Fall 12-18-2014

A Hash-Cash Based Music Streaming Payment System

Timothy Chen
San Jose State University

CS 298 Report

A Hash-Cash Based Music Streaming Payment System

By

Timothy Chen

Project Advisor: Dr. Chris Pollett

Department of Computer Science

San José State University

One Washington Square

San José, CA 95112

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett

Professor of Computer Science Department, San Jose State University, San Jose, California

---

Dr. Suneuy Kim

Professor of Computer Science Department, San Jose State University, San Jose, California

---

Dr. Thomas Austin

Professor of Computer Science Department, San Jose State University, San Jose, California

Abstract

This project develops a hash-cash based, streaming music payment system. In our system, musicians are paid based on how long their works are listened to. Artists can upload their works to our proof-of-concept service so that people can discover and listen to them. While their works are being listened to, a mining process is run in parallel. The mining process discovers a "listening coin" based on the hash-cash algorithm. Users of our service would pay a monthly fee to access the music library. The monthly fees are then distributed to all artists proportionate to the number of virtual coins they received from users who have listened to the songs they have contributed to the library. The virtual coins are available for public inspection so that the artists can be assured that they are getting a fair share of the subscription fees.

ACKNOWLEDGEMENTS

I would like to thank Dr. Chris Pollett, my advisor, for helping and guiding me through this entire project, and my committee members Dr. Thomas Austin and Dr. Suneuy Kim for providing me help and feedback for my project. I also like to thank my friend Brian for helping me by proofreading my report and giving me feedback.

Table of Contents

List of Figure

List o f Table

1. Introduction

Recently, crypto-currencies such as bitcoin have become popular and more retailers are starting to accept those digital currency. Currently, 70,000 to 80,000 bitcoin transactions occur a day [1]. There are also many increasingly popular music streaming services such as Pandora, Spotify, etc. [2]. Unfortunately, music artists who upload their songs to these music streaming services are paid using the an royalty system such as described in article:[3]. In addition, there is no way for music artists to check how many times their music has been streamed by users. This project explores a hash-cash based music streaming payment system for music artists who allow their songs to be streamed by my software which will address all the problems that music artists currently face.

My software allows music artists, via a web front end, to upload their music in MP3 format to a website I have implemented. This site uses the artist's name, content of music, timestamp, and the music listener's IP address as seed in a hash-cash SHA256 function for the artist to earn the new crypto-currency I have created. The amount earned depends on how long the users of the website listen to the music. While the music artists' works are being streamed to a listener, a mining process is run in parallel. This code use HTML5 Web Workers to allow the streaming and the mining processes to run in parallel. The mining process discovers a virtual coin when a hash-cash algorithm has generated a hash that meets a certain criterion such as a minimum number of leading zeros. The artists and the website are paid a share of the monthly access fees for the music library by the public. The monthly fees are distributed to all artists proportionate to the number of

crypto-coins each artist earned while their work was listened to. My software also has a ranking tool and a tool for verifying crypto-coins. The ranking tool feature ranks artists by the amount of coins earned on a monthly basis. The ranking may be used by record labels or radio stations as a top of the charts list or for invitations to a concert or other events. The verification tool allows artists to check the crypto-coin count for a given month.

This semester I worked on various experiments for the new currency system and music system, basically to combine them together. As part of my earlier CS297, I researched the background of Bitcoin and how it works. I have also implemented WebCL and Web Worker mechanisms to make my music function and hash function run in a parallel processing manner.

2. Background

This section reviews the current payment mechanisms in the music industry, how they work, and what their problems are. This review is needed to help understand my new payment system.

2.1 Music Industry

The most common payment mechanisms in the music industry are basically royalty systems. Many organizations and web sites are involved in the development of, management of, and distribution of the royalties. Below is a review of some of these organizations. One such organization is ASCAP, American Society of Composers, Authors, and Publishers. Their About Page [4] describes their role and history as follows. It was created by a group of prominent music creators at the Hotel Claridge in NYC on February 13th, 1914. As of December 2014, its members include more than 500,000 U.S. composers, songwriters, lyricists, and music publishers of all kinds of music. Any decisions that ASCAP makes can influence society because it was created and controlled by composers, songwriters, and music publishers. The Board of Directors is selected by election by the members. The ASCAP protects its members' rights by licensing, distributing royalties, and copyright for the music publicly.

Rights holders have come a long way since the early day of radio when they would have to pay for their music to be heard. Payola is a term for the practice where in radio stations are paid for playing music promoted by artists. Today, radio stations have to disclose when a piece of music is being promoted, that is, when the radio station

receives compensation to play music. In the past there were troubles for stations receiving the money from a record company or artist:

> As the Payola hearings got under way in February 1960, the public was treated to tales of a lavish disk-jockey convention in Miami bought and paid for by various record companies. One disk jockey, Wesley Hopkins of KYW in Cleveland, admitted to receiving, over the course of 1958 and 1959, $12,000 in "listening fees" from record companies for "evaluating the commercial possibilities" of records [5].

2.1.1 Pandora

Pandora is free web radio station based on the music genome project. It started in the year 2000, and according to their company website: "*Calendar year 2013 GAAP total revenue of $637.9 million and non-GAAP total revenue of $647.5 million, both growing 56% year-over-year" [http://investor.pandora.com/phoenix.zhtml?c=227956&p=irol-newsArticle&ID=1897339]*. Pandora analyzes music based on many attributes such as characteristics and style. It provides a better match with a listener's interests and therefore retains audience and generates revenue through advertisement. Artists are paid by dividing revenue divided from the royalty system.

2.1.2 Spotify

Spotify is free web music site that attracts listeners by providing free access to music from all kinds of devices: desktops, mobile, tablets, etc. Advertising is its source of revenue. According to the following quote from its website:

By bringing listeners into our free, ad-supported tier, we migrate them away from piracy and less monetized platforms and allow them to generate far greater royalties than they were before. Once they are using our free tier, we drive users to our premium subscription tier, at least doubling the amount that they spend on music, from less than $5 per month (the average spent by download consumers in The US) to $9.99 per month for Spotify.[6]

Spotify also uses a royalty system to pay the artist.

2.2 Distribution of Royalty System

From the above section, we see that most music websites pay artists who upload their music according to a royalty system. The graph below shows how the Spotify royalty system works and most music websites use a similar method:

Figure 1: Royalty System from [6]

To explain each section in more detail:

1. Total revenues generated by month from advertising and subscription fees.

2. Artist's total stream (how many people have listened to this artist's music) divided by Spotify's total stream of the music (the total music listen by all users from all artists).
3. Fee for labels and publisher in each territory. In the graph above, Spotify takes away 70% of what the artist earns.
4. The label or the publisher pays each artist according to that artist's contractual royalty rates, so that each artist receives deals with a deduction from their respective labels and publisher (only independent artists can receive 100% of the royalty rate).
5. At the end, royalty is paid out to the artist after those deductions by the web sites, record labels, and publishers.

2. 3 Royalty System Issues

The above method has some problems. For example, if a website is more popular, then the artists will be paid less as explained by the first two boxes. If a music website earns $500 in revenue, the total number of songs streamed is 10,000, and if an artist only had one song listened to by two users, this artist will earn five hundred multiplied by two, divided by one thousand. However, if the website was twice as popular and the artists with songs on the website produce a lot of music each month, then the same artist on that website will earn five hundred multiplied by two divided by two thousand. So, artists actually are paid less on more popular sites. The graph below, from the website, "The Streaming Price Bible", proves my point:

Music Streaming price Index as of Feb 1, 2014

Store	Per Stream	Total in website Downloads
Nokia	0.07411	9
Google Play	0.04573	15
Xbox Music	0.03212	22
simfy	0.01626	43
Napster	0.01578	44
MediaNet	0.01140	61
Rhapsody	0.01122	62
Muve Music	0.00875	80
Deezer	0.00754	93
Rdio	0.00692	101
Spotif	0.00521	134
MySpace Music	0.00094	745
Amazon Cloud	0.00012	5,862

Figure 2: Music Stream Price Index from [7]

The above shows us that the more popular a music website is, the less an artist is paid per song. For example: Amazon Cloud has the most downloads, but their artists are paid less than any other music website. The main problem is that artists receive pay by the number of streams, which means if an artist has a good reputation for their music, but the music is horrible, people will just listen for a few seconds and then choose another one, but the artist who creates horrible music will still be paid as well. The way my project will solve this problem is by having listening coins that are created by a hash-cash algorithm when it has generated a digest with the proper number of leading zeros (which in my implementation is four leading zeros). The generation takes time and luck. The

most important factor is time, so the longer a user listens to a song, the more chance the artist will have in earning a virtual coin. This encourages artists to focus on quality rather than quantity, so that people listen for a long time without stopping or else they will end up earning nothing. One artist can just upload one song and have a lot of people listen for long time, and another artist can produce a hundred songs every month, but have terrible music and still earn less. When there are new songs released, people will like to listen to new music, which will be unfair for good artists. My virtual coin combined music method will solve this problem. People can only receive a virtual coin when others listen to his/her music long enough.

2.4 My System vs Spolifys

For my proposed system, I will take only ten percent of the total revenue for label and hosting the website. Popular artists can participate and make more money. Customers will like to visit to my website because they know that the quality of the music on this website is very good. For example, the chart below shows the comparison of my system vs the royalty system. For example, if my website and the royalty system generate the same revenue, assume that this artist is very good and one hundred users listen to his music for at least 5 minutes. Let us also assume that every minute, one virtual coin will be generated, so this artist has 500 virtual coins. If there are five hundred other artists who upload one song to each of the websites and their music is very bad and only one person listens to each of their songs for around one minute, generating an additional five hundred virtual coins, my website will have generated a total of one thousand virtual coins. Assume all the artists are independent so they receive 100% of the royalty payout.

