Cert Human Documentation

Release 1.0.7

Jim Olsen

May 30, 2019

Contents

CHAPTER 1

Description

Somebody said something about over-engineering. So I obviously had to chime in.

No, but seriously, I was in the midst of rewriting another project of mine, and I wanted to incorporate a method to get an SSL certificate from a server, show the user the same kind of information as you'd see in a browser, prompt them for validity, then write it to disk for use in further requests using `requests` to a server.

I was unable to find any great / easy ways that incorporated all of these concepts into one neat thing. So I made a thing.

Originally this was based off of the lovely over-engineered solution in get-ca-py by Josh Peak.

I wound up wanting a more offically supported way of patching urllib3 to have access to the certificate attributes in the raw attribute of a `requests.Response` object. So I wrote *Replacement Connect and Response subclasses* for `urllib3.HTTPSConnectionPool`, and a *patcher, unpatcher, and context manager* to enable/disable the new classes.

I also wanted some generalized utility functions to get the certificates, so I wrote some *get certificate functions*.

I then wanted an easier, more *human* way of accessing all of the information in the certificates. And that wound up turning into a whole thing. So *CertStore and CertChainStore classes* were born.

CHAPTER 2

Python Versions Supported

I only focused on writing and testing for the latest versions of 2.7 and 3.7. It might work on other versions, have fun.

CHAPTER 3

Installation

Install into your system wide site-packages:

```
$ pip install cert_human
```

Or install into your pipenv:

```
$ pipenv install cert_human
```

CHAPTER 4

Get the Source Code

Cert Human is actively developed on GitHub, where the code is always available.

You can clone the public repository:

```
$ git clone git://github.com/lifehackjim/cert_human.git
```

Once you have a copy of the source, you can embed it in your own Python package, or install it into your site-packages easily:

```
$ cd cert_human
$ pip install .
```

CHAPTER 5

Table of Contents

5.1 CLI

5.1.1 CLI Examples

5.1.1.1 Getting certs/cert chains

Use requests to get cert

```
bash-3.2$ ./cert_human_cli.py cyborg
Issuer: Common Name: cyborg
Subject: Common Name: cyborg
Subject Alternate Names: cyborg
Fingerprint SHA1: 67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7
Fingerprint SHA256: FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8 E8 38␣
↪14 7F 32 CE 78 DC 26 B0 84 EA
Expired: False, Not Valid Before: 2008-11-15 06:32:10+00:00, Not Valid After: 2028-11-
↪15 02:56:10+00:00
Self Signed: maybe, Self Issued: True
```

Use socket to get cert

```
bash-3.2$ ./cert_human_cli.py cyborg --method socket
Issuer: Common Name: cyborg
Subject: Common Name: cyborg
Subject Alternate Names: cyborg
Fingerprint SHA1: 67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7
Fingerprint SHA256: FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8 E8 38␣
↪14 7F 32 CE 78 DC 26 B0 84 EA
```

(continues on next page)

(continued from previous page)

```
Expired: False, Not Valid Before: 2008-11-15 06:32:10+00:00, Not Valid After: 2028-11-
↪15 02:56:10+00:00
Self Signed: maybe, Self Issued: True
```

Use requests to get cert chain

```
bash-3.2$ ./cert_human_cli.py cyborg --chain

  - CertStore #1
    Issuer: Common Name: cyborg
    Subject: Common Name: cyborg
    Subject Alternate Names: cyborg
    Fingerprint SHA1: 67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7
    Fingerprint SHA256: FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8␣
↪E8 38 14 7F 32 CE 78 DC 26 B0 84 EA
    Expired: False, Not Valid Before: 2008-11-15 06:32:10+00:00, Not Valid After:␣
↪2028-11-15 02:56:10+00:00
    Self Signed: maybe, Self Issued: True
```

Use socket to get cert chain

```
bash-3.2$ ./cert_human_cli.py cyborg --chain --method socket

  - CertStore #1
    Issuer: Common Name: cyborg
    Subject: Common Name: cyborg
    Subject Alternate Names: cyborg
    Fingerprint SHA1: 67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7
    Fingerprint SHA256: FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8␣
↪E8 38 14 7F 32 CE 78 DC 26 B0 84 EA
    Expired: False, Not Valid Before: 2008-11-15 06:32:10+00:00, Not Valid After:␣
↪2028-11-15 02:56:10+00:00
    Self Signed: maybe, Self Issued: True
```

Get a cert and write it to a file

```
bash-3.2$ ./cert_human_cli.py cyborg --write /tmp/cyborg.pem
** Wrote cert in pem format to: '/tmp/cyborg.pem'
```

Get a cert chain and write it to a file

```
bash-3.2$ ./cert_human_cli.py cyborg --chain --write /tmp/cyborg_chain.pem
** Wrote cert chain in pem format to: '/tmp/cyborg_chain.pem'
```

5.1.1.2 Validating certs

Use correct cert to validate host

```
bash-3.2$ ./cert_human_cli.py cyborg --verify /tmp/cyborg.pem
Issuer: Common Name: cyborg
Subject: Common Name: cyborg
Subject Alternate Names: cyborg
Fingerprint SHA1: 67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7
Fingerprint SHA256: FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8 E8 38␣
↪14 7F 32 CE 78 DC 26 B0 84 EA
Expired: False, Not Valid Before: 2008-11-15 06:32:10+00:00, Not Valid After: 2028-11-
↪15 02:56:10+00:00
Self Signed: maybe, Self Issued: True
```

Use wrong cert to validate host

```
bash-3.2$ ./cert_human_cli.py cyborg --verify /tmp/google.pem
SSL Validation Failed:
  HTTPSConnectionPool(host='cyborg', port=443)
  Max retries exceeded with url
  / (Caused by SSLError(SSLError("bad handshake
  Error([('SSL routines', 'tls_process_server_certificate', 'certificate verify failed
↪')],)",),))
```

5.1.1.3 Print cert info

Print public key for cert

```
bash-3.2$ ./cert_human_cli.py cyborg --print_mode key
Public Key Algorithm: rsa, Size: 2048, Exponent: 65537, Value:
    EC 79 B9 78 66 C2 C9 F1 F6 55 E9 F4 BD C5 91 9B 55 F3 A7 55
    FA F8 30 FE B2 BF 4E A8 01 BA A1 64 6D 63 B6 5A 99 7E 60 A3
    C5 E1 E8 E5 A9 F5 13 99 58 C5 E2 83 D0 99 47 08 F2 8A A4 CB
    9A 8A 29 55 BD A3 A6 76 E3 2D 54 17 D2 DA CD C2 6A 2D FF 5E
    C6 BF AC 0A A5 46 E3 6F E5 36 DC A1 1F 81 42 E8 3B 95 5F 90
    4C 85 F3 3B 01 26 2E 6F C5 1B 47 0A B5 7C 88 14 E9 86 BB 3C
    11 55 D1 14 38 6C C2 3E 47 09 F8 F0 AC 8D 63 43 13 18 AA 2C
    3D D8 64 F1 9F 67 9F 89 FB 5A 60 46 7A 6E 9E DA A3 6E 70 D1
    A8 DC 80 99 24 21 91 D9 2D 1E 53 7F 8C EC D4 05 C0 81 4F 14
    3D EB 63 31 40 04 3D C9 9D E7 FD 9F 69 C9 2C AD B8 92 AD FD
    F8 AB 03 88 4C 2B 2E 03 31 37 25 52 3D 2C 4C 2D FE A2 6A 62
    F6 7E 6B 5C 6C 37 AF 2B 10 DC 6A E4 BC 47 CF 2E 40 47 12 1E
    53 8F 30 A9 34 58 77 07 E1 F6 50 C1 0E 37 99 A5

Signature Algorithm: sha256_rsa, Value:
    C1 0A 57 A7 FA 15 4C BD 1D EA B6 5E 74 DD 7E 01 83 BF A0 23
    EA D3 96 66 49 06 5D 4D 02 C7 D2 92 08 A6 01 18 36 D8 66 95
    8C D9 19 77 F3 FA 55 14 DF 1B 23 83 77 F4 0F 69 8B D6 0E DA
    2A 08 9C 34 00 5A 43 56 7D 19 18 8A E1 8B B4 80 3A AA BC 35
    B7 99 77 60 85 83 A6 88 6A A1 AD 9B 12 13 F2 4D BF CA 4F 18
    3D 02 9B DE 40 A5 A6 CB F3 E5 6B F7 28 EF 85 B3 B4 D5 03 F3
    E6 08 D6 59 91 92 6D D0 7D D1 C1 B0 48 51 D2 5D A5 1D F1 26
    6B 36 CD 14 5B 6B 13 C8 0D F6 24 83 C2 AE B4 2E 12 C9 E8 60
```

(continues on next page)

(continued from previous page)

```
    B6 0D BD 1F 34 D5 54 E4 6B EA 4D C1 AF 19 B7 77 5C C1 AD 9C
    A2 2E 04 DD 3E 5C 2E 66 DD 17 41 57 E8 28 EB 4E 89 DE D7 AA
    00 80 7D B6 4C 00 76 6B 7A 00 E3 8C 9F 9B C8 BE 06 B9 14 C3
    D4 A7 78 A0 17 C1 B4 17 6E E2 6E 8D AA 79 69 FE 18 39 8A 19
    FF 9C 36 1A 3C A3 66 EF D0 5F 4D 7C 54 FD 4D A1

Serial Number:
    B7 83 C6 92 09 1E 1A 32 79 D0 7B 5E EE D6 F5 FC 6D E8 CA 01
    50 62 37 91 5E 2A 00 C9 66 82 44 A6
```

Print extensions for cert

```
bash-3.2$ ./cert_human_cli.py cyborg --print_mode extensions
Extensions:
    Extension 1, name=subjectKeyIdentifier,␣
↪value=E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93
    Extension 2, name=subjectAltName, value=DNS:cyborg
```

