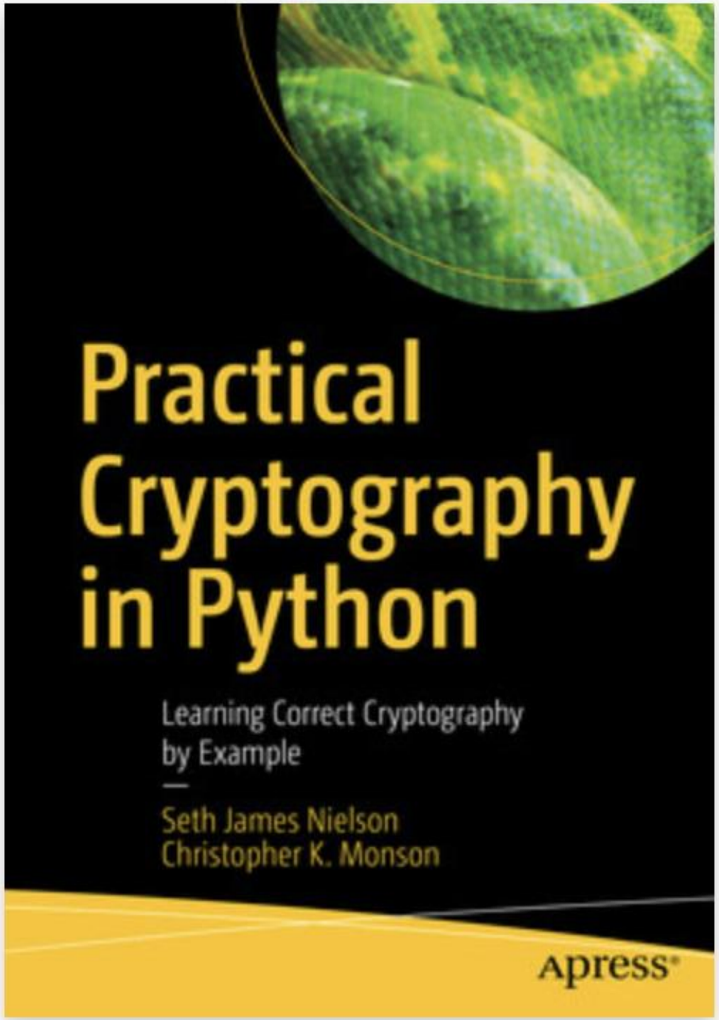
Systems Security Group | Lancaster University

Week 12 Hashing

Recommended reading


Practical Cryptography in Python
Learning Correct Cryptography by Example
Seth James Nielson
Christopher K. Monson
Apress

The book is available to you via the library

Technology stack

- Python 3
  Link to a Python Cheat Sheet
- cryptography.io
  Link to the library

Topics

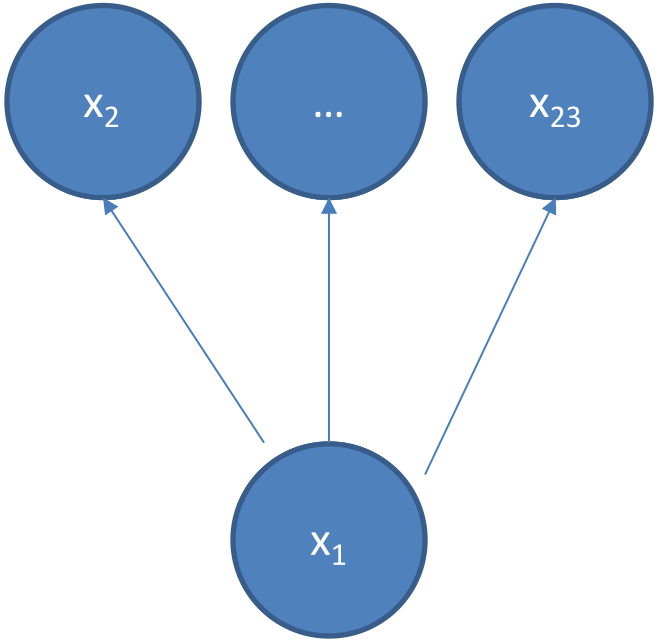
- Hashing & collisions
- How to create an avalanche calculator
- Other applications of hash functions

Recommended reading: Chapters 2 and 5 from the book of "Practical Cryptography in Python"

Reminder!