	100 user listen to 5 min	Artist earn
My System	$1000*(500 the artist own coin /1000 total coin) * *(my cut 70%)	$150
Royalty	$1000*(100 stream /600 stream) *(70% website cut)	$50

Table 1: My System vs Spotify table

Even if my system charges the same rate as they did, which is 70%(my system only charges 10%) of artist's income, the artist will end up with $150 dollars, which is still three times more than the royalty system for good artists.

3. Preliminary Work

This section describes various experiments for we conducted new currency system. I researched the background of bitcoin and how it works, and implemented a hash function and tied it to a music player. The work was divided into four deliverables. The first deliverable involved doing research to understand Bitcoin virtual currency system, installing the Bitcoin wallet, and involved trying to do bitcoin mining. The second deliverable was using JavaScript to demonstrate a hash function to use for the new currency. The third deliverable was using JavaScript to implement SHA256 hash. The fourth deliverable was using JavaScript to implement function to compute hashes as long as audio is being played.

3.1 Bitcoin Research

The Bitcoin virtual currency system uses a peer-to-peer technology to operate without any central authority and there is no need to use banks to manage transactions. The issuing of bitcoins is done by the network. Bitcoin is open-source; anyone can own or mine for bitcoins. bitcoins are spent like real currency. Each transaction is broadcast to the Bitcoin framework with details such as the amount, source, destination, timestamp of the transaction, and the public keys of the bitcoins involved in the transaction. Each bitcoin has a public key and a private key. The private key is used to determine who has ownership of the bitcoin and the public key is used to sign the bitcoin for owner verification during transactions. For example, suppose Owner 1 needs to buy a car from Owner 2. Owner 1 needs to transfer the some of bitcoins to owner 2 by digitally signing a hash of the previous transaction's detail as mentioned above to change ownership of the bitcoins to Owner 2 using Owner 1's private key. The public key will then be used by a

third party to verify that the bitcoin changed ownership to Owner 2 as the graph below shows:

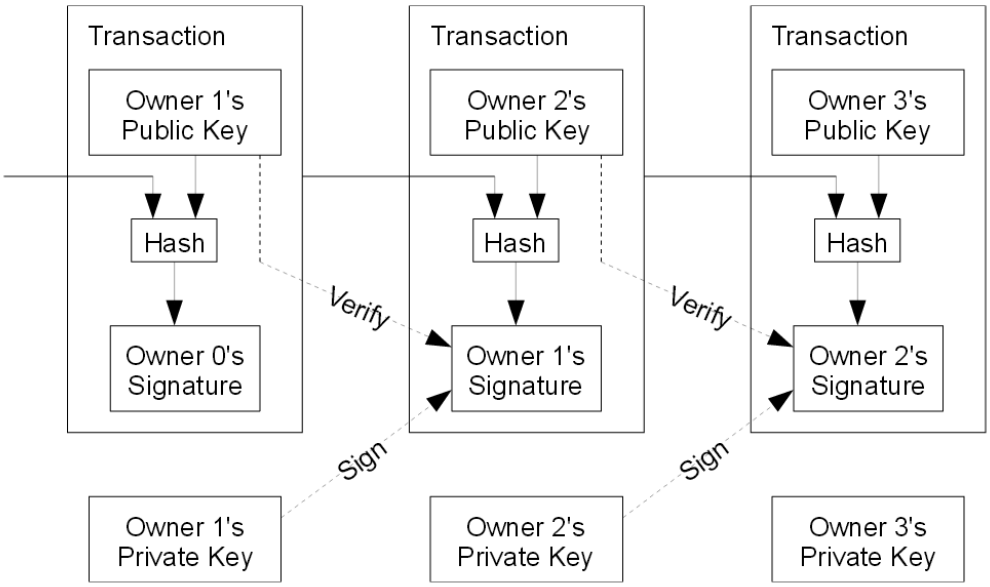


Figure 3: Bitcoin Transaction from [8]

How do we prevent Owner 1 from double-spending the same bitcoin to buy other stuff from Owner 3? The solution is to use a timestamp. As mentioned before, the bitcoin system is peer-to-peer, so anyone who gains ownership of a bitcoin will know the transaction history of the bitcoin. When someone gives someone else a bitcoin, then everyone else knows at what time the transaction occurred. This will prevent the use of the same bitcoin twice because once a transaction occurs, the record of the transaction will go public. Everyone will know what time that bitcoin was used, and the bitcoin will not be able to be reused by the same person twice in a row.

How do we get bitcoins? There are two ways to get bitcoins: first, they can be bought from a bitcoin exchange website such as https://www.mtgox.com/ (now defunct) and https://coinbase.com/. The price for bitcoins fluctuates on a daily basis from about $100 to $1,000, similar to stocks. Second, one can get bitcoins by bitcoin mining. The mining system uses SHA256 to generate a digest of 32 hexidecimal digits. If the digest begins with a certain number of zeros, then you can do verification. Whenever there is any transaction moving bitcoins, people need to verify whether this coin was previously used or not by utilizing their computer's processing power. Whoever is notified and verifies the transaction first will be paid a small reward, which is typically a small part of a bitcoin such as .00001 bitcoin or even less, depending on the amount of bitcoin in the transaction. Over time, more people are doing bitcoin mining, so it will become harder and harder to mine bitcoins. Previously, one could mine bitcoin using a string such as 'abc' and by generating a 32 byte hexidecimal character string that starts with two zeroes like 00af21ac06aceb9cdd0575e82d0d85fc39bed0a7a1d71970ba1641666a44f530. However, now you would have to generate a hash string like with six leading zeros such as: 000000c71f1bda5b63f5165243e10394bc9ebf62e394ef7c6e049c920ea1b181 to successfully mine a bitcoin. Bitcoin is a framework of coins made from digital signatures, which provides strong control of ownership and prevents double-spending. The only way to attack this framework may be to use fake nodes to send a lot of verify requests. The verifying will be very slow and during that time, the attacker can double-send while verifications are not finished. bitcoin exchange websites need to defend against this attack by coding carefully to prevent attackers from selling anything while the bitcoin is not finished verifying prior transactions.

The next part of this deliverable was to install a bitcoin wallet and mine bitcoins. I went to the https://bitminter.com/ website and created an account, so that I have a wallet at the website. All the bitcoins I mine or receive are saved to this wallet. Then I installed the "BitMiner Client" application, which generates different hashes for me to mine bitcoins and do verification. All the bitcoins I obtained from mining are saved to my account in bitminter.com. A screenshot of the "BitMiner Client" application is shown below:

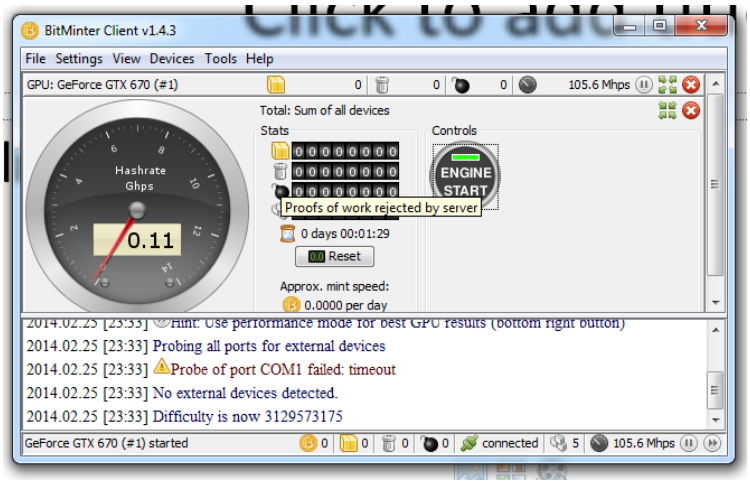
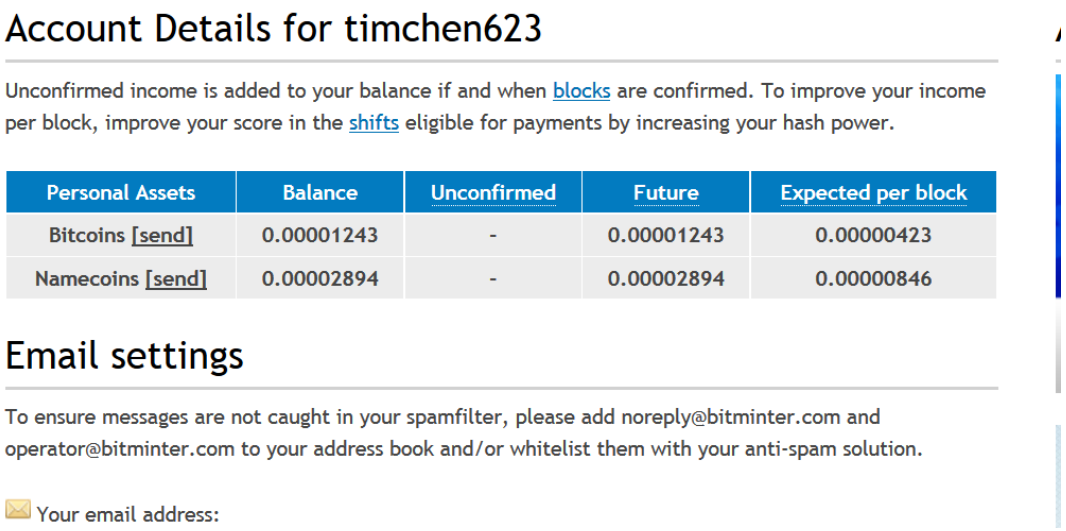


Figure 4: BitMiner Client from https://bitminter.com/

The screenshot below shows how many bitcoins I received from mining:



Account Details for timchen623

Unconfirmed income is added to your balance if and when blocks are confirmed. To improve your income per block, improve your score in the shifts eligible for payments by increasing your hash power.