Print all info for cert

```
bash-3.2$ ./cert_human_cli.py cyborg --print_mode all
Extensions:
    Extension 1, name=subjectKeyIdentifier,␣
↪value=E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93
    Extension 2, name=subjectAltName, value=DNS:cyborg

Public Key Algorithm: rsa, Size: 2048, Exponent: 65537, Value:
    EC 79 B9 78 66 C2 C9 F1 F6 55 E9 F4 BD C5 91 9B 55 F3 A7 55
    FA F8 30 FE B2 BF 4E A8 01 BA A1 64 6D 63 B6 5A 99 7E 60 A3
    C5 E1 E8 E5 A9 F5 13 99 58 C5 E2 83 D0 99 47 08 F2 8A A4 CB
    9A 8A 29 55 BD A3 A6 76 E3 2D 54 17 D2 DA CD C2 6A 2D FF 5E
    C6 BF AC 0A A5 46 E3 6F E5 36 DC A1 1F 81 42 E8 3B 95 5F 90
    4C 85 F3 3B 01 26 2E 6F C5 1B 47 0A B5 7C 88 14 E9 86 BB 3C
    11 55 D1 14 38 6C C2 3E 47 09 F8 F0 AC 8D 63 43 13 18 AA 2C
    3D D8 64 F1 9F 67 9F 89 FB 5A 60 46 7A 6E 9E DA A3 6E 70 D1
    A8 DC 80 99 24 21 91 D9 2D 1E 53 7F 8C EC D4 05 C0 81 4F 14
    3D EB 63 31 40 04 3D C9 9D E7 FD 9F 69 C9 2C AD B8 92 AD FD
    F8 AB 03 88 4C 2B 2E 03 31 37 25 52 3D 2C 4C 2D FE A2 6A 62
    F6 7E 6B 5C 6C 37 AF 2B 10 DC 6A E4 BC 47 CF 2E 40 47 12 1E
    53 8F 30 A9 34 58 77 07 E1 F6 50 C1 0E 37 99 A5

Signature Algorithm: sha256_rsa, Value:
    C1 0A 57 A7 FA 15 4C BD 1D EA B6 5E 74 DD 7E 01 83 BF A0 23
    EA D3 96 66 49 06 5D 4D 02 C7 D2 92 08 A6 01 18 36 D8 66 95
    8C D9 19 77 F3 FA 55 14 DF 1B 23 83 77 F4 0F 69 8B D6 0E DA
    2A 08 9C 34 00 5A 43 56 7D 19 18 8A E1 8B B4 80 3A AA BC 35
    B7 99 77 60 85 83 A6 88 6A A1 AD 9B 12 13 F2 4D BF CA 4F 18
    3D 02 9B DE 40 A5 A6 CB F3 E5 6B F7 28 EF 85 B3 B4 D5 03 F3
    E6 08 D6 59 91 92 6D D0 7D D1 C1 B0 48 51 D2 5D A5 1D F1 26
    6B 36 CD 14 5B 6B 13 C8 0D F6 24 83 C2 AE B4 2E 12 C9 E8 60
    B6 0D BD 1F 34 D5 54 E4 6B EA 4D C1 AF 19 B7 77 5C C1 AD 9C
    A2 2E 04 DD 3E 5C 2E 66 DD 17 41 57 E8 28 EB 4E 89 DE D7 AA
```

(continues on next page)

(continued from previous page)

```
    00 80 7D B6 4C 00 76 6B 7A 00 E3 8C 9F 9B C8 BE 06 B9 14 C3
    D4 A7 78 A0 17 C1 B4 17 6E E2 6E 8D AA 79 69 FE 18 39 8A 19
    FF 9C 36 1A 3C A3 66 EF D0 5F 4D 7C 54 FD 4D A1

Serial Number:
    B7 83 C6 92 09 1E 1A 32 79 D0 7B 5E EE D6 F5 FC 6D E8 CA 01
    50 62 37 91 5E 2A 00 C9 66 82 44 A6

Issuer: Common Name: cyborg
Subject: Common Name: cyborg
Subject Alternate Names: cyborg
Fingerprint SHA1: 67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7
Fingerprint SHA256: FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8 E8 38␣
↪14 7F 32 CE 78 DC 26 B0 84 EA
Expired: False, Not Valid Before: 2008-11-15 06:32:10+00:00, Not Valid After: 2028-11-
↪15 02:56:10+00:00
Self Signed: maybe, Self Issued: True
```

5.1.2 CLI Reference

Command line interface to request a URL and get the server cert or cert chain.

cert_human_cli.**cli**(*argv*)

Parse arguments.

Parameters **argv** (list) – sys.argv or manual list of args to parse.

Returns (argparse.Namespace)

cert_human_cli.**main**(*cli_args*)

Process arguments and run the workflows.

Parameters **cli_args** (argparse.Namespace) – Parsed args from sys.argv or list.

5.1.2.1 Help

```
bash-3.2$ ./cert_human_cli.py --help
usage: cert_human_cli.py [-h] [--port PORT] [--method {requests,socket}]
                         [--chain] [--print_mode {info,key,extensions,all}]
                         [--write WRITE] [--overwrite] [--verify VERIFY]
                         HOST

Command line interface to request a URL and get the server cert or cert chain.

positional arguments:
  HOST                  Host to get cert or cert chain from

optional arguments:
  -h, --help            show this help message and exit
  --port PORT           Port on host to connect to (default: 443)
  --method {requests,socket}
                        Use requests.get a SSL socket to get cert or cert
                        chain. (default: requests)
  --chain               Print/write the cert chain instead of the cert.
                        (default: False)
```

(continues on next page)

(continued from previous page)

```
  --print_mode {info,key,extensions,all}
                        When no --write specified, print this type of
                        information for the cert. (default: info)
  --write WRITE         File to write cert/cert chain to (default: )
  --overwrite           When writing to --write and file exists, overwrite.
                        (default: False)
  --verify VERIFY       PEM file to verify host, empty will disable verify,
                        for --method requests. (default: )
```

5.2 API

5.2.1 High level API Examples

Different examples of working with certs and cert chains using *cert_human.CertStore* and *cert_human.CertChainStore*.

5.2.1.1 Getting certs

Using sockets

```
>>> store = cert_human.CertStore.from_socket(host="cyborg")
>>> chain_store = cert_human.CertChainStore.from_socket(host="cyborg")
```

Using requests

```
>>> store = cert_human.CertStore.from_request(host="cyborg")
>>> chain_store = cert_human.CertChainStore.from_request(host="cyborg")
```

Using a requests response object

```
>>> cert_human.enable_urllib3_patch()
>>> response = requests.get("https://cyborg")
>>> store = cert_human.CertStore.from_response(response=response)
>>> chain_store = cert_human.CertChainStore.from_response(response=response)
```

5.2.1.2 Writing PEM certs to disk

```
>>> store_path = store.to_disk("/tmp/certs/cyborg.pem", overwrite=True)
>>> store_path.is_file()
True

>>> chain_path = chain_store.to_disk("/tmp/certs/cyborg_chain.pem", overwrite=True)
>>> chain_path.is_file()
True
```

5.2.1.3 Creating CertStore from various types

```
>>> # auto CertStore from a requests.Response object
>>> r = cert_human.get_response("cyborg")
>>> auto_response = cert_human.CertStore.from_auto(r)

>>> # auto CertStore from a asn1crypt.X509.Certificate object
>>> auto_asn1 = cert_human.CertStore.from_auto(auto_response.asn1)

>>> # auto CertStore from a PEM string
>>> auto_pem = cert_human.CertStore.from_auto(auto_response.pem)

>>> # auto CertStore from a OpenSSL.crypto.X509 object
>>> auto_x509 = cert_human.CertStore.from_auto(auto_response.x509)

>>> # CertStore from a pem file on disk
>>> from_path = cert_human.CertStore.from_path("tests/certs/cert.pem")
```

5.2.1.4 CertStore: Attributes

PEM format

```
>>> print(store.pem)
-----BEGIN CERTIFICATE-----
MIIC8TCCAdmgAwIBAgIhALeDxpIJHhoyedB7Xu7W9fxt6MoBUGI3kV4qAMlmgkSm
MA0GCSqGSIb3DQEBCwUAMBExDzANBgNVBAMMBmN5Ym9yZzAeFw0wODExMTUwNjMy
MTBaFw0yODExMTUwMjU2MTBaMBExDzANBgNVBAMMBmN5Ym9yZzCCASIwDQYJKoZI
hvcNAQEBBQADggEPADCCAQoCggEBAOx5uXhmwsnx9lXp9L3FkZtV86dV+vgw/rK/
TqgBuqFkbWO2Wpl+YKPF4ejlqfUTmVjF4oPQmUcI8oqky5qKKVW9o6Z24y1UF9La
zcJqLf9exr+sCqVG42/lNtyhH4FC6DuVX5BMhfM7ASYub8UbRwq1fIgU6Ya7PBFV
0RQ4bMI+Rwn48KyNY0MTGKosPdhk8Z9nn4n7WmBGem6e2qNucNGo3ICZJCGR2S0e
U3+M7NQFwIFPFD3rYzFABD3Jnef9n2nJLK24kq39+KsDiEwrLgMxNyVSPSxMLf6i
amL2fmtcbDevKxDcauS8R88uQEcSHlOPMKk0WHcH4fZQwQ43maUCAwEAAaM0MDIw
HQYDVR0OBBYEFOco2mIkLtDOWE1gNHeFHw9/xvKTMBEGA1UdEQQKMAiCBmN5Ym9y
ZzANBgkqhkiG9w0BAQsFAAOCAQEAwQpXp/oVTL0d6rZedN1+AYO/oCPq05ZmSQZd
TQLH0pIIpgEYNthmlYzZGXfz+lUU3xsjg3f0D2mL1g7aKgicNABaQ1Z9GRiK4Yu0
gDqqvDW3mXdghYOmiGqhrZsSE/JNv8pPGD0Cm95ApabL8+Vr9yjvhbO01QPz5gjW
WZGSbdB90cGwSFHSXaUd8SZrNs0UW2sTyA32JIPCrrQuEsnoYLYNvR801VTka+pN
wa8Zt3dcwa2coi4E3T5cLmbdF0FX6CjrTone16oAgH22TAB2a3oA44yfm8i+BrkU
w9SneKAXwbQXbuJujap5af4YOYoZ/5w2GjyjZu/QX018VP1NoQ==
-----END CERTIFICATE-----
```

