Zero Knowledge Checking for Legal Age with Lurk

Christiano Braga
cbraga@ic.uff.br

Computer Science Department
Universidade Federal Fluminense

November 14, 2023

1 Objective

This example applies Lurk to check for legal age (in Brazil) where someone must be over 17 years old. Here, one does not want to disclose one's age but would like to prove to be of legal age.

2 Setup

You just need to install Lurk. My suggestion is to work with lurk-rs, the Rust implementation of Lurk.

3 Protocol

3.1 Create the secret

Make age a secret. One must do it or an issuer.

```
$ lurk
commit: 2023-11-08 2278cf2ad36f6b68849565fe97ebef718d120074
Lurk REPL welcomes you.
user> !(commit 53)
```

```
Hash: 0x29cefdde5e5fdbbc03022b590ad4861c956f236020c73fca291e26a6ba
2eb616
```

The above hash number now represents one's age.

3.2 Fetch the test

One fetches the legal age test from the institution that needs your age to be checked. (This will come in the form of a hash together with the binary of the commitment hash stands for or plain code if the test is publicly available.) And then applies the test to the secret data.

In this example, the test is simply

```
;; test.lurk
!(def test (lambda (x) (> 17 data))
!(commit test)
```

which produces the following commit.

```
$ lurk test.lurk
commit: 2023-11-08 2278cf2ad36f6b68849565fe97ebef718d120074
Loading test.lurk
test

Hash: 0x07d5377323e87cb5b313d007f9e973bad049f19e7b9c7ee0efb2a68fda
b97e86
```

3.3 Generate the proof

Now one is ready to prove being of legal age: this is accomplished by running the `call` command on the hashes of the test and the secret data. First, it's necessary to `fetch` the commits for `test` and the secret data, in order to make this call, shouldn't the data be in Lurk's environment yet.

This is done as follows:

```
$ lurk
commit: 2023-11-08
2278cf2ad36f6b68849565fe97ebef718d120074
Lurk REPL welcomes you.

user> !(fetch 0x29cefdde5e5fdbbc03022b590ad4861c956f236020c73fca29
1e26a6ba2eb616)
Data is now available
user> !(fetch 0x07d5377323e87cb5b313d007f9e973bad049f19e7b9c7ee0ef
b2a68fdab97e86)
Data is now available
user> !(call 0x07d5377323e87cb5b313d007f9e973bad049f19e7b9c7ee0efb
2a68fdab97e860x29cefdde5e5fdbbc03022b590ad4861c956f236020c73fca291
e26a6ba2eb616)
[9 iterations] => t
```

Recall that hash `0x07d5377323e87cb5b313d007f9e973bad049f19e7b9c7ee0ef b2a68fdab97e86` denotes the test function and hash `0x29cefdde5e5fdbbc030 22b590ad4861c956f236020c73fca291e26a6ba2eb616` denotes the secret data. When the test is called with the secret data, it yields `t`, for true.

One can then generate a proof of legal age (that is, the application of the test to the secret data) by executing the command `!(prove)`, as shown below. Lurk generates the proof key `Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704 b2750a8e4276554f392077f8f5e9b7743`.

```
user> !(prove)
Claim hash:

0x2dcf52eb7250fbcf38ebc8c242a7704b2750a8e4276554f392077f8f5e9b7743

Proof key:
Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704b2750a8e4276554f3920
77f8f5e9b7743
```

This proof key, together with the binary file that encodes it, saved as `~/.lurk/proofs/Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704b2750a8e4276554f392077f8f5e9b7743.proof`, and the proof meta information, saved as `~/.lurk/proofs/Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704b2750a8e4276554f392077f8f5e9b7743.meta`, must be sent to the inquiring institution.

3.4 Proof validation

Finally, the inquiring institution may check the proof by running

```
$ lurk verify Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704b2750a
8e4276554f392077f8f5e9b7743 --proofs-dir .
```

It produces the following output:

```
commit: 2023-11-08 2278cf2ad36f6b68849565fe97ebef718d120074
✓ Proof " Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704b2750a8e42
76554f392077f8f5e9b7743 " verified,
```

assuming files `Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704b2750a8e4276554f392077f8f5e9b7743.proof` and `Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704b2750a8e4276554f392077f8f5e9b7743.meta` are present in the same directory as the Lurk command is executed.

However, one question may still arise.

