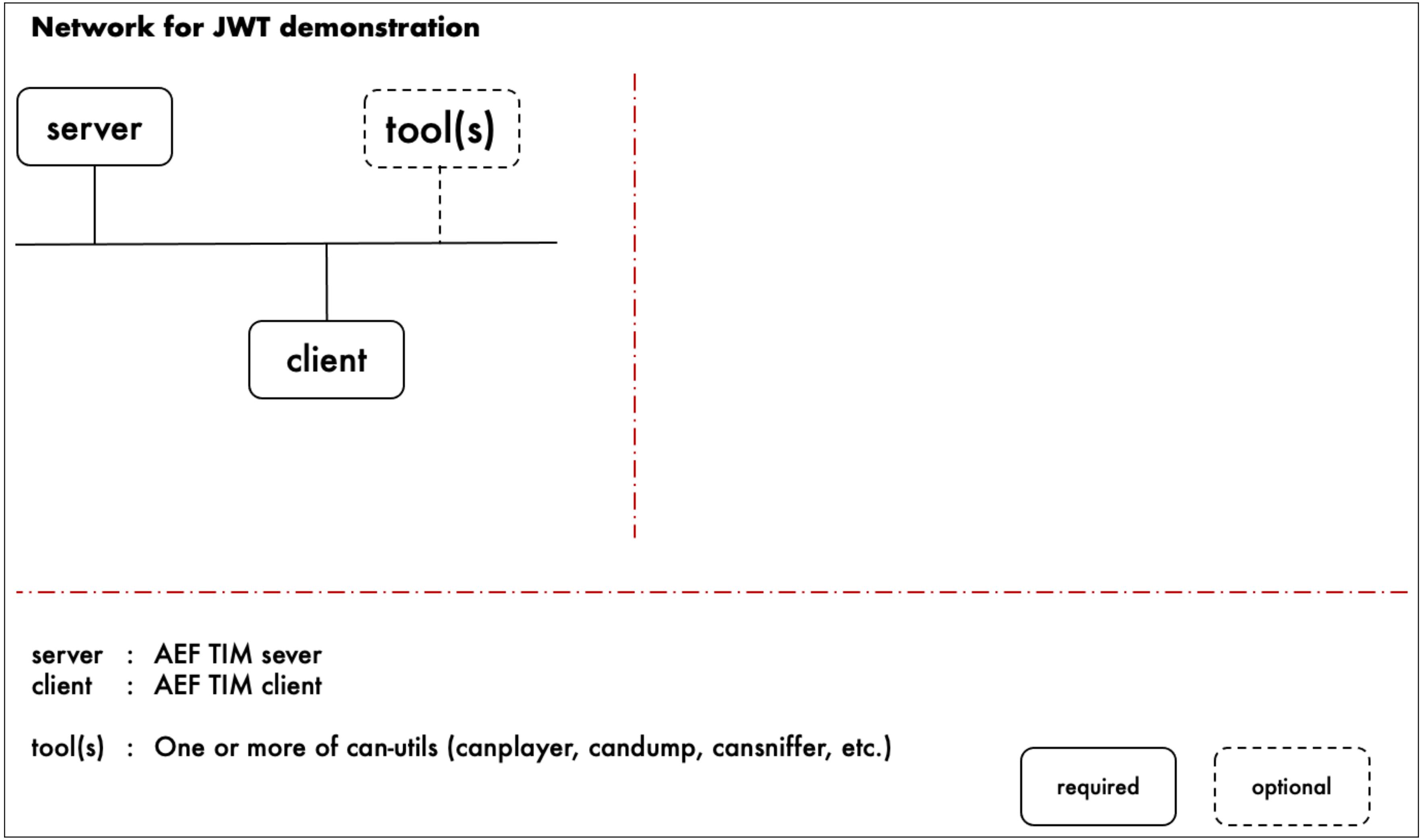
```
$ isotprecv -s 40 -d 30 -l vcan0

D2 84 43 A1 01 27 A1 04 49 69 73 73 75 65 72 2D 30 31 58 6E A7
01 67 6F 65 6D 2E 63 6F 6D 02 69 62 61 6C 65 72 31 30 30 31 03
67 74 72 61 63 74 6F 72 08 A1 01 A6 01 01 02 4C 70 72 65 73 65
6E 74 65 72 2D 30 31 03 27 04 81 02 20 06 21 58 20 A3 49 A1 1D
7F 2B 89 3C 6E B6 36 AF 6C F7 E0 A9 66 D2 5C F3 98 60 20 F7 7A
0E 96 55 45 9C 5B 74 04 1A 64 2B 3A EC 05 1A 64 2B 2C DC 06 1A
64 2B 2C DC 58 40 E0 23 E3 86 91 F9 7E E2 C7 E0 3E 33 B8 DB C3
09 1E 54 DD 7B D0 28 B2 4C 7F 3C 22 82 C4 3D 8C 30 D1 46 71 5E
C6 E4 6E D9 75 46 A2 C7 9A E9 D9 6A A9 D5 A1 26 6D 1B 5E 75 0A
20 B0 69 7B CF 8B 06
```




```
$ xxd -c 1 -p  pop.cwt | isotpsend -s 30 -d 40 vcan0
```

