Introduction to Cryptography: Homework 5

Alvin Lin

January 2018 - May 2018

Hash Algorithm

This hash function uses a pseudorandom number generator to generate a 16-bit hash and uses the input to the hash function as a seed to the random number generator.

Pseudocode

```
def string_to_seed(s):
  seed = ""
  for character in s:
    seed += ascii_code(character)
  return int(seed)

def hash(s):
  seed = string_to_seed(s)
  random.seed(s)
  bits = []
  for i in 1..16:
    bit[i] = random.randint(1)
  return bits
```

This hash function defers the job of generating relatively unique sequences to the pseudorandom number generator, which makes this hash function work differently depending on the underlying `random` library implementation.

Analysis

The Python `random` library uses the Mersenne Twister algorithm to generate pseudorandom numbers. Because the bit length of the hash is only 16 bits, collisions were found almost instanteously.

```
hash("356") == hash("24") == 0x6378
hash("440") == hash("313") == 0x4384
hash("841") == hash("197") == 0xc18
hash("1013") == hash("679") == 0x26a7
hash("1014") == hash("186") == 0x87ce
hash("1060") == hash("624") == 0xafa2
hash("1097") == hash("952") == 0x2882
hash("1219") == hash("760") == 0x929a
hash("1348") == hash("391") == 0xba6
hash("1439") == hash("244") == 0x92f0
hash("1487") == hash("1248") == 0xb645
```

$\text{hash}(\text{"1511"}) == \text{hash}(\text{"678"}) == 0\text{xf1cc}$
$\text{hash}(\text{"1624"}) == \text{hash}(\text{"1350"}) == 0\text{xd2d7}$
$\text{hash}(\text{"1683"}) == \text{hash}(\text{"376"}) == 0\text{x7b59}$
$\text{hash}(\text{"1700"}) == \text{hash}(\text{"1459"}) == 0\text{xc10c}$
$\text{hash}(\text{"1789"}) == \text{hash}(\text{"1039"}) == 0\text{x9626}$
$\text{hash}(\text{"1812"}) == \text{hash}(\text{"92"}) == 0\text{xb39d}$
$\text{hash}(\text{"1897"}) == \text{hash}(\text{"206"}) == 0\text{xff3}$
$\text{hash}(\text{"1933"}) == \text{hash}(\text{"1391"}) == 0\text{x85a9}$
$\text{hash}(\text{"1994"}) == \text{hash}(\text{"1181"}) == 0\text{xa5bc}$

This hash function's properties will depend on the underlying PRNG. If a linear shift feedback register is used, then the input (seed) is used as the start state in the LSFR. Many start states are congruent in that the will have the same period and yield the same sequence of numbers, leading to collision in the hash function above.

Exercise 11.5

We consider three different hash functions which produce outputs of lengths 64, 128, and 160 bit. After how many random inputs do we have a probability of $\epsilon = 0.5$ for a collision? After how many random inputs do we have a probability of $\epsilon = 0.1$ for a collision.

$$t \approx 2^{\frac{n+1}{2}} \sqrt{\ln(\frac{1}{1-\epsilon})}$$

Output length 64 bit:

$$\begin{aligned} t_{0.5} &\approx 2^{\frac{65}{2}} \sqrt{\ln \frac{1}{1-0.5}} \\ &= 2^{32.5} 2^{\log_2(0.832)} \\ &= 2^{32.5-0.264} \\ &= 2^{32.23} \text{ random inputs} \\ t_{0.1} &\approx 2^{\frac{65}{2}} \sqrt{\ln \frac{1}{1-0.1}} \\ &= 2^{32.5} 2^{\log_2(0.324)} \\ &= 2^{32.5-1.623} \\ &= 2^{30.87} \text{ random inputs} \end{aligned}$$

Output length 128 bit:

$$\begin{aligned} t_{0.5} &\approx 2^{\frac{129}{2}} \sqrt{\ln \frac{1}{1-0.5}} \\ &= 2^{64.5} 2^{\log_2(0.832)} \\ &= 2^{64.5-0.264} \\ &= 2^{64.236} \text{ random inputs} \\ t_{0.1} &\approx 2^{\frac{129}{2}} \sqrt{\ln \frac{1}{1-0.1}} \\ &= 2^{64.5-1.623} \\ &= 2^{62.877} \text{ random inputs} \end{aligned}$$

Output length 160 bit:

$$\begin{aligned} t_{0.5} &\approx 2^{\frac{161}{2}-0.264} \\ &= 2^{80.236} \text{ random inputs} \\ t_{0.1} &\approx 2^{\frac{161}{2}-1.623} \\ &= 2^{78.877} \text{ random inputs} \end{aligned}$$

If you have any questions, comments, or concerns, please contact me at alvin@omgimanerd.tech