PWL NYC LIGHTNING TALK

SHAMWOW

HASHES, HASHES, HASHES!

- Desirable properties of hashes:
  - One-way (an arbitrary hash tells you nothing of its input)
  - “Avalanche Effect” (changing the input slightly results in a huge change in output)
- Fast (memory- and time-efficient to compute)
- Unique (it should be hard to find two bitstrings that share a hash)

SHAMWOW


KITTY PURRY
@rabcyr_alt
Follow
I fought the SHA but the SHA1
9:22 PM - 3 Oct 2017
1,366 Retweets 4,118 Likes
65
1.4K
4.1K

SLY AND THE FAMILY SHA

- SHA stands for Secure Hash Algorithm
  - An NIST-promoted spec, FIPS PUB 180-4
    - SHA0 (withdrawn): 1993
    - SHA1: 1995
    - SHA-256 (and friends): 2001
- http://dx.doi.org/10.6028/NIST.FIPS.180-4

UGH, DETAILS

- Merkle–Damgård construction
  - Proposed in Ralph Merkle's Ph.D thesis in 1979
  - Basic steps:
    - Pad message to a standard size, *including initial message length*, then chunk into blocks
    - Compress each block
    - Combine result of compression with previous output and repeat

OK, HOW DO WE IMPLEMENT THIS?

- First rule: DON’T
  - Seriously, unless you know what you’re doing, never implement cryptographic functions on your own (for actual use) unless you have a really good reason
- Luckily, while I don’t know what I’m doing, I’m also not using this for anything other than exploration!

GETTING DOWN TO BIT-NESS

```
"Hello World".unpack("B*")[0]
```

```
H 01001000
E 01100101
L 01101100
L 01101100
O 01101111
  00100000
W 01010111
O 01101111
R 01110010
L 01101100
D 01100100
```

TRY A LITTLE RANDOMNESS

The first 32 bits of the fractional parts of the square roots of the first 8 primes 2 through 19

```
h0 = 0x6a09e667
h1 = 0xbb67ae85
h2 = 0x3c6ef372
h3 = 0xa54ff53a
h4 = 0x510e527f
h5 = 0x9b05688c
h6 = 0x1f83d9ab
h7 = 0x5be0cd19
```

NOTHING UP MY SLEEVE

- Hashes need ‘seed’ numbers for doing the permutations
- These numbers should be meaningless…
  - …but not arbitrary.
- DES was considered suspect for years because its magic numbers were, well, magic – no explanation was given for them by the NSA
- It turns out they were chosen specifically to *avoid* certain theoretical attacks

OK, A LOT OF RANDOMNESS

The first 32 bits
of the fractional parts
of the cube roots
of the first 64 primes
2 through 311

THIS WON'T BE ON THE FINAL

```
k = [
0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b,
0x59f111f1, 0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01,
0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7,
0xc19bf174, 0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da, 0x983e5152,
0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147,
0x06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc,
0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819,
0xd6990624, 0xf40e3585, 0x106aa070, 0x19a4c116, 0x1e376c08,
0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f,
0x682e6ff3, 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
]
```

PADDING THE NUMBERS

```
len = message_in_bits.length

bits = message_in_bits

bits << "1"

bits << "0" * (512 - ((bits.length + 64) % 512))

bits << "%064b" % len
```

BITS, CHUNKS, AND WORDS

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

I THOUGHT THIS WAS CRYPTOGRAPHY, NOT LATIN CLASS

4.1.2 SHA-224 and SHA-256 Functions

SHA-224 and SHA-256 both use six logical functions, where *each function operates on 32-bit words*, which are represented as x, y, and z. The result of each function is a new 32-bit word.

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \tag{4.2}$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{4.3}$$

$$\sum_0^{\{256\}}(x) = ROTR^{2}(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \tag{4.4}$$

$$\sum_1^{\{256\}}(x) = ROTR^{6}(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \tag{4.5}$$

$$\sigma_0^{\{256\}}(x) = ROTR^{7}(x) \oplus ROTR^{18}(x) \oplus SHR^{3}(x) \tag{4.6}$$

$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \tag{4.7}$$

A LITTLE BIT OF RUBY

```
(16..63).each { |w|
  s0 = ror(m[w-15], 7) ^ ror(m[w-15], 18) ^ (m[w-15] >> 3)
  s1 = ror(m[w-2], 17) ^ ror(m[w-2],  19) ^ (m[w-2] >> 10)
  m[w] = (m[w-16] + s0 + m[w-7] + s1) & 0xFFFFFFFF
}
```

A LITTLE BIT OF RUBY

```
(16..63).each { |w|
  s0 = ror(m[w-15], 7) ^ ror(m[w-15], 18) ^ (m[w-15] >> 3)
  s1 = ror(m[w-2], 17) ^ ror(m[w-2],  19) ^ (m[w-2] >> 10)
  m[w] = (m[w-16] + s0 + m[w-7] + s1) & 0xFFFFFFFF
}
```