Network Configuration

Network for this Lab



Lab

- $ pip3 install cwt

Lab - CWT

```
> cat -n signed_cwt.py
# source: https://github.com/dajiaji/python-cwt#signed-cwt
#         https://github.com/dajiaji/python-cwt#cwt-with-user-defined-claims

from cwt import encode, decode, COSEKey, Claims, set_private_claim_names

# filename of OEM private and public signing keys
oem_k = "./oem_ed25519-priv.pem"
oem_p = "./oem_ed25519-pub.pem"

# TIM specific claims
tim_claim_names = {
    "func1": -70001,
    "func2": -70002,
    "func3": -70003,
    "func4": -70004,
    "func5": -70003,
    "func6": -70004,
}
set_private_claim_names(tim_claim_names)

# the sender side:
with open(oem_k) as key_file:
    private_key = COSEKey.from_pem(key_file.read(), kid="01")
    token = encode(
      {"iss": "oem.com", "sub": "baler1001", "aud": "TIMServer",
       "func5": True }, private_key
    )

# the recipient side:
with open(oem_p) as key_file:
    public_key = COSEKey.from_pem(key_file.read(), kid="01")
    decoded = decode(token, public_key)    # raw form
    readable = Claims.new(decoded, private_claim_names=tim_claim_names)
    print(decoded)
    print("iss: ", readable.get("iss"))
    print("sub: ", readable.get("sub"))
    print("aud: ", readable.get("aud"))
    print("exp: ", readable.get("exp"))
    print("func1: ", readable.get("func1"))
    print("func5: ", readable.get("func5"))
```

```
> python3 signed_cwt.py
{1: 'oem.com', 2: 'baler1001', 3: 'TIMServer', -70003: True, 4: 1680554505, 5: 1680550905,
6: 1680550905}
iss:  oem.com
sub:  baler1001
aud:  TIMServer
exp:  1680554505
func1:  None
func5:  True
```

Lab – PoP CWT

1 of 3

(1) lines 13-17
(2) lines 20-41

```
+--------------+
|              |
|              |--(3) Presentation of -->+--------------+
|              |      JWT w/ Public       |              |
|  Presenter   |      PoP Key             |              |
|              |                          |              |
|              |<-(4) Communication ----->|              |
|              |      Authenticated by    |              |
+--------------+      PoP Key             |              |
   |         ^                            |              |
   |         |                            |              |
(1) Public (2) JWT w/                     |  Recipient   |
   | PoP     |  Public                    |              |
   | Key     |  PoP Key                   |              |
   v         |                            |              |
+--------------+                          |              |
|              |                          |              |
|              |                          |              |
|    Issuer    |                          |              |
|              |                          |              |
|              |                          |              |
+--------------+                          +--------------+
```