- A cryptographic hash function *H* must provide
  - Compression: e.g, *H*: {0,1}* → $\{0,1\}^{160}$
  - Efficiency: H: h(*x*) easy to compute for any *x*
  - One-way: given *y* it is infeasible to find *x*: h(*x*)=*y* (preimage resistance)
  - Weak collision resistance: for any given *x,* it should be difficult to find x' , *x'≠x* so that h(*x'*)=h(*x*) (2nd preimage resistance)
  - Strong collision resistance: it should be difficult to find any pair (*x, x')* with *x≠x'* so that h(*x*)=h(*x'*) (collision resistance)
- Check message integrity!

Weak collision resistance



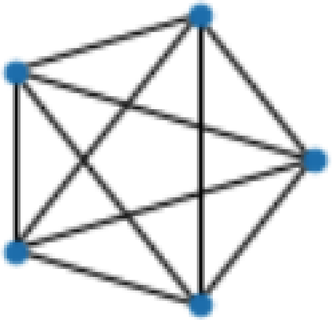
For any given *x*, it should be difficult to find x' , *x'*≠*x* so that h(*x'*)=h(*x*) (2nd preimage resistance)

Attack example

A: x1 has the same b/d as x2, or
x1 has the same b/d as x3 or
…
x1 has the same b/d as x23
(mutually exclusive)

$$P(A) = \underbrace{\frac{1}{365} + \ldots + \frac{1}{365}}_{22} \approx 0.06$$

Strong collision resistance



It should be difficult to find
any pair (*x, x’)* with *x≠x’* so that h(*x*)=h(*x’*)
(collision resistance)

Birthday attack example

$$C(23,2) = \frac{23!}{21!\,2!} = \frac{21!\,22 * 23}{21!\,2} = 11 * 23 = 253$$

A:2 people having b/d on a different day

$$P(A) = 1 - \frac{1}{365} = \frac{364}{365} \approx 0.99$$

B: All people having a different d/b

$$P(B) = P(A)^{253} \approx 0.49$$

C: At least one has the same b/d

$$P(C) = 1 - P(B) \approx 0.51$$

Hashing example

How to hash a string?

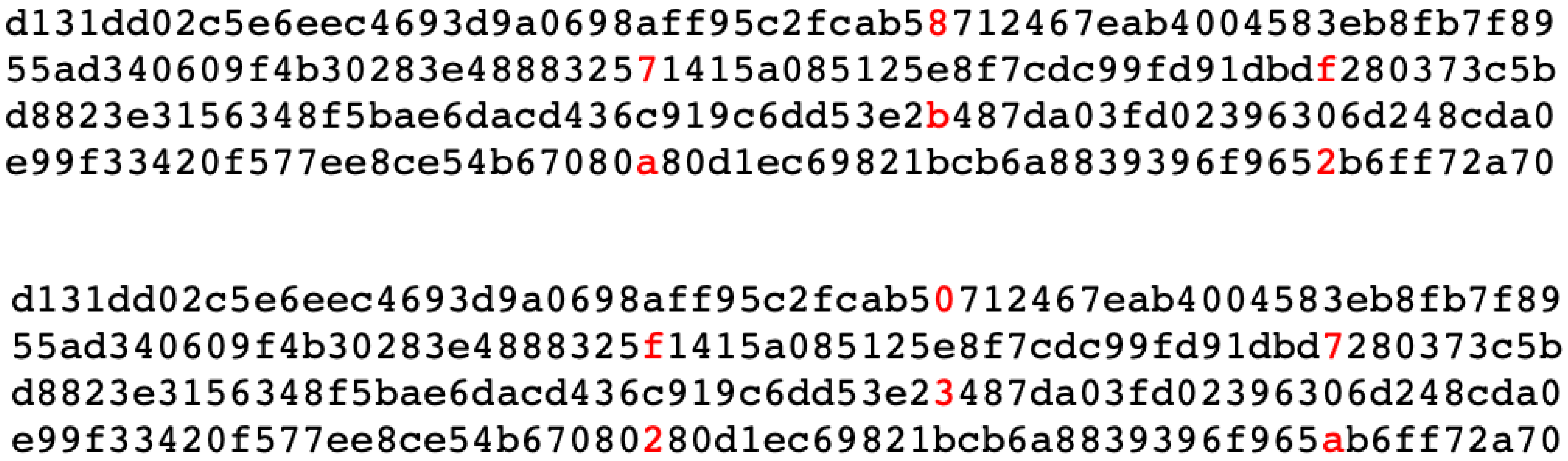
```
import hashlib
str1 = b”Hello World!”
hash1MD5 = hashlib.md5()
hash1MD5.update(str1)
hash2MD5 = hashlib.md5()
hash2MD5 = hashlib.md5(str1*100)
```

```
>>> print(hash1MD5.hexdigest())
‘ed076287532e86365e841e92bfc50d8c’
>>> print(hash1MD5.digest())
b'\xed\x07b\x87S.\x866^\x84\x1e\x92\xbf\xc5\r\x8c'
>>> len(hash1MD5.digest())
16
```

```
>>> print(hash2MD5)
c252ff6f54841f4970a9dd60aac5f5a2
>>> hash2MD5.digest_size
16
```