Personal Assets	Balance	Unconfirmed	Future	Expected per block
Bitcoins [send]	0.00001243	-	0.00001243	0.00000423
Namecoins [send]	0.00002894	-	0.00002894	0.00000846

Email settings

To ensure messages are not caught in your spamfilter, please add noreply@bitminter.com and operator@bitminter.com to your address book and/or whitelist them with your anti-spam solution.

Your email address:

Figure 5: Bitcoins I received from mining from https://bitminter.com/

The application examines my GPU processing power to determine how many hashes I can generate every second to send to the bitcoin framework and see if we can get any strings of hashes with starting zero the bitcoin machine wants. The mining tool can use: ASIC card, our CPU, and graphics card. The speed at which bitcoins are mined depends on how powerful our computer graphics card or ASIC machine we use to generate the hash is. The chart below shows how different ASIC machines, Nvidia graphic card, and AMD(ATI) graph card can mine bitcoins rate, and their current price. I got the information below from [9], [10], [11], and http://www.Amazon.com/[12] :

Product Name	Generate Mhash/s	Price
ASIC Avalon Asic #1	107	$1299
ASIC Avalon Asic #2	117	$1499

ASIC Avalon Asic #3	117	$1499
AMD(ATI) Radeon HD 7970	603.8	$350
AMD(ATI) Radeon HD 7950	517	$229.99
AMD(ATI) Radeon HD 6970	389.9	$169.99
Nvidia GTX 770	123	$370
Nvidia GTX 670	112	$289
Nvidia GTX 660 Ti	96	$189

Table 2: Chart for AISC machines, Nvidia graph card, and AMD (ATI) graph card mining bitcoins rate

From the above, ATI graphics cards seem to be the best for bitcoin mining. For the same price, they are much faster than Nvidia. For example, the Nvidia GTX770 and ATI Radeon HD 7970 both cost around $350, but the GTX770 can only generate 123 hash strings while the Radeon HD 7970 can generate 603 hash strings. The Radeon HD7970 is four to five times faster at generating hashes than the GTX 770 for the same price.

3.2 Implementing a Hash Function

I am using JavaScript to demonstrate a hash function for use in a new currency. After the research I did on Deliverable 1, I found out how to generate hashes by having the user enter a string and how many zeroes they want at the start of the hash string, and then the hash function will return a hash string with the requested number of zeroes at the start. Below is a screenshot of my test application:

Figure 6 My test application for Hash Function

The “text to hash” input box allows a user to enter the text they want to hash and the “enter zero” input box will allows the user to choose how many zeros at the beginning are required to mine for a coin. The “calculate” section uses the MD5 hash algorithm [19] to generate the 32 hex digit to be displayed in the "result" text box from the string that the user entered in the "Text to hash" text box. The code was obtained from [13].

```
<input type="button" onclick="document.getElementById('hash').value =
hex_md5(document.getElementById('userString').value)" value="MD5">
```

Clicking “ADDSTRING” button, generates the hash based on the user inputs in the “text to hash” and “enter zero” input boxes, using a function called “hex_add.” For example, if

a user entered ‘abc’, this function will add characters after it. I use a loop to determine the ASCII code from 0 to 255 to put into the MD5 function to see if the number of zeroes in the start of the hash string generated will match the value of the “Enter Zero” input box. If adding one character cannot generate a hash string as specified, then the function will keep changing and adding additional characters until the generated hash string satisfies the “Enter Zero” input. For example, given input ‘abc’ and ‘2’, character would result in string ‘bac1’, which will be changed to ‘abc2’ if ‘abc1’ does not generate a hash string as specified. If just adding one character does not generate a hash string starting with two zeroes, we will add another character at end, and it will be abc11, then abc12, and so on. Basically, it tries all 256 ASCII codes for each added character until we generate the desired hash string. The screenshot below shows how this code works.

```
for (var i=0; i<len; i++)
{
	var currentone = (str.charAt(i)).charCodeAt(0);
	if (currentone <= 255){
		currentone = currentone+1;
		var res = String.fromCharCode(currentone);
		str = str.replaceAt(i,String.fromCharCode(currentone));
		var hash = hex_md5(userstring+str);
		if (hash.charAt(0) == "0"){
			found = true;
		}
		for (var i=1; i<zerolength; i++)
		{
			if (hash.charAt(i) != "0"){
			   found = false;
			}
		}
		if (found){
			break;
		}
	}
	else
	{
		currentone = 0;
		var init = String.fromCharCode(currentone);
		str = str.replaceAt(i,String.fromCharCode(currentone));
		str += String.fromCharCode(0);
	}
}
```

Figure 7: Hash function implements

3.3 Implementing Sha256

As part of my CS297 preliminary work, I explored some hash functions, I decided to implement SHA256 using the pseudo code from [14]. SHA256 is a one way hashing method, meaning it can encrypt text but cannot be used to generate the original text. The implementation works like this: when a user enters a string, the input string is divided into 512 bit message blocks. Each message block and its prior intermediate hash value are processed by a message schedule and a compression function to produce a 256 bit intermediate hash value. The initial hash value is the square root of the first eight primes 2…19. Each message block is further broken down into 16 32-bit words. The 16 words are extended to a 64 entry message schedule array[20]. The message schedule function is to improve the compression function's quality because the compression function is operated on a longer message schedule array. The compression function consists of bitwise operations such as XOR, AND, OR, SHIFT operations, and so on. I use the pseudo code for my implementation of the SHA256 algorithm as below:

```
Initialize hash values:
(first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19):
var hash = new Array(0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A, 0x510E527F,
0x9B05688C, 0x1F83D9AB, 0x5BE0CD19);
Initialize array of round constants:
(first 32 bits of the fractional parts of the cube roots of the first 64 primes 2..311):
var K = new Array(0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5, 0x3956C25B, 0x59F111F1,
0x923F82A4, 0xAB1C5ED5, 0xD807AA98, 0x12835B01, 0x243185BE, 0x550C7DC3, 0x72BE5D74,
0x80DEB1FE, 0x9BDC06A7, 0xC19BF174, 0xE49B69C1, 0xEFBE4786, 0xFC19DC6, 0x240CA1CC,
0x2DE92C6F, 0x4A7484AA, 0x5CB0A9DC, 0x76F988DA, 0x983E5152, 0xA831C66D, 0xB00327C8,
0xBF597FC7, 0xC6E00BF3, 0xD5A79147, 0x6CA6351, 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);

Process the message in successive 512-bit chunks:
function preProcess (str, l) {
    var binarylist = Array();
    var move = (1 << 8) - 1;
```

```
   for(var i = 0; i < str.length * 8; i += 8) {
      binarylist[i>>5] |= (str.charCodeAt(i / 8) & move) << (24 - i%32);
   }

                        binarylist[l >> 5] |= 0x80 << (24 - l % 32);
   binarylist[((l + 64 >> 9) << 4) + 15] = l;
   return binarylist;
}
```

(The initial values in w[0..63] don't matter, so many implementations zero them here)

copy chunk into first 16 words w[0..15] of the message schedule array

Extend the first 16 words into the remaining 48 words w[16..63] of the message schedule array:

```
for i from 16 to 63
   if (j < 16)
         W[j] = m[j + i];
else
```

Initialize working variables to current hash value:

```
a = hash[0];
      b = hash[1];
      c = hash[2];
      d = hash[3];
      e = hash[4];
      f = hash[5];
      g = hash[6];
      h = hash[7];
```

Compression function main loop:

```
for ( var j = 0; j<64; j++) {
        if (j < 16)
          W[j] = m[j + i];
        else

          W[j] = safe_add(safe_add(safe_add(s1(W[j - 2]), W[j - 7]), s0(W[j - 15])), W[j - 16]);

        temp1 = safe_add(safe_add(safe_add(safe_add(h, S1(e)), ch(e, f, g)), K[j]), W[j]);
        temp2 = safe_add(S0(a), maj(a, b, c));
        h = g;
        g = f;
        f = e;
        e = safe_add(d, temp1);
        d = c;
        c = b;
        b = a;
        a = safe_add(temp1, temp2);
      }
```

Add the compressed chunk to the current hash value:

```
hash[0] = safe_add(a, hash[0]);
      hash[1] = safe_add(b, hash[1]);
      hash[2] = safe_add(c, hash[2]);
      hash[3] = safe_add(d, hash[3]);
      hash[4] = safe_add(e, hash[4]);
      hash[5] = safe_add(f, hash[5]);
      hash[6] = safe_add(g, hash[6]);
```

```
        hash[7] = safe_add(h, hash[7]);
```

Produce the final hash value (big-endian):
digest := hash := h0 **append** h1 **append** h2 **append** h3 **append** h4 **append** h5 **append** h6 **append** h7

Figure 8: Sha256 Implementation

My implementation produced an identical hash results as test values from[15]. When I entered the string 'abc' as my input text and also enter the same input string to that website, both give the same result of ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad.