Figure 2: Proof of Possession with an Asymmetric Key

```
> python3 pop_cwt.py
o0mhHX8riTxutjavbPfgqWbSXPOYYCD3eg6WVUWcW3Q=
```

```
> cat -n pop_cwt.py
 1  # source: https://github.com/dajiaji/python-cwt#cwt-with-pop-key
 2  # docs:   https://python-cwt.readthedocs.io/en/stable/index.html
 3
 4
 5  # ========================================================================
 6  #       the OEM has a server ISSUING the CWT
 7  # ========================================================================
 8
 9  import cwt
10  from cwt import COSEKey, Claims
11  import base64
12
13  # the public signing key here came from copy/paste of baler_ed25519-pub.txt
14  Pb_s_bytes = bytes.fromhex("a349a11d7f2b893c6eb636af6cf7e0a966d25cf3986020f77a0e9655459c5b74")
15  Pb_s_b64 = base64.b64encode(Pb_s_bytes)
16  Pb_s = str(Pb_s_b64, encoding="ascii")  # converts bytes to ascii expected in 'encode'
17  print(Pb_s)
18
19  # Read's the OEM's private signing key
20  with open("./oem_ed25519-priv.pem") as key_file:
21      private_key = COSEKey.from_pem(key_file.read(), kid="issuer-01")
22
23      # Sets the PoP key to a CWT for the presenter.
24      token = cwt.encode(
25          {
26              "iss": "oem.com",
27              "sub": "baler1001",
28              "aud": "tractor",
29              "cnf": {
30                  "jwk": {  # Provided by the CWT presenter.
31                      "kty": "OKP",
32                      "use": "sig",
33                      "crv": "Ed25519",
34                      "kid": "presenter-01",
35                      "x":   Pb_s,
36                      "alg": "EdDSA",
37                  },
38              },
39          },
40          private_key,
41      )
42
43  # Issues the token to the presenter.
44
45
```

Lab – PoP CWT

(3) lines 51-59

```
+--------------+
|              |--(3) Presentation of -->+--------------+
|              |      JWT w/ Public      |              |
|  Presenter   |      PoP Key            |              |
|              |                         |              |
|              |<-(4) Communication ---->|              |
|              |      Authenticated by   |              |
+--------------+      PoP Key            |              |
  |         ^                            |              |
  |         |                            |              |
 (1) Public (2) JWT w/                   |  Recipient   |
  |  PoP     |  Public                   |              |
  |  Key     |  PoP Key                  |              |
  v          |                           |              |
+--------------+                         |              |
|              |                         |              |
|              |                         |              |
|   Issuer     |                         |              |
|              |                         |              |
|              |                         |              |
|              |                         +--------------+
+--------------+
```

Figure 2: Proof of Possession with an Asymmetric Key

```
47  # ==========================================================================
48  #      the baler uses the CWT to demonstrate "proof-of-possession" (PoP)
49  # ==========================================================================
50
51  # Prepares a private PoP key in advance.
52  with open("./baler_ed25519-priv.pem") as key_file:
53      pop_key_private = COSEKey.from_pem(key_file.read(), kid="presenter-01")
54
55      # Receives a message (e.g., nonce)  from the recipient.
56      msg = b"could-you-sign-this-message?"  # Provided by recipient.
57
58      # Signs the message with the private PoP key.
59      sig = pop_key_private.sign(msg)
60
61  # Sends the msg and the sig with the CWT to the recipient.
62
63  # create an invalid signature, in parallel; will 'verify' catch this? Yes!
64  invalidsig = bytes.fromhex("00") + sig[2:]
65
```

Lab – PoP CWT


```
66 # ================================================================================
67 #      the tractor confirms the baler's "proof-of-possession" (PoP)
68 # ================================================================================
69
70 # Read the OEM's public key; so tractor can confirm CWT is valid
71 with open("./oem_ed25519-pub.pem") as key_file:
72     public_key = COSEKey.from_pem(key_file.read(), kid="issuer-01")
73
74     # Validates and decodes the CWT received from the baler.
75     raw = cwt.decode(token, public_key)
76     decoded = Claims.new(raw)
77
78     # Extracts the PoP key from the CWT.
79     extracted_pop_key = COSEKey.new(decoded.cnf)  # = raw[8][1]
80
81     # Then, verifies the message sent by the presenter
82     # with the signature which is also sent by the presenter as follows:
83     print(extracted_pop_key.verify(msg, sig))
84     print(extracted_pop_key.verify(msg, invalidsig))
```