Collisions in MD5

Example of 2 different sequences of 128 bytes that have the same MD5 hexdigest



```
d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

```
d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965ab6ff72a70
```

Source: https://www.mscs.dal.ca/~selinger/md5collision/
Paper: X. wang, H. Y, “How to Break MD5 and Other Hash Functions”,
http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf

How to…

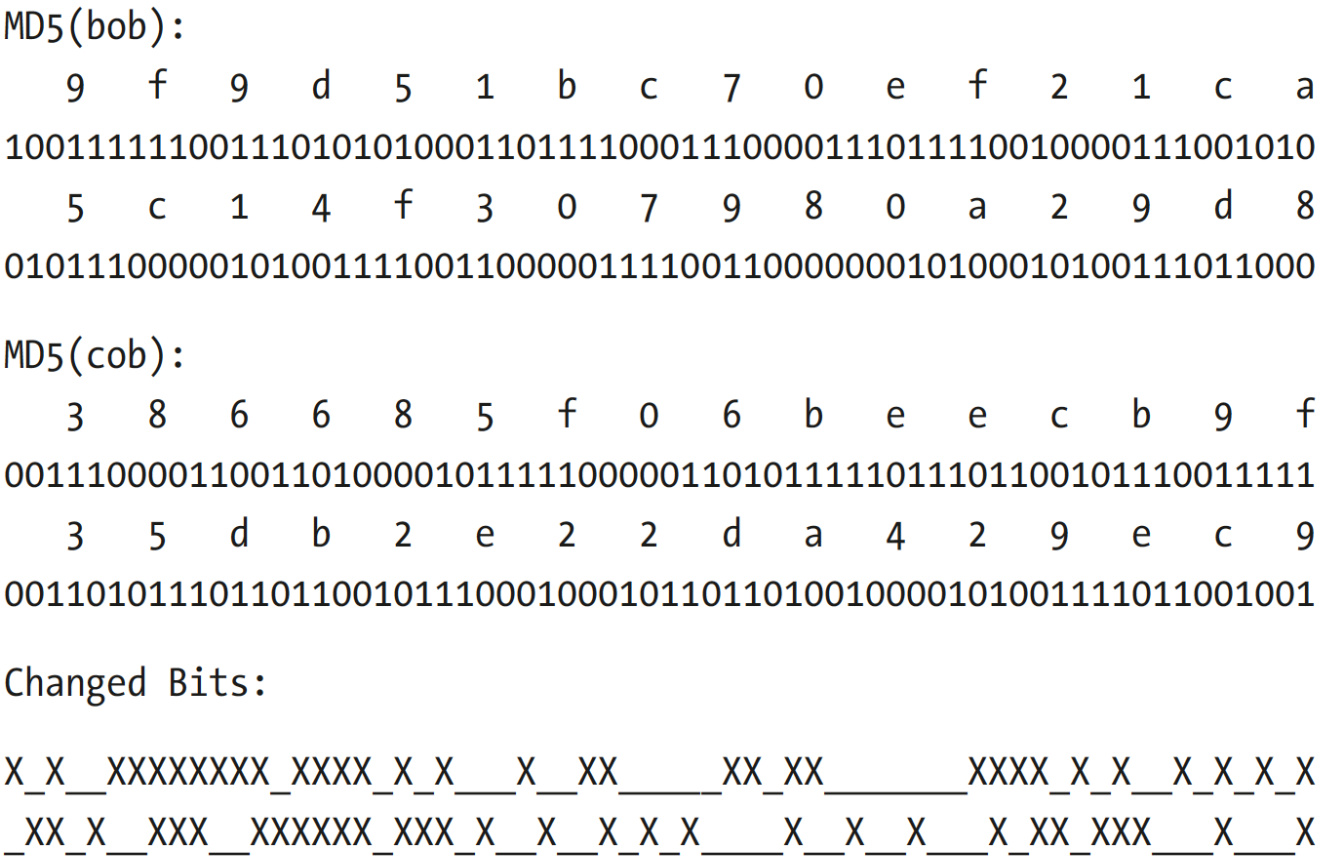
```
>>> fout = open('bin1', 'wb')

