Code Listing for Trustery

Final Project Report Supplement

Mustafa Al-Bassam

King's College London

April 24, 2016

Contents

A Code **2**
A.1 trustery/build-contract.sh 2
A.2 trustery/contract/trustery.sol 2
A.3 trustery/contract/trustery-alt.sol 4
A.4 trustery/pytrustery/setup.py 5
A.5 trustery/test/pgp.py 6
A.6 trustery/userconfig.py 6
A.7 trustery/res/pgp_attribute_1.pickle 7
A.8 trustery/res/pgp_attribute_2.pickle 14
A.9 trustery/res/pgp_attribute_3.pickle 15
A.10 trustery/pytrustery/__init__.py 16
A.11 trustery/pytrustery/console.py 16
A.12 trustery/pytrustery/consoleutil.py 23
A.13 trustery/pytrustery/ethapi.py 24
A.14 trustery/pytrustery/gpgapi.py 25
A.15 trustery/pytrustery/ipfsapi.py 28
A.16 trustery/pytrustery/trustery_abi.json 29
A.17 trustery/pytrustery/events.py 35
A.18 trustery/pytrustery/transactions.py 41
A.19 trustery/pytrustery/userconfig.py 45

Appendix A

Code

A.1 trustery/build-contract.sh

```
#!/bin/bash

if [ "$1" == "--alt" ]
then
    solc contract/trustery-alt.sol --abi | tail -2 | python -m json.
        tool > pytrustery/trustery/trustery_abi.json
else
    solc contract/trustery.sol --abi | tail -2 | python -m json.tool
         > pytrustery/trustery/trustery_abi.json
fi
```

A.2 trustery/contract/trustery.sol

```
contract Trustery {
    struct Attribute {
        address owner;
        string attributeType;
        bool has_proof;
        bytes32 identifier;
        string data;
        string datahash;
    }
```

```
struct Signature {
    address signer;
    uint attributeID;
    uint expiry;
}

struct Revocation {
    uint signatureID;
}

Attribute[] public attributes;
Signature[] public signatures;
Revocation[] public revocations;

event AttributeAdded(uint indexed attributeID, address indexed
    owner, string attributeType, bool has_proof, bytes32 indexed
     identifier, string data, string datahash);
event AttributeSigned(uint indexed signatureID, address indexed
    signer, uint indexed attributeID, uint expiry);
event SignatureRevoked(uint indexed revocationID, uint indexed
    signatureID);

function addAttribute(string attributeType, bool has_proof,
    bytes32 identifier, string data, string datahash) returns (
    uint attributeID) {
    attributeID = attributes.length++;
    Attribute attribute = attributes[attributeID];
    attribute.owner = msg.sender;
    attribute.attributeType = attributeType;
    attribute.has_proof = has_proof;
    attribute.identifier = identifier;
    attribute.data = data;
    attribute.datahash = datahash;
    AttributeAdded(attributeID, msg.sender, attributeType,
        has_proof, identifier, data, datahash);
}

function signAttribute(uint attributeID, uint expiry) returns (
    uint signatureID) {
    signatureID = signatures.length++;
```

```
        Signature signature = signatures[signatureID];
        signature.signer = msg.sender;
        signature.attributeID = attributeID;
        signature.expiry = expiry;
        AttributeSigned(signatureID, msg.sender, attributeID, expiry
            );
    }

    function revokeSignature(uint signatureID) returns (uint
        revocationID) {
        if (signatures[signatureID].signer == msg.sender) {
            revocationID = revocations.length++;
            Revocation revocation = revocations[revocationID];
            revocation.signatureID = signatureID;
            SignatureRevoked(revocationID, signatureID);
        }
    }
}
```

A.3 trustery/contract/trustery-alt.sol

```
contract Trustery {
    struct Signature {
        address signer;
    }

    uint public attributes;
    Signature[] public signatures;
    uint public revocations;

    event AttributeAdded(uint indexed attributeID, address indexed
        owner, string attributeType, bool has_proof, bytes32 indexed
         identifier, string data, string datahash);
    event AttributeSigned(uint indexed signatureID, address indexed
        signer, uint indexed attributeID, uint expiry);
    event SignatureRevoked(uint indexed revocationID, uint indexed
        signatureID);

    function addAttribute(string attributeType, bool has_proof,
        bytes32 identifier, string data, string datahash) returns (
```

```
        uint attributeID) {
        attributeID = attributes++;
        AttributeAdded(attributeID, msg.sender, attributeType,
            has_proof, identifier, data, datahash);
    }

    function signAttribute(uint attributeID, uint expiry) returns (
        uint signatureID) {
        signatureID = signatures.length++;
        Signature signature = signatures[signatureID];
        signature.signer = msg.sender;
        AttributeSigned(signatureID, msg.sender, attributeID, expiry
            );
    }

    function revokeSignature(uint signatureID) returns (uint
        revocationID) {
        if (signatures[signatureID].signer == msg.sender) {
            revocationID = revocations++;
            SignatureRevoked(revocationID, signatureID);
        }
    }
}
```

A.4 trustery/pytrustery/setup.py

```
from setuptools import setup

setup(
    name='trustery',
    version='0.1',
    packages=['trustery'],
    package_data={'trustery': ['trustery/trustery_abi.json']},
    install_requires=[
        'click',
        'jsonrpc-requests',
        'ethereum',
        'rlp',
        'configobj',
        'appdirs',
```

```
        'ethereum-rpc-client',
        'python-gnupg',
        'ipfs-api'
    ],
    entry_points='''
        [console_scripts]
        trustery=trustery.console:cli
    ''',
)
```

A.5 trustery/test/pgp.py

```
import pickle
import unittest

from trustery.events import Events


class TestPGP(unittest.TestCase):
    def test_verify_attribute_pgp_proof(self):
        events = Events()

        attribute = pickle.load(open('res/pgp_attribute_1.pickle'))
        valid = events.verify_attribute_pgp_proof(attribute)
        self.assertTrue(valid)

        attribute = pickle.load(open('res/pgp_attribute_2.pickle'))
        valid = events.verify_attribute_pgp_proof(attribute)
        self.assertIsNone(valid)

        attribute = pickle.load(open('res/pgp_attribute_3.pickle'))
        valid = events.verify_attribute_pgp_proof(attribute)
        self.assertFalse(valid)

if __name__ == '__main__':
    unittest.main()
```

A.6 trustery/userconfig.py

```
import unittest

from trustery.userconfig import *


class TestUserConfig(unittest.TestCase):
    def test_truststore(self):
        self.assertFalse(is_trusted('foo'))

        trust('foo')
        self.assertTrue(is_trusted('foo'))

        untrust('foo')
        self.assertFalse(is_trusted('foo'))

if __name__ == '__main__':
    unittest.main()
```