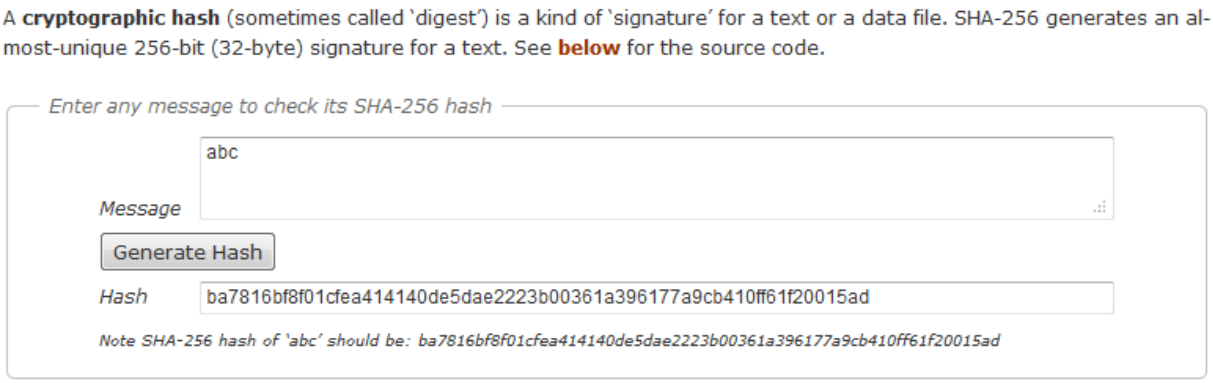


Figure 9: Comparison of my sha256 code and website sha256 code

3.4 Music function

As part of my experiment, I began working on the play music proof of concert. This deliverable combined the two functions that I implemented and described in the previous two sections. I am using JavaScript to implement a function to compute hashes while playing audio. I used the function I implemented in Deliverable 2 to add a music bar into the GUI, which ran the hash function while the music player is played. When the user presses play audio, it will ignore the "Enter Zero" input field. As it plays, my implementation will run a while loop and continually generate hashes with leading zeroes and keeping track of the hash value that has the highest number of leading zeros until the music is finished playing or the pause audio button has been pressed..

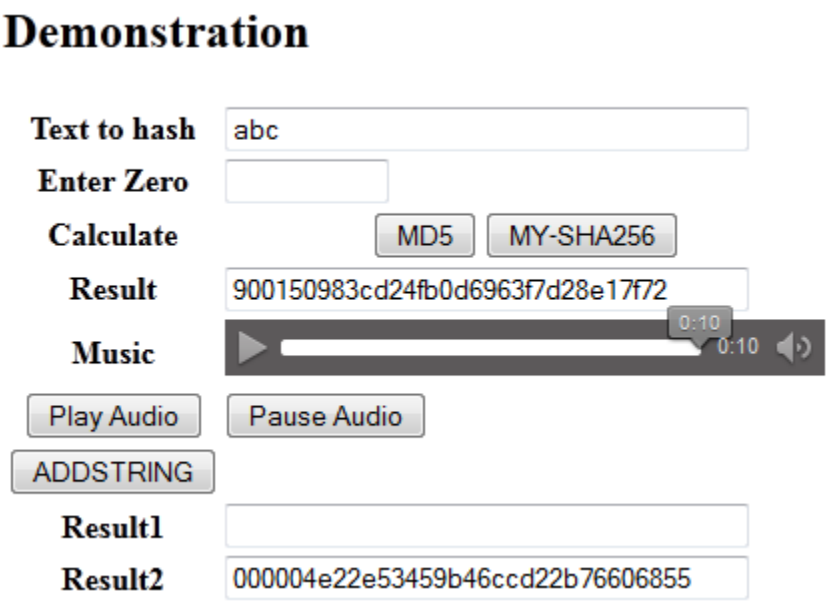


Figure 10: My test application for music function

Once the music stops, the "Result2" field will contain the hash value with the highest number of leading zeros it has generated while the music was playing. For example, the above screenshot shows in the 'Result2' field that during the ten second interval, it has

calculated a hash value with five zeros by use SHA256 function which I implemented in for the last section. I used an HTML5 audio tag to get music from a local file and for the play audio button, I implemented a function called clickplay. When I press the play audio button, it will use the value from the 'text to hash' inputbox, which is used by my clickplay function. While the audio is playing, it will keep incrementing the zeroes to pass to the function hex add starting with only one zero. If it returns a value and the music is still playing, it will add another zero to hex_add function. If the music stops or is paused then it will return the last value generated to the 'Result2' output text box.

This function seems to be working perfectly, but it has a problem. When it runs, I cannot press the pause button because the JavaScript does not support multi-tasking. So when I try to press the pause button, it will not work because it is still running in the while loop as well as playing audio and it creates a problem. The screenshot below shows the results:

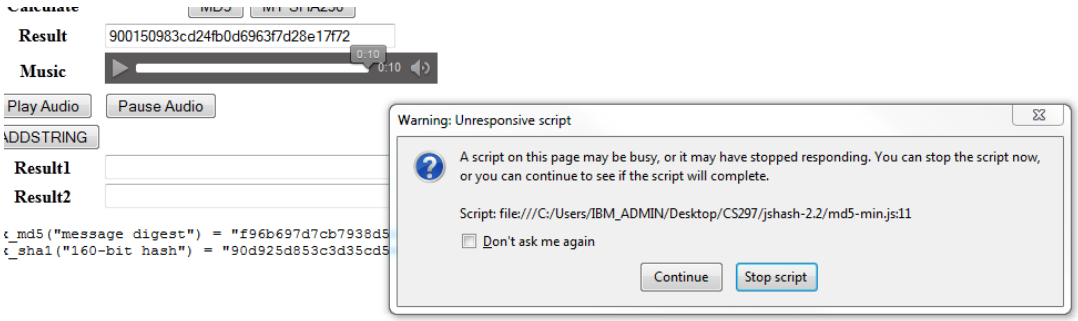


Figure 11: Browser stop

Because JavaScript is single-threaded, I cannot use sleep() function in this method; I have to use setTimeout function instead. During the setTimeout, the code runs smoothly, but

once the timer expires, the problem shows up again. The only solution for this problem is to use WebCL; it supports parallel computing in HTML5 web browsers and will allow me to run two threads at same time, which are play audio function and clickplay function. It uses the GPU and CPU, unlike my example website that only uses the CPU, so it will be much faster.

According to WebCL [16]:

"WebCL 1.0 defines a JavaScript binding to the Khronos OpenCL standard for heterogeneous parallel computing. WebCL enables web applications to harness GPU and multi-core CPU parallel processing from within a Web browser, enabling significant acceleration of applications such as image and video processing and advanced physics for WebGL games. WebCL has been developed in close cooperation with the Web community and provides the potential to extend the capabilities of HTML5 browsers to accelerate computationally intensive and rich visual computing applications."

The problem with this approach is anyone who wants to run it must have both the OpenCL driver and the WebCL browser extension. I use WebCL for my thesis implementation.

4. An MP3-based Currency System

The technologies I am going to use are JavaScript for the front end of the website, PHP for the back end of the website, and new technologies like web worker to do the hash encryption for the currency coin. My proposed MP3-based currency system will mimic the bitcoin architecture with a modified mining process and a slightly different transaction process. Instead of using the hash-cash based proof-of-work system in mining bitcoin, this MP3-based currency system will award artists after they upload an MP3-based song and users listen to it. Like the bitcoin system, the amount of currency transferred is based on the length of time an artist's song has been listened to. I also created a ranking system to rank the artists from highest rank to lowest. All artists who have their music listened to by users will be in the ranking system. First place will be the most popular artist. The artist with the highest rank will receive the most listening coins, which implies that their music is popular. The rank depends on the amount of the virtual coins the artist owns, meaning the artist needs to create good music that encourages people to listen for a long time instead of just listening to a few seconds and then stoping. The record company will want to invite popular music artists to concerts so they can become famous quickly. In addition there is a verification tool for artists to check whether another crypto coin is valid or not.

4.1 Requirements and Design

The MP3-based currency system is a large framework. In my CS297 I was thinking that when users played music the artists would receive some kind of crypto-coin. After trying to come up with a workable system, I decided to use a subscription service

payment model. Users are required to pay a five dollar per month subscription fee for listen the music. At end of the month, the artist will be paid depending on their virtual coin amount divided by the total virtual coin amount and multiplied by revenue generate by subscription fees. The operating cost for this project is around the $2000 to build a server which can store a lot of music and $200 for a business account with an internet service provider which can provide fast upload and download speed. I will offer a seven day free trial period and our music quality will be very good because artists with bad music cannot make money, which may discourage them from uploading more bad music. I anticipate starting with one hundred users which will generate five hundred dollars of revenue. The website will earn ten percent of that revenue which fifty dollars which will be a net loss. Once this service becomes more popular and server is paid off, it will become profitable with at least four hundred and one users.

The design will have the following requirements:

1. System to upload the music
2. A string for each individual virtual coin
3. Use the Sha256 method to mine for virtual coins
4. System to save the coin after the artist receives it
5. Rank for who has the highest virtual coin count for each month
6. To verify each coin

After the requirements were gathered, the design was started. The requirements were converted into classes using object-oriented design. In MP3 system, all the classes and database can be seen in the figure below.

