Vid 17 – Hash Functions

Ch 2 - Main Objectives

- Writing a Blockchain from scratch
- Using cryptography module for hash functions

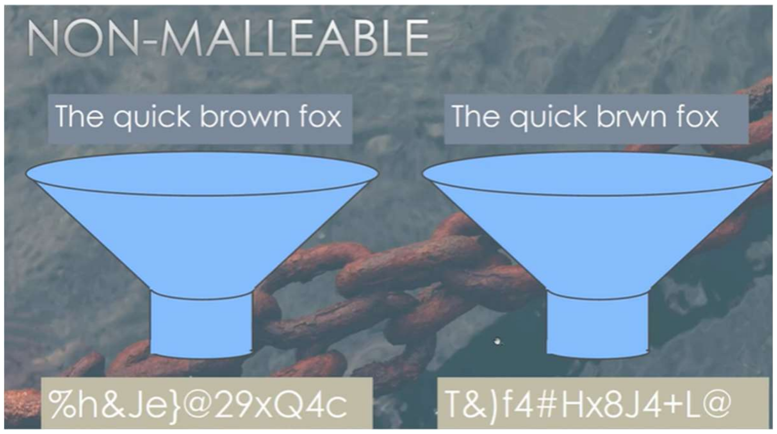
Blocks in the Blockchain will be able to store any data that can be converted to a string, and like any good Blockchain, we can quickly detect if there is any tampering going on.

What is a Hash Function for? What makes a good Hash Function?

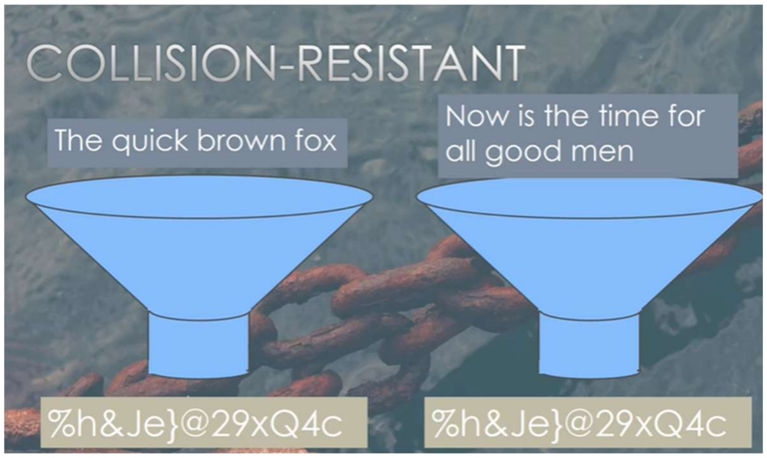
An algorithm that converts a string of arbitrary length into a string of bits of a fixed length in a predictable way.

Hashes are used in computer science to speed up certain algorithms

Cryptographic Hashes have additional requirements: should be **non-malleable** → meaning that 2 very similar inputs should not produce similar outputs. NEED ANY SLIGHT CHANGES INSTANTLY VISIBLE



Collision resistance means that the only sensible way to find two inputs that produce the same output is to compute the hash of many input functions until you stumble onto one.



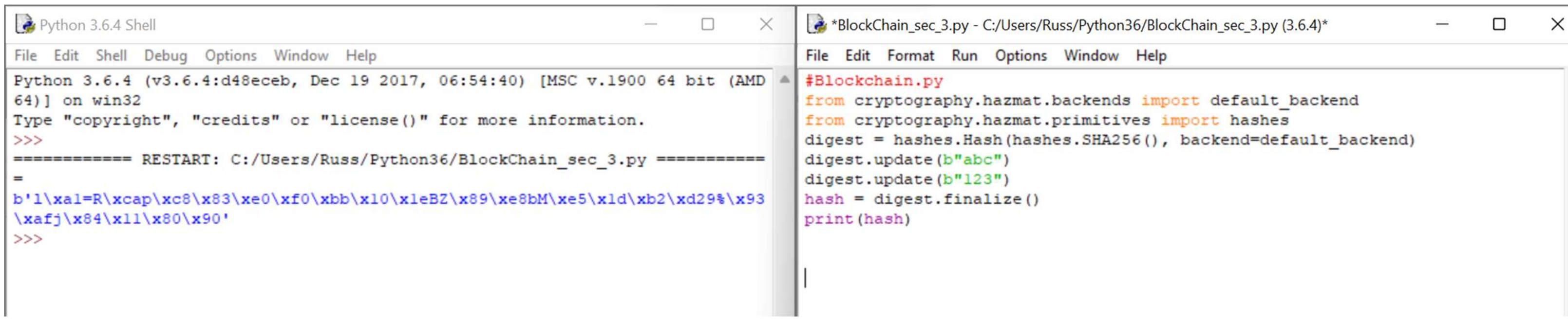
These are two very different messages that hash to the same hash string. (Inevitable b/c there are many messages we can pass and only a finite # of hashes we can have) → however it shouldn't be easy to find something like this

The security of the blockchain is going to depend on the fact that the only way to find these collisions is by passing a bunch of messages until we find two collisions.