A.7 trustery/res/pgp_attribute_1.pickle

```
(dp0
Vhas_proof
p1
I01
sS'_event_type'
p2
S'AttributeAdded'
p3
sVdatahash
p4
S''
p5
sS'signatures_status'
p6
(dp7
S'status'
p8
(dp9
S'valid'
p10
```

```
I0
sS'invalid'
p11
I0
ssS'signatures'
p12
(lp13
ssVattributeID
p14
I9
sVattributeType
p15
S'pgp-key'
p16
sVowner
p17
S'db57ccafaa5198089af9c69499e8d3d67cb463a3'
p18
sVidentifier
p19
S'\x14KY\xad\x81\x8b\x0ecHp\x9b\xd0a\xce\xb38\x8f!\xea\x17\x00\x00\
    x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
p20
sVdata
p21
V-----BEGIN PGP PUBLIC KEY BLOCK-----\u000aVersion: GnuPG v1\u000a\
    u000amQINB
FLUFRsBEAC7LSFzH4isppSXlZnuvxqV99Nip1F2umAps+qrp+L3dMFJdkZs\
    u000a1debUrH0tfNcTF
kkkzLiXiJjODWI1kIYKd15nY4fscsjkazC6byg9iHWXsACdKp0\u000aVlN11/
    XrQ5zEfxNp8kU0mJT
GGJTTYOYKEHdf9Ylui1eBZOVmvcrasC8OIMj16VmB\u000atlzi1QRGotl1E55KEZYJ
    /TJK4SDB5B+f
f+r44HvGJA+H1BdfkStJ9YkYtrsb46yL\u000aga540lAZGPTpfRBr7REqmB5lCg+
    jJSka7oRUkNsTp
KcSQ0aJ4Eri5Ki2kCMuFWrj\
    u000axhr5IIgCdx2w9ARHqygsJ0nzLagAfV87qC9Q6OQgsHG6jiC8oS
teaZot+RkZ4I3Y\u000aRbG9eEeSPwF++gRtWkCncuyobNgK9noshp+9H96AGMhVm/
    E9o40TT24JSEr
cqCqE\u000aACQQif+6VvdMcofpwXm1Wz/g9XxXPTkQqzskv/
```

```
    EvWDAIyF5EzVX8ESaib8J5zJUj\u0
00azS7vM+F7op6C9UhiAV4PyC89wus4Z8vXHUbx2xPwthFPZoNJEAr/kX+av0MiWtC1
    \u000a96VUUt
sUCdrKEdewU8sMqY1RS1ay0vv/5gXjPJgsyohJVapqwe1kQgxQHVqKdM48\u000a+
    iDrD1ytv4odI8D
2D+dgu2ZqjF/NheLHISxe4kkjASNOolFCvZuByhPxSwARAQAB\
    u000atCRNdXN0YWZhIEFsLUJhc3Nh
bSA8bXVzQG11c2FsYmFzLmNvbT6JAkEEEwECACsC\
    u000aGwMGCwkIBwMCBhUIAgkKCwQWAgMBAh4BA
heAAhkBBQJWTyNCBQkHPXUZAAoJEGHO\u000asziPIeoXzJgP/ipNZHSgwmJpRFs+
    ZJ2nUvC6CQG5fI
nGu0kieCK4i+FRyQdwyhkr\u000adczoNW/wGteA9N7wZfMCQ9V8XL+
    B05ssxVVxOGkATzbq+ziUA5n
IvHA6a65nu+tS\u000a3iAHZIsZB4zwKb6STR11GTFk/
    rWsH2MdeL7mClZCKQbLk50eqPkxJ9vJx06i
2vU8\
    u000aLdfEOOlKC4vkjdUxOK6YPOdqcSvso1KwEHMhk1KS6s2JMoh2lW2JtvP96nEJQZYI
    \u00
0aEcqkbaCSoQJr2Htw9dBgSH9o3p9mV+jb5917T3vSy3tClSdZ89sWCNEIAs0IJg8X\
    u000afjP6CyO
5kRdddfOlyVTOArJ6vPqgStj5ATdsssQmLLqECIfE1kbABQVt3hL6eOav\u000an7zF
    /s9viiL5V7aX
o0B/D4s7WeWIuyKeA4HZ0wD/OFMXjd+mtKfDV7enEh5ud+Y9\u000atA1/
    gQrVX3OEig27bgGD6NfOx
xpeKE5gsnC1Kk1UVWADQJL/3U8yg0r0nR2f4V0K\
    u000a1lbjTXCWKjwh2dZewHhDcNbNcCGlwQHniD
NV21N9NqqZme28VHUUnhbSnM9x9Jfa\
    u000aeJ9SI8jms3ZKd7yvBFlg8DNZTkDSBpZQfCn6WmriWZg
gmORWb4r/1i7boWOLTbFM\
    u000aQYqGbnU5GuGXhfgfwJKy3KAUjolmZk9woCMb0mR0nApNP2FqJOx1
UwpNtCdNdXN0\
    u000aYWZhIEFsLUJhc3NhbSA8bXVzYWxiYXNAcmlzZXVwLm5ldD6JAj4EEwECACgCG

wMG\
    u000aCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheABQJWTyNCBQkHPXUZAAoJEGHOsziPIeoX
    \u000
a11sP/2alfIS9TyN7TVddk0WBrNlT3Nsn4TnKONPMxs24u6ZPH6lkCDYr/kr5Ecnb\
    u000a264/1XYu
AFtUU/56x3cya7dWF6/6Q0o7n1Ond5HBniFIrF5zyr0FNrGzrStmTw9x\
    u000a1OEvdJQZeN5DjVLPw
```

```
UeRR57/B2AkJwkGmzzClhxqB+NUTgOdsTfRIdjDZv2JiUTy\
    u000aRyWsddl94RfCQbpOKdtQfCSBg0
LSW9E441YkGeARXb+rss6mckWSXrtlGJangCu4\
    u000ajEP4AiMBqmuBynkhM01L4vO3yGnzCdgfWEc
20a9FB2b5fcdqYrSvRHB3hNJk+YAw\u000aO9Mjw/yb/
    qOLaPqf8wQrr4d6tgApurQchMr4j4uswy1+
F7UD/0IJPf9v0wrhr1k5\u000aot8xAmRAN88ToLhOVky6GGGmv0u8HmiqmXdc/
    thzS0OkeLB5m/dE4
iBgj3zosteN\
    u000aDfKGGa9Qnuww6ZjpW1LcjVNYvGNwLdZL1BGAHNIMSYGoKOAW8Ubrl9ypbDFkbQ

cP\u000aX8+GKEMk6BjAdJbDNwOyrh1J99lr+gfN1qkXTLiM13nsVkO+
    eaUz48T2VKSu2kUZ\u000a
H7VlsjxSIy9ZQN8t4r8tmbb48dBs/mvFkzrdY8Sfd/+/x3A0dO8q+D4jqh0m7KLE\
    u000akse+rpP2Q
4tbj1bDTIlDnnCbM1Z0hT7xjuEcUEYtMW8NI2VZtC9NdXN0YWZhIEFs\
    u000aLUJhc3NhbSA8bXVzdG
FmYS5hbC1iYXNzYW1Aa2NsLmFjLnVrPokCPgQTAQIAKAIb\
    u000aAwYLCQgHAwIGFQgCCQoLBBYCAwE
CHgECF4AFAlZPI0IFCQc9dRkACgkQYc6zOI8h\u000a6heeTg//
    ZYjEjEVeCK6GpsRiaURoW0cTlngO
fz+3akGavfhfOJRSTTok+MSWlKVc\u000a/T0tyyzYGAIhZYx/
    K0aSpFs2Sd5xEfamWfC0loMyLHq5n
1m6/kiQyv6immF+Rpq3\u000aHAj0+/rsR1tOJl0j7E4trcQ7/
    lWudremfZxVXmt2pqb9VVNr403yFD
49m3dmr1Jq\
    u000aB4yLUX8j28ki45Qchfts8zguY4J1neO5pKmhVwsgSxjwNHTGdrqWWrdz1+
    iXtRt
7\
    u000a2KENLas6PEdAlJcJDT10wyiMuHBkaQpVNCsWtwWAz6DdlmSCeiLn8Lrrvnns0iV6
    \u000a/
bPEhKEv3yr9fJl7gLPUEI2U3pOcxhEUMInmodIgGrnUX6bFvZpdTXgamnpmJvRX\
    u000aQEdYkFcgkR
7IPmeNBoEcCJU5xqOtnd3zj+763pAQnknx1u6Xoyuh2YwE/dqINLzQ\
    u000avyzVsKtqU3ckUw1tCtq
+3TTDbdgH+yYWpFTm054bLUYQpcvt2Vnixq84+UavsOdX\
    u000ae9692St1Ea5W3LEotwNui6zZbXU5
vjdObxXpKNaPVrQA1pLoxV7R8ereRYVM6R7d\u000aDzwvU/
    QcIZUVKT8UzKdSUpRvwSbvtRVMOkIj/
SSk5Z2lZUxSHhJWFX1AuVfPrOsU\u000aSMz2722sxx+
```

```
    xzUkPBiBCvNx0EFPdyFZHLMuyDOIhUgjEhY
BFWuS0NE11c3RhZmEg\
    u000aQWwtQmFzc2FtIDxtdXN0YWZhQHByaXZhY3lpbnRlcm5hdGlvbmFsLm9
yZz6JAj4E\
    u000aEwECACgCGwMGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheABQJWTyNCBQkHPXUZAAoJ

\u000aEGHOsziPIeoXOPUQAJsqFx+AZkwnq2bNmX1yw5sh5cCWdrBf69ivjPBV/m+