Issuer

```
>>> store.issuer
{'common_name': 'cyborg'}
>>> store.issuer_str
'Common Name: cyborg'
```

Subject

```
>>> store.subject
{'common_name': 'cyborg'}
>>> store.subject_str
'Common Name: cyborg'
>>> store.subject_alt_names
['cyborg']
```

Fingerprints

```
>>> store.fingerprint_sha1
'67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7'
>>> store.fingerprint_sha256
'FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8 E8 38 14 7F 32 CE 78 DC␣
↪26 B0 84 EA'
```

Public Key

```
>>> store.public_key

↪'EC79B97866C2C9F1F655E9F4BDC5919B55F3A755FAF830FEB2BF4EA801BAA1646D63B65A997E60A3C5E1E8E5A9F513995
↪'
>>> print(store.public_key_str)
EC 79 B9 78 66 C2 C9 F1 F6 55 E9 F4 BD C5 91 9B 55 F3 A7 55
FA F8 30 FE B2 BF 4E A8 01 BA A1 64 6D 63 B6 5A 99 7E 60 A3
C5 E1 E8 E5 A9 F5 13 99 58 C5 E2 83 D0 99 47 08 F2 8A A4 CB
9A 8A 29 55 BD A3 A6 76 E3 2D 54 17 D2 DA CD C2 6A 2D FF 5E
C6 BF AC 0A A5 46 E3 6F E5 36 DC A1 1F 81 42 E8 3B 95 5F 90
4C 85 F3 3B 01 26 2E 6F C5 1B 47 0A B5 7C 88 14 E9 86 BB 3C
11 55 D1 14 38 6C C2 3E 47 09 F8 F0 AC 8D 63 43 13 18 AA 2C
3D D8 64 F1 9F 67 9F 89 FB 5A 60 46 7A 6E 9E DA A3 6E 70 D1
A8 DC 80 99 24 21 91 D9 2D 1E 53 7F 8C EC D4 05 C0 81 4F 14
3D EB 63 31 40 04 3D C9 9D E7 FD 9F 69 C9 2C AD B8 92 AD FD
F8 AB 03 88 4C 2B 2E 03 31 37 25 52 3D 2C 4C 2D FE A2 6A 62
F6 7E 6B 5C 6C 37 AF 2B 10 DC 6A E4 BC 47 CF 2E 40 47 12 1E
53 8F 30 A9 34 58 77 07 E1 F6 50 C1 0E 37 99 A5
>>> store.public_key_algorithm
'rsa'
>>> store.public_key_exponent
65537
>>> store.public_key_parameters
>>> store.public_key_size
2048
```

Signature

```
>>> store.signature

↪'C10A57A7FA154CBD1DEAB65E74DD7E0183BFA023EAD3966649065D4D02C7D29208A6011836D866958CD91977F3FA5514D
↪'
```

(continues on next page)

(continued from previous page)

```
>>> print(store.signature_str)
C1 0A 57 A7 FA 15 4C BD 1D EA B6 5E 74 DD 7E 01 83 BF A0 23
EA D3 96 66 49 06 5D 4D 02 C7 D2 92 08 A6 01 18 36 D8 66 95
8C D9 19 77 F3 FA 55 14 DF 1B 23 83 77 F4 0F 69 8B D6 0E DA
2A 08 9C 34 00 5A 43 56 7D 19 18 8A E1 8B B4 80 3A AA BC 35
B7 99 77 60 85 83 A6 88 6A A1 AD 9B 12 13 F2 4D BF CA 4F 18
3D 02 9B DE 40 A5 A6 CB F3 E5 6B F7 28 EF 85 B3 B4 D5 03 F3
E6 08 D6 59 91 92 6D D0 7D D1 C1 B0 48 51 D2 5D A5 1D F1 26
6B 36 CD 14 5B 6B 13 C8 0D F6 24 83 C2 AE B4 2E 12 C9 E8 60
B6 0D BD 1F 34 D5 54 E4 6B EA 4D C1 AF 19 B7 77 5C C1 AD 9C
A2 2E 04 DD 3E 5C 2E 66 DD 17 41 57 E8 28 EB 4E 89 DE D7 AA
00 80 7D B6 4C 00 76 6B 7A 00 E3 8C 9F 9B C8 BE 06 B9 14 C3
D4 A7 78 A0 17 C1 B4 17 6E E2 6E 8D AA 79 69 FE 18 39 8A 19
FF 9C 36 1A 3C A3 66 EF D0 5F 4D 7C 54 FD 4D A1
>>> store.signature_algorithm
'sha256_rsa'
```

Serial Number

```
>>> store.serial_number
'B783C692091E1A3279D07B5EEED6F5FC6DE8CA01506237915E2A00C9668244A6'
>>> store.serial_number_str
'B7 83 C6 92 09 1E 1A 32 79 D0 7B 5E EE D6 F5 FC 6D E8 CA 01\n50 62 37 91 5E 2A 00 C9␣
↪66 82 44 A6'
```

Validity

```
>>> store.is_expired
False
>>> store.is_self_issued
True
>>> store.is_self_signed
'maybe'
>>> store.not_valid_before
datetime.datetime(2008, 11, 15, 6, 32, 10, tzinfo=datetime.timezone.utc)
>>> store.not_valid_after
datetime.datetime(2028, 11, 15, 2, 56, 10, tzinfo=datetime.timezone.utc)
>>> store.not_valid_before_str
'2008-11-15 06:32:10+00:00'
>>> store.not_valid_after_str
'2028-11-15 02:56:10+00:00'
```

Extensions

```
>>> store.extensions
{'subjectKeyIdentifier': 'E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93
↪', 'subjectAltName': 'DNS:cyborg'}
>>> print(store.extensions_str)
Extension 1, name=subjectKeyIdentifier,␣
↪value=E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93
Extension 2, name=subjectAltName, value=DNS:cyborg
```

Info in string format

Basic cert info:

```
>>> print(store)
CertStore:
    Issuer: Common Name: cyborg
    Subject: Common Name: cyborg
    Subject Alternate Names: cyborg
    Fingerprint SHA1: 67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7
    Fingerprint SHA256: FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8␣
↪E8 38 14 7F 32 CE 78 DC 26 B0 84 EA
    Expired: False, Not Valid Before: 2008-11-15 06:32:10+00:00, Not Valid After:␣
↪2028-11-15 02:56:10+00:00
    Self Signed: maybe, Self Issued: True

>>> print(store.dump_str_info)  # same as print(store)
Issuer: Common Name: cyborg
Subject: Common Name: cyborg
Subject Alternate Names: cyborg
Fingerprint SHA1: 67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7
Fingerprint SHA256: FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8 E8 38␣
↪14 7F 32 CE 78 DC 26 B0 84 EA
Expired: False, Not Valid Before: 2008-11-15 06:32:10+00:00, Not Valid After: 2028-11-
↪15 02:56:10+00:00
Self Signed: maybe, Self Issued: True
```

Public key info:

```
>>> print(store.dump_str_key)
Public Key Algorithm: rsa, Size: 2048, Exponent: 65537, Value:
    EC 79 B9 78 66 C2 C9 F1 F6 55 E9 F4 BD C5 91 9B 55 F3 A7 55
    FA F8 30 FE B2 BF 4E A8 01 BA A1 64 6D 63 B6 5A 99 7E 60 A3
    C5 E1 E8 E5 A9 F5 13 99 58 C5 E2 83 D0 99 47 08 F2 8A A4 CB
    9A 8A 29 55 BD A3 A6 76 E3 2D 54 17 D2 DA CD C2 6A 2D FF 5E
    C6 BF AC 0A A5 46 E3 6F E5 36 DC A1 1F 81 42 E8 3B 95 5F 90
    4C 85 F3 3B 01 26 2E 6F C5 1B 47 0A B5 7C 88 14 E9 86 BB 3C
    11 55 D1 14 38 6C C2 3E 47 09 F8 F0 AC 8D 63 43 13 18 AA 2C
    3D D8 64 F1 9F 67 9F 89 FB 5A 60 46 7A 6E 9E DA A3 6E 70 D1
    A8 DC 80 99 24 21 91 D9 2D 1E 53 7F 8C EC D4 05 C0 81 4F 14
    3D EB 63 31 40 04 3D C9 9D E7 FD 9F 69 C9 2C AD B8 92 AD FD
    F8 AB 03 88 4C 2B 2E 03 31 37 25 52 3D 2C 4C 2D FE A2 6A 62
    F6 7E 6B 5C 6C 37 AF 2B 10 DC 6A E4 BC 47 CF 2E 40 47 12 1E
    53 8F 30 A9 34 58 77 07 E1 F6 50 C1 0E 37 99 A5

Signature Algorithm: sha256_rsa, Value:
    C1 0A 57 A7 FA 15 4C BD 1D EA B6 5E 74 DD 7E 01 83 BF A0 23
    EA D3 96 66 49 06 5D 4D 02 C7 D2 92 08 A6 01 18 36 D8 66 95
    8C D9 19 77 F3 FA 55 14 DF 1B 23 83 77 F4 0F 69 8B D6 0E DA
    2A 08 9C 34 00 5A 43 56 7D 19 18 8A E1 8B B4 80 3A AA BC 35
    B7 99 77 60 85 83 A6 88 6A A1 AD 9B 12 13 F2 4D BF CA 4F 18
    3D 02 9B DE 40 A5 A6 CB F3 E5 6B F7 28 EF 85 B3 B4 D5 03 F3
    E6 08 D6 59 91 92 6D D0 7D D1 C1 B0 48 51 D2 5D A5 1D F1 26
    6B 36 CD 14 5B 6B 13 C8 0D F6 24 83 C2 AE B4 2E 12 C9 E8 60
    B6 0D BD 1F 34 D5 54 E4 6B EA 4D C1 AF 19 B7 77 5C C1 AD 9C
    A2 2E 04 DD 3E 5C 2E 66 DD 17 41 57 E8 28 EB 4E 89 DE D7 AA
    00 80 7D B6 4C 00 76 6B 7A 00 E3 8C 9F 9B C8 BE 06 B9 14 C3
```

