Christopher LaJon Morgan
CS465: Kent Seamsons
Project #2 Report

Summery of Experiment

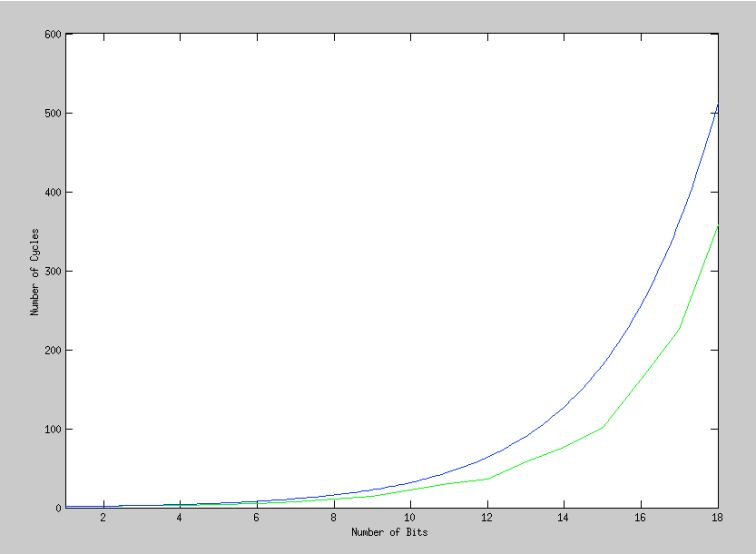
This experiment was intended to test the theoretical cost of two classic hash attacks: collision and pre-image attacks. This report shows that the theoretical runtime of $2^{n/2}$ for a collision attack and 2^n for pre-image attacks was observed, where n is the number of bits in the hash and runtime is measured in hashes generated.

These attacks were performed using hashes of bit lengths ranging from one bit to eighteen bits. Once a bit length was chosen, fifty attacks were run for each bit length. Then the results for the fifty tests were averaged together to give an average run time for each bit length selected.

Experiment Results

Please note that the scales of these graphs are not identical.

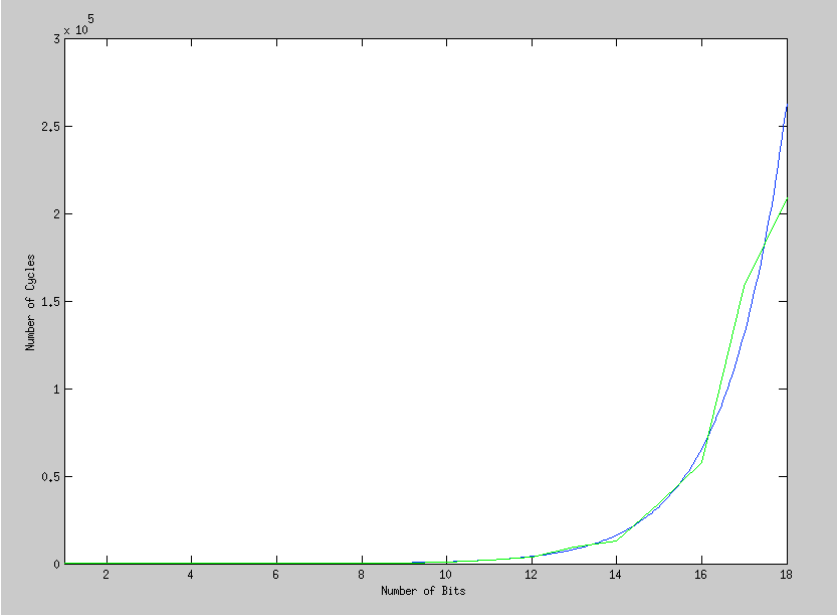
Collision Attack



Blue == Theoretical Runtime
Green == Observed Runtime

The above figure summarizes the collision attack results. The blue line represents the theoretical result of $2^{n/2}$, while the green line is the observed average runtime over 50 experiments. This graph shows that my experiment of implementing a hash collision attack runs in a similar amount of time as the collision attack theoretical runtime.

Pre-Image Attack



Blue == Theoretical Runtime
Green == Observed Runtime

The above figure summarizes the pre-image attack results. The blue line represents the theoretical result of 2^n, while the green line is the observed average runtime over 50 experiments. This graph shows that my experiment implementing a hash pre-image attack runs in a similar amount of time as the pre-image attack theoretical runtime. There is a weird blip in the data around the 16 to 18 bit range, but this could be do to several implementation factors: one, truncating the hash bits, two, generating two hashes per guess round, and three, the specific hash chosen to resolve a fabricated message to. However, the results still show that the implementation follows its theoretical runtime.

Raw Data

Number of Bits in Hash	1	2	3	4	5	6	7	8	9
Collision Attack 50 run Average	1	2	2	3	4	6	8	11	14
Pre-Image Attack 50 run Average	2	4	8	15	31	73	132	292	444
Number of Bits in Hash	10	11	12	13	14	15	16	17	18
Collision Attack 50 run Average	22	31	36	58	76	102	162	226	358
Pre-Image Attack 50 run Average	1097	1963	3984	9673	13027	34744	57734	158076	208757

