# *Cryptographic Primitives: Hash Functions*

*Packer Collegiate Institute*
*Dr. Anthony Schultz*

*March 4 2016*

Project folder available at:
https://github.com/Trismeg/Packer

Figure 1: Computer scientists hashing

This is an introduction to hash functions. Hash functions are awesome because they are cryptic and abstract. They are also great because they are useful and help make the internet run properly. We will learn the following:

- what is a hash function and what are its properties
- how to use the SHA-256 hash function in Python and from shell
- applications of hash functions including cryptographic verification, proof of work and proof of existence

## HASHING

Hashing is the systematic process of adding entropy to number/data in a repeatable but irreversible and unpredictable way. A cryptographic hash function works like the culinary technique. It chops up numbers, any data, into a random scramble. The process is computationally irreversible but fully deterministic and repeatable.

A hash function maps a number $X$ to another number $Y$.

$$\text{Hash}(X) \longrightarrow Y$$

There is no reverse mapping or "un-hashing."

$$X \nleftarrow \text{Hash}^{-1}(Y)$$

A hash function is a non-invertible function.

The word **hash** entered English in the mid 1600's, during the dawn of science. It meant *to chop into small pieces* and represented a brilliant culinary advancement. It came from the French ***hacher***, *chop up*. This deriving from Old French ***hache***, *ax*.

Figure 2: Breakfast hash

## SHA2

SHA-2 (Secure Hash Algorithm 2) is a set of cryptographic hash functions designed by the NSA. SHA stands for Secure Hash Algorithm. SHA-256 and SHA-512 are novel hash functions. They use different shift amounts and additive constants, but their structures are otherwise virtually identical, differing only in the number of rounds.

The SHA-256 hash function is implemented in some widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec.

SHA-256 returns a 64 digit, base 16 hexadecimal number. The SHA-256 hash of the string "SHA-2 Electric Boogaloo" returns the following:

6f1dceb60c1e2251379c78059a07ec98d6ac368ee2f115c020d753158b16617c

A good hash function is a method of randomizing numbers so that no two inputs ever produce the same output. Such an occurrence is called is a collision. It is bad, very bad.

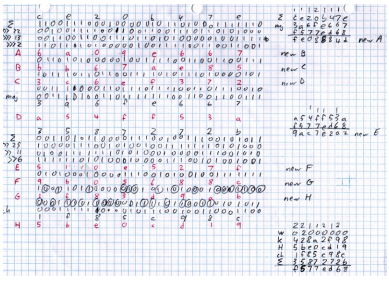

Figure 3: SHA-256 by hand

## HASHING IN PYTHON

At the command line type the following to get a python shell open.

```
$ python
```

Hexadecimal or hex numbers are base 16. The numerals are written:

```
01234567890abdef
```

Once the python shell is open, import the hash library and hash a test message.

```
>>> import hashlib
>>> hashlib.sha256("testing").hexdigest()
'cf80cd8aed482d5d1527d7dc72fceff84e6326592848447d2dc0b0e87dfc9a90'
>>> hashlib.sha256("testing1").hexdigest()
'3e3d6a28351293395ba3a345e79593de3780723a42321204c54b7da49bf3da45'
```

The SHA-256 output is 64 digits. Since the output is hex there are $16^{64}$ possible numbers. This is on the order of the number of atoms in the whole universe.

Note that the hash of "testing" and "testing1" is completely different despite the small difference of the input.

This code shows how to hash a file. This particular program, named "hasher.py" hashes itself.

```
import hashlib

hasher = hashlib.sha256()
with open("hasher.py", 'rb') as afile:
    buf = afile.read()
    hasher.update(buf)
print(" the hash is ")
print(hasher.hexdigest())
```

To create this file:

```
$ touch hasher.py
$ nano hasher.py
```

**Copy/Paste CTRL+X**

```
$ python hasher.py
```

It will output the following.

```
 the hash is
a37d71db6c43c3650ade82ccac7107bad8c268a9b94d03e36d6895eb2edbdd99
```

Figure 4: A hash of file is like a digital fingerprint. It functions as a unique identifier.

***CHALLENGE!!!!!***

***Find a number whose hash begins with "87".***

## APPLICATIONS

*Verification* SHA-256 is used as part of the process of authenticating Debian GNU/Linux software packages. If we are downloading software by torrent or loading a page from the internet we can check if the downloaded files are the intended downloads by checking a listed hash for the file.