>>> data1 =
b"\xd1\x31\xdd\x02\xc5\xe6\xee\xc4\x69\x3d\x9a\x06\x98\xaf
\xf9\x5c\x2f\xca\xb5\x87\x12\x46\x7e\xab\x40\x04\x58\x3e\x
b8\xfb\x7f\x89\x55\xad\x34\x06\x09\xf4\xb3\x02\x83\xe4\x88
\x83\x25\x71\x41\x5a\x08\x51\x25\xe8\xf7\xcd\xc9\x9f\xd9\x
1d\xbd\xf2\x80\x37\x3c\x5b\xd8\x82\x3e\x31\x56\x34\x8f\x5b
\xae\x6d\xac\xd4\x36\xc9\x19\xc6\xdd\x53\xe2\xb4\x87\xda\x
03\xfd\x02\x39\x63\x06\xd2\x48\xcd\xa0\xe9\x9f\x33\x42\x0f
\x57\x7e\xe8\xce\x54\xb6\x70\x80\xa8\x0d\x1e\xc6\x98\x21\x
bc\xb6\xa8\x83\x93\x96\xf9\x65\x2b\x6f\xf7\x2a\x70"

>>> fout.write(data1)
>>> fout.close()
```

Size of binary values

```
>>> hash1MD5.update(b"Hello World1")

>>> bin1=bin(int (hash1MD5.hexdigest(),16))

