CSCI 104L Lecture 17: Cryptography

Modular Exponentiation

Question 1. What is 3^{500} **mod** 10?

Suppose we want to calculate b^n **mod** m. It is impractical to calculate b^n; instead, start by writing n in binary: $n = (a_{k-1}...a_1a_0)_2$

$$b^n = b^{a_{k-1}2^{k-1}+...+a_1\cdot 2+a_0} = b^{a_{k-1}2^{k-1}}...b^{a_1\cdot 2} \cdot b^{a_0}$$

Remember that the a values are either 1 or 0, so we could calculate $b, b^2, b^4, b^8, ...$ and multiply together the values where $a_i = 1$. For example: $3^{11} = 3^8 3^2 3^1$

Remember: if $a \equiv b$ (**mod** m) and $c \equiv d$ (**mod** m), then $ac \equiv bd$ (**mod** m).

$$b^{a_{k-1}2^{k-1}}...b^{a_1\cdot 2} \cdot b^{a_0} \textbf{ mod } m = (b^{a_{k-1}2^{k-1}} \% m) \cdot ... \cdot (b^{a_1\cdot 2} \% m) \cdot (b^{a_0} \% m)$$

And we can calculate b^2 **mod** m by calculating b **mod** $m \cdot b$ **mod** m.

```
ModularExponentiation(b, n = (a_{k-1}a_{k-2}...a_1a_0), m)

  x ← 1
  power = b % m
  for i = 0 → k − 1 do
    if a_i = 1 then
      x ← (x · power ) % m
    power = (power · power) % m
  return x
```

Private Key Cryptography

One of the earliest known uses of cryptography was by Julius Caesar. He made messages secret by shifting each letter three letters forward in the alphabet.

$f(p) = (p + 3)\%26$

This is an example of a private key cryptosystem. In this example, "FIGHT ON" would be encrypted as "ILJKW RQ."

Knowing the encryption key lets you quickly find the decryption key. The decryption key would be

$f(c) = (c - 3)\%26$

Therefore, any pair of people who want to communicate secretly must have the same key. They must securely exchange this key, which can pose a problem if they are limited to digital communication. You would have to send the key unencrypted, as the recipient would not yet know how to decrypt it.

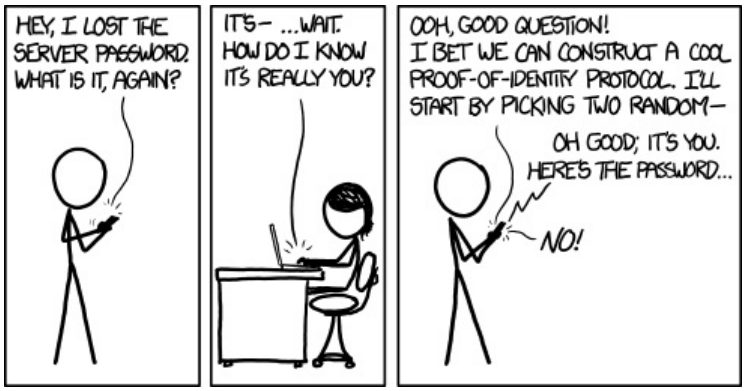

Figure 1: XKCD # 1121 : Identity. Not sure why I just taught everyone to flawlessly impersonate me to pretty much anyone I know. Just remember to constantly bring up how cool it is that birds are dinosaurs and you'll be set.

Public Key Cryptography

A more interesting form of cryptography is a public key cryptosystem. In these systems, knowing how to send a message does not help you decrypt the message. Therefore, you can make your encryption key public. You must only keep the decryption key private.

How could two people set up a secure communication, using public key cryptography?

The RSA public key cryptosystem was first discovered in secret government research. It was discovered independently, and introduced to the world, by Rivest, Shamir, and USC's very own Leonard Adleman.

RSA Encryption

Suppose I want everyone in the world to be able to send me a message that only I can read. I am going to select values e, n and publish them. If you want to send me a message M, compute $C = M^e$ **mod** n, and send C as your message. Of course, your message isn't usually a number, but you can take a block of letters and translate them to an integer M. These will be large values in practice: n will be around 400 digits long. This is why we want the Modular Exponentiation algorithm!

Suppose we want to encrypt the message STOP, with $n = 2537$ and $e = 13$. We don't want to send a given message whose value is at least n, so we'll break it into blocks, each of which is less than n. In this case, $STOP = 18 - 19 - 14 - 15$. Group this into 2 blocks: $1819 - 1415$. Now calculate 1819^{13} **mod** $2537 = 2081$ and 1415^{13} **mod** $2537 = 2182$. Therefore you would send the message $2081 - 2182$.