*Partial Knowledge Systems* Unix and Linux vendors are moving to using 256- and 512-bit SHA-2 for secure password hashing. When storing user login information it is prudent security practice not to directly store passwords but rather hashes of passwords. If the database is compromised the information will not allow accounts to be compromised.

*Proof of Work* Hashing requires computation and this takes energy. A solution to a hashing challenge shows a cryptographic proof of work.

*Proof of Existence* A hash of a file may be encoded in a time-stamped public record in order to document the existence of the file at a specific time.

**Verification**
Note the hash code that comes up to verify the software version when you type "python" in the command line.

```
$ python
Python 2.7.6 v2.7.6:3a1db0d2747e
```

**Proof of Work**
Hashing requires computing and computing requires energy. The most efficient computing machinery for the executing SHA-256 is the 16nm BitFury ASIC. It runs at an efficiency of 0.1 J/GH.

**Mining**
The process of Bitcoin mining takes bitcoin transaction data and hashes it. The miner who discovers a hash with a sufficient number of leading zeros wins a bitcoin prize. The set of transactions that are hashed together constitute a block. The transactions are hashed with the previous prize-winning hash and a random number called a "nonce." The inclusion of the previous block hash orders the blocks in time as a chain of hashes. This data structure is called a blockchain.

**Digital Fingerprinting**
Since a hash can be used to uniquely identify a file it can be useful as a document reference. When I write student evaluations every semester I include the SHA-256 hash of their major research papers. Git, the open source version control system, also uses hashes to uniquely identify every change to a project over time.

## PROOF OF WORK SPAM FILTER

```
import hashlib

text="Hash this"
recipient="schultz@packer.edu"
nonce=0
state=True

num=3
lead=""
for i in range(num):
    lead=lead+"0"

while state:
    message=recipient+" "+text+" "+str(nonce)
    hashe=hashlib.sha256(message).hexdigest()
    if hashe[0:num]==lead:
        state = False
    else:
        nonce=nonce+1

print "The proper nonce is " + str(nonce)
print message
print hashe
```

To create this file:

```
$ touch email.py
$ nano email.py
```

**Copy/Paste CTRL+X**

```
$ python email.py
```

Imagine we want to develop an email protocol that will block spam. One solution to this problem is only to accept email messages which show sufficient work. Spammers send out millions of messages at a time because they can, email is cheap. If sending an email required some computational work the cost would make it prohibitive for spammers to send out millions of spam emails.

Figure 5: Hashing = Energy = Money

The above code implements such a protocol. Imagine only message whose hash has a sufficient number of leading zeros are accepted by the recipient server. In the code the variable `num` determines the number of leading zeros required. Notice the higher this number is set, the longer it takes to generate an acceptable message. The nonce value allows the sender to change the hash output without changing the recipient or message content. [1]

[1] This type of proof-of-work email system was proposed in May 1997 by Adam Back under the project name Hashcash.

## PROOF OF EXISTENCE OF OUR SPAM FILTER

Imagine that we want to protect our intellectual property for our spam fighting email system. We hay take the hash of the file "email.py".

```
import hashlib

hasher = hashlib.sha256()
with open("email.py", 'rb') as afile:
    buf = afile.read()
    hasher.update(buf)
print(" the hash is ")
print(hasher.hexdigest())
```

We may also achieve this using the following shell command.

```
$ shasum -a 256 email.py
```

It will output the following:

```
 the hash is
217c72a2b3a355bec2bda20e5280eaabae8fb445f8cf01666cd213fee7be2cde
```

Many services provide the ability to hash a document and record the resulting hash in the blockchain. It is a notary service. One such service started by Manuel Araoz:
https://www.proofofexistence.com

As a unique identifier for the file "email.py" the hash may be used as a proof of existence for the file without revealing the file itself. All that is required is a trusted timestamp. We could seal this in an envelope and mail it to ourselves. The postmark could serve as a trusted timestamp. Another option is that we could take out an add out in the New York Times, our trusted paper of record. The recording of the hash in the paper could be proof, at a later date, that the hash existed and thus the file existed.

Inclusion of a hash in a blockchain constitutes legal proof of existence in the state of Vermont.
http://goo.gl/ZpzPjv

Another interesting solution is to take the hash value and encode it in a transaction in a blockchain. This has been done for this file.[2]

[2] https://goo.gl/ULQwKD