THE HEBREW
UNIVERSITY
OF JERUSALEM

Advanced Topics In Online Privacy and Cybersecurity

Public Key Infrastructure

06/06/2022

Amit Roth

API

We have 3 different modules in our program, but the CA module is fully controlled by the Entity module, as we chose to wrap every CA with an Entity class. The modules uses sockets to communicate and you can add calls in main.py.

We have a an assumption int the project that who ever requests to be a ca nor requests a certification, we will give without performing any checks.

- entity/server.py - the server wraps the Entity class and lets us interact with it using sockets. We will describe each call:
    1. create_entity - creates an entity object with a given name
    2. sign - sign a message with the entity's private key
    3. request_cert - returns the entity's cert
    4. make_root_ca - turns the entity into a root ca
    5. is_ca - returns a boolean
    6. get_cert - returns the entity's certification
    7. pk - returns the entity's public key
    8. generate_cert - generate and signs a certificate for the entity, need to provide a signer
    9. revocate - revocates a cert, more details about revocation below
    10. check_if_revocated - returns a boolean

- validator/server.py - the server wraps the Validator class
    1. create_validator - creates a validator object
    2. verify - verify specific message with signature and public key
    3. add_root_ca - adds a ca as a root ca to the list of the validator
    4. validate - checks if a cert is validated by a root ca or a child of him

Design

entity.py - An entity can be anything in our network, a little e-commerce site or the root ca of the network. The Entity class and the CA class have a “composition” relations. If an entity is also a ca (in our case, anyone can be without any validation) then it has a field ca which points for a CA object.

ca.py - CA can generate a cerificate for the entity and can sign it with his own secret key. You can also add certificates to the revocation list of a ca.

validator.py - Validator class saves a list of root CAs and can recursively check if an entity is a ca or not. Also can check if any of the CAs from the entity path to the root has been revocated the cert or not.

All of the signing process is made with RSA.

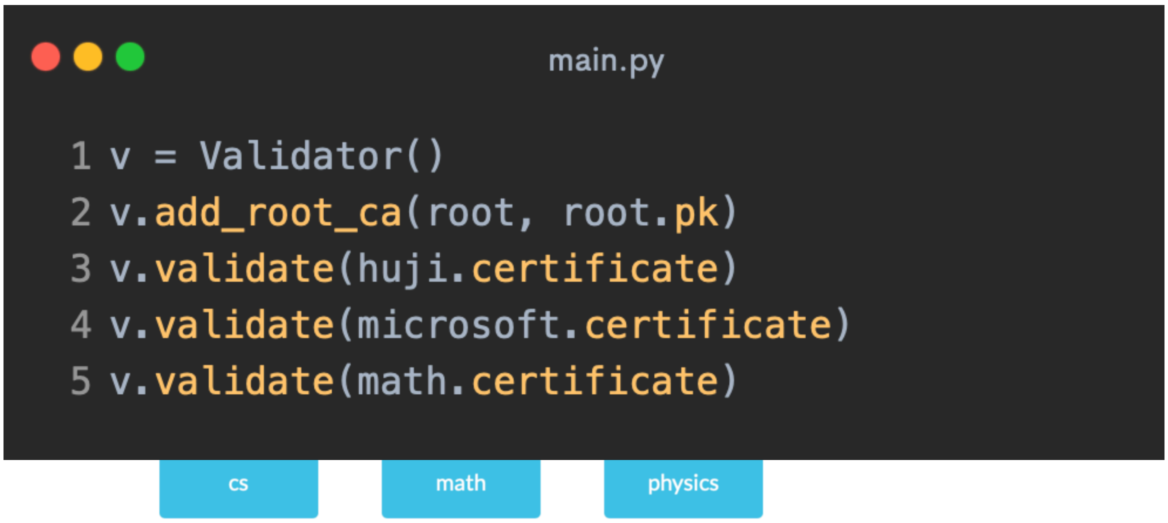
Revocation

Each CA saves a list of the certificates that he has been revocated.
When ever a validator validates a cert, it checks on all of the CAs in the list if the certificate is in their revocation list.
If a certificate is expired it will be removed from the revocation list.

Example

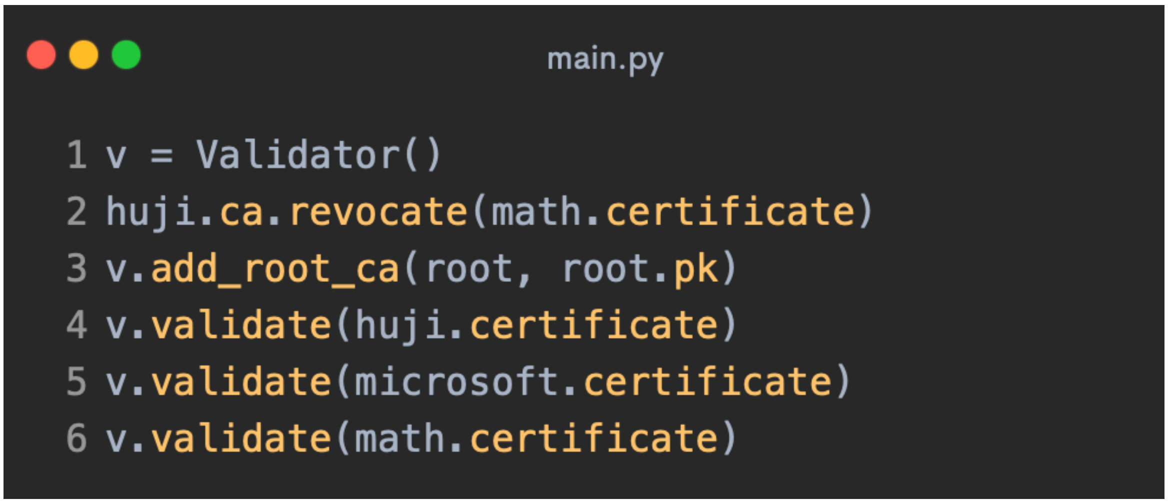
We will create the following tree, which the nodes are CAs and the leaves are regular entities.