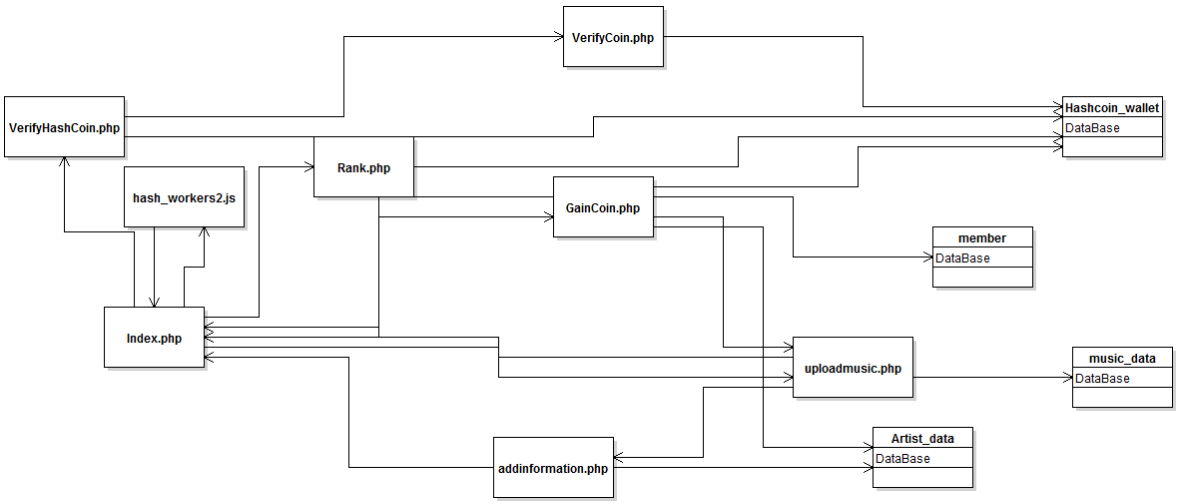


Figure 12: My system design

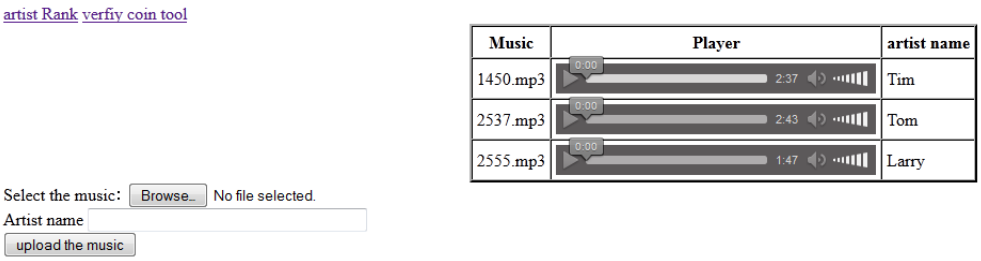


Figure 13: My system's application

4.2 System to upload the music

If an artist wants to upload their music to my website, they click browse, select their music, and click the "upload the music" button.

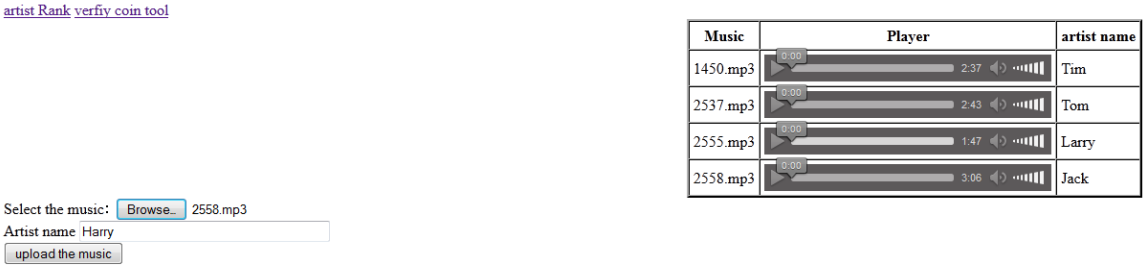


Figure 14: Use my application upload music

For example, if you are Harry, like the above screenshot, and have uploaded music before, clicking the "upload the music" button will take you to uploadmusic.php. In there, you will save your songs in a folder call fileupload, and connect to the database music _data, save the artist name, song names, path of the song, and musicData hashString, which is generated from an MP3 file. We use PHP function file_get_contents(the mp3 file) to convert it to a single long string, then use PHP library Sha256 function to change it into a hex string. It then takes you back to the main page with an extra music line. If you have never uploaded music onto the website before, after you upload the music, it will go to addinformation.php and will ask your email and password as screenshot below:

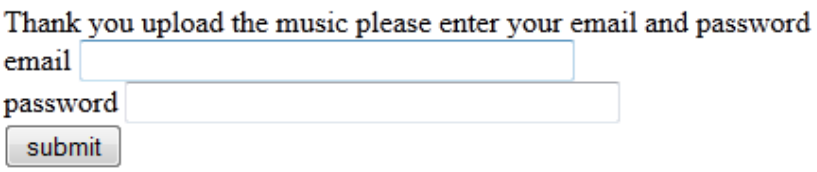


Figure 15: Addinformation.php application

After you enter your email and password, it will go back to the home page.

4.3 A string for each individual coin

Once the user clicks the play button on the music, the name of the artist who uploaded and their music data will be passed to the function audiPlay(name, music data). Inside audiPlay, the play button will make a string for artistname+music_data+timestamp (when the user pressed to play) +user IP address (converted into a hash string by the SHA256 function), which will be passed to the method startworker (string). The timestamp is created in PHP by first setting the time zone date_default_timezone_set (‘America/Los_Angeles) and then creating a new DateTime() variable. This new DateTime variable will contain the current timestamp.

4.4 Use the Sha256 method to mine the virtual coin

The startworker method calls Web Worker to run:

w = new Worker ("hash_workers2.js").

The Inside hash_workers2.js file has hash function and implements Sha256. As shown below, the w.possMessage(a) will pass the string to hash_workers2.js. It will keep generating the hash hex string until it finds 4 leading zero then will return the string (string pass to this method) + nonce (ASCII character added to the string, which will create the 4 leading zero hex string) using the function w.onmessage. For example, if it found one coin, it will return a string without parentheses

(artistname)(music_data)(timestamp)(userIP address) (1)(nonce).

If it found two virtual coin at this music play length of time, it will return a string in this format without parentheses

(artistname)(music_data)(timestamp)(userIpaddress)(1)(nonce)--(artistname)(music_data)(timestamp)(userIp address)(2)(nonce).

It should look like:

Larry8eac221e13834defb2e14d636e1a2417b30009dee009a6a07dd7b862c1b579b02014-11-30 21:22:2012ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a01n½

Then, we save those messages in variable counNumber.

```
w.onmessage = function(event) {
    zeros = event.data;
    var hashString = ""+event.data;
    countNumber = event.data;

};
w.postMessage(a);
```

Figure 16: Web Worker implementation

Once the music ends, or the user clicks stop, it will go to the method stopWorker(). It will terminate the Web Worker and stop generates the hash string and sends the variable countNumber to GainCoin.php file.

4.5 System to save the coin after the artist receives it.

GainCoin.php will split the countNumber variable string by splitting on pattern "--". Because the database cannot read some of the ASCII characters, I used the PHP function **base64_encode($string)** to encode the string so the database can read it and

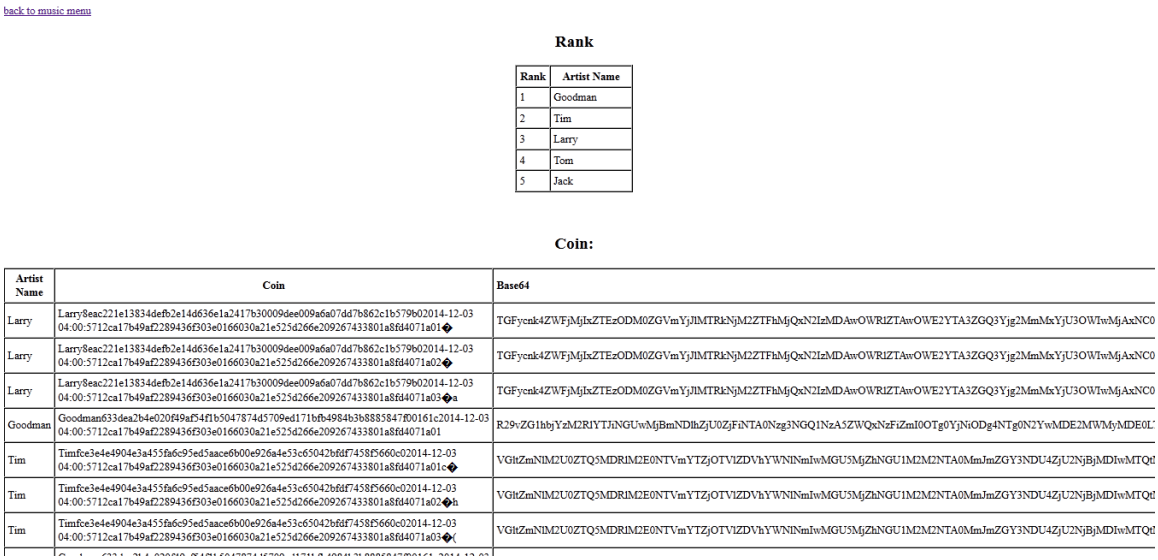
increased the character length for the column for hash_coin to ten thousand so it can be saved in the database by using SQL

"**Alter Table Hashcoin_wallet modify hash_coin varchar(10000)".**

After doing that, I then save each hashcoin one by one to the Hashcoin_wallet database with the artistName. If the current day is at the end of the month, then GainCoin.php will also connect to the member database and get the number of subscribers, multiply that by 5, which is the subscription fee in dollars, to get the revenue. Then it will connect to the Artist_data database to get each artist's name and update the percentage the artist will receive. Finally, it will connect to the Hashcoin_wallet database, calculate the sum of all the virtual coins that all of the artists have earned, then multiply that sum by the percentage each artist is due.