>>> len(bin1)
129

>>> print(bin1)
0b1011000101010001010010000100110110010111111
010100100111101100100000111100100001011111111
101000111111101111000001010001101111010
```

Avalanche effect

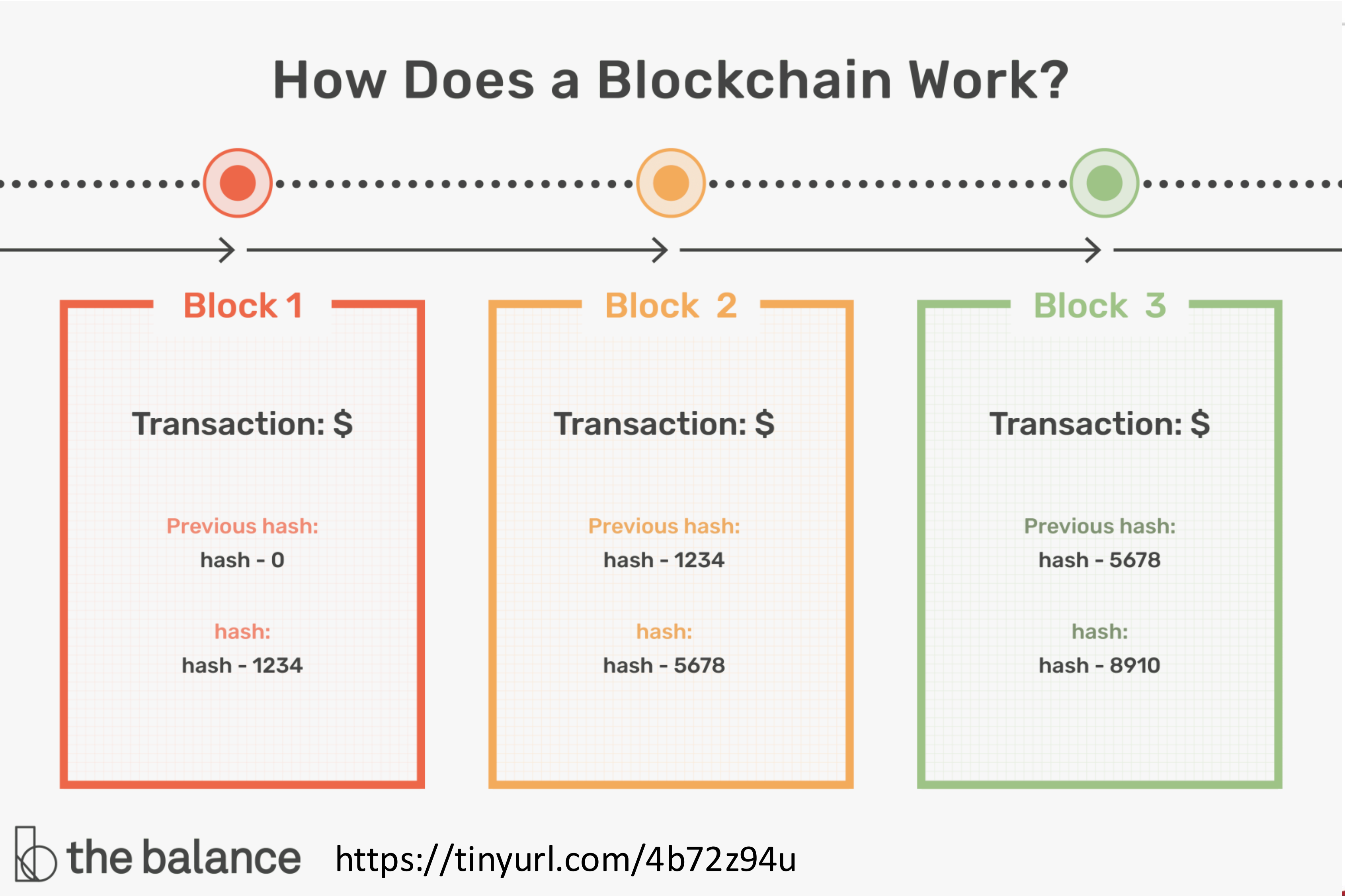
Example from “Practical Cryptography in Python”



```
MD5(bob):
   9   f   9   d   5   1   b   c   7   0   e   f   2   1   c   a
1001111110011101010100011011110001110000111011110010000111001010
   5   c   1   4   f   3   0   7   9   8   0   a   2   9   d   8
0101110000010100111100110000011110011000000010100010100111011000

MD5(cob):
   3   8   6   6   8   5   f   0   6   b   e   e   c   b   9   f
0011100001100110100001011111000001101011111011101100101110011111
   3   5   d   b   2   e   2   2   d   a   4   2   9   e   c   9
0011010111011011001011100010001011011010010000101001111011001001

Changed Bits:

