ClaimChain Documentation

Bogdan Kulynych, Marios Isaakidis, George Danezis

May 23, 2018

Contents:

This is a brief documentation for *claimchain-core*, an experimental prototype implementation of ClaimChain, a cryptographic data structure. See the web page to learn about ClaimChain.

CHAPTER 1

Installing

For the moment, the package needs to be installed manually from Github:

```
git clone git@github.com:claimchain/claimchain-core.git
cd claimchain-core
pip install -r requirements/base.txt
pip install -e .
```

To run the tests, first install *dev* requirements:

```
pip install -r requirements/dev.txt
```

And then run `pytest`. To test against both Python 2 and Python 3, run `tox`.

CHAPTER 2

Usage guide

High-level interface for ClaimChain consists of two classes, `State` for building claimchains, and `View` for parsing and interpreting claimchains.

2.1 Building chains

The core abstraction for a ClaimChain user is a *state*. The *state* contains information about the user, and claims they make about other users or objects. Currently, this package only supports private claims, which means the owner of a chain needs to explicitly make every claim readable by intended readers. Once the *state* is constructed it can be committed to the chain.

Here is how user *Alice* would prepare her *state*:

```
from claimchain import State

state = State()

# Alice adds information about herself
state.identity_info = "Hi, I'm Alice"

# Alice adds private claims
state['bob'] = 'Bob is a good lad'
```

Making claims accessible requires knowing the DH public key of each authorized reader. The way to obtain the DH keys of other users is described later. Assuming Alice has Carol's DH public key, `carol_dh_pk`, she can allow Carol to access her claim about Bob:

```
state.grant_access(carol_dh_pk, ['bob'])
```

Note that the second argument must be an iterable of claim labels, not a single label.

To commit the state, first, a chain needs to be built, and second, the cryptographic keys have to be generated:

```
from hippiehug import Chain
from claimchain import LocalParams, State

state = State()

# Generate cryptographic keys
params = LocalParams.generate()

store = {}
chain = Chain(store)

with params.as_default():
    head = state.commit(chain)
```

The chain can then be published or transmitted to other users by publishing the `store` and communicating the chain's `head`. Other users will be able to interpret the chain using the `View` interface, described below.

2.2 Interpreting chains

Having access to the store (dictionary) containing other user's chain, and a head of this user's chain, one can use the `View` interface.

Here is how Carol can interpret Alice's claimchain, assuming Alice's store is `alice_store`, the head of her chain is `alice_head`, and `params` is Carol's `LocalParams` object:

```
from hippiehug import Chain
from claimchain import View

alice_chain = Chain(alice_store, root_hash=alice_head)

with params.as_default():
    alice_view = View(alice_chain)

    # Try to get claim with label 'bob'
    claim = alice_view['bob']

    assert claim == b'Bob is a good lad'
```

Finally, this is how Carol can retrieve Alice's DH public key:

```
alice_dh_pk = alice_view.params.dh.pk
```

This DH public key can be later used to grant Alice rights to read claims on Carol's chain.

CHAPTER 3

API

3.1 High-level interface

High-level ClaimChain interface.

class claimchain.state.**Metadata**(*params*, *identity_info=None*)
Block metadata.

Parameters

- **params** – Owner's cryptographic parameters.
- **identity_info** – Owner's identity info (public key)

class claimchain.state.**Payload**(*mtr_hash*, *metadata*, *nonce=False*, *timestamp=NOTHING*, *version=1*)
Block payload.

Parameters

- **mtr_hash** (*bytes*) – Hash of the Merkle tree root
- **metadata** (Metadata) – Block's metadata
- **nonce** (*bytes*) – Nonce
- **timestamp** – Unix-format timestamp
- **version** (*int*) – Protocol version

static build(*tree*, *nonce*, *identity_info=None*)
Build a payload.

Parameters

- **tree** – Tree object
- **nonce** (*bytes*) – Nonce
- **identity_info** – Owner's identity info (public key)

export()
Export to dictionary.

static from_dict(*exported*)
Import payload from dictionary.

Parameters **exported** (*dict*) – Exported payload.

class claimchain.state.**State**(*identity_info=None*)
ClaimChain owner state.

Parameters **identity_info** – Owner's identity info (public key)

__getitem__(*label*)
Get queued claim by label.

Parameters **label** – Claim label

__setitem__(*claim_label*, *claim_content*)
Add a claim with given label and content to be committed.

Parameters

- **claim_label** (*bytes*) – Claim label
- **claim_content** (*bytes*) – Claim content

clear()
Clear buffer.

commit(*target_chain*, *tree_store=None*, *nonce=None*)
Commit state to a chain.

Constructs a new block and appends to a chain.

Parameters

- **target_chain** (*hippiehug.Chain*) – Chain to which a block will be appended.
- **tree_store** (*utils.ObjectStore*) – Object store to hold tree nodes.
- **nonce** (*bytes*) – Nonce to include in the new block.

compute_evidence_keys(*reader_dh_pk*, *claim_label*)
List hashes of all nodes that prove inclusion of a claim label.