4.6 Rank for who has highest coin of each month

If people or any record company wants to get the most famous music artist to play at their concert, then they can click on the 'Rank' link to get the current most popular artist, It will print out the all the virtual coins and which virtual coins belong to which artist. It is also has ranks to show who has the most virtual coins from most to least.



back to music menu

Rank

Rank	Artist Name
1	Goodman
2	Tim
3	Larry
4	Tom
5	Jack

Coin:

Artist Name	Coin	Base64
Larry	Larry8eac221e13834defb2e14d636e1a2417b30009dee009a6a07dd7b862c1b579b02014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a01	TGFycnk4ZWFjMjIxZTEzODM0ZGVmYjJlMTRkNjM2ZTFhMjQxN2IzMDAwOWRlZTAwOWE2YTA3ZGQ3Yjg2MmMxYjU3OWIwMjAxNC0
Larry	Larry8eac221e13834defb2e14d636e1a2417b30009dee009a6a07dd7b862c1b579b02014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a02	TGFycnk4ZWFjMjIxZTEzODM0ZGVmYjJlMTRkNjM2ZTFhMjQxN2IzMDAwOWRlZTAwOWE2YTA3ZGQ3Yjg2MmMxYjU3OWIwMjAxNC0
Larry	Larry8eac221e13834defb2e14d636e1a2417b30009dee009a6a07dd7b862c1b579b02014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a03a	TGFycnk4ZWFjMjIxZTEzODM0ZGVmYjJlMTRkNjM2ZTFhMjQxN2IzMDAwOWRlZTAwOWE2YTA3ZGQ3Yjg2MmMxYjU3OWIwMjAxNC0
Goodman	Goodman633dea2b4e020f49af54f1b5047874d5709ed171bfb4984b3b8885847f00161c2014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a01	R29vZG1hbjYzM2RlYTJiNGUwMjBmNDlhZjU0ZjFiNTA0Nzg3NGQ1NzA5ZWQxNzFiZmI0OTg0YjNiODg4NTg0N2YwMDE2MWMyMDE0L
Tim	Timfce3e4e4904e3a455fa6c95ed5aace6b00e926a4e53c65042bfdf7458f5660c02014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a01c	VGltZmNlM2U0ZTQ5MDRlM2E0NTVmYTZjOTVlZDVhYWNlNmIwMGU5MjZhNGU1M2M2NTA0MmJmZGY3NDU4ZjU2NjBjMDIwMTQt
Tim	Timfce3e4e4904e3a455fa6c95ed5aace6b00e926a4e53c65042bfdf7458f5660c02014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a02h	VGltZmNlM2U0ZTQ5MDRlM2E0NTVmYTZjOTVlZDVhYWNlNmIwMGU5MjZhNGU1M2M2NTA0MmJmZGY3NDU4ZjU2NjBjMDIwMTQt
Tim	Timfce3e4e4904e3a455fa6c95ed5aace6b00e926a4e53c65042bfdf7458f5660c02014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a03(	VGltZmNlM2U0ZTQ5MDRlM2E0NTVmYTZjOTVlZDVhYWNlNmIwMGU5MjZhNGU1M2M2NTA0MmJmZGY3NDU4ZjU2NjBjMDIwMTQt

Figure 17: Rank.php application

Inside rank.php, I connect to the hashCoin_wallet database using PHP to perform the following query:

```
$query ="SELECT * FROM hashcoin_wallet GROUP BY artist_Name ORDER BY COUNT(*) desc"
```

then process the results to display the rank of the artist as shown at the top of Figure 17. On the bottom part of Figure 17, I used the following sql statement:

```
SELECT * FROM hashcoin_wallet
```

to then display all of the artist names, coins, and encoding base64 version of the hashcoins (for the purpose of verifying described in the next section).

4.7 Verify the coin tool

After I created the ‘Rank’ table and showed all the HashStrings for each coin, I made a verify tool to test the hexString. This is for the artist; if some the artists feel like

they should get more, or they think others' music crypto-coin is invalid, they can use this tool to check other coin hexStrings. On the home page, if you click "verify coin tool" link, it will take you to the verify tool page to test the hexString in testHashCoin.php, which will display the hex string for any input strings. The screenshot below shows my tool for which I put string "88484" in the "Text to hash" field and clicked on "MY-SHA256" button. It will display the hex string with 4 leading zeroes in the result. Thus, this tool can be used to verify the virtual coins created when users listen to enough music.

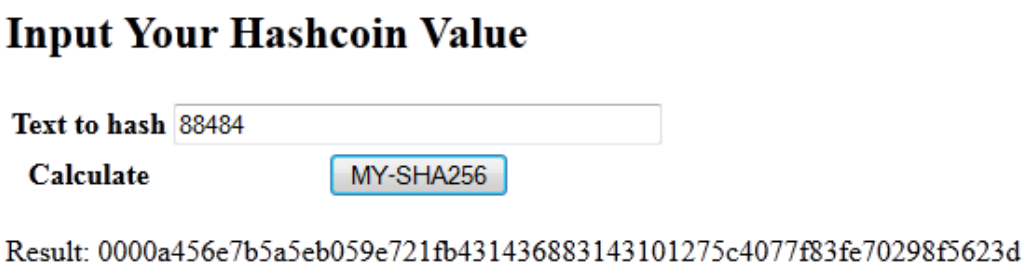


Figure 18: Verify tool application

There is another tool written for testing the base64 encoding version of the hashcoins. Some of the hashcoins cannot be read by the browser because some ASCII characters are unsupported by the browser. Figure 19 shows my tool for which I input a string to encode using base64 of the hashcoin which I got from the verify coin base64 column in the "Text to hash" field and clicked on "MY-SHA256" button. It will display the hex string with 4 leading zeroes in the result as expected.

Figure 19: Verify tool application for Base 64

Then I used the SHA256 function I created and made a textbox allowing a user to enter a string. After they click on the "MY-SHA256 button", the text is passed to the SHA256 method and the result will be displayed below the button. In the verify tool function, I also made a search tool that allows users to search for whomever they want to search to see the virtual coins that person owns.

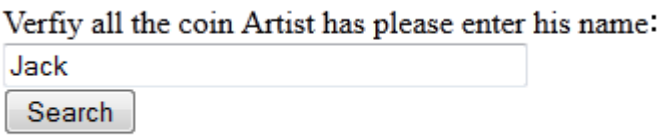


Figure 20: Verify tool search application

Once the user enters an existing artist name, they are taken to verifyCoin.php with the artist name they have entered on previous page and put into SQL statement:

$query ="SELECT * FROM Hashcoin_wallet where artist_Name='$artist_name'".

This statement searches for all the virtual coins that the artist, Jack, has and uses PHP function **base64_decode($string)** to decode the hash_coin string to display what the coins really look like in coin column, and uses PHP library SHA256 function to display

all of the hex strings for the artist to verify that Jack's coins are all valid. The results show as below:

Artist Name	Coin	
Jack	Jack0e2371b4fd4ec412c868762c583cc5e4888bbf3f40404dd4a08686d98458cc032014-12-03 04:36:5812ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a01	00005
Jack	Jack0e2371b4fd4ec412c868762c583cc5e4888bbf3f40404dd4a08686d98458cc032014-12-03 13:36:3112ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a01Z	0000b
Jack	Jack0e2371b4fd4ec412c868762c583cc5e4888bbf3f40404dd4a08686d98458cc032014-12-03 13:36:3112ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a02k$	0000e
Jack	Jack0e2371b4fd4ec412c868762c583cc5e4888bbf3f40404dd4a08686d98458cc032014-12-03 13:36:3112ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a03+	00002

Figure 21: Verify tool search result

5 Experiments

5.1 WebCL: Implement Hash Function by using webcl

I experimented using WebCL to implement the hash function used by my music player. As mentioned before, WebCL will utilize the GPU for processing instead of the CPU. In figure 10, the “calculate” section uses for the hash function algorithm to generate the 32 hex digit to "result" section from the string the user entered. Let us start with the kernel description using OpenCL C language [21]. The idea is to add sixteen vectorIn1 element vectors in global memory, vectorIn1 to vectorIn16, and store the result to the vectorOut, vectorOut2. The kernel code is shown below. All code written in WebCL must inside. Because we can’t pass a struct that contains pointers (mention in [18]section 6.9) we need to declare each variable one by one instead of using **pass vector array()**.

<script id="clProgramVectorAdd" type="text/x-opencl">

....

```
kernel void ckVectorAdd(global uint* vectorIn1,
                        global uint* vectorIn2,
                        global uint* vectorIn3,
                        global uint* vectorIn4,
                       global uint* vectorIn5,
                       global uint* vectorIn6,
                       global uint* vectorIn7,
                       global uint* vectorIn8,
                       global uint* vectorIn9,
                       global uint* vectorIn10,
                       global uint* vectorIn11,
                       global uint* vectorIn12,
                       global uint* vectorIn13,
                       global uint* vectorIn14,
                       global uint* vectorIn15,
                       global uint* vectorIn16,

                         global uint* vectorOut,
                         global uint* vectorOut2,
                         uint uiVectorWidth) {
  uint x = get_global_id(0);
  uint hash = hex_shA256(vectorIn1[x], vectorIn2[x], vectorIn3[x],
  vectorIn4[x], vectorIn5[x], vectorIn6[x], vectorIn7[x],
  vectorIn8[x], vectorIn9[x], vectorIn10[x], vectorIn11[x], vectorIn12[x],
  vectorIn13[x], vectorIn14[x], vectorIn15[x], vectorIn16[x]);

  if (hash < 65536)
  {
      vectorOut[x] = 0 ;
  }

  else
  {
      vectorOut[x] = 1 ;
  }
  vectorOut2[x] = hash;
}
```

....

