Justin Reyes

CS 131 Project - Cryptographic Hashing

Introduction

Hello Everyone! My name is Justin Reyes and this is my CS 131 semester project. This project is based around one of the main components of Bitcoin and cryptocurrencies in general. Here's a brief outline of what I'm going to be talking about. Pause if you need to, otherwise, I'll go on ahead because there is a lot to talk about.

Preface

To begin with, I have to talk a little bit about Bitcoin, which is a cryptocurrency, used for committing transactions to an online network. Here we see that prefix crypto- since these cryptocurrencies involve a lot of cryptography.

Blockchain

For us to get from bitcoin to hashing, we first need to learn a little bit about blockchain technology, which was implemented for the first time by the creator of Bitcoin.

So what is Blockchain and what makes it so good?

Well, Blockchain, in simplest terms, is a way of storing data using a network of computers and there's three main things that make it good.

- It is secure - Each block (containing data) is encrypted individually.
- Concrete Record - Basically, for Bitcoin, the entire blockchain is public to all people involved with it, and all previous transactions can be reviewed and validated if necessary.
- Decentralized - Instead of committing transactions with middlemen such as banks, lawyers, etc, people deal with each other directly. And because of this, the chain is all linked and attacking one block means attacking all the other blocks.

Of course if the encryption is simple then the attack is simple.

Hashes In The Chain

Here in the next slide, we have a very simple representation of the blockchain. We have data, we have a chain, each block has a hash representing it, and this hash is used for the hash of the next block.

What is in the Hash?

Let's look a little bit deeper into what's in the hash. Well, the hash itself is made up of different types of data. So we can have the usernames of people involved, the transaction being concluded including the amount of money and who it is sent to, the previous hash from the preceding block, and we also have this thing called a proof of work which controls a big part of the hash and validates it. For now this is not important since I'm not here to fully explain blockchain, I'm here to explain the concept of hashing specifically.

So, all of that information is fed through the hashing function, and we end up with a seemingly random combination of binary bits.

Characteristics of Hashes

So, researching all of this information gave me a nice general view of hashing functions, but I wanted to understand them better. That is why I made a program that basically incorporates the concepts I've just described into an algorithm. In that process, I found four main things that I needed to incorporate in my hash:

(1) The hash must be seemingly random. The letters in a hash pretty much mean nothing, they are a representation of the original data. These aren't actually random but just looking at the hash will get you nowhere.

(2) Irreversible. So, for someone to get the information within the hash, they would have to guess and check an unfathomable amount of times until they get the right hash.

(3) Even the slightest changes to the initial message completely changes the encryption, making it impossible to get a pattern. This also relates to number 2.

(4) The length of the hash never differs no matter how long the original data is.

Of course these four things aren't the only things that hashing functions must have. There's also things involving time complexity and code efficiency but for now we will look at those four things only.

Switching to the CPP file

You can see here in my cpp file that I've already commented in some brief descriptions of the functions in my code, and I even made a version of the hashing algorithm that prints out each step to take a closer look, but for this presentation we will take a look more on the idea behind the code rather than the code itself.

Running the Program

Let's see what my program can output. First, let's put in a message, let's do my name: Justin

And this is what we get: 6B80E6E5.

And then let's make some small changes. And we get 3 totally different encryptions, all seemingly random and with the same length, so we know my code works.

Hashing Function Graph

Here I've made a graph that looks deeper into the algorithm which hashes the message "DOG" all in uppercase.

We see here that the algorithm first converts each character from the string text into ASCII. Adding up the ASCII numbers for use later, we'll call that total variable "a".

The ASCII numbers are converted to binary and concatenated. This concatenated string is padded by adding 0s to its end until it reaches a length that is a multiple of 32. This one is initially 21 bits long, so it adds 11 extra zeros.

If it is more than 32 at this point, then it will be padded to the next multiple: 64, or 96, or 128, etc. If it does reach this case, then the string is split into equal parts and XOR'd to get a final length of 32.

It then gets the length of the binary string at this stage for later use. For now we are already at 32 so it does none of that.

This 32 bit block gets copied twice, and the copies are rotated a certain number of times. The first copy rotates using the binary length we took a while ago, and mods by 3. In this case it rotates by 2 to the right.

Then it rotates right again this time using the ASCII total modded by 5, so it rotates right by 3.

It does a similar thing for the second copy, but modded by 7 and 11 and rotating to the left.

These rotated copies are XOR'd, and the result is XOR'd again to the 32 bit block that we initially copied from. Then this value is reversed.

Here you see why it needed to be 32 bits. This final 32 bit block is cut into 8 blocks each having 4 binary bits and each block is converted to hexadecimal. This is a conversion technique we learned in class.

Now that we've converted all of the bits to hexadecimal, we get the final hash value.

Going back to the Characteristics

Taking the 4 requirements that I listed a while ago, I've succeeded in making my own hash function. In reality, there would be a lot more requirements for creating hash functions. Since computers these days are fast, and getting even faster, the method of guessing and checking is actually becoming more viable as computers can guess and check billions of times per second.

This is also why Bitcoin is considered non-energy efficient.

Conclusion

Before I end my presentation, now that we know the characteristics of hashes, let's extend the use of hash functions to an idea simpler than blockchains. For example, let's say there is a website that you can log in to. This website can store your password as a hash, so that if you input the right password, it always corresponds to the right hash and you can log in.

If there is somehow a leak in the website, the only thing hackers would get are lists of hashes which they cannot reverse to the original passwords, so they can't get your account without guessing and checking. In the end, it is a waste of time. This is really the beauty of hash functions.

And that concludes my presentation.

Thank you for watching.