    uucDZ\u000awK
qb/u8k51Npm2FJ35/tJei1zEca7RZku0qhCczvmnJxKhhnS258NZyNn+o42HV0\
    u000agh4u/4djxoQ
UzW6rMTzHDaawzYGRybXNfv/SVuoCWWgrQabN6or6dezRI7fOVvCx\
    u000aXHV4REy45Bo11v5KaYIL
Cu9wUNGf6Qe2xPr0BhvUWV1f9qxNTNOll7UiQ6jZ+mnc\u000aJFuNJTjZC4E+
    tATvAt9t67fy6Ss/d
DEGEdX17ZsgxqNB4ieS+T51ixOQFL8+49Ok\u000aC/6WVP/
    mgNNKMTuI2GY0ZRedJqqFrSWd/8RdLT
+7rVkHvSOiUesEkI30qEWjD72a\
    u000a3a3auD9f7HGzKmMp1HwR8V6jua39bLo8sCx4ClhKd1NMu2n
veV5f138iJy1sdGDK\u000afDnaEiXK/qEPyydhjG2/zlDZFB/sdIG1I7qpgQ8+7
    C6LqFpbsKLSCBH9
bwZiTZGd\
    u000aLixRmLQJRZOsxpxIldAkbysXllBABeK99K9QuHEOq20Ht1n2CcGsgrkBykBBHg4Y
    \
u000aZ8QKXjZzi35zmmIhgcpkKzpgxgB4s3HcbNL+v1n0vx+8/zSQl/
    tp7qHH9FrAO8Vr\u000aE0h
zBKJLj14UDbvtMaZj9bbcZEG1nhEzml/URAOVrIF8v6AUEoluCGP60/u0tCtN\
    u000adXN0YWZhIEFs
LUJhc3NhbSA8bXVzYWxiYXNAbWlsbGl3YXlzLmluZm8+iQI+BBMB\
    u000aAgAoAhsDBgsJCAcDAgYVC
AIJCgsEFgIDAQIeAQIXgAUCVk8jQgUJBz11GQAKCRBh\u000azrM4jyHqF2sjD/4kw
    +8aMiSMSOxvE9
QOqOWSV6KGJgt/28fKl8rakut06EDBaSvE\
    u000a6PMbEka1JGgYO6fFPjwZNlHhfTnrzlKw/oP78Bh
mqGEJ++nq7NfwYD9bEBiRKFTS\u000adT/B92luX3wc7mQY+
    uRQBUc4eFZv1Htqmqoz1jGyVccEeZLa
2Rtf+fzKNWKz/LGQ\u000aMrI0E+ki/ryKC229RHkaQ1BcRHwK3xNEWxXuUg+
    CHaBJ7XIAVugx+u0vW
PhLqP4+\u000alBflws5z2OHVTyEdKj6XRroI19Uz6Z7UbH/
    GNwfb9UKd59M8L9o8ToTTpHQuvyvM\
u000ajjBf2VOpXe5R8uHsxKX3fl6ZHxffbT1JroVwjb3+KHJELksWi+CL5x+
```

```
    TgcM651Fq\u000a+Rgw
p6ticMbOAQ2qL4BY4dH9JbMGSZEdzHY8MIumxlp4/UzTUQdna/kcHajC9o4x\
    u000a5aT39jnWdVIeH
VJH2jmuVPmxLtre+5iwbDUU89KK5/dl+uIfSN/0NjbarTVwsVnb\
    u000a2T1HVBXBVSxt2FIK93IJbL
/CmbxbcG26shBORpYWa+6h3Cf0Z/fzyrmd0NyUP9Hy\u000a32IE3EIRRBY1/
    ohSFRPFI4U2i7t/s89
yFeUYwDJscHljE4Z/TwsnoFiuXhaVMABV\u000at/
    fjTUtkH8DReRJdJ0FpXF4XI5zgWHnD4rkbZeaP
0J94Ynr4J2hzZKZKZLkCDQRS\u000a1BUbARAAx9QwESbja6EMl/
    QUh5J630mIcUwdWavQWK2HhdNJc
SQhHV5ZlKCJxc2w\u000a95IoG52nrFBvPL83YEHc8qPoRUZ5W/dy+
    p6CX88lgE2Bd7njYqhys83Ovc
6HHTd4\
    u000aQXHERsjP9yOUIdeUndhynhX5zoaESBFrJNaP7e5YuaUx7YUJlycP77uJPozjKyA
    /\u
000a5KkvbOLDCG/tsa2c4JuDAANrXfozAbOi+
    r0vacS5CMp8I8TkXXHEU7JdvK0BmXi7\u000aSsHW+
m+a+n867Ab2NDg4kPChHLZqsLsuhMyyRU5LNoF5yfUKpnhG26UYTpmNqt85\
    u000aOi8ttrQflgrEWb
gnJrVQbf2mJPExpCrWEsz7ArGEznqEhNWvVQyopCiENgJWLZsi\
    u000acyQ5eRiEVg7rKr2M8O2VtaO
JxO6jxbeUrqXsr0pvhwYBj7/uhQRL1Xtfsj0mqtAO\u000a4h94PkN/
    mm2g75WBKZS1ejnoFoDiCqvq
f0CPnn80d9hiHTHw5vIQny9Fk6c2FU/E\u000aVPo2Xhjt2iS4++
    w4fUrNT7NBfj5ncGXNoYQE8Qky9
pFu33/y17vX7/a1P4leuCZn\u000aA2b8/
    FUVmRttNgZNlkCMtljJazQnybtXwCOvNZbfjW0Mh7ia5N
YEssNWnj2GiVyV\u000aOHzytBWCBxmLrL5AshyTevPAynzqxX+
    T8HaRPKQ1tudmJtDhqIsAEQEAAYk
CJQQY\u000aAQIADwIbDAUCVk8jwQUJBz11ogAKCRBhzrM4jyHqF/HSD/43RB/7
    GWH6Uj4S4oQM\u0
00aHOadHGcvevVslbo4jyly5rOHaDNcdoovtb3TE2AjQ9CvqfEpHPzatyY4TRu0IAT
    +\u000all4JfA
1XeShDtXtdn7kevSBv7dNRjBaV52YhS2ZRLvfttCelIzmEHYQdrdSfV24I\
    u000afgcSWFYjkavryFL
2wIbtlC3TYhO9CjQRnZJ1OQVnU736yBc1y7swO9V06HaKcWJy\
    u000axIixZzU9MhPpEdUG2i9BKVfh
xlXuCzmGsQw30Dw7/ZjiduFbZoEwVQnVNDDEW9FE\u000aQQbyfyidQNg5w/99
    B5idON4E1vheJRuI+
```

```
Ksi9FchJBJIg9hHWOjPjOFn12LhnIrr\
    u000aKC7WeXpuR9EpT2H7X6KpTcreEFGQwdoiXnM4vPwezl
ex0gu6Bc0PzLP26yLj101P\
    u000audwo4jwrYWlHKsGS7sZdrUqpJbhpcW6yCssvhjgzd15IN8vY8GP
HDh8kQXq9VyNl\u000aQ1BgcnQ3lO+qkYqBXERWqTQAW1G0oyG69zubp5WXQE+
    L4igLx/yaIBdPC3c+
fty5\u000aifIYTH1OCQ+
    REJIRuk8viz3YrK4cr22UUc4ffbsmmgdSlaXRCB23DoFQybORp4iU\u00
0ajGdSI83lBduL+dJFBuWjMxm1OdcGtlwN57GAT9BJeLxQZGzef+GbuDEx+6VcwxIk\
    u000aBpBgsW5
Yp7rXHYZJXHW/37IsBw==\u000a=QMUh\u000a-----END PGP PUBLIC KEY BLOCK
    -----\u000
a\u000a-----BEGIN PGP SIGNED MESSAGE-----\u000aHash: SHA1\u000a\
    u000aEthereu
m address: 0xdb57ccafaa5198089af9c69499e8d3d67cb463a3\u000a-----
    BEGIN PGP SIGNA
TURE-----\u000aVersion: GnuPG v1\u000a\u000aiQIcBAEBAgAGBQJXC+
    liAAoJEGHOsziPI
eoXo3gP/2LopkFRWTjPjrhBZa/BUThc\u000aFRD/B+
    D2iV2OPZATRbQa9SfyzZVWDgLJQsv5ujg1ZH
tSYn3PwRvdHDpG5PHiZ2uU\u000abE38PtVV1jt47NZT/LSxoRVfu9J404jXo/3
    HLUoGzzTL0jELZfU
EINt7Twhn1OL9\u000ajwmsbLSBsMhFvl8MwfE9ZLs/Bp+hhmzkzWBISUZZX10HfBL
    /3hFc3Y9A7XuT
XuHv\u000aV2Od5a63dg+vDjnb5Q/PvcOsNySeaSU/JXlrcS7G14EYa1vseepzIbkG+
    J5fPUJP\u00
0aH0+3P0q9pA/eNrFfUQYD5oK3JTDRn3n2A0OnIvzSQfgD4Eux0RQ4l9O5mtMlnlOe\
    u000aikQ9XSf
6LYUdFEoJbE/pDWDVNwgCCuzCuP1T+aiiHubPEejlpnjlbfGmd4Jh3zDm\
    u000aZE5gFR3/dTy+Tiwy
/0Wx2BECG2R3RNtM6WvG/zlsv2hJMt5avftssY/pCVG0Qyuk\u000aoNcekq+/
    jfqgGTQHQc0Icss7u
3P5WwL9g9Mu+0kw9NKOuOT+ktOgjNkdjv/a8pLW\
    u000aZAqAWSsD1QnNqdVwNLv44X3wu7Xi62Bv9h
TnCID9aKeXvyv/nyq7ltHWiXep0cLH\
    u000aN0NL2ZwZzEilD4xuTXDhw0hHqZQwIcrrWCjScHamkJK
4kewmFrlByZYXddGJbXEG\u000aAWERulViEAVjzmph2WtT\u000a=4KYh\u000a
    -----END PGP
SIGNATURE-----\u000a
p22
```