</script>

Figure 22: WebCL implementation

The reason for passing 32-bit integers instead of strings is because WebCL only allows integers to be passed. A work item is an instance of the kernel and 16 array elements can be executed in parallel to run in the hex_Sha256 function to generate the hash string, which I create in OpenCL C language. The function call "get_global_id(0)" returns the identifier of the processed work item. After calculating the hash value, it will check it has how many leading zero after converting to hexadecimal:

2 leading zero if less then decimal 16777216

3 leading zero if less then decimal 1048576
4 leading zero if less then decimal 65536
5 leading zero if less then decimal 4096
6 leading zero if less then decimal 256

If the hash value has more than 4 leading zeroes, vectorOut1 will return 1 or else it will return 0. Vectorout2 will just return a 32 bit integer num, which can be converted to a hex string in Javascript so we can verify that if has four leading zeroes for testing purposes. OpenCL kernel needs to be passed to OpenCL device so, therefore, we need a JavaScript utility function for finding and loading the kernel according to given id. The function shown below returns the kernel source code.

```
function loadKernel(id) {
    var kernelElement = document.getElementById(id);
    var kernelSource = kernelElement.text;
    if (kernelElement.src != "") {
        var mHttpReq = new XMLHttpRequest();
        mHttpReq.open("GET", kernelElement.src, false);
        mHttpReq.send(null);
        kernelSource = mHttpReq.responseText;
    }
    return kernelSource;
}
```

Figure 23: LoadKernel implementaton

The function vectorAdd represents an OpenCL host program. Before proceeding with actual host program, let us first check that WebCL is installed and generate 16 256-element arrays to act as inputs to run the Sha256 algorithm. Those 16 arrays are created by preprocessing method from previous Sha256 implementation. We will add another character at end, and it will be ‘abc1’, then ‘abc2’, and so on. Basically, it tries all 255 ASCII, adding additional ASCII characters if necessary to put into preprocessing method.

It will create 16 32-bit integers, which will put into an array. They will end up with 16 arrays, each containing 255 elements.

```
function vectorAdd(name) {
    var output = document.getElementById("output");
    output.innerHTML = "";
    try {
        if (window.webcl == undefined) {
            alert("Unfortunately your system does not support WebCL. " +
                "Make sure that you have both the OpenCL driver " +
                "and the WebCL browser extension installed.");
            return false;
        }
        var vectorLength = 256;
        var ulvector = new Array();

        for (var i = 0; i < 16; i++) {
            ulvector[i] = new Uint32Array(vectorLength);
        }
```

Figure 24: JavaScript declare WebCL step 1

Hosting OpenCL computation starts with reserving the required resources. WebCL context is created using the default device of the first available platform. In addition, add 18 buffers. 16 read only buffers for the inputs and two write only buffers for the output. The size of the buffers is given as bytes.

```
var ctx = webcl.createContext();

        var bufSize = vectorLength * 4;
        for (var i = 0; i < 16; i++) {
            bufIn[i] = ctx.createBuffer(WebCL.MEM_READ_ONLY, bufSize);
        }
        var bufOut = ctx.createBuffer(WebCL.MEM_WRITE_ONLY, bufSize);
        var bufOut2 = ctx.createBuffer(WebCL.MEM_WRITE_ONLY, bufSize);
```

Figure 25: JavaScript declare WebCL step 2

Next, create a program object. The kernel code is loaded with the loadKernel function and built for the defined device. Then, the kernel code "clProgramVectorAdd" is selected for the kernel object. It will load coded between the script tags

```
<script id="clProgramVectorAdd" type="text/x-opencl"> </script>

    var kernelSrc = loadKernel("clProgramVectorAdd");
    var program = ctx.createProgram(kernelSrc);
    var device = ctx.getInfo(WebCL.CONTEXT_DEVICES)[0];

    try {
       program.build([device], "");
    } catch (e) {
       alert("Failed to build WebCL program. Error "
            + program.getBuildInfo(device,
                                   WebCL.PROGRAM_BUILD_STATUS)
            + ":  "
            + program.getBuildInfo(device,
                                   WebCL.PROGRAM_BUILD_LOG));
       throw e;
    }


    var kernel = program.createKernel("ckVectorAdd");
    for (var i = 0; i < 16; i++) {
       kernel.setArg(i, bufIn[i]);
    }
    kernel.setArg(16, bufOut);
    kernel.setArg(17, bufOut2);

    kernel.setArg(18, new Uint32Array([vectorLength]));
```

Figure 26: JavaScript declare WebCL step 3

Next, create a command queue then local and global work sizes are defined. The execution is enqueued with enqueueNDRangeKernel. After the execution, the results can be read from the OpenCL device with enqueueReadBuffer. The command queue is flushed with cmdQueue.finish. Finally, cmdQueue.release() to release all the memory obtained during the execution.

```
var localWS = [8];
var globalWS = [Math.ceil(vectorLength / localWS) * localWS];
output.innerHTML += "<br>Global work item size: " + globalWS;
output.innerHTML += "<br>Local work item size: " + localWS;
cmdQueue.enqueueNDRangeKernel(kernel, globalWS.length, null,
                        globalWS, localWS);
          outBuffer = new Uint32Array(vectorLength);
outBuffer2 = new Uint32Array(vectorLength);
cmdQueue.enqueueReadBuffer(bufOut, false, 0, bufSize, outBuffer);
cmdQueue.enqueueReadBuffer(bufOut2, false, 0, bufSize, outBuffer2);
cmdQueue.finish();

cmdQueue.release();
```

Then, we check for zeroes in the outBuffer in a loop, which we will break out of once a zero is found.

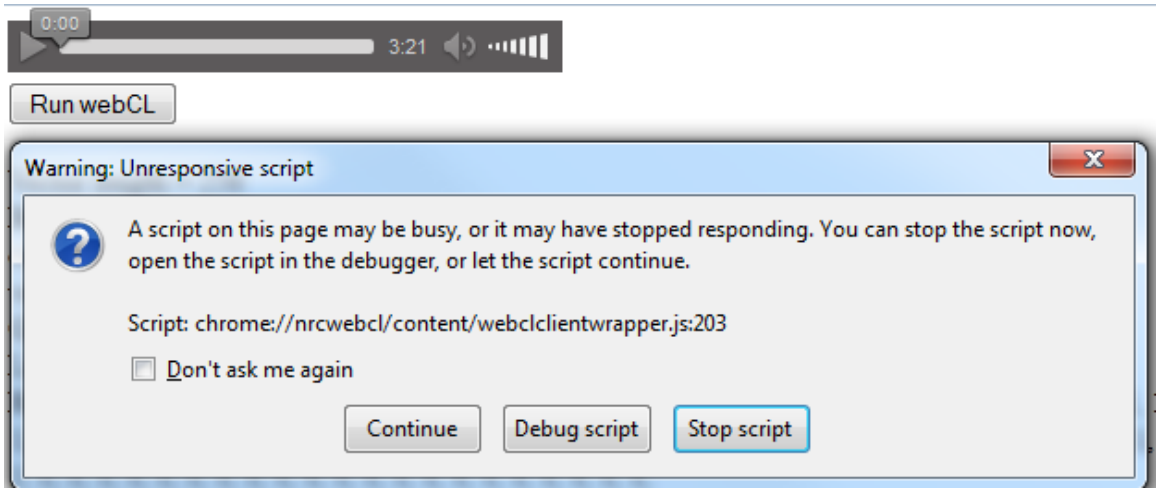
```
for (var i = 0; i < vectorLength; i = i + 1) {
            if (outBuffer[i] == 0) {
               notFound = false;
               break;
            }
            output.innerHTML += outBuffer[i] + ", ";

         }
```

Figure 27: JavaScript declare WebCL step 4

If all the loops have finished running and we still cannot find any zeroes, I will use the function we created called the plusOne method in the section on implementing a hash function. It will add one character at the end of the string and do it again from beginning. If just adding one character does not generate any zeros from the outBuffer, another character will be added. Basically, it tries all 255 ASCII codes for each added character until we generate a zero at the buffer. The webCL is good because it uses the GPU and runs everything in parallel, so the run time will be a lot faster because it is testing 255 elements for hex_Sha256 at same time.

It seems that WebCL will make the hashing algorithm run much faster than before, however, if we add the music function from the Music function section, we still have the same problem as below:



The reason is that the browser is running the music and WebCL at same time, and the WebCL still gets called from JavaScript. For Javascript, if some scripts take too long time to execute, it will complain about an unresponsive script. Then, I need to use a time interval in java script to solve it, so I will stop the script before the page freezes.

5.2 TimeInterval function For WebCL

The idea to combine the music and WebCL together is to run the music and simultaneously run the WebCL for one second. For WebCL part, if it finds 4 leading zeroes as we wish, or one second has passed, it will stop. After the one second later, it will resume the webCL part. Because the browser does not consider it an unresponsive script anymore, the music and browser will run smoothly without stopping. The code is below:

```
function audiPlay(a)
{
    setTimeout(function(){timedCount(a)}, 1000);
}

function audiPause()
{
    clearInterval(myHash);
}

function timedCount(a) {
    var musicCoin = vectorAdd(a);
    document.getElementById("result").innerHTML = musicCoin;
    myHash = setInterval(function(){timedCount2(a)}, 1000);
}

function timedCount2(a) {
    var musicCoin = vectorAdd(a);
    document.getElementById("result").innerHTML = musicCoin;
}
```

Figure 28: Use TimeInterval for WebCL

But this creates another problem. It is okay to generate the first virtual coin, and after that, it starts generating the second one if the music is still playing, but the problem is that playing the music still eats up a lot memory. If the music is not stopped, the memory will not be released and at the same time, WebCL does need a lot of memory to generate the virtual coin, so the browser will run out of memory. Even through WebCL is using the GPU, it also needs to use Javascript to call it. In addition, every time an additional ASCII character is added, additional memory is required.