(continues on next page)

(continued from previous page)

```
    D4 A7 78 A0 17 C1 B4 17 6E E2 6E 8D AA 79 69 FE 18 39 8A 19
    FF 9C 36 1A 3C A3 66 EF D0 5F 4D 7C 54 FD 4D A1

Serial Number:
    B7 83 C6 92 09 1E 1A 32 79 D0 7B 5E EE D6 F5 FC 6D E8 CA 01
    50 62 37 91 5E 2A 00 C9 66 82 44 A6
```

Extensions info:

```
>>> print(store.dump_str_exts)
Extensions:
    Extension 1, name=subjectKeyIdentifier,␣
↪value=E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93
    Extension 2, name=subjectAltName, value=DNS:cyborg
```

All info blocks as one:

```
>>> print(store.dump_str)
Extensions:
    Extension 1, name=subjectKeyIdentifier,␣
↪value=E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93
    Extension 2, name=subjectAltName, value=DNS:cyborg

Public Key Algorithm: rsa, Size: 2048, Exponent: 65537, Value:
    EC 79 B9 78 66 C2 C9 F1 F6 55 E9 F4 BD C5 91 9B 55 F3 A7 55
    FA F8 30 FE B2 BF 4E A8 01 BA A1 64 6D 63 B6 5A 99 7E 60 A3
    C5 E1 E8 E5 A9 F5 13 99 58 C5 E2 83 D0 99 47 08 F2 8A A4 CB
    9A 8A 29 55 BD A3 A6 76 E3 2D 54 17 D2 DA CD C2 6A 2D FF 5E
    C6 BF AC 0A A5 46 E3 6F E5 36 DC A1 1F 81 42 E8 3B 95 5F 90
    4C 85 F3 3B 01 26 2E 6F C5 1B 47 0A B5 7C 88 14 E9 86 BB 3C
    11 55 D1 14 38 6C C2 3E 47 09 F8 F0 AC 8D 63 43 13 18 AA 2C
    3D D8 64 F1 9F 67 9F 89 FB 5A 60 46 7A 6E 9E DA A3 6E 70 D1
    A8 DC 80 99 24 21 91 D9 2D 1E 53 7F 8C EC D4 05 C0 81 4F 14
    3D EB 63 31 40 04 3D C9 9D E7 FD 9F 69 C9 2C AD B8 92 AD FD
    F8 AB 03 88 4C 2B 2E 03 31 37 25 52 3D 2C 4C 2D FE A2 6A 62
    F6 7E 6B 5C 6C 37 AF 2B 10 DC 6A E4 BC 47 CF 2E 40 47 12 1E
    53 8F 30 A9 34 58 77 07 E1 F6 50 C1 0E 37 99 A5

Signature Algorithm: sha256_rsa, Value:
    C1 0A 57 A7 FA 15 4C BD 1D EA B6 5E 74 DD 7E 01 83 BF A0 23
    EA D3 96 66 49 06 5D 4D 02 C7 D2 92 08 A6 01 18 36 D8 66 95
    8C D9 19 77 F3 FA 55 14 DF 1B 23 83 77 F4 0F 69 8B D6 0E DA
    2A 08 9C 34 00 5A 43 56 7D 19 18 8A E1 8B B4 80 3A AA BC 35
    B7 99 77 60 85 83 A6 88 6A A1 AD 9B 12 13 F2 4D BF CA 4F 18
    3D 02 9B DE 40 A5 A6 CB F3 E5 6B F7 28 EF 85 B3 B4 D5 03 F3
    E6 08 D6 59 91 92 6D D0 7D D1 C1 B0 48 51 D2 5D A5 1D F1 26
    6B 36 CD 14 5B 6B 13 C8 0D F6 24 83 C2 AE B4 2E 12 C9 E8 60
    B6 0D BD 1F 34 D5 54 E4 6B EA 4D C1 AF 19 B7 77 5C C1 AD 9C
    A2 2E 04 DD 3E 5C 2E 66 DD 17 41 57 E8 28 EB 4E 89 DE D7 AA
    00 80 7D B6 4C 00 76 6B 7A 00 E3 8C 9F 9B C8 BE 06 B9 14 C3
    D4 A7 78 A0 17 C1 B4 17 6E E2 6E 8D AA 79 69 FE 18 39 8A 19
    FF 9C 36 1A 3C A3 66 EF D0 5F 4D 7C 54 FD 4D A1

Serial Number:
    B7 83 C6 92 09 1E 1A 32 79 D0 7B 5E EE D6 F5 FC 6D E8 CA 01
    50 62 37 91 5E 2A 00 C9 66 82 44 A6
```

(continues on next page)

(continued from previous page)

```
Issuer: Common Name: cyborg
Subject: Common Name: cyborg
Subject Alternate Names: cyborg
Fingerprint SHA1: 67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7
Fingerprint SHA256: FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8 E8 38␣
↪14 7F 32 CE 78 DC 26 B0 84 EA
Expired: False, Not Valid Before: 2008-11-15 06:32:10+00:00, Not Valid After: 2028-11-
↪15 02:56:10+00:00
Self Signed: maybe, Self Issued: True
```

5.2.1.5 Exporting all attributes in a CertStore

Get all the JSON friendly attributes as a JSON string:

```
>>> print(store.dump_json)
{
  "issuer": {
    "common_name": "cyborg"
  },
  "issuer_str": "Common Name: cyborg",
  "subject": {
    "common_name": "cyborg"
  },
  "subject_str": "Common Name: cyborg",
  "subject_alt_names": [
    "cyborg"
  ],
  "subject_alt_names_str": "cyborg",
  "fingerprint_sha1": "67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF B7",
  "fingerprint_sha256": "FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 F7 A8␣
↪E8 38 14 7F 32 CE 78 DC 26 B0 84 EA",
  "public_key":
↪"EC79B97866C2C9F1F655E9F4BDC5919B55F3A755FAF830FEB2BF4EA801BAA1646D63B65A997E60A3C5E1E8E5A9F5139958
↪",
  "public_key_str": "EC 79 B9 78 66 C2 C9 F1 F6 55 E9 F4 BD C5 91 9B 55 F3 A7 55\nFA␣
↪F8 30 FE B2 BF 4E A8 01 BA A1 64 6D 63 B6 5A 99 7E 60 A3\nC5 E1 E8 E5 A9 F5 13 99␣
↪58 C5 E2 83 D0 99 47 08 F2 8A A4 CB\n9A 8A 29 55 BD A3 A6 76 E3 2D 54 17 D2 DA CD␣
↪C2 6A 2D FF 5E\nC6 BF AC 0A A5 46 E3 6F E5 36 DC A1 1F 81 42 E8 3B 95 5F 90\n4C 85␣
↪F3 3B 01 26 2E 6F C5 1B 47 0A B5 7C 88 14 E9 86 BB 3C\n11 55 D1 14 38 6C C2 3E 47␣
↪09 F8 F0 AC 8D 63 43 13 18 AA 2C\n3D D8 64 F1 9F 67 9F 89 FB 5A 60 46 7A 6E 9E DA␣
↪A3 6E 70 D1\nA8 DC 80 99 24 21 91 D9 2D 1E 53 7F 8C EC D4 05 C0 81 4F 14\n3D EB 63␣
↪31 40 04 3D C9 9D E7 FD 9F 69 C9 2C AD B8 92 AD FD\nF8 AB 03 88 4C 2B 2E 03 31 37␣
↪25 52 3D 2C 4C 2D FE A2 6A 62\nF6 7E 6B 5C 6C 37 AF 2B 10 DC 6A E4 BC 47 CF 2E 40␣
↪47 12 1E\n53 8F 30 A9 34 58 77 07 E1 F6 50 C1 0E 37 99 A5",
  "public_key_parameters": null,
  "public_key_algorithm": "rsa",
  "public_key_size": 2048,
  "public_key_exponent": 65537,
  "signature":
↪"C10A57A7FA154CBD1DEAB65E74DD7E0183BFA023EAD3966649065D4D02C7D29208A6011836D866958CD91977F3FA5514DE
↪",
  "signature_str": "C1 0A 57 A7 FA 15 4C BD 1D EA B6 5E 74 DD 7E 01 83 BF A0 23\nEA␣
↪D3 96 66 49 06 5D 4D 02 C7 D2 92 08 A6 01 18 36 D8 66 95\n8C D9 19 77 F3 FA 55 14␣
↪DF 1B 23 83 77 F4 0F 69 8B D6 0E DA\n2A 08 9C 34 00 5A 43 56 7D 19 18 8A E1 8B B4␣
↪80 3A AA BC 35\nB7 99 77 60 85 83 A6 88 6A A1 AD 9B 12 13 F2 4D BF CA 4F 18\n3D 02␣
↪9B DE 40 A5 A6 CB F3 E5 6B F7 28 EF 85 B3 B4 D5 03 F3\nE6 08 D6 59 91 92 6D D0 7D␣
↪D1 C1 B0 48 51 D2 5D A5 1D F1 26\n6B 36 CD 14 5B 6B 13 C8 0D F6 24 83 C2 AE B4 2E␣
↪12 C9 E8 60\nB6 0D BD 1F 34 D5 54 E4 6B EA 4D C1 AF 19 B7 77 5C C1 AD 9C\nA2 2E 04␣
↪DD 3E 5C 2E 66 DD 17 41 57 E8 28 EB 4E 89 DE D7 AA\n00 80 7D E6 4C 80 76 6B 7A 00␣
↪E3 8C 9F 9B C8 BE 06 B9 14 C3\nD4 A7 78 A0 17 C1 B4 17 6E E2 6E 8D AA 79 69 FE 18␣
↪39 8A 19\nFF 9C 36 1A 3C A3 66 EF D0 5F 4D 7C 54 FD 4D A1",
```