s.

A.8 trustery/res/pgp_attribute_2.pickle

```
(dp0
Vhas_proof
p1
I00
sS'_event_type'
p2
S'AttributeAdded'
p3
sVdatahash
p4
S'foo'
p5
sS'signatures_status'
p6
(dp7
S'status'
p8
(dp9
S'valid'
p10
I0
sS'invalid'
p11
I0
ssS'signatures'
p12
(lp13
ssVattributeID
p14
I10
sVattributeType
p15
S'pgp-key'
p16
sVowner
p17
```

```
S'db57ccafaa5198089af9c69499e8d3d67cb463a3'
p18
sVidentifier
p19
S'foo\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
    x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
p20
sVdata
p21
S'foo'
p22
s.
```

A.9 trustery/res/pgp_attribute_3.pickle

```
(dp0
Vhas_proof
p1
I01
sS'_event_type'
p2
S'AttributeAdded'
p3
sVdatahash
p4
S'foo'
p5
sS'signatures_status'
p6
(dp7
S'status'
p8
(dp9
S'valid'
p10
I0
sS'invalid'
p11
I0
ssS'signatures'
```

```
p12
(lp13
ssVattributeID
p14
I11
sVattributeType
p15
S'pgp-key'
p16
sVowner
p17
S'db57ccafaa5198089af9c69499e8d3d67cb463a3'
p18
sVidentifier
p19
S'foo\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
    x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
p20
sVdata
p21
S'foo'
p22
s.
```

A.10 trustery/pytrustery/__init__.py

```
"""Console client and API for Trustery that communicates with the
    Ethereum JSON RPC."""
```

