zenroom.py Documentation

Release 0.0.2

Samuel Mulube

Nov 21, 2018

Contents:

CHAPTER 1

Introduction

This library contains a very thin Python wrapper for the Zenroom crypto virtual machine that has been developed as part of the DECODE project.

1.1 What is DECODE?

DECODE is a European Commission funded project exploring and piloting new technologies that give people more control over how they store, manage and use personal data generated online. We will test the technology we develop in two pilot sites and will explore the social benefits of widespread open data commons.

1.2 What is Zenroom?

Zenroom is a brand new virtual machine for fast cryptographic operations on Elliptic Curves. The Zenroom VM has no external dependencies, includes a cutting edge selection of C99 libraries and builds a small executable ready to run on: desktop, embedded, mobile, cloud and browsers (webassembly). It also compiles unikernel (without Linux).

It provides an execution environment for a domain specific language of cryptographic primitives using a dialect of Lua. The aim of this is to provide an environment which makes it very easy to write safe encryption logic, in the form of a Lua script. This script can then be passed into the Zenroom VM for execution.

For more details of the DSL please see the Zenroom API documentation.

1.3 What is this library?

This library simply provides a very thin wrapper around the core Zenroom crypto virtual machine, that aims to make its functionality slightly easier to use from Python code.

1.4 License

This library has currently been released under the GNU AGPL 3.0 License. This is a strong, free copyleft license published by the Free Software Foundation in 2007, which is based on the GNU General Public License.

1.5 Installation

The package has been published to PyPi, so should be installable via the following command:

```
$ pip install zenroom
```

1.6 Limitations

The library includes a static binary containing the Zenroom virtual machine, compiled for Linux (amd64) systems. As such this library will only work within this restricted environment.

CHAPTER 2

Usage

The core Zenroom virtual machine exposes two primary functions by which cryptographic operations may be invoked:

- `zenroom_exec` which takes as input a script, as well as optional keys and data fields, and when invoked the output of the script is written to stdout.
- `zenroom_exec_tobuf` which takes the same basic parameters, but the output is written to a buffer provided by the caller.

For use in a library, this secondary function is the one that this library attempts to expose, simplified slightly to expose a more idiomatically Pythonic API surface.

2.1 Executing a simple script

The library only exposes a single function called `execute`, which passes its input parameters into the Zenroom VM, so to execute a simple script that does not require any data or keys, the library can be used simply as follows:

```
from zenroom import zenroom

script = b'print("Hello world!")'

output = zenroom.execute(script)

print(output)
```

This is a trite example obviously, but we wanted to demonstrate the simplest possible Zencode script execution, that a caller can capture.

Important: Note that the script value that is passed into the execute function must be a byte string. This is also true for any data or keys that are passed into Zenroom.

2.2 Executing a script with keys and data

The example below is more involved, and shows how we can execute a script with keys and data that must be fed into Zenroom to enable the script.

```
from zenroom import zenroom

script = b"""
-- define data schema
msg = SCHEMA.Record {
   msg = SCHEMA.String
}

-- read and validate the data
data = read_json(DATA, msg)

-- read keys without validating
keys = read_json(KEYS)

-- now import recipient public key
recipient_key = ECDH.new()
recipient_key:public(base64(keys.recipient_public))

-- now import our own private key (we are the data subject)
own = ECDH.new()
own:private(base64(keys.own_private))


-- encrypt the fields
out = {}
out = LAMBDA.map(data,function(k,v)
  header = MSG.pack({key=k, pubkey=own:public()})
  enc = ECDH.encrypt(own,recipient_key,str(v), header)
  oct = MSG.pack( map(enc,base64) )
  return str(oct):base64()
end)

-- print out result
print(JSON.encode(out))
"""

keys = b"""
{
  "own_private": "DYgWghvJuClxvHVCQSAfDpWQmeMQ4zh1/mGoNjM8UX0=",
  "own_public": "BAXRFKfMNSMge11U/cP+mCW2al166qIAY/
↪cETmToEGmqe+4JnMdhmJ1FURvtUU+gA4QiEP+C7QFy/eoH+FDSRbw=",
  "recipient_public":
↪"BCj962CsLq0Ey9Ibe6DEFSak4KqnQ5FhbNMv7MaMr6OzZZsnncUVOTrFK4Ym9WItAEMbpkGIOIjgfPESblAKlbw=
↪"
}
"""

data = b'{"msg":"top secret"}'

result = zenroom.execute(script, keys=keys, data=data)

# Here we'd actually do something with the output
```

(continues on next page)

(continued from previous page)

```
print(result)
```

Note that the script parameter is the single required parameter, so here I pass it it first as a positional argument; `keys` and `data` in contrast are optional, so we choose to pass them as keyword arguments meaning we can label them when calling the function, so reducing the risk of getting the parameters in the wrong order.