(continues on next page)

(continued from previous page)

```
  "signature_algorithm": "sha256_rsa",
  "x509_version": "v3",
  "serial_number": "B783C692091E1A3279D07B5EEED6F5FC6DE8CA01506237915E2A00C9668244A6",
  "serial_number_str": "B7 83 C6 92 09 1E 1A 32 79 D0 7B 5E EE D6 F5 FC 6D E8 CA␣
↪01\n50 62 37 91 5E 2A 00 C9 66 82 44 A6",
  "is_expired": false,
  "is_self_signed": "maybe",
  "is_self_issued": true,
  "not_valid_before_str": "2008-11-15 06:32:10+00:00",
  "not_valid_after_str": "2028-11-15 02:56:10+00:00",
  "extensions": {
    "subjectKeyIdentifier":
↪"E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93",
    "subjectAltName": "DNS:cyborg"
  },
  "extensions_str": "Extension 1, name=subjectKeyIdentifier,␣
↪value=E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93\nExtension 2,␣
↪name=subjectAltName, value=DNS:cyborg"
}
```

Get all the attributes as a dict:

```
>>> pprint.pprint(store.dump)
{'extensions': {'subjectAltName': 'DNS:cyborg',
                'subjectKeyIdentifier':
↪'E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93'},
 'extensions_str': 'Extension 1, name=subjectKeyIdentifier, '

↪'value=E7:28:DA:62:24:2E:D0:CE:58:4D:60:34:77:85:1F:0F:7F:C6:F2:93\n'
                  'Extension 2, name=subjectAltName, value=DNS:cyborg',
 'fingerprint_sha1': '67 FD F1 7A 02 26 C7 AB 77 AD CD CB 63 76 19 AD 83 0C BF '
                     'B7',
 'fingerprint_sha256': 'FA BF 9D EC CF 6C 3F 8A 08 89 29 04 5E 9E B5 A8 28 A9 '
                       'F7 A8 E8 38 14 7F 32 CE 78 DC 26 B0 84 EA',
 'is_expired': False,
 'is_self_issued': True,
 'is_self_signed': 'maybe',
 'issuer': {'common_name': 'cyborg'},
 'issuer_str': 'Common Name: cyborg',
 'not_valid_after': datetime.datetime(2028, 11, 15, 2, 56, 10, tzinfo=datetime.
↪timezone.utc),
 'not_valid_after_str': '2028-11-15 02:56:10+00:00',
 'not_valid_before': datetime.datetime(2008, 11, 15, 6, 32, 10, tzinfo=datetime.
↪timezone.utc),
 'not_valid_before_str': '2008-11-15 06:32:10+00:00',
 'public_key':
↪'EC79B97866C2C9F1F655E9F4BDC5919B55F3A755FAF830FEB2BF4EA801BAA1646D63B65A997E60A3C5E1E8E5A9F513995
↪',
 'public_key_algorithm': 'rsa',
 'public_key_exponent': 65537,
 'public_key_parameters': None,
 'public_key_size': 2048,
 'public_key_str': 'EC 79 B9 78 66 C2 C9 F1 F6 55 E9 F4 BD C5 91 9B 55 F3 A7 '
                   '55\n'
                   'FA F8 30 FE B2 BF 4E A8 01 BA A1 64 6D 63 B6 5A 99 7E 60 '
                   'A3\n'
                   'C5 E1 E8 E5 A9 F5 13 99 58 C5 E2 83 D0 99 47 08 F2 8A A4 '
```

(continues on next page)

(continued from previous page)

```
                  'CB\n'
                  '9A 8A 29 55 BD A3 A6 76 E3 2D 54 17 D2 DA CD C2 6A 2D FF '
                  '5E\n'
                  'C6 BF AC 0A A5 46 E3 6F E5 36 DC A1 1F 81 42 E8 3B 95 5F '
                  '90\n'
                  '4C 85 F3 3B 01 26 2E 6F C5 1B 47 0A B5 7C 88 14 E9 86 BB '
                  '3C\n'
                  '11 55 D1 14 38 6C C2 3E 47 09 F8 F0 AC 8D 63 43 13 18 AA '
                  '2C\n'
                  '3D D8 64 F1 9F 67 9F 89 FB 5A 60 46 7A 6E 9E DA A3 6E 70 '
                  'D1\n'
                  'A8 DC 80 99 24 21 91 D9 2D 1E 53 7F 8C EC D4 05 C0 81 4F '
                  '14\n'
                  '3D EB 63 31 40 04 3D C9 9D E7 FD 9F 69 C9 2C AD B8 92 AD '
                  'FD\n'
                  'F8 AB 03 88 4C 2B 2E 03 31 37 25 52 3D 2C 4C 2D FE A2 6A '
                  '62\n'
                  'F6 7E 6B 5C 6C 37 AF 2B 10 DC 6A E4 BC 47 CF 2E 40 47 12 '
                  '1E\n'
                  '53 8F 30 A9 34 58 77 07 E1 F6 50 C1 0E 37 99 A5',
 'serial_number': 'B783C692091E1A3279D07B5EEED6F5FC6DE8CA01506237915E2A00C9668244A6',
 'serial_number_str': 'B7 83 C6 92 09 1E 1A 32 79 D0 7B 5E EE D6 F5 FC 6D E8 '
                      'CA 01\n'
                      '50 62 37 91 5E 2A 00 C9 66 82 44 A6',
 'signature':
↪'C10A57A7FA154CBD1DEAB65E74DD7E0183BFA023EAD3966649065D4D02C7D29208A6011836D866958CD91977F3FA5514DE
↪',
 'signature_algorithm': 'sha256_rsa',
 'signature_str': 'C1 0A 57 A7 FA 15 4C BD 1D EA B6 5E 74 DD 7E 01 83 BF A0 '
                  '23\n'
                  'EA D3 96 66 49 06 5D 4D 02 C7 D2 92 08 A6 01 18 36 D8 66 '
                  '95\n'
                  '8C D9 19 77 F3 FA 55 14 DF 1B 23 83 77 F4 0F 69 8B D6 0E '
                  'DA\n'
                  '2A 08 9C 34 00 5A 43 56 7D 19 18 8A E1 8B B4 80 3A AA BC '
                  '35\n'
                  'B7 99 77 60 85 83 A6 88 6A A1 AD 9B 12 13 F2 4D BF CA 4F '
                  '18\n'
                  '3D 02 9B DE 40 A5 A6 CB F3 E5 6B F7 28 EF 85 B3 B4 D5 03 '
                  'F3\n'
                  'E6 08 D6 59 91 92 6D D0 7D D1 C1 B0 48 51 D2 5D A5 1D F1 '
                  '26\n'
                  '6B 36 CD 14 5B 6B 13 C8 0D F6 24 83 C2 AE B4 2E 12 C9 E8 '
                  '60\n'
                  'B6 0D BD 1F 34 D5 54 E4 6B EA 4D C1 AF 19 B7 77 5C C1 AD '
                  '9C\n'
                  'A2 2E 04 DD 3E 5C 2E 66 DD 17 41 57 E8 28 EB 4E 89 DE D7 '
                  'AA\n'
                  '00 80 7D B6 4C 00 76 6B 7A 00 E3 8C 9F 9B C8 BE 06 B9 14 '
                  'C3\n'
                  'D4 A7 78 A0 17 C1 B4 17 6E E2 6E 8D AA 79 69 FE 18 39 8A '
                  '19\n'
                  'FF 9C 36 1A 3C A3 66 EF D0 5F 4D 7C 54 FD 4D A1',
 'subject': {'common_name': 'cyborg'},
 'subject_alt_names': ['cyborg'],
 'subject_alt_names_str': 'cyborg',
 'subject_str': 'Common Name: cyborg',
```

(continues on next page)

(continued from previous page)

```
 'x509_version': 'v3'}
```

5.2.1.6 CertChainStore: Attributes

```
>>> print(chain_store.pem)  # concatted string of pem from each cert in chain store
>>> chain_store.asn1  # list of each cert in cert chain in asn1 format
>>> chain_store.der  # list of each cert in cert chain in der format
>>> chain_store.dump_str  # concatted and indexed output of dump_str on each cert in␣
↪chain store
>>> chain_store.dump_str_exts  # concatted and indexed output of dump_str_exts on␣
↪each cert in chain store
>>> chain_store.dump_str_info  # concatted and indexed output of dump_str_info on␣
↪each cert in chain store
>>> chain_store.dump_str_key  # concatted and indexed output of dump_str_key on each␣
↪cert in chain store
>>> chain_store.dump_json  # json string with list of dicts from dump_json on each␣
↪cert in chain store
>>> chain_store.dump  # list of dict from dump on each cert in chain store
```

5.2.2 Low level API Examples

Different examples of getting certs and cert chains manually using cert_human.