What if one sends a verifiable proof about something else entirely?

For this example, the institution needs to make sure that the test it gave away (encoded by hash `0x07d5377323e87cb5b313d007f9e973bad049f19e7b9c7ee0efb2a68fdab97e86`) was applied to the secret data encoded by hash `0x29cefdde5e5fdbbc03022b590ad4861c956f236020c73fca291e26a6ba2eb616`. The institution needs to *trust* that the encoding for the secret data is indeed about one's age, hence the need for an issuer.

Under this assumption, that is, that the hash for the secret data is indeed one's age, one can check that the proof is about the proper commits by running Lurk's command `inspect`. It reveals, in the command line, information, in textual form, about the proof as stored in the `.meta` file, sent to the institution, together with the proof key and the proof itself. Such information includes the *input expression* used to generate the proof.

In this example, the expression was
`(call 0x07d5377323e87cb5b313d007f9e973bad049f19e7b9c7ee0efb2a68fdab97e86 0x29cefdde5e5fdbbc03022b590ad4861c956f236020c73fca291e26a6ba2eb616)`,
where the first hash is for the test and the second for the secret data. Now, if one runs the command
`$ lurk inspect Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704b2750a8e4276554f392077f8f5e9b7743--proofs-dir .`,
it will produce the following output

```
commit: 2023-11-08 2278cf2ad36f6b68849565fe97ebef718d120074
Input:
```

```
((open 0x07d5377323e87cb5b313d007f9e973bad049f19e7b9c7ee0efb2a68fd
ab97e86) 0x29cefdde5e5fdbbc03022b590ad4861c956f236020c73fca291e26a
6ba2eb616)
```

```
Output:
  t
Iterations: 9
```

(The expression `(call x y)` is inlined to `((open x) y)`.) One may then check if the proof was generated from the appropriate commits by matching the output of command `inspect` with the appropriate hashes. Note that **no** secret data is revealed as the proof only knows about hashes and nothing about the secret value they conceal.

This can be easily automated by running `grep` on the output of `inspect`. An example script is given in the Appendix.

A Searching for hashes in a proof

The following code is an example script to search for hashes in the output of a proof inspection.

```bash
# in_proof.sh
#!/usr/bin/bash

RED='\033[0;31m'
GREEN='\033[0;32m'
NC='\033[0m' # No Color

if [ $# != 3 ]
then
    echo "Wrong number of arguments."
    echo "USE: in_proof.sh <proof key> <query hash> <secret data hash>"
    exit 1
fi
```

```
PK=$1  # Proof key
QH=$2  # Query hash
SDH=$3 # Secret data hash

LOG=`lurk inspect --full $PK --proofs-dir .`

function check() {
    # The first parameter of `check` is the output of ispect.
    # The second parameter is the hash we are looking for.
    echo $1 | grep -w -q $2
    if [[ $? == 0 ]]
    then
        echo -e "* Hash ${GREEN}$2${NC} found in proof $PK."
        return 0
    else
        echo -e "* Hash ${RED}$2${NC} not found in proof $PK."
        return 1
    fi
}

check "$LOG" $QH
FOUND_QH=$?
check "$LOG" $SDH
FOUND_SDH=$?

if [ $((FOUND_QH + FOUND_SDH)) == 0 ]
then
    echo -e "${GREEN}Proof check successful.${NC}"
else
    echo -e "${RED}Proof check unsuccessful.${NC}"
fi
```

The shell script can be used as follows:

```
$ ./in_proof.sh Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a7704b275
0a8e4276554f392077f8f5e9b7743 0x07d5377323e87cb5b313d007f9e973bad0
49f19e7b9c7ee0efb2a68fdab97e86 0x29cefdde5e5fdbbc03022b590ad4861c9
56f236020c73fca291e26a6ba2eb616.
```

It will produce the following output:

```
✓ Hash 0x07d5377323e87cb5b313d007f9e973bad049f19e7b9c7ee0efb2a68fd
ab97e86 found in proof Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a77
04b2750a8e4276554f392077f8f5e9b7743.
✓ Hash 0x29cefdde5e5fdbbc03022b590ad4861c956f236020c73fca291e26a6b
a2eb616 found in proof Nova_Pallas_10_2dcf52eb7250fbcf38ebc8c242a77
```

```
04b2750a8e4276554f392077f8f5e9b7743.
Proof check successful.
```