Valid signature

Invalid signature

```
None
Traceback (most recent call last):
  File "/home/john/.local/lib/python3.10/site-packages/cwt/algs/okp.py", line 26
7, in verify
    self._public_key.verify(sig, msg)
  File "/home/john/.local/lib/python3.10/site-packages/cryptography/hazmat/backe
nds/openssl/ed25519.py", line 77, in verify
    raise exceptions.InvalidSignature
cryptography.exceptions.InvalidSignature

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/home/john/HVOCLabs/lab8/classroom/pop_cwt.py", line 84, in <module>
    print(extracted_pop_key.verify(msg, invalidsig))
  File "/home/john/.local/lib/python3.10/site-packages/cwt/algs/okp.py", line 26
9, in verify
    raise VerifyError("Failed to verify.") from err
cwt.exceptions.VerifyError: Failed to verify.
```

(4) lines 70-84

```
+--------------+
|              |--(3) Presentation of -->+--------------+
|              |      JWT w/ Public      |              |
|  Presenter   |      PoP Key            |              |
|              |                         |              |
|              |<-(4) Communication ---->|              |
|              |      Authenticated by   |              |
+--------------+      PoP Key            |              |
  |         ^                            |              |
  |         |                            |              |
 (1) Public (2) JWT w/                   |  Recipient   |
  |  PoP    |   Public                   |              |
  |  Key    |   PoP Key                  |              |
  v         |                            |              |
+--------------+                         |              |
|              |                         |              |
|              |                         |              |
|              |                         |              |
|    Issuer    |                         |              |
|              |                         |              |
|              |                         |              |
|              |                         +--------------+
+--------------+
```

Figure 2: Proof of Possession with an Asymmetric Key

https://cbor.me

Import bookmarks... JLPLabs mybookmarks.html ISU University Museums ISU DIGITAL AG BioCentury Research F... ABE Home senior design bash

CBOR playground. See RFC 8949 for the CBOR specification, and cbor.io for more background informatio

CBOR

Diagnostic → plain hex

← 196 Bytes as text utf8 emb cbor cborseq

enter hex below or Browse... No file selected.

cnf key

signature

```
18([<< {1: -8} >>, {4: 'issuer-01'}, << {1: "oem.com", 2: "baler1001", 3:
"tractor", 8: {1: {1: 1, 2: 'presenter-01', 3: -8, 4: [2], -1: 6, -2:
h'A349A11D7F2B893C6EB636AF6CF7E0A966D25CF3986020F77A0E9655459C5B74'}}, 4:
1680554732, 5: 1680551132, 6: 1680551132} >>,
h'E023E38691F97EE2C7E03E33B8DBC3091E54DD7BD028B24C7F3C2282C43D8C30D146715EC
6E46ED97546A2C79AE9D96AA9D5A1266D1B5E750A20B0697BCF8B06'])
```

```
D2                                      # tag(18)
   84                                   # array(4)
      43                                # bytes(3)
         A10127                         # "\xA1\u0001'"
      A1                                # map(1)
         04                             # unsigned(4)
         49                             # bytes(9)
            6973737565722D3031          # "issuer-01"
      58 6E                             # bytes(110)

A701676F656D2E636F6D026962616C6572313030310367747261637746F7
208A101A60101024C70726573656E7465722D3031032704810220062158
20A349A11D7F2B893C6EB636AF6CF7E0A966D25CF3986020F77A0E96554
59C5B74041A642B3AEC051A642B2CDC061A642B2CDC #
"\xA7\u0001goem.com\u0002ibaler1001\u0003gtractor\b\xA1
\u0001\xA6\u0001\u0001\u0002Lpresenter-01\u0003'\u0004
\x81\u0002 \u0006!X \xA3I\xA1\u001D\u007F+\x89<n\xB66\xAFl
\xF7\xE0\xA9f\xD2\\\xF3\x98` \xF7z\u000E\x96UE\x9C[t\u0004
\u001Ad+:\xEC\u0005\u001Ad+,\xDC\u0006\u001Ad+,\xDC"
      58 40                             # bytes(64)

E023E38691F97EE2C7E03E33B8DBC3091E54DD7BD028B24C7F3C2282C43
D8C30D146715EC6E46ED97546A2C79AE9D96AA9D5A1266D1B5E750A20B0
697BCF8B06 # "\xE0#ト\xF9~\xE2\xC7\xE0>3\xB8\xDB\xC3\t
\u001ET\xDD{\xD0(\xB2L\u007F<\"\x82\xC4=\x8C0\xD1Fq^\xC6
\xE4n\xD9uF\xA2ű\xE9\xD9j\xA9u&m\e^u\n \xB0i{ü\u0006"
```