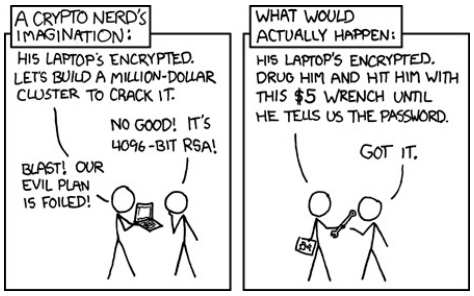

Figure 2: XKCD # 538: Security. Actual actual reality: nobody cares about his secrets. (Also, I would be hard-pressed to find that wrench for $5.)

RSA Decryption

The variables e and n form the public key portion of the encryption, which can be made publicly known. But these values are selected in a very specific manner. Most importantly, n should be the product of two large (and fairly random) prime numbers p, q. Furthermore, e should be relatively prime to $(p-1)(q-1)$. In the earlier example, $2537 = 43 \cdot 59$. If you know p and q, you can crack the code using the Euclidean Algorithm (why? SUPER complicated). To crack the code, you would have to factor n. The most efficient factorization methods known require billions of years to factor 400-digit integers.

How do we read a message? If we know e, p, and q, there is an efficient algorithm to calculate the decrypting key d. If we receive a message C, we then compute C^d. It turns out that $C^d \equiv M \pmod{n}$, which means we have recovered the original message.

Digital Signatures

Here's the rudimentary idea behind how digital signatures work.

Suppose Alice's public RSA key is n and e, and her private key is d. She sends her message as normal. However, she also attaches her name (and some extra information, such as the date and time), which has been run through her **decryption** protocol. Anyone can use the "encryption" protocol to decrypt the signature and see ALICE sent it, but no one could create this without knowing the private key.

Note that the signature is not secure: anyone can read it, but only Alice can write it!

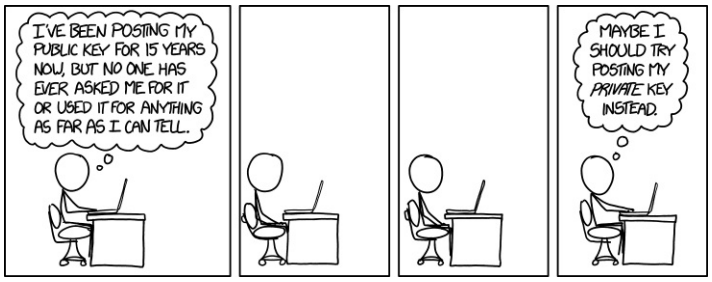

Figure 3: XKCD # 1553: Public Key. I guess I should be signing stuff, but I've never been sure what to sign. Maybe if I post my private key, I can crowdsource my decisions about what to sign.

Hash Tables

A **hash function** takes a valid input (in the case of the last question, a word in the English language) and outputs the entry in the array to store it. To be a good hash function it must:

- be efficient to calculate
- distribute the inputs well
- be consistent

Question 2. Is $h(k) = 0$ a good hash function? Why or why not?

Question 3. Is $h(k) = k\%m$, where m is the size of the table, a good hash function? Why or why not?

Question 4. Is $h(k) =$ a random integer between 0 and $m-1$, where m is the size of the table, a good hash function? Why or why not?

The goal is to design a hash function where the probability of collision is $\leq \frac{1}{m}$. Any "good" hash function that satisfies this is called a "Universal Hash Function."
Question 5. What do we mean when we say that a good hash function is pseudo-random?

Question 6. How can a good hash function be used to store passwords?

Question 7. Suppose we make a hash table to implement the set or map ADT. What is the *worst case* running time for create, insert, delete, contains?

Question 8. Suppose we make a hash table to implement the set or map ADT, and use a Universal Hash Function. What is the *expected* running time for create, insert, delete, contains?

Of course, for that to be useful, we need a Universal Hash Function. Here you go: assume that all inputs are base p, for some prime p. p will be the size of our Hash Table. Now, English words are base 26, but we can translate them to base p quite easily. So an English word will be $w = w_1 w_2 ... w_x, 0 \leq w_i < p$.

Suppose that the longest english word has length x. Then we will choose a random number (base p) $a = a_1 a_2 ... a_x$. We choose this random number ONCE when we create our hash table, and then keep that random number until we delete the hash table. The hash function is then this:

$h(w) = (\sum_i a_i w_i) \bmod p$.