Parameters

- **reader_dh_pk** (*petlib.EcPt*) – Reader's DH public key
- **claim_label** (*bytes*) – Claim label

get_capabilities(*reader_dh_pk*)
List all labels accessibly by a reader.

Parameters **reader_dh_pk** (*petlib.EcPt*) – Reader's DH public key

grant_access(*reader_dh_pk*, *claim_labels*)
Grant access for given claims a reader.

Parameters

- **reader_dh_pk** (*petlib.EcPt*) – Reader's DH public key
- **claim_labels** (*iterable*) – List of claim labels

revoke_access(*reader_dh_pk*, *claim_labels*)
Revoke access for given claims to a reader.

Parameters

- **reader_dh_pk** (*petlib.EcPt*) – Reader's DH public key
- **claim_labels** (*iterable*) – List of claim labels

tree
Corresponding Merkle tree holding the claims and capabilities.

class claimchain.state.**View**(*source_chain*, *source_tree=None*)
View of an existing ClaimChain.

__getitem__(*claim_label*)
Get claim by label.

Parameters **claim_label** (*bytes*) – Claim label

Raises KeyError if claim not found or not accessible

__hash__()
Return hash(self).

get(*claim_label*)
Get claim by label.

Parameters **claim_label** (*bytes*) – Claim label

Returns Claim or None if not found or not accessible.

head
Chain's head (latest block hash).

validate()
Validate the chain.

Note: Don't use this method. It is broken. ¯_()_/¯

3.2 Low-level operations

Low-level operations for encoding and decoding claims and capabilities.

claimchain.core.**decode_capability**(*owner_dh_pk*, *nonce*, *claim_label*, *encrypted_capability*)
Decode capability.

Parameters

- **owner_dh_pk** (*petlib.EcPt*) – Owder's VRF public key
- **nonce** (*bytes*) – Nonce
- **claim_label** (*bytes*) – Corresponding claim label
- **encrypted_capability** (*bytes*) – Encrypted capability

claimchain.core.**decode_claim**(*owner_vrf_pk*, *nonce*, *claim_label*, *vrf_value*, *encrypted_claim*)
Decode claim.

Parameters

- **owner_vrf_pk** (*petlib.EcPt*) – Owner's VRF public key
- **nonce** (*bytes*) – Nonce

- **claim_label** (*bytes*) – Claim label
- **vrf_value** (*bytes*) – Exported VRF value (hash)
- **encrypted_claim** (*bytes*) – Claim content

claimchain.core.**encode_capability**(*reader_dh_pk*, *nonce*, *claim_label*, *vrf_value*)
Encode capability.

Parameters

- **reader_dh_pk** (*petlib.EcPt*) – Reader's VRF public key
- **nonce** (*bytes*) – Nonce
- **claim_label** (*bytes*) – Corresponding claim label
- **vrf_value** (*bytes*) – Exported VRF value (hash)

claimchain.core.**encode_claim**(*nonce*, *claim_label*, *claim_content*)
Encode claim.

Parameters

- **nonce** (*bytes*) – Nonce
- **claim_label** (*bytes*) – Claim label
- **claim_content** (*bytes*) – Claim content

claimchain.core.**get_capability_lookup_key**(*owner_dh_pk*, *nonce*, *claim_label*)
Compute capability lookup key.

Parameters

- **owner_dh_pk** (*petlib.EcPt*) – Owner's DH public key
- **nonce** (*bytes*) – Nonce
- **claim_label** (*bytes*) – Corresponding claim label

3.3 Cryptography

3.3.1 Containers

Containers for key material.

class claimchain.crypto.params.**Keypair**(*pk*, *sk=None*)
Asymmetric key pair.

Parameters

- **pk** – Public key
- **sk** – Private key

static generate()
Generate a key pair.

3.3.2 Signatures

claimchain.crypto.sign.**sign**(*message*)
Sign a message.

Parameters **message** (*bytes*) – Message

Returns Tuple of bignums (petlib.bn.Bn)

claimchain.crypto.sign.**verify_signature**(*sig_pk*, *sig*, *message*)
Verify a signature.

Parameters

- **sig_pk** (*petlib.EcPt*) – Signature verification key
- **sig** (tuple of bignums (petlib.bn.Bn)) – Signature
- **message** (*bytes*) – Message

3.3.3 Verifiable random functions

Implementation of a CONIKS's verifiable random function scheme.

class claimchain.crypto.vrf.**VrfContainer**(*value*, *proof*)
VRF value (hash) and proof.

Parameters

- **value** (*bytes*) – Exported VRF value (hash)
- **proof** (*bytes*) – Exported VRF proof

claimchain.crypto.vrf.**compute_vrf**(*message*)
Compute VRF.

Produces a VRF value (hash) and a proof.

Parameters **message** (*bytes*) – Message

Returns *VrfContainer*

claimchain.crypto.vrf.**verify_vrf**(*pub*, *vrf*, *message*)
Verify a VRF.

Checks whether a VRF value and a proof correspond to the message.

Parameters

- **pub** (*petlib.EcPt*) – VRF public key
- **vrf** (VrfContainer) – VRF value and proof
- **message** (*bytes*) – Message

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

B

C

D

E

F

G

H

K

M

P

R

S

T

V