Now when checking on a validator which root has been added as a root ca, all of the entities will be verified successfully.



```python
v = Validator()
v.add_root_ca(root, root.pk)
v.validate(huji.certificate)
v.validate(microsoft.certificate)
v.validate(math.certificate)
```

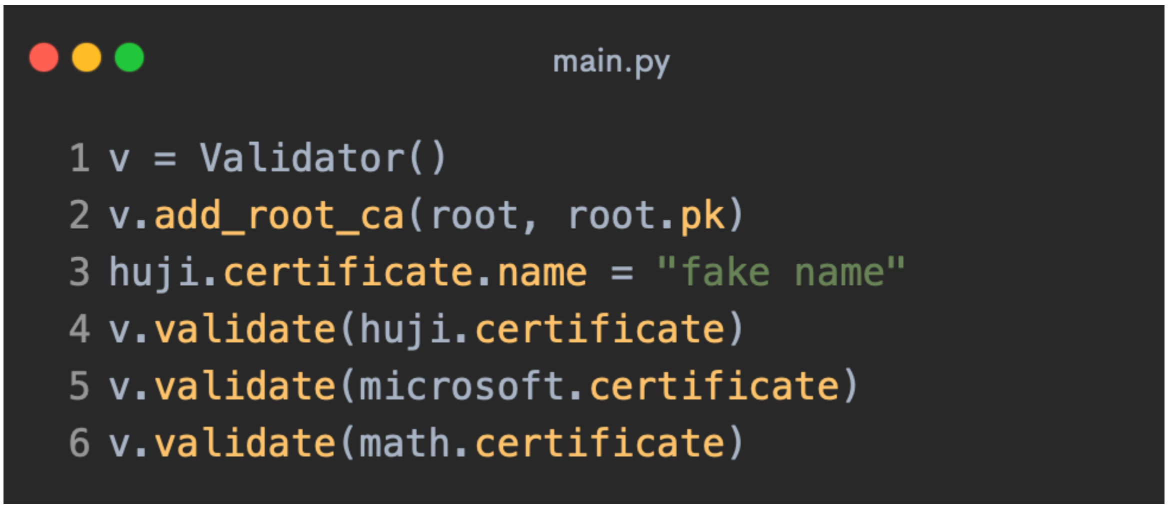
After revocating math we will get a RevocedCertificateError



```python
v = Validator()
huji.ca.revocate(math.certificate)
v.add_root_ca(root, root.pk)
v.validate(huji.certificate)
v.validate(microsoft.certificate)
v.validate(math.certificate)
```

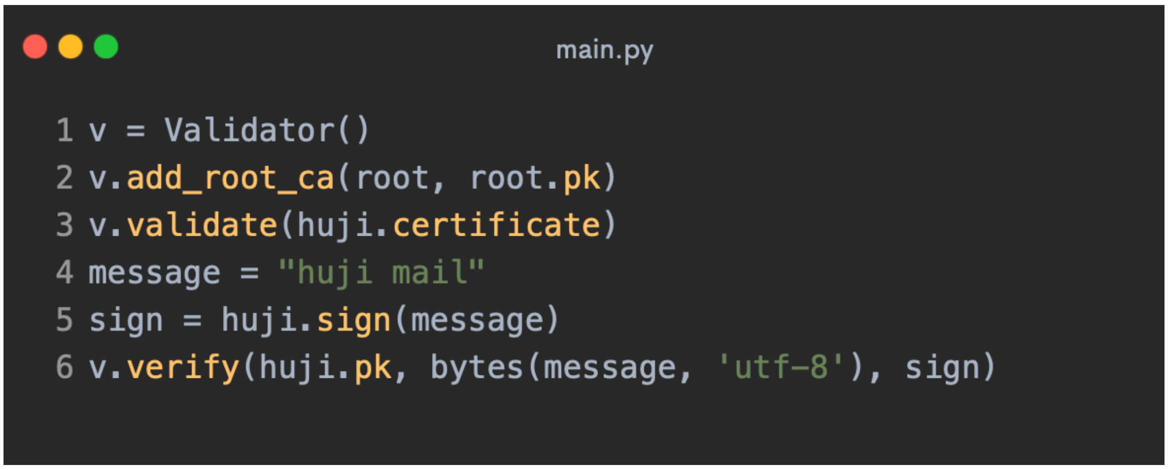
But if we would sign math with an higher ranked CA, like il, we won't get any error.

If we change one of the fields of the cert after signing we would get InvalidSignature Error.



```python
v = Validator()
v.add_root_ca(root, root.pk)
huji.certificate.name = "fake name"
v.validate(huji.certificate)
v.validate(microsoft.certificate)
v.validate(math.certificate)
```

After the Entity is validated, here is how we validate the origin of the messages:



```python
v = Validator()
v.add_root_ca(root, root.pk)
v.validate(huji.certificate)
message = "huji mail"
sign = huji.sign(message)
v.verify(huji.pk, bytes(message, 'utf-8'), sign)
```