Cryptographically Signed License Issuance with Payment in Cryptocurrency

Perry Kundert

2022-01-25 12:32:00

Licensing software and getting paid for it has become extremely difficult, due to government, regulatory and banking interference.

The `crypto-licensing` Python module allows you automatically and securely issue licenses, and get paid in various cryptocurrencies.

Contents

1 Software Licensing Using Ed25519 Signatures

Providing software to a client usually includes more than just the code; you need a way to configure the software for each client's licensed set of capabilities.

The `crypto_licensing` module provides a way to securely transmit some authorizations or configurations specific to each client.

These configurations are signed using Ed25519 Public/Private keys, and are shipped to the client either with the software, or separately, for example via email, after a standard online purchase.

Your organization's or product's Authoring public key is online (using the same model as for email's DKIM signature checking), so your software can **verify** the License in the field, by securely accessing DNS `TXT` records from your organization's domain, and checking the License' signature.

Your software can also sign and save this verification for later runs, so it can be assured that the License has been verified – even if the software is not normally "online". Your software just

has to have access to the Internet *once*, after the new License is installed, to verify the License and remember its decision.

1.1 Issuing A License: Your Authoring (Signing) Key

To begin authoring Licenses, you need to be able to sign them; you need to create and save an encrypted Ed25519 keypair, so you (and only you) can obtain it later to sign new Licenses.

The public key `dqFZIESm5PURJlvKc6YE2QsFKdHfYCvjChmpJXZg0fU=` (related to the private key consisting of all `1` bits) may be created via the API or CLI. It should be stored securely, so a `KeypairEncrypted` might be appropriate. Both the `KeypairPlaintext` and `KeypairEncrypted` contain the public "verifying" key `.vk`. The `KeypairEncrypted` also contains a `.vk_signature`, proving that the `.vk` was signed by the corresponding private key at the time of creation.

1.1.1 `crypto_licensing.authoring`: Create an Authoring Keypair in Python

The raw `ed25519.Keypair` from `authoring` isn't serializable, so get a `crypto_licensing` `KeypairEncrypted` or `KeypairPlaintext` and save its `str( <Keypair...> )` output to a file.

```
import crypto_licensing as cl

username = 'admin@awesome-inc.com'
password = 'password'

auth_keypair = cl.authoring( seed=b'\xff' * 32 ) # Supply None, unless you really have a random seed!
encr_keypair = cl.KeypairEncrypted( auth_keypair, username=username, password=password )
decr_keypair = cl.KeypairPlaintext( encr_keypair.into_keypair( username=username, password=password ))

# How can I know that the KeypairEncrypted holds a real private key?  Because the public key was signed by it!
from crypto_licensing import ed25519
try:
    ed25519.crypto_sign_open( encr_keypair.vk_signature + encr_keypair.vk, encr_keypair.vk )
    valid = True
except Exception as exc:
    valid = exc

[
    [ "Encrypted:", "" ],
    [ "Public Key", encr_keypair['vk'] ],
    [ "Salt", encr_keypair['salt'] ],
    [ "Ciphertext", encr_keypair['ciphertext'] ],
    [ "Signature", encr_keypair['vk_signature'] ],
    [ "Valid?", repr(valid) ],
    [],
    [ "Plaintext:", "" ],
    [ "Public Key", decr_keypair['vk'] ],
    [ "Private Key", decr_keypair['sk'] ],
]
```

Encrypted:	
Public Key	dqFZIESm5PURJlvKc6YE2QsFKdHfYCvjChmpJXZg0fU=
Salt	f48f08e16c88f83ecf3a6099
Ciphertext	21b92062f70882a480eed7e7ab7fffcc4c288d30715c12c85c56a30e6de14c470c3fb3258148ba191b885cbdaa5b0e49
Signature	h44cyYJvofemshmvizrN0+LVisMSTcPD1BGBVkwHVbEKbz+zHsNMjczQh91mLgwv8A6mzlbF7jQqznJOQwcxDA==
Valid?	True
Plaintext:	
Public Key	dqFZIESm5PURJlvKc6YE2QsFKdHfYCvjChmpJXZg0fU=
Private Key	//92oVkgRKbk9REmW8pzpgTZCwUp0d9gK+MKGakldmDR9Q==

1.1.2 `crypto_licensing registered`: Load or Create an Authoring Keypair

But the simplest way to manage creating and then (later) obtaining your Authoring Keypair is to use the CLI to check if one is already `registered` and saved in your `~/.crypto-licensing`

directory under a given name, using some encryption credentials.