Hash function used for our blockchain is the same as the one that we use for reducing the message size for our digital signatures → called SHA-256

Vid 18: Computing hash functions with Python

Fairly sensible hash, with some hexadecimal characters (hexadecimal character → \x83, the slash specifies that its hexadecimal and then the two characters that follow are the actual character) , regular characters, and looks generally random

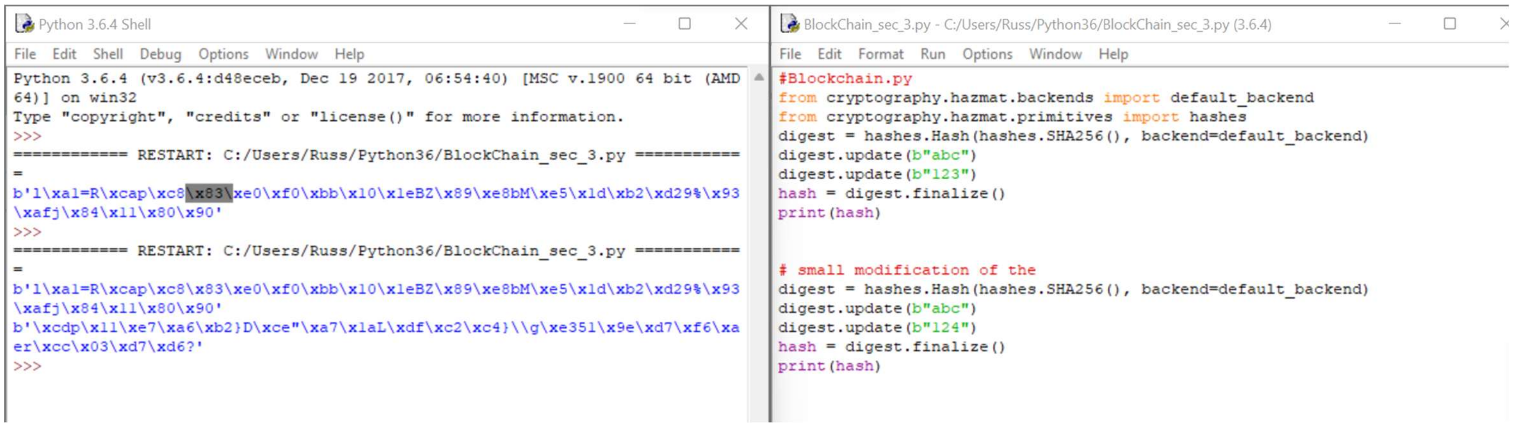

Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
BlockChain_sec_3.py - C:/Users/Russ/Python36/BlockChain_sec_3.py (3.6.4)
File Edit Format Run Options Window Help

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD
64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
============ RESTART: C:/Users/Russ/Python36/BlockChain_sec_3.py ===========
=
b'l\xal=R\xcap\xc8\x83\xe0\xf0\xbb\x10\xleBZ\x89\xe8bM\xe5\xld\xb2\xd29%\x93
\xafj\x84\x11\x80\x90'
>>> 
```