A.11 trustery/pytrustery/console.py

```
"""Console application for Trustery."""

import atexit
import logging
import time

import click
```

```
from trustery.consoleutil import echo_attribute_block
from trustery.events import Events
from trustery.transactions import Transactions
from trustery import userconfig


class StrParamType(click.ParamType):
    """Click parameter type that converts data using str()."""
    name = 'STR'

    def convert(self, value, param, ctx):
        return str(value)

STR = StrParamType()


@click.group()
def cli():
    """Ethereum-based identity system."""
    # Prevent the requests module from printing INFO logs to the
        console.
    logging.getLogger("requests").setLevel(logging.WARNING)

    # Save the configuration on exit.
    atexit.register(userconfig.config.write)


@cli.command()
@click.option('--attributetype', prompt=True, type=STR)
@click.option('--has_proof', prompt=True, type=bool)
@click.option('--identifier', prompt=True, type=STR)
@click.option('--data', prompt=True, type=STR)
@click.option('--datahash', prompt=True, type=STR)
def rawaddattribute(attributetype, has_proof, identifier, data,
    datahash):
    """(Advanced) Manually add an attribute to your identity."""
    transactions = Transactions()
    transactions.add_attribute(attributetype, has_proof, identifier,
         data, datahash)
```

```
    click.echo()
    click.echo("Transaction sent.")


@cli.command()
@click.option('--attributeid', prompt=True, type=int)
@click.option('--expiry', prompt=True, type=STR)
def rawsignattribute(attributeid, expiry):
    """(Advanced) Manually sign an attribute about an identity."""
    transactions = Transactions()
    transactions.sign_attribute(attributeid, expiry)

    click.echo()
    click.echo("Transaction sent.")


@cli.command()
@click.option('--signatureid', prompt=True, type=STR)
def rawrevokeattribute(signatureid):
    """(Advanced) Manaully revoke your signature of an attribute."""
    transactions = Transactions()
    transactions.revoke_signature(signatureid)

    click.echo()
    click.echo("Transaction sent.")


@cli.command()
@click.option('--attributetype', prompt='Attribute type', help='
    Attribute type', type=STR)
@click.option('--identifier', prompt='Attribute identifier', help='
    Attribute identifier', type=STR)
@click.option('--data', prompt='Attribute data', default='', help='
    Attribute data', type=STR)
def add(attributetype, identifier, data):
    """Add an attribute to your identity."""
    transactions = Transactions()
    transactions.add_attribute_with_hash(attributetype, False,
        identifier, data)
```

```
    click.echo()
    click.echo("Transaction sent.")


@cli.command()
@click.option('--attributetype', prompt='Attribute type', help='
    Attribute type', type=STR)
@click.option('--identifier', prompt='Attribute identifier', help='
    Attribute identifier', type=STR)
@click.option('--data', prompt='Attribute data', default='', help='
    Attribute data', type=unicode)
def ipfsadd(attributetype, identifier, data):
    """Add an attribute to your identity over IPFS."""
    transactions = Transactions()
    transactions.add_attribute_over_ipfs(attributetype, False,
        identifier, data)

    click.echo()
    click.echo("Transaction sent.")


@cli.command()
@click.option('--attributeid', prompt='Attribute ID', help='
    Attribute ID', type=int)
@click.option('--expires', prompt='Signature days to expire',
    default=365, help='Signature days to expire', type=int)
def sign(attributeid, expires):
    """Sign an attribute."""
    transactions = Transactions()

    expiry = int(time.time()) + expires * 60 * 60 * 24
    transactions.sign_attribute(attributeid, expiry)

    click.echo()
    click.echo("Transaction sent.")


@cli.command()
@click.option('--signatureid', prompt='Signature ID', help='
    Signature ID', type=int)
```

```
def revoke(signatureid):
    """Revoke one of your signatures."""
    transactions = Transactions()
    transactions.revoke_signature(signatureid)

    click.echo()
    click.echo("Transaction sent.")


@cli.command()
@click.option('--address', prompt='Ethereum address', help='
    Ethereum address', type=STR)
def trust(address):
    """Trust an Ethereum address."""
    click.echo()

    if userconfig.is_trusted(address):
        click.echo("Address " + address + " is already trusted.")
    else:
        userconfig.trust(address)
        click.echo("Address " + address + " trusted.")


@cli.command()
@click.option('--address', prompt='Ethereum address', help='
    Ethereum address', type=STR)
def untrust(address):
    """Untrust an Ethereum address."""
    click.echo()

    if not userconfig.is_trusted(address):
        click.echo("Address " + address + " is already not trusted
            .")
    else:
        userconfig.untrust(address)
        click.echo("Address " + address + " untrusted.")


@cli.command()
def trusted():
```

```
    """View the list of trusted Ethereum addresses."""
    for address in userconfig.get_trusted():
        click.echo(address)


@cli.command()
@click.option('--attributeid', prompt='Attribute ID', help='
    Attribute ID', type=int)
def retrieve(attributeid):
    """Retrieve an attribute."""
    events = Events()
    attribute = events.retrieve_attribute(attributeid)

    if attribute is None:
        click.echo("No such attribute.")
        return

    click.echo()

    echo_attribute_block(attribute)
    click.echo()

    if 'proof_valid' in attribute:
        click.echo("Proof status for attribute ID #" + str(attribute
            ['attributeID']) + ':')
        if attribute['proof_valid'] is None:
            click.echo("\tUnknown")
        elif attribute['proof_valid']:
            click.echo("\tValid")
        else:
            click.echo("\tINVALID")

        click.echo()

    click.echo("Signatures for attribute ID #" + str(attribute['
        attributeID']) + ':')
    for signature in attribute['signatures_status']['signatures']:
        sig_line = "\t#" + str(signature['signatureID'])

        if signature['revocation']:
```

```
            sig_line += " [revoked]"
        elif signature['expired']:
            sig_line += " [expired]"
        elif signature['valid']:
            sig_line += " [valid]"

        sig_line += " by " + signature['signer']
        sig_line += (" [trusted]" if userconfig.is_trusted(attribute
            ['owner']) else " [untrusted]")
        click.echo(sig_line)

    click.echo()
    click.echo("--ATTRIBUTE DATA:")
    click.echo(attribute['data'])


@cli.command()
@click.option('--attributetype', help='Attribute type', type=STR)
@click.option('--identifier', help='Attribute identifier', type=STR
    )
@click.option('--owner', help='Attribute owner', type=STR)
def search(attributetype, identifier, owner):
    """Search for attributes."""
    # Pad identifiers with zeros.
    if identifier is not None:
        if identifier.startswith('0x'): # Hex data.
            identifier = identifier.ljust(66, '0')
        else:
            identifier = identifier.ljust(32, '\x00')

    events = Events()
    attributes = events.filter_attributes(None, owner, identifier)

    for attribute in attributes:
        if attributetype is not None and attributetype != attribute
            ['attributeType']:
            continue

        signatures_status = events.get_attribute_signatures_status(
            attribute['attributeID'])
```

```
        echo_attribute_block(attribute, signatures_status)
        click.echo()


@cli.command()
@click.option('--keyid', prompt='Key ID', help='Key ID', type=STR)
def ipfsaddpgp(keyid):
    """Add a PGP key attribute to your identity over IPFS."""
    transactions = Transactions()
    click.echo()

    try:
        transactions.add_pgp_attribute_over_ipfs(keyid)
    except ValueError as e:
        click.echo("Error: " + e.message)
        return

    click.echo("Transaction sent.")
```

A.12 trustery/pytrustery/consoleutil.py

```
"""Console utility functions."""

import click

from trustery import userconfig


def echo_attribute_block(attribute, signatures_status=None):
    """Echo a console block representing basic data about the
        attribute."""
    if signatures_status is None and 'signatures_status' in
        attribute:
        signatures_status = attribute['signatures_status']

    # Encode attribute identifier as hex if it contains non-ASCII
        characters.
    if not all(ord(c) < 128 for c in attribute['identifier']):
        attribute['identifier'] = '0x' + attribute['identifier'].
```

```
            rstrip('\x00').encode('hex')

    click.echo("Attribute ID #" + str(attribute['attributeID']) +
        ':')
    click.echo("\tType: " + attribute['attributeType'])
    click.echo("\tOwner: " + attribute['owner']
        + (" [trusted]" if userconfig.is_trusted(attribute['owner'])
             else " [untrusted]"))
    click.echo("\tIdentifier: " + attribute['identifier'])

    if signatures_status is not None:
        valid_signatures = signatures_status['status']['valid']
        click.echo("\t[" + str(valid_signatures) + " valid signature
            "
            + ("]" if valid_signatures == 1 else "s]"))
```

