Assignment: Due Monday, 11:59pm

Chris Betsill

October 24, 2016

1. Birthday Theorem examples

1. Assuming that Social Security numbers behave as a random oracle, what is the probability that at least two people in a group of 350 have the same last four digits of their SSN's? Compute the answer exactly and approximately, using the formulas given in the Birthday Theorem.

```
n <- 10^4
prob <- 1
for(i in 1:349){
 prob <- prob*((n-i)/n)
}
print(1-prob)
```

```
[1] 0.9979294
```

Around $1 - e^{-350^2/2*10^4} = .9978$

2. Approximately how many messages would have to be hashed to do a birthday attack on SHA-256 that has a probability of 0.25 of succeeding? (Use the Birthday Theorem.)

$.75 = 1 - e^{-r/2N}$

$ln(.75) = ln(1) - (-r^2/2N)$

$ln(.75) = -r^2/2N$

$-r^2 = ln(.75) * 2N$

$r = \sqrt{(ln(.75) * 2^{257}/-1)}$

$2.581 * 10^{38}$ messages

3. Approximately how many messages would have to be hashed to do a birthday attack on SHA-256 that has a probability of 0.99 of succeeding? (Use the Birthday Theorem.)

$.01 = 1 - e^{-r/2N}$

$ln(.01) = ln(1) - (-r^2/2N)$

$ln(.01) = -r^2/2N$

$-r^2 = ln(.01) * 2N$

$r = \sqrt{(ln(.01) * 2^{257}/-1)}$

$1.032 * 10^{39}$ messages

2. Find a collision

Find a collision for your hash function `miniSHA`. That is, find two strings `x1` and `x2` such that `miniSHA(x1)` equals `miniSHA(x1)`. Use a birthday attack with randomness (everyone should get different answers). Do your work in a separate R script, outside of this file and your project. Once you find the collision, show the code that you used for the attack below, along with the strings that you found.

```
require(stringi)
while(1){
  random <- stri_rand_strings(2,5)
  if(miniSHA(random[1]) == miniSHA(random[2])){
    print(random)
    break()
  }
}
```

The strings "LwXdC", and "fnpNq" both have a miniSHA hash as "5c45"

3. Find a document with the same hash

Suppose Alice and Bob are using a signature scheme using the SHA-1 hash function, but instead of using the whole hash, they are just using the first two bytes of the SHA-1 hash. Bob sends the following message to Alice to sign.

```
m1 <- "On the first day of 2017, I agree to pay Bob five dollars."
```

Explain how Bob could modify the following message `m2` to have the same two-byte hash as `m1` by adding no more than 16 additional whitespace characters between the words. Your answer should explain why Bob is guaranteed to find a collision. (Apply the Pigeonhole Principle.)

```
m2 <- "On the first day of 2017, I agree to pay Bob one million dollars in small unmarked bills."
```

There are 16^2 possible two-byte hashes, and we can make 601,080,390 (c(32,16)) different stings just by adding whitspace between characters. Since this is far greater than the 16^2 possible two-byte hashes, we can guarantee (using the Pigeonhole Principle) that there will be a collison when hashing.

4. Find a multicollision

Let $p = 953945908395498035409837 59$. Consider the hash function $h(x) = x^2 \bmod p$, defined on the set of natural numbers. Find five different messages (i.e., numbers) that have the same hash. (Multiple collisions are called *multicollisions*.)

You can add or subtract any number to p, and then add any multiple of p back to that number, resulting in the same hash.

```
p <- as.bigz("95394590839549803540983759")
m <- p-17
hash <- function(num){
  powm(num, 2, p)
}
```

```
for(i in 7:13){
  print(m+i*p)
  print(hash(m+i*p))
}
```

```
Big Integer ('bigz') :
[1] 763156726716398428327870055
Big Integer ('bigz') :
[1] 289
Big Integer ('bigz') :
[1] 858551317555948231868853814
Big Integer ('bigz') :
[1] 289
Big Integer ('bigz') :
[1] 953945908395498035409837573
Big Integer ('bigz') :
[1] 289
Big Integer ('bigz') :
[1] 104934049923504783895082133 2
Big Integer ('bigz') :
[1] 289
Big Integer ('bigz') :
[1] 114473509007459764249180509 1
Big Integer ('bigz') :
[1] 289
Big Integer ('bigz') :
[1] 124012968091414744603278885 0
Big Integer ('bigz') :
[1] 289
Big Integer ('bigz') :
[1] 133552427175369724957377260 9
Big Integer ('bigz') :
[1] 289
```

5. Authenticated Erequire(stringi)ncryption

Suppose Alice and Bob both have public encryption functions E_A and E_B, and private decryption functions E_A^{-1} and E_B^{-1}. Let m be a message that Alice wants to send to Bob. Alice sends $c = E_B(E_A^{-1}(m))$. Then Bob applies $E_A(E_B^{-1}(c))$.

1. Explain why Alice has access to the functions she used and Bob has access to the functions he used.

Both parties have access to their own functions, and have made their public functions available to the other party. This allows for either party to both send encrypted messages and decrypt recieved messages.

2. What will be the result of Bob's calculation?

Bob's calculation will result in the plaintext message m.

3. Could anyone other than Alice have sent the message? How does Bob know?

No, Bob can be sure that Alice sent the message since when he decrypts the message with her public key, it will only work if the ciphertext was encrypted with her private key.

4. Could anyone other than Bob have received (and decrypted) this message?

Since the message was encrypted using Bob's public key and Alice's private key, only someone who knows both Bob's private key and Alice's public key can decrypt the message (even if the message was intercepted).

5. Make this algorithm into an authenticated encryption scheme using RSA. Since Alice and Bob both need private and public keys, start by making both Alice and Bob do steps 1 through 4 of the RSA algorithm (p. 165) to get n_A, n_B, etc. Then explain what Alice sends and what Bob does.

a) Alice and Bob pick private keys (large primes) p and q and compute $n = p * q$

b) They then choose an exponent $(p-1)(q-1)$

c) They then find d, the inverse of e in mod $(p-1)(q-1)$

d) They then make n and e public

e) To prove that Alice is indeed who she says she is, she first runs her decryption function (only known to her) on a message, and encrypts that with Bob's encryption function (which is now public), and sends this to Bob. To verify, Bob can then use Alice's public encryption function, then his own private decryption function.