5.2.2.1 Using cert_human.get_response to get cert and cert chain

cert_human.get_response does a number of things:

- It will automatically construct a url based on `host` and `port` arguments as follows:
  - if no `://` in host, prepend host with the default scheme `https://`
  - if no `:int` in host, append host with the default port `:443`
- You can supply a value to the `host` argument a number of different ways:
  - `google.com`
  - `google.com:443`
  - `https://google.com:443`
  - `https://www.google.com`
- Uses a context manager to disable warnings from requests about SSL certificate validation.
- Uses a context manager to patch urllib3 to add SSL certificate attributes to the HTTPSResponse object.
- Makes a request to a server using `requests.get()`.
- Returns the response object, with the SSL certificate objects available via:
  - `response.raw.peer_cert`: The servers actual certificate as an `OpenSSL.crypto.X509` object.
  - `response.raw.peer_cert_chain`: The servers certificate chain as a list of `OpenSSL.crypto.X509` objects
  - `response.raw.peer_cert_dict`: A dictionary that seems to only contain the subject info from the servers certificate, and therefore has limited usefulness, but is included for completeness sake.

Valid certs

You can get a valid cert and cert chain (signed by known CA):

```
>>> # get the response object
>>> response = cert_human.get_response(host="www.google.com")

>>> # access the cert chain from the response.raw object
>>> print(response.raw.peer_cert_chain)
[<OpenSSL.crypto.X509 object at 0x104d1a5f8>, <OpenSSL.crypto.X509 object at␣
↪0x104d1a6a0>]

>>> # access the cert from the response.raw object
>>> print(response.raw.peer_cert.get_subject().get_components())
[(b'C', b'US'), (b'ST', b'California'), (b'L', b'Mountain View'), (b'O', b'Google LLC
↪'), (b'CN', b'www.google.com')]

>>> # convert the cert from x509 object to PEM string
>>> pem = cert_human.x509_to_pem(response.raw.peer_cert)

>>> # write the PEM to a file
>>> path = cert_human.write_file(path="/tmp/certs/google.pem", text=pem,␣
↪overwrite=True, mkparent=True, protect=True)
>>> path.is_file()
True
```

Invalid certs

You can get an invalid cert (self-signed/self-issued/signed by unknown CA):

```
>>> # get the response object
>>> response = cert_human.get_response(host="cyborg")

>>> # access the cert from the response.raw object
>>> print(response.raw.peer_cert.get_subject().get_components())
[(b'CN', b'cyborg')]

>>> # access the cert chain from the response.raw object
>>> print(x.raw.peer_cert_chain)
[<OpenSSL.crypto.X509 object at 0x10e7a7908>]

>>> # convert the cert from x509 object to PEM string
>>> pem = cert_human.x509_to_pem(response.raw.peer_cert)

>>> # write the PEM to a file
>>> path = cert_human.write_file(path="/tmp/certs/cyborg.pem", text=pem,␣
↪overwrite=True, mkparent=True, protect=True)
>>> path.is_file()
True
```

5.2.2.2 Using cert_human.ssl_socket to get cert and cert chain

cert_human.ssl_socket does a number of things:

- Allows fetching a cert or cert chain using a socket.socket wrapped with OpenSSL.SSL.Context.

- This means you don't need to patch `urllib3.connectionpool.HTTPSConnectionPool` so that requests can access the certificate attributes on the raw object.
- By default does not allow SSLv2 connections.

Valid or invalid certs

Getting certs with this method performs NO verification, so you can just get your cert on.

```
>>> # get the cert and cert chain
>>> with cert_human.ssl_socket(host="cyborg") as sock:
...     cert = sock.get_peer_certificate()
...     cert_chain = sock.get_peer_cert_chain()
...

>>> print(cert_chain)
[<OpenSSL.crypto.X509 object at 0x10bccd6d8>]

>>> # access whatever cert info manually from the OpenSSL.crypto.x509 object
>>> print(cert.get_subject().get_components())
[(b'CN', b'cyborg')]

>>> # convert the cert from x509 object to PEM string
>>> pem = cert_human.x509_to_pem(cert)

>>> # write the PEM to a file
>>> path = cert_human.write_file(path="/tmp/certs/cyborg.pem", text=pem,␣
↪overwrite=True, mkparent=True, protect=True)
>>> path.is_file()
True
```

```
>>> # get the cert and cert chain
>>> with cert_human.ssl_socket(host="google.com") as sock:
...     cert = sock.get_peer_certificate()
...     cert_chain = sock.get_peer_cert_chain()

>>> print(cert_chain)
[<OpenSSL.crypto.X509 object at 0x10bccd7b8>, <OpenSSL.crypto.X509 object at␣
↪0x10bccd860>]

>>> # access whatever cert info manually from the OpenSSL.crypto.x509 object
>>> print(cert.get_subject().get_components())
[(b'C', b'US'), (b'ST', b'California'), (b'L', b'Mountain View'), (b'O', b'Google LLC
↪'), (b'CN', b'*.google.com')]

>>> # convert the cert from x509 object to PEM string
>>> pem = cert_human.x509_to_pem(cert)

>>> # write the PEM to a file
>>> path = cert_human.write_file(path="/tmp/certs/cyborg.pem", text=pem,␣
↪overwrite=True, mkparent=True, protect=True)
>>> path.is_file()
True
```

5.2.2.3 Using requests methods to get cert and cert chain

These examples just patch urllib3 for requests so you can use any normal `requests` method to get a response.

Valid certs using urllib3 patch tools

Using the *cert_human.urllib3_patch* context manager:

```
>>> # get the response with the cert attributes set
>>> with cert_human.urllib3_patch():
...     response = requests.get("https://www.google.com")
...
>>> print(response.raw.peer_cert.get_subject().get_components())
[(b'C', b'US'), (b'ST', b'California'), (b'L', b'Mountain View'), (b'O', b'Google LLC
↪'), (b'CN', b'www.google.com')]
```

Using *cert_human.enable_urllib3_patch* to patch urllib3:

```
>>> # patch urllib3
>>> cert_human.enable_urllib3_patch()

>>> # get a response using whatever method in requests
>>> response = requests.get("https://www.google.com")

>>> # access whatever cert info manually from the OpenSSL.crypto.x509 object
>>> print(response.raw.peer_cert.get_subject().get_components())
[(b'C', b'US'), (b'ST', b'California'), (b'L', b'Mountain View'), (b'O', b'Google LLC
↪'), (b'CN', b'www.google.com')]

>>> # convert the cert from x509 object to PEM string
>>> pem = cert_human.x509_to_pem(response.raw.peer_cert)

>>> # write the PEM to a file
>>> path = cert_human.write_file(path="/tmp/certs/google.pem", text=pem,␣
↪overwrite=True, mkparent=True, protect=True)
>>> path.is_file()
True

>>> # optionally disable the urllib3 patch once you no longer need
>>> # to get responses with the cert attributes attached
>>> cert_human.disable_urllib3_patch()
```

Invalid certs using urllib3 patch tools

Same as valid certs using urllib3 patch tools, but you need to set verify=False in your requests method (and optionally disable requests warnings as well):

```
>>> cert_human.enable_urllib3_patch()
>>> response = requests.get("https://cyborg", verify=False)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/urllib3/
↪connectionpool.py:847: InsecureRequestWarning: Unverified HTTPS request is being␣
↪made. Adding certificate verification is strongly advised. See: https://urllib3.
↪readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning)
```

(continues on next page)

(continued from previous page)

```
>>> print(response.raw.peer_cert.get_subject().get_components())
[(b'CN', b'cyborg')]
```

5.2.3 API Reference

5.2.3.1 Classes

Store Classes

class cert_human.**CertStore**(*x509*)

Bases: object

Make SSL certs and their attributes generally more accessible.

Examples

```
>>> # x509 cert from any number of methods
>>> cert = CertStore(OpenSSL.crypto.X509)
>>> # not echoing any of these due to length
>>> print(cert)  # print the basic info for this cert
>>> x = cert.issuer  # get a dict of the issuer info.
>>> print(cert.issuer_str)  # print the issuer in str form.
>>> x = cert.subject  # get a dict of the subject info.
>>> print(cert.subject_str)  # print the subject in str form.
>>> print(cert.dump_str_exts)  # print the extensions in str form.
>>> print(cert.pem) # print the PEM version.
>>> print(cert.public_key_str)  # print the public key.
>>> print(cert.dump_str_key)  # print a bunch of public key info.
>>> print(cert.dump_str_info)  # print what str(cert) prints.
>>> x = cert.dump  # get a dict of ALL attributes.
>>> x = cert.dump_json_friendly  # dict of JSON friendly attrs.
>>> print(cert.dump_json)  # JSON str of JSON friendly attrs.
>>> # and so on
```

Notes

The whole point of this was to be able to provide the same kind of information that is seen when looking at an SSL cert in a browser. This can be used to prompt for validity before doing "something". For instance:

- If no cert provided, get the cert and prompt user for validity before continuing.
- If no cert provided, get cert, prompt for validity, then write to disk for using in further connections.
- ... to print it out and hang it on the wall???

__str__()

Show dump_str_info.

__repr__()

Use str() for repr().

classmethod from_socket(*host*, *port=443*)

Make instance of this cls using socket module to get the cert.

Examples

```
>>> cert = CertStore.from_socket("cyborg")
>>> print(cert)
```

Parameters

- **host** (str) – hostname to connect to.
- **port** (str, optional) – port to connect to on host. Defaults to: 443.

Returns (*CertStore*)

classmethod from_request(*host*, *port=443*)

Make instance of this cls using requests module to get the cert.

Examples

```
>>> cert = CertStore.from_request("cyborg")
>>> print(cert)
```

Parameters

- **host** (str) – hostname to connect to.
- **port** (str, optional) – port to connect to on host. Defaults to: 443.