The first time you do this, one will be created for you; subsequently, the existing one will be opened, displaying the file path, the public key, and (with `-v`) the `KeypairEncrypted`:

```
python3 -m crypto_licensing -v --name "Awesome-Inc" registered --username admin@awesome-inc.com --password password \
    --seed "0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"

    [
 "/Users/perry/.crypto-licensing/Awesome-Inc.crypto-keypair",
 "dqFZIESm5PURJlvKc6YE2QsFKdHfYCvjChmpJXZg0fU=",
 {
     "ciphertext":"aef7cf9884bc13bd7b4ee0cde402d2b666a084335f5d7b2bb6d2c31a8910499b5b19d450b2ccab03b83e9bb586612fb2",
     "salt":"a84d6df719af9f50dcdc1416",
     "vk":"dqFZIESm5PURJlvKc6YE2QsFKdHfYCvjChmpJXZg0fU=",
     "vk_signature":"h44cyYJvofemshmvizrN0+LVisMSTcPD1BGBVkwHVbEKbz+zHsNMjczQh91mLgwv8A6mzlbF7jQqznJOQwcxDA=="
 }
    ]
```

You may instead obtain the decrypted private signing key instead of the public key, using the `--private` option, for you to use in toolchains requiring it

```
python3 -m crypto_licensing -v --private --name "Awesome-Inc" registered --username admin@awesome-inc.com --password password

    [
 "/Users/perry/.crypto-licensing/Awesome-Inc.crypto-keypair",
 "////////////////////////////////////////92oVkgRKbk9REmW8pzpgTZCwUp0d9gK+MKGakldmDR9Q==",
 {
     "ciphertext":"aef7cf9884bc13bd7b4ee0cde402d2b666a084335f5d7b2bb6d2c31a8910499b5b19d450b2ccab03b83e9bb586612fb2",
     "salt":"a84d6df719af9f50dcdc1416",
     "vk":"dqFZIESm5PURJlvKc6YE2QsFKdHfYCvjChmpJXZg0fU=",
     "vk_signature":"h44cyYJvofemshmvizrN0+LVisMSTcPD1BGBVkwHVbEKbz+zHsNMjczQh91mLgwv8A6mzlbF7jQqznJOQwcxDA=="
 }
    ]
```

Use `jq` to process the JSON output:

```
python3 -m crypto_licensing -v --private --name "Awesome-Inc" registered --username admin@awesome-inc.com --password password \
 | jq '.[1]'

"////////////////////////////////////////92oVkgRKbk9REmW8pzpgTZCwUp0d9gK+MKGakldmDR9Q=="
```

Of course, if you get the password wrong, then you'll get an error (we'll never over-write existing files):

```
python3 -m crypto_licensing -v --name "Awesome-Inc" registered --username admin@awesome-inc.com --password wrong 2>&1

2024-12-25 08:27:17 WARNING  licensing  load_keypa Cannot load Keypair(s) from /Users/perry/.crypto-licensing/Awesome-Inc.crypto
2024-12-25 08:27:17 WARNING  doh.cli    <module>   Failed: '/Users/perry/.crypto-licensing/Awesome-Inc.crypto-keypair'
```

We've provided the (very poor) `--seed 0xff...` option above for consistency with the API calls in the example above, but you shouldn't; a random seed will be used to create it, unless you specify `--no-registering` to prevent creation:

```
python3 -m crypto_licensing -v --name "Awesome-Again" registered --username admin@awesome-inc.com --password password \
    --no-registering 2>&1

2024-12-25 08:27:18 WARNING  doh.cli    <module>   Failed: Failed to find a admin@awesome-inc.com Keypair; registering a new one
```

But don't worry; if an existing `KeypairEncrypted` file with the specified name `Awesome-Inc.crypto-keypair` exists anywhere in your `crypto_licensing` search paths, we won't re-create it if you specify the wrong password, but will instead report a failure.

It is not recommended to use the `--password ...` command-line option; specify the password in the `CRYPTO_LIC_PASSWORD` environment variable. `CRYPTO_LIC_USERNAME` may be used instead of `--username`.

1.1.3 `issue`: Signing a License

A License can be as simple, free-standing authorization with no other License dependencies, or it may have a tree of sub-Licenses that must also be confirmed as valid.

1.1.4 `verify`: Confirm License (and sub-License) Validity

1.2 Using Licenses

1.2.1 `load_keys`: Find all Ed25519 Signing Keys

1.2.2 `load`: Find all Licenses

1.2.3 `check`: Find all Keys and Valid Licenses

Loads every available Ed25519 Keypairs (with the provided credentials), and all available Licenses, yielding all <Keypair>,<LicenseSigned> that are valid in the current environment.

If no valid License is available for some key found, then <Keypair>,None is yielded, allowing the caller to use the Key to issue a License if desired.

If nothing at all is yielded, then this indicates that **no** Keypairs were found; either you need to "register" (create and save) one, or provide different credentials.

1.3 Running A `crypto_licensing.licensing` Server

Supply the `username` and `password` to the `KeypairEncrypted` via environment variables `CRYPTO_LIC_USERNAME` and `CRYPTO_LIC_PASSWORD`.

2 Payment with Cryptocurrencies

3 Issuance via Web API