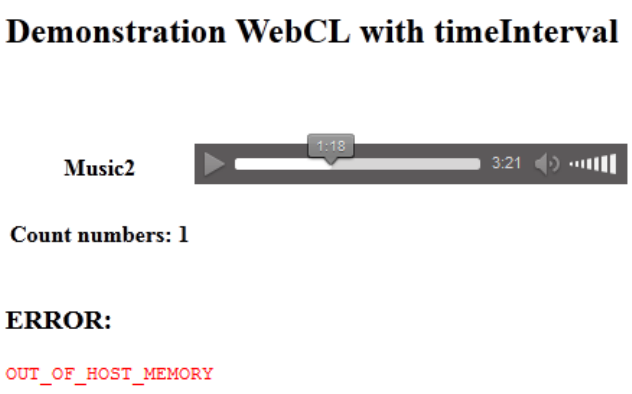


Figure 29: Out of memory issue for WebCL

5.3 Web Worker Combine the music function and Hash function with Web Worker

Because the WebCL has problems when it is called from JavaScript, I did some research about how to separate the music function and hash function. HTML5 has new technology call Web Workers, which allows you to spawn new thread, providing true asynchrony. The new worker can run in the background while the main thread processes UI events. Even if the worker thread is busy processing a heavy amount of data, it will not effect the background workers running. Worker threads can pass the message in parallel, which is ideal for this project.

Web Worker runs in an isolated thread. As result, the code it executes needs to be contained in a separate file. I created a new worker object in my main page, and put my hash function and Sha256 method in the hash_workers2.js file. Script should be invoked like this:

w = new Worker("hash_workers2.js").

After a worker is created, I call the worker and pass the input string to hash_worker2.js to w.postMessage (name). Name is input string from this main file. It will keep returning the result:

w.onmessage = function(event) {var hashString = ""+event.data;} .

It will continue finding inputs that generate 4 leading zeros in the hash value until it is terminated by **w. terminate();** It will not bother the UI music player so everything runs smoothly. It actually runs faster than WebCL even when I was using the old Javascript Sha256 method to generate. The WebCL version is supposed to be faster because it uses

the GPU, but the reason is because I use time interval, which stops every second and starts again. Web Worker can run without stopping. The best way to do this project is put WebCL inside the Web Worker, but, unfortunately, the Web Worker and WebCL are new technologies, so Web Worker cannot support WebCL.

5.4 Verification tool issue

After a virtual coin is created, it saves it to a data base. I created a tool for a user to input their virtual coin string to the function I created to test whether it really has 4 leading zeroes. I found I cannot do that because some of ASCII characters are unreadable by the web browser. So when I paste it, it will show up as below and the result will not have four leading zeroes.

Figure 30: Verify tool issue

I compared the result between my hash256 hash function with the php library sha256 function. In the screenshot below, it still prints out the 4 leading zeroed.

```
print ($hashValue ."<br>");
$hash_name = hash('sha256', $hashValue);
    echo $hash_name;
    echo "<br>";
```

It will show that it has 4 leading zero as below:

```
Tim453b006675b48441d45e52028a435507e76297f089ff389b1d9c59b662b3c5f12014-11-29 15:50:0012
0000d975bb7c055febe0f5c88edf50b5f7b4ed35334fbe582d13f4bddd84c7b1
```

Figure 31: Proof of my code is working

I have also saved them into the database, but I found out the database also does not support some of the ASCII characters because I made a function below. After querying to the database, the value does not match what was stored.

Artist Name	Coin	
Tim	Timfce3e4e4904e3a455fa6c95ed5aace6b00e926a4e53c65042bfdf7458f5660c02014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a01c|	f263dfa
Tim	Timfce3e4e4904e3a455fa6c95ed5aace6b00e926a4e53c65042bfdf7458f5660c02014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a02h	206fbe
Tim	Timfce3e4e4904e3a455fa6c95ed5aace6b00e926a4e53c65042bfdf7458f5660c02014-12-03 04:00:5712ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a03(	c4080a

Figure 32: Store value

The only way to fix this is to use base64 which was taken from this website[17]:

> Base64 is a generic term for a number of similar encoding schemes that encode binary data by treating it numerically and translating it into a base 64 epresentation. The Base64 term originates from a specific MIME content transfer encoding.

Once it transfers the encoded binary data, it will support both web browsers and the database. I use base64 function in JavaScript and PHP in my project.

6 Conclusion

My hash-based music streaming system allows music artists to upload their songs to my website. The artist's name, content of their music, timestamp, and music artist's IP address are used as a seed to the hash function. The "Play music" function has the ability to play songs while running the hash function to generate virtual coins in parallel, which can determine the amount earned by an artist based on how long the users of the website listen to their music. Artists who upload their work on this system will be rewarded based on the number of virtual coins earned. At the end of each month, each artist's earnings in dollars will be calculated by taking the number of virtual coins they earned that month, dividing that by the number of the total number of virtual coins earned that month and then multiplying that by the total revenue. In addition, a ranking page is available for people to see who the most popular artist in the website is, based on who has most virtual coins. The most popular artists' music should be better rewarded by my system because people listened to them for a long time without stopping. Other features I implemented include verification tool, which lets anyone check the virtual coins an artist has earned and result of string to hexString after using the SHA256 function.

Currently, there are numerous music websites like Spotify, Xbox Music, Amazon Cloud, etc., where music artists can upload their songs and be paid when people listen to their music. They all use a similar system called the royalty system.

The royalty system has flaws, despite the fact its widespread use for a long time. The main flaw is its unfairness in that:

1. Royalty may decrease when a web site becomes more popular.

2. Royalty is not necessarily dependent on the length of time a song is listened to but, to some extent, depends on the frequency a song is sampled.

The MP3-based payment system, on the other hand, gives virtual coins to the artists while a song is being played. The longer a song is played, the more virtual coins are paid out to the artist of the song. This system addresses the problems the royalty system has.

A possible future enhancement for the MP3-based payment system is adding support to WebCL for Web Worker, which should increase the mining speed. Another is to increase the number of leading zeroes because everyone will be mining faster. One problem to adding WebCL is that users who have better graphics card in their computer will generate virtual coins at a faster rate and will be unfair to the artists because everyone has a different computer. One issue I have experienced so far is that the browser is unable to display some ASCII characters, but I believe that in the future, browsers will be enhanced to have the ability to support all ASCII characters so that using copy and paste for my verify coin tool can show that all the virtual coins have the exact number of leading zeroes as expected.

7 Reference:

[1] Gupta Shalen. (2014, November) Bitcoin Buzz bypasses Shopper. Retrieved December 10, 2014, from http://fortune.com/2014/11/28/bitcoin-buzz-bypasses-shoppers/

[2] On-demand: 11 subscription music streaming services compared Retrieved from (November 24, 2014) http://thenextweb.com/apps/2014/07/21/11-music-subscription-services/13/

[3] The Streaming Price Bible – Spotify (November 12, 2014). , YouTube and What 1 Million Plays Means to You! Retrieved from (November 24, 2014)http://thetrichordist.com/2014/11/12/the-streaming-price-bible-spotify-youtube-and-what-1-million-plays-means-to-you/

[4] About ASCAP, Retrieved from http://www.ascap.com/about/(December 8, 2014)

[5] The Payola scandal heats up(Feb 11, 1960) , Retrieved from (December 5, 2014). http://www.history.com/this-day-in-history/the-payola-scandal-heats-up

[6] How is Spotify contributing to the music business? Retrieved from http://www.spotifyartists.com/spotify-explained/ (December 7, 2014).

[7] The Streaming Price Bible(November 12, 2014) – Spotify, YouTube and What 1 Million Plays Means to You! Retrieved from (December 12, 2014). http://thetrichordist.com/2014/11/12/the-streaming-price-bible-spotify-youtube-and-what-1-million-plays-means-to-you/.

[8] Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System. Bitcoin, page 2-3 Retrieved from **https://bitcoin.org/bitcoin.pdf** (2009)

[9] Mining Hardware comparison. Retrieved from (March , 8, 2014) https://en.bitcoin.it/wiki/Mining_hardware_comparison

[10] Videocard Benchmarks Retrived from (March , 10, 2014) http://www.videocardbenchmark.net/video_lookup.php?gpu=GeForce+GTX+670

[11] GeForce GTX 660 Ti Review: Nvidia's Trickle-Down Keplernomics. Retrieved from (March , 10, 2014) http://www.tomshardware.com/reviews/geforce-gtx-660-ti-benchmark-review,3279-12.html

[12] Amazon video card price, Retrieved from (February , 30, 2014) http://www.amazon.com/s/ref=nb_sb_noss?url=search-alias%3Daps&field-keywords=graphic+card

[13] JavaScript MD5 Retrived from (April, 30, 2014) http://pajhome.org.uk/crypt/md5/

[14] SHA-2 Retrieved from (February, 23, 2014) http://en.wikipedia.org/wiki/SHA-2

[15] SHA-256 Cryptographic Hash Algorithm. Retrieved from (May, 9, 2014) http://www.movable-type.co.uk/scripts/sha256.html

[16] WebCL Heterogeneous parallel computing in HTML5. web browsers Retrieved from (September, 9, 2014) https://www.khronos.org/webcl/

[17] BASE64 Decode and Encode, Retrieved from (December, 9, 2014) https://www.base64decode.org/

[18] Aaftab Munshi (November 14, 2014)The OpenCL Specification, Khronos Group, section 6.9, page 233-235. Retrieved from (December, 10, 2014) https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf

[19] Bruce Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C. , 436-440. Canada. John Wiley & Sons, Inc 1996

[20] National Institute of Standards and Technology. Descriptions of SHA-256, SHA-384, and SHA-512, Computer Security Division Computer Security Resource Center., page 4- 15, Retrieved from (December, 15, 2014) http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf

[21] Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, Dana Schaa. Heterogeneous Computing with OpenCL, 2nd Edition. Chapter 1- 7. Morgan Kaufmann. December 31, 2012www