verify

December 2, 2024

[ ]:
```
from datetime import datetime
import json
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric.utils import encode_dss_signature
from cryptography.hazmat.primitives import serialization

# Helper function for converting strings into byte arrays needed by␣
 ↪cryptographic functions
def string_to_bytes(s):
    return s.encode('utf-8')

# This function will ensure that we represent the JSON dictionary as exactly the
# same string every time, otherwise we'd get different hashes while signing
def canonicalize_json(j):
    return json.dumps(j, sort_keys=True)

def verify(ca_identity, signed_message_filename):
    print("Trying to verify " + signed_message_filename)

    # Load the signed message data
    with open(signed_message_filename, 'r') as fh:
        signed_message = json.load(fh)

    # Read out the identity of the signer and load their certificate
    signer_identity = signed_message['signer identity']
    with open(signer_identity + '.cert', 'r') as fh:
        signer_cert = json.load(fh)

    # Format the certificate body for signing as a byte array in a canonical␣
 ↪order
    cert_body_to_be_signed =␣
 ↪string_to_bytes(canonicalize_json(signer_cert["body"]))

    # Read out the identity of the issuer and load their public key
    issuer_identity = signer_cert['body']['issuer identity']
```

```
    signer_pk = serialization.
↪load_pem_public_key(string_to_bytes(signer_cert['body']['public key']))
    with open(ca_identity + '.pk', 'r') as fh:
        ca_public_key = serialization.load_pem_public_key(string_to_bytes(fh.
↪read()))

    # YOUR SOLUTION STARTS HERE

    # Functions that might be of use to you:
    # - datetime.fromisoformat (https://docs.python.org/3/library/datetime.
↪html#datetime.date.fromisoformat)
    # - datetime.now
    # - encode_dss_signature (https://cryptography.io/en/latest/hazmat/
↪primitives/asymmetric/utils/#cryptography.hazmat.primitives.asymmetric.utils.
↪encode_dss_signature)
    # - ca_public_key.verify and signer_pk.verify (see https://cryptography.io/
↪en/latest/hazmat/primitives/asymmetric/ec/
↪#elliptic-curve-signature-algorithms)

    cert_signature = signer_cert["signature"]
    encoded_cert_signature = encode_dss_signature(cert_signature['r'],␣
↪cert_signature['s'])
    ca_public_key.verify(
        encoded_cert_signature,
        cert_body_to_be_signed,
        ec.ECDSA(hashes.SHA256())
    )

    validity_start = datetime.fromisoformat(signer_cert["body"]["validity␣
↪start"])
    validity_end = datetime.fromisoformat(signer_cert["body"]["validity end"])
    if not (validity_start <= datetime.now() <= validity_end):
        print("Certificate is not valid.")
        return False

    message_signature = signed_message["signature"]
    encoded_message_signature = encode_dss_signature(message_signature['r'],␣
↪message_signature['s'])
    signer_pk.verify(
        encoded_message_signature,
        string_to_bytes(signed_message['message']),
        ec.ECDSA(hashes.SHA256())
    )
    print("Message is valid!")
    return True
```

[ ]:
```
# Verify all signed messages
verify("dstebila", "message1.signed.txt")
```

```
Trying to verify message1.signed.txt
```

```
---------------------------------------------------------------------------
InvalidSignature                          Traceback (most recent call last)
Cell In[2], line 2
      1 # Verify all signed messages
----> 2 verify("dstebila", "message1.signed.txt")
      3 verify("dstebila", "message2.signed.txt")
      4 verify("dstebila", "message3.signed.txt")

Cell In[1], line 41, in verify(ca_identity, signed_message_filename)
     39 cert_signature = signer_cert["signature"]
     40 encoded_cert_signature = encode_dss_signature(cert_signature['r'],␣
 ↪cert_signature['s'])
---> 41 ca_public_key.verify(
     42     encoded_cert_signature,
     43     cert_body_to_be_signed,
     44     ec.ECDSA(hashes.SHA256())
     45 )
     47 # Check certificate validity
     48 validity_start = datetime.fromisoformat(signer_cert["body"]["validity␣
 ↪start"])

InvalidSignature:
```

[3]:
```
verify("dstebila", "message2.signed.txt")
```

```
Trying to verify message2.signed.txt
```

```
---------------------------------------------------------------------------
InvalidSignature                          Traceback (most recent call last)
Cell In[3], line 1
----> 1 verify("dstebila", "message2.signed.txt")

Cell In[1], line 57, in verify(ca_identity, signed_message_filename)
     55 message_signature = signed_message["signature"]
     56 encoded_message_signature = encode_dss_signature(message_signature['r'] ␣
 ↪message_signature['s'])
---> 57 signer_pk.verify(
     58     encoded_message_signature,
     59     string_to_bytes(signed_message['message']),
     60     ec.ECDSA(hashes.SHA256())
     61 )
     62 print("Message is valid!")
```

```
     63 return True

InvalidSignature:
```

[4]:
```
verify("dstebila", "message3.signed.txt")
```

```
Trying to verify message3.signed.txt
```

```
---------------------------------------------------------------------------
InvalidSignature                          Traceback (most recent call last)
Cell In[4], line 1
----> 1 verify("dstebila", "message3.signed.txt")

Cell In[1], line 41, in verify(ca_identity, signed_message_filename)
     39 cert_signature = signer_cert["signature"]
     40 encoded_cert_signature = encode_dss_signature(cert_signature['r'],␣
 ↪cert_signature['s'])
---> 41 ca_public_key.verify(
     42     encoded_cert_signature,
     43     cert_body_to_be_signed,
     44     ec.ECDSA(hashes.SHA256())
     45 )
     47 # Check certificate validity
     48 validity_start = datetime.fromisoformat(signer_cert["body"]["validity␣
 ↪start"])

InvalidSignature:
```

[5]:
```
verify("dstebila", "message4.signed.txt")
```

```
Trying to verify message4.signed.txt
Certificate is not valid.
```

[5]: False

[6]:
```
verify("dstebila", "message5.signed.txt")
```

```
Trying to verify message5.signed.txt
Message is valid!
```

[6]: True

[7]:
```
verify("dstebila", "message6.signed.txt")
```

```
Trying to verify message6.signed.txt
Message is valid!
```

[7]: True