Returns (*CertStore*)

classmethod from_response(*response*)

Make instance of this cls using a requests.Response object.

Examples

```
>>> enable_urllib3_patch()
>>> response = requests.get("https://cyborg", verify=False)
>>> cert = CertStore.from_response(response)
>>> print(cert)
```

Notes

This relies on the fact that *enable_urllib3_patch()* has been used to add the SSL attributes to the requests.Response.raw object.

Parameters **response** (requests.Response) – response object to get raw.peer_cert.

Returns (*CertStore*)

classmethod from_auto(*obj*)

Make instance of this cls from a number of types.

:param obj (str or bytes or OpenSSL.crypto.X509 or: X509.Certificate or requests.Response):
Object to create CertStore.

Returns (*CertStore*)

classmethod from_path(*path*)
Make instance of this cls from a file containing a PEM.

Parameters **path** (str or pathlib.Path) – Path to file containing PEM.

Returns (*CertStore*)

pem
Return the PEM version of the original x509 cert object.

Returns (str)

x509
Return the original x509 cert object.

Returns (OpenSSL.crypto.X509)

der
Return the DER bytes version of the original x509 cert object.

Returns (bytes)

asn1
Return the ASN1 version of the original x509 cert object.

Returns (x509.Certificate)

to_path(*path*, *overwrite=False*, *mkparent=True*, *protect=True*)
Write self.pem to disk.

Examples

```
>>> # get a cert using sockets:
>>> cert = CertStore.from_socket("cyborg")
>>> # or, get a cert using requests:
>>> cert = CertStore.from_request("cyborg")
```

```
>>> # ideally, do some kind of validation with the user here
>>> # i.e. use ``print(cert.dump_str)`` to show the same
>>> # kind of information that a browser would show
```

```
>>> # then write to disk:
>>> cert_path = cert.to_path("~/cyborg.pem")
```

```
>>> # use requests with the newly written cert
>>> # no SSL warnings or SSL validation errors happen
>>> # even though it's self signed:
>>> response = requests.get("https://cyborg", verify=cert_path)
```

Parameters **path** (str or pathlib.Path) – Path to write self.pem.

Returns (pathlib.Path)

test (*host*, *port=443*, ***kwargs*)
Test that a cert is valid on a site.

Parameters

- **host** (str) – hostname to connect to. can be any of: "scheme://host:port", "scheme://host", or "host".
- **port** (str, optional) – port to connect to on host. If no :PORT in host, this will be added to host. Defaults to: 443
- **kwargs** – passed thru to requests.get()

Returns True / False if cert was valid. Exception that was thrown if cert not valid, or None if successful.

Return type (tuple of (bool, Exception))

issuer
Get issuer parts.

Returns (dict)

issuer_str
Get issuer parts as string.

Returns (str)

subject
Get subject parts.

Returns (dict)

subject_str
Get subject parts as string.

Returns (str)

subject_alt_names
Get subject alternate names.

Returns (list of str)

subject_alt_names_str
Get subject alternate names as CSV string.

Returns (str)

fingerprint_sha1
SHA1 Fingerprint.

Returns (str)

fingerprint_sha256
SHA256 Fingerprint.

Returns (str)

public_key
Public key in hex format.

Notes

EC certs don't have a modulus, and thus public_key in asn1 obj is not a dict, just the cert itself.

Returns (str)

public_key_str
Public key as in hex format spaced and wrapped.

Returns (str)

public_key_parameters
Public key parameters, only for 'ec' certs.

Returns (str)

public_key_algorithm
Algorithm of public key ('ec', 'rsa', 'dsa').

Returns (str)

public_key_size
Size of public key in bits.

Returns (int)

public_key_exponent
Public key exponent, only for 'rsa' certs.

Returns (int)

signature
Signature in hex format.

Returns (str)

signature_str
Signature in hex format spaced and wrapped.

Returns (str)

signature_algorithm
Algorithm used to sign the public key certificate.

Returns (str)

x509_version
Version of x509 this certificate is using.

Returns (str)

serial_number
Certificate serial number.

Returns int if algorithm is 'ec', or hex str.

Return type (str or int)

serial_number_str
Certificate serial number.

Returns int if algorithm is 'ec', or spaced and wrapped hex str.

Return type (str or int)

is_expired
Determine if this certificate is expired.

Returns (bool)

is_self_signed
Determine if this certificate is self_sign.

Returns ('yes', 'maybe', or 'no')

Return type (str)

is_self_issued
Determine if this certificate is self issued.

Returns (bool)

not_valid_before
Certificate valid start date as datetime object.

Returns (datetime.datetime)

not_valid_before_str
Certificate valid start date as str.

Returns (str)

not_valid_after
Certificate valid end date as datetime object.

Returns (datetime.datetime)

not_valid_after_str
Certificate valid end date as str.

Returns (str)

extensions
Certificate extensions as dict.

Notes

Parsing the extensions was not easy. I sort of gave up at one point. Resorted to using str(extension) as OpenSSL returns it.

Returns (dict)

extensions_str
Certificate extensions as str with index, name, and value.

Returns (str)

dump
Dump dictionary with all attributes of self.

Returns (dict)

dump_json_friendly
Dump dict with all attributes of self that are JSON friendly.

Returns (dict)

dump_json
Dump JSON string with all attributes of self that are JSON friendly.

Returns (str)

dump_str
Dump a human friendly str of the all the important bits.

Returns (str)

dump_str_info
Dump a human friendly str of the important cert info bits.

Returns (str)

dump_str_exts
Dump a human friendly str of the extensions.

Returns (str)

dump_str_key
Dump a human friendly str of the public_key important bits.

Returns (str)

_extension_str(*ext*)
Format the string of an extension using str(extension).

Returns (str)

_extensions
List mapping of extension name to extension object.

Returns (list of list)

_public_key_native
Access self.asn1.public_key.

Returns (dict)

_cert_native
Access to self.asn1.

Returns (dict)

_is_ec
Determine if this certificates public key algorithm is Elliptic Curve ('ec').

Returns (bool)

class cert_human.**CertChainStore**(*x509=None*)
Bases: object

Make SSL cert chains and their attributes generally more accessible.

This is really just a list container for a cert chain, which is just a list of x509 certs.

__str__()
Show most useful information of all certs in cert chain.

__repr__()
Use str() for repr().

__getitem__(*i*)
Passthru to self._certs[n].

__len__()
Passthru to len(self._certs).

append(*value*)
Passthru to self._certs.append with automatic conversion for PEM or X509.

Parameters **value** (str or OpenSSL.crypto.X509 or *CertStore*) –

classmethod from_socket(*host*, *port=443*)

Make instance of this cls using socket module to get the cert chain.

Examples

```
>>> cert_chain = CertChainStore.from_socket("cyborg")
>>> print(cert_chain)
```

Parameters

- **host** (str) – hostname to connect to.
- **port** (str, optional) – port to connect to on host. Defaults to: 443.

Returns (*CertChainStore*)

classmethod from_request(*host*, *port=443*)

Make instance of this cls using requests module to get the cert chain.

Examples

```
>>> cert_chain = CertChainStore.from_request("cyborg")
>>> print(cert_chain)
```

Parameters

- **host** (str) – hostname to connect to.
- **port** (str, optional) – port to connect to on host. Defaults to: 443.
- **timeout** (str, optional) – Timeout for connect/response. Defaults to: 5.

Returns (*CertChainStore*)

classmethod from_response(*response*)

Make instance of this cls using a requests.Response.raw object.

Examples

```
>>> enable_urllib3_patch()
>>> response = requests.get("https://cyborg", verify=False)
>>> cert_chain = CertChainStore.from_response(response)
>>> print(cert_chain)
```

Notes

This relies on the fact that *enable_urllib3_patch()* has been used to add the SSL attributes to the requests.Response.raw object.

Parameters **response** (requests.Response) – response object to get raw.peer_cert_chain from.

Returns (*CertChainStore*)

classmethod from_pem(*pem*)
Make instance of this cls from a string containing multiple PEM certs.

Parameters **pem** (str) – PEM string with multiple pems to convert to x509.

Returns (*CertChainStore*)

classmethod from_path(*path*)
Make instance of this cls from a file containing PEMs.

Parameters **path** (str or pathlib.Path) – Path to file containing PEMs.

Returns (*CertChainStore*)

certs
Expose self._certs list container.

pem
Return all of the joined PEM strings for each cert in self.

Returns all PEM strings joined.

Return type (str)

x509
Return the X509 version of the each CertStore in self.

Returns (list of x509.Certificate)

der
Return the DER bytes version of the each CertStore in self.

Returns (list of bytes)

asn1
Return the asn1crypto X509 version of the each CertStore in self.

Returns (list of x509.Certificate)

to_path(*path*, *overwrite=False*, *mkparent=True*, *protect=True*)
Write self.pem to disk.

Parameters **path** (str or pathlib.Path) – Path to write self.pem to.

Returns (pathlib.Path)

dump_json_friendly
Dump dict with all attributes of each cert in self that are JSON friendly.

Returns (list of dict)

dump_json
Dump JSON string with all attributes of each cert in self that are JSON friendly.

Returns (str)

dump
Dump dictionary with all attributes of each cert in self.

Returns (list of dict)

dump_str
Dump a human friendly str of the all the important bits for each cert in self.

Returns (str)

dump_str_info
Dump a human friendly str of the important cert info bits for each cert in self.

Returns (str)

dump_str_key
Dump a human friendly str of the public_key important bits for each cert in self.

Returns (str)

dump_str_exts
Dump a human friendly str of the extensions for each cert in self.

Returns (str)

WithCert Classes

class cert_human.**HTTPSConnectionWithCertCls**(*host*, *port=None*, *key_file=None*, *cert_file=None*, *key_password=None*, *strict=None*, *timeout=<object object>*, *ssl_context=None*, *server_hostname=None*, ***kw*)
Bases: urllib3.connection.VerifiedHTTPSConnection

connect()
Connect to the host and port specified in __init__.