A.13 trustery/pytrustery/ethapi.py

```
"""Interface for Ethereum client and Trustery contract."""

import json
import os

import eth_rpc_client
from rlp.utils import encode_hex

import trustery

# Trustery contract constants.
TRUSTERY_DEFAULT_ADDRESS = '0
    xe40b13f40e4f9e9f85a0f3ffbb212a4ff1547c6b'
TRUSTERY_ABI = json.load(open(os.path.join(os.path.dirname(trustery
    .__file__), 'trustery_abi.json')))

# Ethereum client interface.
ethclient = eth_rpc_client.Client(host='127.0.0.1', port='8545')


def encode_api_data(data):
    """
```

```
    Prepare arbitrary data to be send to the Ethereum client via the
        API.

    data: the data.
    """
    if data is None:
        return None
    elif type(data) == str and data.startswith('0x'):
        # Return data if it is already hex-encoded.
        return data
    elif type(data) in [bool, int]:
        # Use native hex() to encode non-string data has encode_hex
            () does not support it.
        return hex(data)
    elif type(data) == long:
        # Use native hex() to encode long.
        encoded = hex(data)
        if encoded[-1:] == 'L':
            # Remove the trailing 'L' if found.
            encoded = encoded[:-1]
        return encoded
    else:
        # Encode data using encode_hex(), the recommended way to
            encode Ethereum data.
        return '0x' + encode_hex(data)
```

A.14 trustery/pytrustery/gpgapi.py

```
"""Interface for GPG."""

import shutil
import tempfile

import gnupg

# Initialise GPG interface.
gpgclient = gnupg.GPG()


class TempGPG(object):
```

```
    """A class for creating a temporary GPG instance seperate from
       the user's GPG home directory."""

    def __init__(self):
        """Initialise the temporary GPG instance."""
        # Securely create a temporary directory.
        self.tempdir = tempfile.mkdtemp()

        # Initialise temporary GPG interface.
        self.gpgclient = gnupg.GPG(gnupghome=self.tempdir)

    def destroy(self):
        """Destroy the temporary GPG instance."""
        shutil.rmtree(self.tempdir)


def generate_pgp_attribute_data(keyid, address):
    """
    Generate the data field (the PGP public key and cryptographic
       proof) for a PGP attribute.

    Returns a (fingerprint, data) tuple.

    keyid: the ID of the PGP key.
    address: Ethereum address to generate cryptographic proof for.
    """
    # Export public key.
    public_key = gpgclient.export_keys(keyid, minimal=True)

    # Use temporary GPG interface to check that only one key has
       been exported and to get its fingerprint.
    tempgpg = TempGPG()
    try:
        # Import public key.
        import_results = tempgpg.gpgclient.import_keys(public_key)

        # Check that only one key has been imported.
        if import_results.count != 1:
            raise ValueError("invalid PGP key ID specified")
```

```
        # Get key fingerprint.
        fingerprint = str(import_results.fingerprints[0])
    finally:
        # Destroy temporary GPG interface.
        tempgpg.destroy()

    # Generate cryptographic proof signature.
    proof = gpgclient.sign('Ethereum address: ' + address, keyid=
        fingerprint).data

    # Check that a proof was actually generated.
    if not proof:
        raise ValueError("a PGP key was specified that does not have
             a corresponding secret key")

    # Concatenate public key and cryptographic proof.
    data = public_key + '\n' + proof

    # Return data and fingerprint.
    return (fingerprint, data)


def process_proof(data):
    """
    Process cryptographic proof of PGP attribute data.

    Returns a (Ethereum address, PGP key fingerprint) tuple the
        proof is associated with if the signature is valid,
        otherwise False.

    data: the PGP attribute data.
    """
    # Extract key, signature and Ethereum address.
    key = ''
    signature = ''
    address = ''
    key_mode = False
    signature_mode = False
    for line in data.split('\n'):
        line = line.strip()
```

```
        if line == '-----END PGP PUBLIC KEY BLOCK-----':
            key_mode = False
            key += line + '\n'
        elif line == '-----BEGIN PGP PUBLIC KEY BLOCK-----' or
            key_mode:
            if not key_mode:
                key_mode = True
            key += line + '\n'
        elif line == '-----END PGP SIGNATURE-----':
            signature_mode = False
            signature += line + '\n'
        elif line == '-----BEGIN PGP SIGNED MESSAGE-----' or
            signature_mode:
            if not signature_mode:
                signature_mode = True
            if line.startswith('Ethereum address: '):
                address = line[len('Ethereum address: '):]
            signature += line + '\n'

    # Create temporary keychain and import key.
    tempgpg = TempGPG()
    import_results = tempgpg.gpgclient.import_keys(key)

    verified = tempgpg.gpgclient.verify(signature)

    tempgpg.destroy()

    if not verified:
        return False

    return (address, verified.fingerprint)
```

A.15 trustery/pytrustery/ipfsapi.py

```
"""Interface for IPFS."""

import ipfsApi


# Initialise IPFS interface.
```

```
ipfsclient = ipfsApi.Client('127.0.0.1', 5001)
```