X_X__XXXXXXXX_XXXX_X_X___X__XX_____XX_XX_______XXXX_X_X__X_X_X_X
_XX_X__XXX__XXXXXX_XXX_X__X__X_X_X____X__X__X___X_XX_XXX___X___X
```

- Difference is 64 bits
- Avalanche helps collision resistance

Application of hash functions



How Does a Blockchain Work?

Block 1

Transaction: $

Previous hash:
hash - 0

hash:
hash - 1234

Block 2

Transaction: $

Previous hash:
hash - 1234

hash:
hash - 5678

Block 3

Transaction: $

Previous hash:
hash - 5678

hash:
hash - 8910

the balance

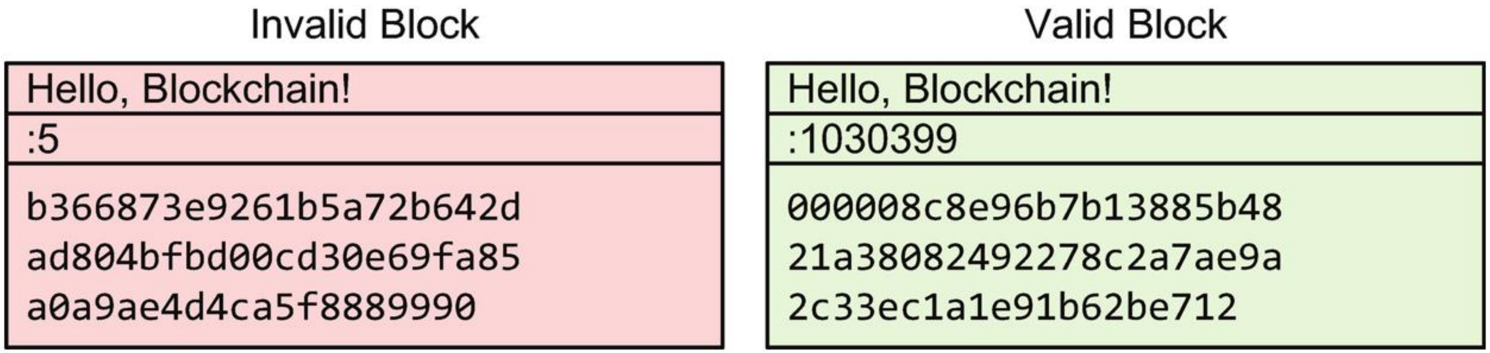
https://tinyurl.com/4b72z94u

Application of hash functions (2)

- Main idea
  - Protect each block with a hash
- Incentive
  - Give an award when a new block is added, but make it difficult to produce it.
- Sequence of actions
  - A user can request a transaction
  - Miners get the request and create a candidate block
  - The block has the transactions, metadata, etc.
  - It’s added to the blockchain when the miner solves a puzzle!

What’s the puzzle?

- Find a SHA-256 hash value that is smaller than a threshold.
- The threshold defined the difficulty of the network
- The puzzle is solved when adding a nonce to the block, which will result in producing a hash value with a certain number of leading zeros.



Invalid Block

Hello, Blockchain!

:5

```
b366873e9261b5a72b642d
ad804bfbd00cd30e69fa85
a0a9ae4d4ca5f8889990
```

Valid Block

Hello, Blockchain!

:1030399

```
000008c8e96b7b13885b48
21a38082492278c2a7ae9a
2c33ec1a1e91b62be712
```

Source: Chapter 2: Applied Cryptography in Python

More applications… Hash-based Message Authentication Codes (HMAC)

- Collision resistance + unforgeability

```
from cryptography.hazmat.primitives import hashes, hmac
import os
key = os.urandom(32)

h_sender = hmac.HMAC(key, hashes.SHA256())
h_sender.update(b"This is my message")
signature = h_sender.finalize()
```

Sender-side code

```
h_receiver = hmac.HMAC(key, hashes.SHA256())
h_receiver.update(b"This is my message")
try:
        h_receiver.verify(signature)
        print(b"OK")
except:
        print(b"Something went wrong")
```

Receiver verification

Structure of your code…

Modules you want to import

```
import XYZ
```

List of functions you implement

```
def  myFunction():
     # TODO

     return # TODO
```

Have a main section to call your functions

```
if __name__ == “__main__”:
     x = myFunction()
```