A LITTLE BIT OF RUBY

```
(16..63).each { |w|
  s0 = ror(m[w-15], 7) ^ ror(m[w-15], 18) ^ (m[w-15] >> 3)
  s1 = ror(m[w-2], 17) ^ ror(m[w-2],  19) ^ (m[w-2] >> 10)
  m[w] = (m[w-16] + s0 + m[w-7] + s1) & 0xFFFFFFFF
}
```

LOSE BITS OFF YOUR WAISTLINE WITH THIS ONE SIMPLE TRICK

```
9 >> 1 = 4

4 >> 1 = 2

2 >> 1 = 1

1 >> 1 = 0
```

LOSE BITS OFF YOUR WAISTLINE WITH THIS ONE SIMPLE TRICK

`1001` >> 1 = `100`

`100` >> 1 = `10`

`10` >> 1 = `1`

`1` >> 1 = `0`

SPIN ME RIGHT ROUND

4. The *rotate right* (circular right shift) operation $\boldsymbol{ROTR}^{n}(\boldsymbol{x})$, where x is a w-bit word and n is an integer with $0 \leq n < w$, is defined by

$$ROTR^{n}(x)=(x >> n) \vee (x << w - n).$$

THERE IS A SEASON, TURN, TURN, TURN

```
9   >>> 1 = 12

12  >>> 1 = 6

6   >>> 1 = 3

3   >>> 1 = 9
```

THERE IS A SEASON, TURN, TURN, TURN

```
`1001` >>> 1 = `1100`
`1100` >>> 1 = `0110`
`0110` >>> 1 = `0011`
`0011` >>> 1 = `1001`
```

THERE IS A SEASON, TURN, TURN, TURN

```
`1001` >>> 1 = `1100`
`1100` >>> 1 = `0110`
`0110` >>> 1 = `0011`
`0011` >>> 1 = `1001`
`1001` >>> 1 = `1100`
`1100` >>> 1 = `0110`
`0110` >>> 1 = `0011`
`0011` >>> 1 = `1001`
```

HOW WIDE IS MY WHAT

```
`1` ror 1 = 1

`01` ror 1 = 2

`00000001` ror 1 = 128

`00000000000000000000000000000001` ror 1 =
2147483648
```

BITMASKS FOR DUMMIES

```
(((num >> shift) | (num << (32-shift))) & ((2 ** 32) - 1))
```

```
        (((1 >> 1) | (1 << (32-1))) & ((2 ** 32) - 1))

00000000000000000000000000000000 |
10000000000000000000000000000000 &
11111111111111111111111111111111

10000000000000000000000000000000

2147483648
```

BITMASKS FOR DUMMIES

```
                    201 = `11001001`

(((num >> shift) | (num << (32-shift))) & ((2 ** 32) - 1))

(((201 >> 2) | (201 << (32-2))) & ((2 ** 32) - 1))

       00000000000000000000000011001001 |
11001001000000000000000000000000 &
       11111111111111111111111111111111

    01000000000000000000000011001001

    4294967496
```

PRE-LOADING THE NUMBERS

```
a = h0
b = h1
c = h2
d = h3
e = h4
f = h5
g = h6
h = h7
```

3. For t=0 to 63:

{

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$

$$T_2 = \sum_0^{\{256\}}(a) + Maj(a,b,c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = T_1 + T_2$$

}

LOSSY COMPRESSION IS A VIRTUE

```
s1 = ror(e, 6) ^ ror(e, 11) ^ ror(e, 25)
ch = (e & f) ^ (~(e) & g)
tmp1 = (h + s1 + ch + k[w] + m[w]) & 0xFFFFFFFF

s0 = ror(a, 2) ^ ror(a, 13) ^ ror(a, 22)
maj = (a & b) ^ (a & c) ^ (b & c)
tmp2 = (s0 + maj) & 0xFFFFFFFF
```

SUPER BOWL SHUFFLE

```
h = g
g = f
f = e
e = (d + tmp1) & 0xFFFFFFFF
d = c
c = b
b = a
a = (temp1 + tmp2) & 0xFFFFFFFF
```

...AND MERGE IT BACK IN

```
h0 = (h0 + a) & 0xFFFFFFFF
h1 = (h1 + b) & 0xFFFFFFFF
h2 = (h2 + c) & 0xFFFFFFFF
h3 = (h3 + d) & 0xFFFFFFFF
h4 = (h4 + e) & 0xFFFFFFFF
h5 = (h5 + f) & 0xFFFFFFFF
h6 = (h6 + g) & 0xFFFFFFFF
h7 = (h7 + h) & 0xFFFFFFFF
```

TURNS OUT A HASH AIN'T NOTHIN' BUT A NUMBER

- Once you've walked all the message blocks, just convert those eight carry variables h0-h7 into hexadecimal and concatenate them, and you're done!