A.16 trustery/pytrustery/trustery_abi.json

```
[
    {
        "constant": false,
        "inputs": [
            {
                "name": "signatureID",
                "type": "uint256"
            }
        ],
        "name": "revokeSignature",
        "outputs": [
            {
                "name": "revocationID",
                "type": "uint256"
            }
        ],
        "type": "function"
    },
    {
        "constant": false,
        "inputs": [
            {
                "name": "attributeID",
                "type": "uint256"
            },
            {
                "name": "expiry",
                "type": "uint256"
            }
        ],
        "name": "signAttribute",
        "outputs": [
            {
                "name": "signatureID",
                "type": "uint256"
            }
```

```
    ],
    "type": "function"
},
{
    "constant": true,
    "inputs": [
        {
            "name": "",
            "type": "uint256"
        }
    ],
    "name": "signatures",
    "outputs": [
        {
            "name": "signer",
            "type": "address"
        },
        {
            "name": "attributeID",
            "type": "uint256"
        },
        {
            "name": "expiry",
            "type": "uint256"
        }
    ],
    "type": "function"
},
{
    "constant": false,
    "inputs": [
        {
            "name": "attributeType",
            "type": "string"
        },
        {
            "name": "has_proof",
            "type": "bool"
        },
        {
```

```
            "name": "identifier",
            "type": "bytes32"
        },
        {
            "name": "data",
            "type": "string"
        },
        {
            "name": "datahash",
            "type": "string"
        }
    ],
    "name": "addAttribute",
    "outputs": [
        {
            "name": "attributeID",
            "type": "uint256"
        }
    ],
    "type": "function"
},
{
    "constant": true,
    "inputs": [
        {
            "name": "",
            "type": "uint256"
        }
    ],
    "name": "attributes",
    "outputs": [
        {
            "name": "owner",
            "type": "address"
        },
        {
            "name": "attributeType",
            "type": "string"
        },
        {
```

```
            "name": "has_proof",
            "type": "bool"
        },
        {
            "name": "identifier",
            "type": "bytes32"
        },
        {
            "name": "data",
            "type": "string"
        },
        {
            "name": "datahash",
            "type": "string"
        }
    ],
    "type": "function"
},
{
    "constant": true,
    "inputs": [
        {
            "name": "",
            "type": "uint256"
        }
    ],
    "name": "revocations",
    "outputs": [
        {
            "name": "signatureID",
            "type": "uint256"
        }
    ],
    "type": "function"
},
{
    "anonymous": false,
    "inputs": [
        {
            "indexed": true,
```

```
            "name": "attributeID",
            "type": "uint256"
        },
        {
            "indexed": true,
            "name": "owner",
            "type": "address"
        },
        {
            "indexed": false,
            "name": "attributeType",
            "type": "string"
        },
        {
            "indexed": false,
            "name": "has_proof",
            "type": "bool"
        },
        {
            "indexed": true,
            "name": "identifier",
            "type": "bytes32"
        },
        {
            "indexed": false,
            "name": "data",
            "type": "string"
        },
        {
            "indexed": false,
            "name": "datahash",
            "type": "string"
        }
    ],
    "name": "AttributeAdded",
    "type": "event"
},
{
    "anonymous": false,
    "inputs": [
```

```
            {
                "indexed": true,
                "name": "signatureID",
                "type": "uint256"
            },
            {
                "indexed": true,
                "name": "signer",
                "type": "address"
            },
            {
                "indexed": true,
                "name": "attributeID",
                "type": "uint256"
            },
            {
                "indexed": false,
                "name": "expiry",
                "type": "uint256"
            }
        ],
        "name": "AttributeSigned",
        "type": "event"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "name": "revocationID",
                "type": "uint256"
            },
            {
                "indexed": true,
                "name": "signatureID",
                "type": "uint256"
            }
        ],
        "name": "SignatureRevoked",
        "type": "event"
```

```
    }
]
```

A.17 trustery/pytrustery/events.py

```
"""API for retrieving Trustery events."""

import time

from ethereum import abi
from ethereum import processblock
from ethereum.utils import big_endian_to_int
from rlp.utils import decode_hex

from ipfsapi import ipfsclient
from gpgapi import process_proof
from ethapi import TRUSTERY_ABI
from ethapi import TRUSTERY_DEFAULT_ADDRESS
from ethapi import ethclient
from ethapi import encode_api_data


class Events(object):
    """API for retrieving Trustery events."""
    def __init__(self, address=TRUSTERY_DEFAULT_ADDRESS):
        """
        Initialise events retriever.

        address: the Ethereum Trustery contract address.
        """
        self.address = address

        # Initialise contract ABI.
        self._contracttranslator = abi.ContractTranslator(
            TRUSTERY_ABI)

    def _get_event_id_by_name(self, event_name):
        """
        Get the ID of an event given its name.
```

```
    event_name: the name of the event.
    """
    for event_id, event in self._contracttranslator.event_data.
        iteritems():
        if event['name'] == event_name:
            return event_id

def _get_logs(self, topics, event_name=None):
    """
    Get logs (events).

    topics: a list of topics to search for.
    event_name: the name of the event.
    """
    # Set the event topic to the event ID if the event name is
        specified.
    if event_name is None:
        event_topic = None
    else:
        event_topic = self._get_event_id_by_name(event_name)

    # Prepent the event type to the topics.
    topics = [event_topic] + topics
    # Encode topics to be sent to the Ethereum client.
    topics = [encode_api_data(topic) for topic in topics]

    # Get logs from Ethereum client.
    logs = ethclient.get_logs(
        from_block='earliest',
        address=self.address,
        topics=topics,
    )

    # Decode logs using the contract ABI.
    decoded_logs = []
    for log in logs:
        logobj = processblock.Log(
            log['address'][2:],
            [big_endian_to_int(decode_hex(topic[2:])) for topic
                in log['topics']],
```

```
            decode_hex(log['data'][2:])
        )
        decoded_log = self._contracttranslator.listen(logobj,
            noprint=True)
        decoded_logs.append(decoded_log)

    return decoded_logs

def filter_attributes(self, attributeID=None, owner=None,
    identifier=None):
    """
    Filter and retrieve attributes.

    attributeID: the ID of the attribute.
    owner: the Ethereum address that owns the attributes.
    identifier: the identifier of the attribute.
    """
    return self._get_logs([attributeID, owner, identifier],
        event_name='AttributeAdded')

def filter_signatures(self, signatureID=None, signer=None,
    attributeID=None):
    """
    Filter and retrieve signatures.

    signatureID: the ID of the signature.
    signer: the Ethereum address that owns the signature.
    attributeID: the ID of the attribute.
    """
    return self._get_logs([signatureID, signer, attributeID],
        event_name='AttributeSigned')

def filter_revocations(self, revocationID=None, signatureID=None
    ):
    """
    Filter and retrieve revocations.

    revocationID: the ID of the revocation.
    attributeID: the ID of the attribute.
    """
```

```
    return self._get_logs([revocationID, signatureID],
        event_name='SignatureRevoked')

def get_attribute_signatures_status(self, attributeID):
    """
    Get all the signatures of an attribute and check whether
        they have been revoked or expired.

    attributeID: the ID of the attribute.

    Returns a dictionaries representing the signatures status of
         the attribute:
        dict['status']['valid']: number of valid signatures.
        dict['status']['invalid']: number of invalid signatures.
        dict['signatures']: a list of signatures.

        For a signature index s:
            dict['signatures'][s]: dictionary representing the
                signature, plus the additional status keys below.
            dict['signatures'][s]['expired']: True if the
                signature is expired.
            dict['signatures'][s]['revocation']: dictionary
                representing the signature's revocation if it was
                 revoked, otherwise False.
            dict['signatures'][s]['valid']: True if the signature
                 is valid.
    """
    # Prepare return dictionary
    signatures = []
    status = {
        'valid': 0,
        'invalid': 0
    }
    signatures_status = {
        'status': status,
        'signatures': signatures
    }

    # Filter signatures for the specified attribute
    rawsignatures = self.filter_signatures(attributeID=
```

```
            attributeID)

        # Process signatures
        for rawsignature in rawsignatures:
            signature = {}

            # Add signature properties to the dictionary
            signature.update(rawsignature)

            # Check if expired
            signature['expired'] = time.time() > signature['expiry']

            # Check if revoked
            rawrevocations = self.filter_revocations(signatureID=
                signature['signatureID'])
            if len(rawrevocations) > 0:
                signature['revocation'] = rawrevocations
            else:
                signature['revocation'] = False

            # Check if valid
            if not signature['expired'] and not signature['revocation
                ']:
                signature['valid'] = True
                status['valid'] += 1
            else:
                signature['valid'] = False
                status['invalid'] += 1

            signatures.append(signature)

        return signatures_status

    def retrieve_attribute(self, attributeID):
        """Get an attribute, its status and signatures status,
            downloading off-blockchain data if necessary.

        attributeID: the ID of the attribute.

        Returns a dictionary representing all of the attribute's
```

```
        properties:
        dict: dictionary representing the attribute itself, plus
            the additional status keys below.
        dict['signatures_status']: the signatures status of the
            attribute.
    """
    rawattributes = self.filter_attributes(attributeID=
        attributeID)

    if not rawattributes:
        return None

    attribute = rawattributes[0]
    attribute['signatures_status'] = self.
        get_attribute_signatures_status(attributeID)

    # Download IPFS data if necessary.
    if attribute['data'].startswith('ipfs-block://'):
        ipfs_key = attribute['data'][len('ipfs-block://'):]
        attribute['data'] = ipfsclient.block_get(ipfs_key)

    # Verify PGP proof.
    if attribute['attributeType'] == 'pgp-key':
        attribute['proof_valid'] = self.
            verify_attribute_pgp_proof(attribute)

    # Set proof validity to unknown if the attribute has a proof
         but we did not know how to process it.
    if attribute['has_proof'] and 'proof_valid' not in attribute
        :
        attribute['proof_valid'] = None

    return attribute

def verify_attribute_pgp_proof(self, attribute):
    """
    Verify the PGP proof of an attribute.

    Return True if valid, False if invalid, or None is unknown
        because a proof is unspecified.
```

```
        attribute: the attribute dictionary.
        """
        # Don't check proof if one was not specified.
        if not attribute['has_proof']:
            return None # Unknown validity.

        # Process the proof.
        proof = process_proof(attribute['data'])

        if not proof:
            return False

        (proof_address, proof_fingerprint) = proof
        if (
            # Check that the fingerprints match.
            proof_fingerprint.decode('hex') == attribute['identifier
                '].rstrip('\x00')
            # Check that the Ethereum addresses match.
            and proof_address == '0x' + attribute['owner']
            ):
            return True

        return False
```