_set_cert_attrs()
Add cert info from the socket connection to a HTTPSConnection object.

Adds the following attributes:

- peer_cert: x509 certificate of the server
- peer_cert_chain: x509 certificate chain of the server
- peer_cert_dict: dictionary containing commonName and subjectAltName

class cert_human.**ResponseWithCertCls**(**args*, ***kwargs*)
Bases: urllib3.response.HTTPResponse

_set_cert_attrs()
Add cert info from a HTTPSConnection object to a HTTPSResponse object.

This allows accessing the attributes in a HTTPSConnectionWithCertCls from a requests.Response.raw object like so:

- requests.Response.raw.peer_cert
- requests.Response.raw.peer_cert_chain
- requests.Response.raw.peer_cert_dict

_abc_impl = <_abc_data object>

Exception Classes

exception cert_human.**CertHumanError**
Bases: Exception

Exception wrapper.

5.2.3.2 Functions

Get Cert Functions

cert_human.**get_response**(*host*, *port=443*, ***kwargs*)
Get a requests.Response object with cert attributes.

Examples

Make a request to a site that has a valid cert:

```
>>> response = get_response(host="www.google.com")
>>> print(response.raw.peer_cert.get_subject().get_components())
>>> print(response.raw.peer_cert_chain)
>>> print(response.raw.peer_cert_dict)
```

Make a request to a site that has an invalid cert (self-signed):

```
>>> response = get_response(host="cyborg")
>>> print(response.raw.peer_cert.get_subject().get_components())
```

Notes

This is to fetch a requests.Response object that has cert attributes.

- Uses a context manager to disable warnings about SSL cert validation.
- Uses a context manager to patch urllib3 to add SSL cert attributes to the HTTPSResponse object, which is then accessible via the `requests.Response`.raw object.
- Makes a request to a server using `requests.get()`

Parameters

- **host** (`str`) – hostname to connect to. can be any of: "scheme://host:port", "scheme://host", or "host".
- **port** (`str`, optional) – port to connect to on host. If no :PORT in host, this will be added to host. Defaults to: 443
- **kwargs** – passed thru to requests.get()

Returns (`requests.Response`)

cert_human.**ssl_socket**(*host*, *port=443*, **args*, ***kwargs*)
Context manager to create an SSL socket.

Examples

Use sockets and OpenSSL to make a request using this context manager:

```
>>> with ssl_socket(host="cyborg", port=443) as sock:
...   cert = sock.get_peer_certificate()
...   cert_chain = sock.get_peer_cert_chain()
...
>>> print(cert.get_subject().get_components())
>>> print(cert_chain)
```

Parameters

- **host** (str) – hostname to connect to.
- **port** (str, optional) – port to connect to on host. Defaults to: 443.

Yields (OpenSSL.SSL.Connection)

cert_human.**test_cert**(*host*, *port=443*, *timeout=5*, ***kwargs*)

Test that a cert is valid on a site.

Parameters

- **host** (str) – hostname to connect to. can be any of: "scheme://host:port", "scheme://host", or "host".
- **port** (str, optional) – port to connect to on host. If no :PORT in host, this will be added to host. Defaults to: 443
- **timeout** (str, optional) – Timeout for connect/response. Defaults to: 5.
- **kwargs** – passed thru to requests.get()

Returns True / False if cert was valid. Exception that was thrown if cert not valid, or None if successfull.

Return type (tuple of (bool, Exception))

WithCert Functions

cert_human.**enable_urllib3_patch**()

Patch HTTPSConnectionPool to use the WithCert Connect/Response classes.

Examples

Make a request using requests and patch urllib3 until you want to unpatch it:

```
>>> enable_urllib3_patch()
>>> response1 = requests.get("https://www.google.com")
>>> # self-signed, don't verify
>>> response2 = requests.get("https://cyborg", verify=False)
>>> print(response1.raw.peer_cert.get_subject().get_components())
>>> print(response2.raw.peer_cert.get_subject().get_components())
>>> # optionally disable the urllib3 patch once you no longer need
>>> # to make requests with the cert attributes attached
>>> disable_urllib3_patch()
```

Notes

Modifies `urllib3.connectionpool.HTTPSConnectionPool` attributes `urllib3.connectionpool.HTTPSConnectionPool.ConnectionCls` and `urllib3.connectionpool.HTTPSConnectionPool.ResponseCls` to the WithCert classes.

`cert_human.`**`disable_urllib3_patch`**`()`
Unpatch HTTPSConnectionPool to use the default Connect/Response classes.

Notes

Modifies `urllib3.connectionpool.HTTPSConnectionPool` attributes `urllib3.connectionpool.HTTPSConnectionPool.ConnectionCls` and `urllib3.connectionpool.HTTPSConnectionPool.ResponseCls` back to original classes.

`cert_human.`**`urllib3_patch`**`()`
Context manager to enable/disable cert patch.

Examples

Make a request using `requests` using this context manager to patch urllib3:

```
>>> import requests
>>> with urllib3_patch():
...   response = requests.get("https://www.google.com")
...
>>> print(response.raw.peer_cert.get_subject().get_components())
```

`cert_human.`**`using_urllib3_patch`**`()`
Check if urllib3 is patched with the WithCert classes.

Returns (`bool`)

`cert_human.`**`check_urllib3_patch`**`()`
Throw exception if urllib3 is not patched with the WithCert classes.

Raises (*`CertHumanError`*) – if using_urllib3_patch() returns False.

Conversion Functions

`cert_human.`**`asn1_to_der`**`(`*`asn1`*`)`
Convert from asn1crypto x509 to DER bytes.

Parameters **`asn1`** (`x509.Certificate`) – asn1crypto x509 to convert to DER bytes

Returns (`bytes`)

`cert_human.`**`asn1_to_x509`**`(`*`asn1`*`)`
Convert from asn1crypto x509 to OpenSSL x509.

Parameters **`asn1`** (`x509.Certificate`) – asn1crypto x509 to convert to OpenSSL x509

Returns (`OpenSSL.crypto.X509`)

cert_human.**der_to_asn1**(*der*)
Convert from DER bytes to asn1crypto x509.

Parameters **der** (bytes) – DER bytes string to convert to x509.Certificate.

Returns (x509.Certificate)

cert_human.**der_to_x509**(*der*)
Convert from DER bytes to OpenSSL x509.

Parameters **der** (bytes) – DER bytes string to convert to x509.Certificate.

Returns (OpenSSL.crypto.X509)

cert_human.**pem_to_x509**(*pem*)
Convert from PEM str to OpenSSL x509.

Parameters **pem** (str) – PEM string to convert to x509 certificate object.

Returns (OpenSSL.crypto.X509)

cert_human.**pems_to_x509**(*pem*)
Convert from PEM str with multiple certs to list of OpenSSL x509s.

Parameters **pem** (str) – PEM string with multiple certs to convert to x509 certificate object.

Returns (list of OpenSSL.crypto.X509)

cert_human.**x509_to_asn1**(*x509*)
Convert from OpenSSL x509 to asn1crypto x509.

Parameters **x509** (OpenSSL.crypto.X509) – x509 object to convert to x509.Certificate.

Returns (x509.Certificate)

cert_human.**x509_to_der**(*x509*)
Convert from OpenSSL x509 to DER bytes.

Parameters **x509** (OpenSSL.crypto.X509) – x509 certificate object to convert to DER.

Returns (bytes)

cert_human.**x509_to_pem**(*x509*)
Convert from OpenSSL x509 to PEM str.

Parameters **x509** (OpenSSL.crypto.X509) – x509 certificate object to convert to PEM.

Returns (str)

Utility Functions

cert_human.**clsname**(*obj*)
Get objects class name.

Parameters **obj** (object) – The object or class to get the name of.

Returns (str)

cert_human.**hexify**(*obj*, *space=False*, *every=2*, *zerofill=True*)
Convert bytes, int, or str to hex and optionally space it out.

Parameters

- **obj** (str or int or bytes) – The object to convert into hex.

- **zerofill** (bool, optional) – Zero fill the string if len is not even. This gets around oddly sized hex strings. Defaults to: True.
- **space** (bool, optional) – Space the output string using join. Defaults to: False.
- **join** (str, optional) – Rejoining str. Defaults to: " ".
- **every** (str, optional) – The number of characters to split on. Defaults to: 2.

Returns (str)

cert_human.**indent**(*txt*, *n=4*, *s=' '*)
Text indenter.

Parameters

- **txt** (str) – The text to indent.
- **n** (str, optional) – Number of s to indent txt. Defaults to: 4.
- **s** (str, optional) – Char to use for indent. Defaults to: " ".

Returns (str)

cert_human.**write_file**(*path*, *text*, *overwrite=False*, *mkparent=True*, *protect=True*)
Write text to path.

Parameters

- **path** (str or pathlib.Path) – The path to write text to.
- **text** (str) – The text to write to path.
- **overwrite** (bool, optional) – Overwite file if exists. Defaults to: False.
- **mkparent** (bool, optional) – Create parent directory if not exist. Defaults to: True.
- **protect** (bool, optional) – Set file 0600 and parent 0700. Defaults to: True.

Raises (*CertHumanError*) – path exists and not overwrite, parent not exist and not mkparent.

Returns (pathlib.Path)

cert_human.**read_file**(*path*)
Read text from path.

Parameters **path** (str or pathlib.Path) – Path to file to read.

Raises (*CertHumanError*) – if path does not exist.

Returns (str)

cert_human.**find_certs**(*txt*)
Split text with multiple certificates into a list of certificates.

Parameters **txt** (str) – the text to find certificates in.

Returns (list of str)

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

C

D

E

F

G

H

I

M

N

P

R

S

T

U

W

X