```
#Blockchain.py
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
digest = hashes.Hash(hashes.SHA256(), backend=default_backend)
digest.update(b"abc")
digest.update(b"123")
hash = digest.finalize()
print(hash)
```

Next, test the **malleability** by making a small modification to the message

- A non-malleable hash should give us a thoroughly different output from the hash function even if there is only a small change
- The outputs look like a **solid non-malleable hash**


Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
BlockChain_sec_3.py - C:/Users/Russ/Python36/BlockChain_sec_3.py (3.6.4)
File Edit Format Run Options Window Help

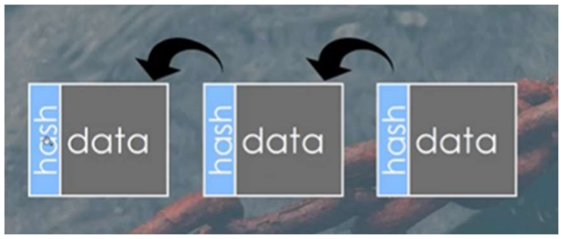
```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD
64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
============ RESTART: C:/Users/Russ/Python36/BlockChain_sec_3.py ===========
=
b'l\xal=R\xcap\xc8\x83\xe0\xf0\xbb\x10\xleBZ\x89\xe8bM\xe5\xld\xb2\xd29%\x93
\xafj\x84\x11\x80\x90'
>>> 
============ RESTART: C:/Users/Russ/Python36/BlockChain_sec_3.py ===========
=
b'l\xal=R\xcap\xc8\x83\xe0\xf0\xbb\x10\xleBZ\x89\xe8bM\xe5\xld\xb2\xd29%\x93
\xafj\x84\x11\x80\x90'
b'\xcdp\x11\xe7\xa6\xb2}D\xce"\xa7\xlaL\xdf\xc2\xc4}\\g\xe351\x9e\xd7\xf6\xa
er\xcc\x03\xd7\xd6?'
>>> 
```

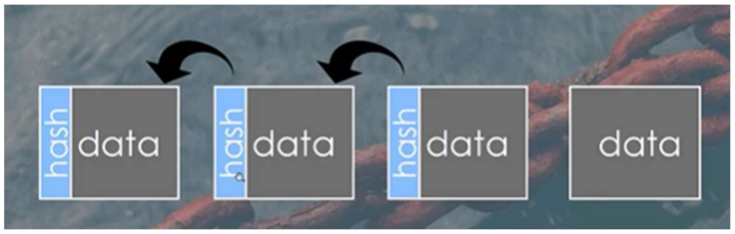
```
#Blockchain.py
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
digest = hashes.Hash(hashes.SHA256(), backend=default_backend)
digest.update(b"abc")
digest.update(b"123")
hash = digest.finalize()
print(hash)


# small modification of the
digest = hashes.Hash(hashes.SHA256(), backend=default_backend)
digest.update(b"abc")
digest.update(b"124")
hash = digest.finalize()
print(hash)
```

Vid 19: Block Chains

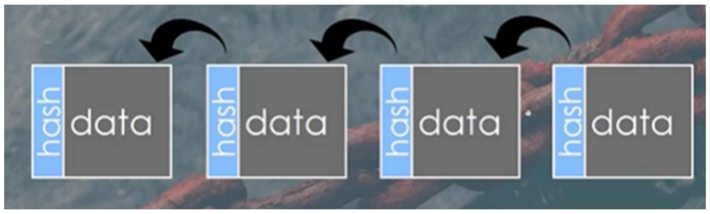
How this affects a block chain?

A blockchain consists of blocks and each of the blocks is going to contain some data and then also the hash of the block that preceded it.

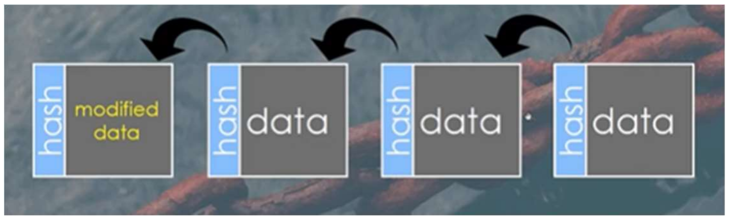


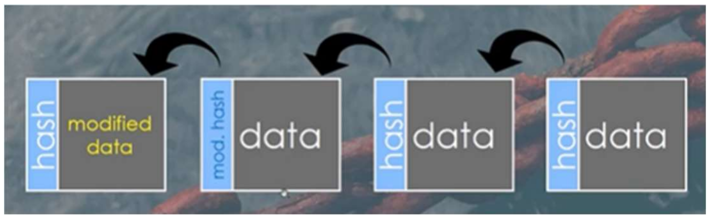


So when you add a new hash with some new data the hash will contain the data itself and then ill compute the hash of the previous block including its stored hash of the block before it → then put that in as a part of the block as seen below.



Advantage: tampering is always apparent very quickly. If someone wanted to tamper with an earlier block, and if our hash is a good one, then the modified data will cause the hash of the next block not to match with a recomputed has of this block. If you recompute the hash and compare it to the stored hash in the block that follows it wont match**.**





To get away with this → attacker would have to modify the next hash as well. But then the next block would detect it. **To modify the data in one block you'd have to modify the data in every block that follows it. Making the blockchain safe from tampering!!!**

Vid 20: Assignment – Blockchain for general data