A.18 trustery/pytrustery/transactions.py

```
"""API for making Trustery tranactions."""

import io

from ethereum import abi

from ipfsapi import ipfsclient
from gpgapi import generate_pgp_attribute_data
from ethapi import TRUSTERY_ABI
from ethapi import TRUSTERY_DEFAULT_ADDRESS
from ethapi import ethclient
from ethapi import encode_api_data
```

```
class Transactions(object):
    """API for making Trustery tranactions."""
    def __init__(self, from_address=None, to_address=
        TRUSTERY_DEFAULT_ADDRESS):
        """
        Initialise transactions.

        from_address: the Ethereum address transactions should be
            sent from.
        to_address: the Ethereum Trustery contract address.
        """
        if from_address is None:
            # Use the first Ethereum account address if no from
                address is specified.
            self.from_address = ethclient.get_accounts()[0]
        else:
            self.from_address = from_address
        self.to_address = to_address

        # Initialise contract ABI.
        self._contracttranslator = abi.ContractTranslator(
            TRUSTERY_ABI)

    def _send_transaction(self, data):
        """
        Send a transaction.

        data: the transactions data.
        """
        return ethclient.send_transaction(
            _from=self.from_address,
            to=self.to_address,
            data=encode_api_data(data),
            gas=2000000, # TODO deal with gas limit more sensibly
        )

    def add_attribute(self, attributetype, has_proof, identifier,
        data, datahash):
        """
```

```
        Send a transaction to add an identity attribute.

        attributetype: the type of address.
        has_proof: True if the attribute has a cryptographic proof,
            otherwise False.
        identifier: the indexable identifier of the attribute.
        data: the data of the attribute.
        datahash: the Keccak hash of the data of the attribute if it
             is stored off-blockchain.
        """
        args = [attributetype, has_proof, identifier, data, datahash
            ]
        data = self._contracttranslator.encode('addAttribute', args)
        return self._send_transaction(data)

    def add_attribute_with_hash(self, attributetype, has_proof,
        identifier, data):
        """
        Send a transaction to add an identity attribute,
            automatically calculating its datahash if the data is
            stored remotely.

        attributetype: the type of address.
        has_proof: True if the attribute has a cryptographic proof,
            otherwise False.
        identifier: the indexable identifier of the attribute.
        data: the data of the attribute.
        """
        datahash = '' # TODO calculate hash for remotely stored data
        return self.add_attribute(attributetype, has_proof,
            identifier, data, datahash)

    def add_attribute_over_ipfs(self, attributetype, has_proof,
        identifier, data):
        """
        Send a transaction to add an identity attribute, storing the
             data on IPFS first.

        attributetype: the type of address.
        has_proof: True if the attribute has a cryptographic proof,
```

```
        otherwise False.
    identifier: the indexable identifier of the attribute.
    data: the data of the attribute.
    """
    #  Store the data as an IPFS block and get its key.
    ipfs_key = ipfsclient.block_put(io.StringIO(data))['Key']

    # Generate Trustery-specific URI for the IPFS block.
    ipfs_uri = 'ipfs-block://' + ipfs_key

    # Add the attribute.
    self.add_attribute(attributetype, has_proof, identifier,
        ipfs_uri, datahash='')

def add_pgp_attribute_over_ipfs(self, keyid):
    """
    Send a transaction to add an identity PGP attribute, storing
         the attribute data on IPFS.

    keyid: the ID of the PGP key.
    """
    # Generate PGP attribute data and get identifier (
        fingerprint).
    (fingerprint, data) = generate_pgp_attribute_data(keyid,
        self.from_address)

    # Express identifier as fingerprint in binary format.
    identifier = fingerprint.decode('hex')

    self.add_attribute_over_ipfs(
        attributetype='pgp-key',
        has_proof=True,
        identifier=identifier,
        data=data,
    )

def sign_attribute(self, attributeID, expiry):
    """
    Send a transaction to sign an identity attriute.
```

```
        attributeID: the ID of the attribute.
        expiry: the expiry time of the attriute.
        """
        args = [attributeID, expiry]
        data = self._contracttranslator.encode('signAttribute', args
            )
        return self._send_transaction(data)

    def revoke_signature(self, signatureID):
        """
        Send a transaction to revoke a signature.

        signatureID: the ID of the signature.
        """
        args = [signatureID]
        data = self._contracttranslator.encode('revokeSignature',
            args)
        return self._send_transaction(data)
```

A.19 trustery/pytrustery/userconfig.py

```
"""Local user configuration management."""

import os

from appdirs import user_config_dir
from configobj import ConfigObj

# Create configuration directory in case it does not exist.
try:
    os.makedirs(user_config_dir('trustery'))
except OSError:
    if not os.path.isdir(user_config_dir('trustery')):
        raise

# Determine cross-platform configuration file path.
configfile = os.path.join(user_config_dir('trustery'), 'config.ini
    ')

# Create configuration object.
```

```
config = ConfigObj(configfile)

# Initialise configuration.
if 'truststore' not in config:
    config['truststore'] = {}


def trust(address):
    """
    Add address to the trust store.

    address: the address to add.
    """
    config['truststore'][address] = True


def untrust(address):
    """
    Remove address from the trust store.

    address: the address to remove.
    """
    del config['truststore'][address]


def is_trusted(address):
    """
    Return True if an address is in the trust store, otherwise False
        .

    address: the address to check.
    """
    return address in config['truststore'] and config['truststore'][
        address]


def get_trusted():
    """Return a list of trusted Ethereum addresses."""
    return config['truststore'].keys()
```