Source Code

The following source code was compiled with the provided sha1.h and sha1.cpp files.

```
//============================================================================
// Name        : HashCrack.cpp
// Author      : Christopher Morgan
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
#include <cstdlib>
#include <vector>

#include "sha1.h"
using namespace std;

/*****************************************************************************
 * From two hashs return if the first numBits are equal.
 *****************************************************************************/
bool hashsAreEqualToNumBits(const unsigned char sha1[20],
                            const unsigned char sha2[20],
                            unsigned int numBits)
{
        unsigned int byte      = 0;
        unsigned char checkBit = 0x80;
        while (numBits)
        {
                unsigned char ans1 = sha1[byte] & checkBit;
                unsigned char ans2 = sha2[byte] & checkBit;

                if (ans1 != ans2) return false;

                checkBit = checkBit >> 1;
                if (checkBit == 0)
                {
                        byte++;
                        checkBit = 0x80;
                }
                numBits--;
        }
```

```
	return true;
}

/*****************************************************************************
 * KLUDGE: Just used to save a hash for future look up.
 *****************************************************************************/
class SavedHash
{
public:
	//Functions
	SavedHash(unsigned char set[20], unsigned int numBits)
	{
		for (int i = 0; i < 5; i++)
			hash[i] = set[i];

		this->numBits = numBits;
	}

	//Variables
	unsigned char hash[5];
	unsigned int numBits;
};


/*****************************************************************************
 * Generates a shaw hash from data.
 *****************************************************************************/
void genShaw(unsigned char* data, unsigned int size, unsigned char sha[20])
{
	CSHA1 hasher;

	hasher.Reset();
	hasher.Update(data, size);
	hasher.Final();

	hasher.GetHash(sha);
	return;
}

/*****************************************************************************
 * From two hashs return if the first numBits are equal.
 *****************************************************************************/
bool hashsAreEqualToNumBits(unsigned char sha1[20],
                            unsigned char sha2[20],
                            unsigned int numBits)
```

```
{
	unsigned int byte      = 0;
	unsigned char checkBit = 0x80;
	while (numBits)
	{
		unsigned char ans1 = sha1[byte] & checkBit;
		unsigned char ans2 = sha2[byte] & checkBit;

		if (ans1 != ans2) return false;

		checkBit = checkBit >> 1;
		if (checkBit == 0)
		{
			byte++;
			checkBit = 0x80;
		}
		numBits--;
	}
	return true;
}

/*****************************************************************************
 * Fill data[] with random data.
 *****************************************************************************/
void genRandData(unsigned char data[10])
{
	for (int i = 0; i < 10; i++)
		data[i] = rand();
	return;
}

/*****************************************************************************
 * KLUDGE : Search -- Find test in saved.
 *****************************************************************************/
bool contains(vector<SavedHash>& saved, SavedHash &test,unsigned int numBitsToTest)
{
	for (unsigned int i = 0; i < saved.size(); i++)
		if (hashsAreEqualToNumBits(saved[i].hash, test.hash,numBitsToTest))
			return true;
	return false;
}

/*****************************************************************************
 * Generate hashes on random data arrays till identical hash is generated.
 *****************************************************************************/
```

```
unsigned int genCollision(unsigned int numBitsToTest)
{
	vector<SavedHash> saved;

	unsigned int timesTillCollision = 0;

	unsigned char data[10];
	unsigned char data2[10];

	unsigned char saw[20];
	unsigned char saw2[20];

	bool collision = false;

	do
	{
		genRandData(data);
		genRandData(data2);

		genShaw(data, 10, saw);
		genShaw(data2, 10, saw2);

		collision = hashsAreEqualToNumBits(saw, saw2, numBitsToTest);
		timesTillCollision += 2;

		SavedHash one(saw, numBitsToTest);
		SavedHash two(saw2, numBitsToTest);

		//Kludge
		collision = collision ||
					contains(saved, one, numBitsToTest) ||
					contains(saved, two, numBitsToTest);

		saved.push_back(one);
		saved.push_back(two);

	}while(!collision);

	return timesTillCollision;
}

/*****************************************************************************
 * Run collision attack by performing experiment with n bits and with
 * each bit selection r times.
 * Print out the average of r runs on n bits.
```

```
 ******************************************************************************/
void collisionAttack()
{
	cout << "CollisionAttack: \n";
	//run experiment r-times with n-bits
	for (unsigned int n = 1; n <= 18; n++)
	{
		double total = 0;
		for (int r = 1; r <= 50; r++)
		{
			total += genCollision(n);
		}
		cout << total / 50.0 << ", ";
	}
	cout << endl;
	return;
}

/*******************************************************************************
 * Generate a saw on random data until it matches saw.
 ******************************************************************************/
unsigned int genImageAttack(unsigned int numBitsToTest)
{
	unsigned int timesTillCollision = 0;

	unsigned char data[] = {0xfa, 0xbc, 0xaf, 0x11, 0x22, 0x55, 0xff, 0xaa, 0xbb, 0xbb};
	unsigned char saw[20];
	genShaw(data, 10, saw);

	unsigned char data2[10];
	unsigned char saw2[20];

	bool collision = false;

	do
	{
		genRandData(data2);
		genShaw(data2, 10, saw2);

		collision = hashsAreEqualToNumBits(saw, saw2, numBitsToTest);
		timesTillCollision++;

	}while(!collision);

	return timesTillCollision;
```

```
}

/*****************************************************************************
 * Run preImage attack by performing experiment with n bits and with
 * each bit selection r times.
 * Print out the average of r runs on n bits.
 *****************************************************************************/
void preimageAttack()
{
	cout << "PreImageAttack: \n";
	//run experiment m-times with n-bits
	for (int n = 1; n <= 18; n++)
	{
		double total = 0;
		for (int r = 1; r <= 50; r++)
		{
			total += genImageAttack(n);
		}
		cout << total / 50.0 << ", ";
	}
	cout << endl;
	return;
}

/*****************************************************************************
 *****************************************************************************/
int main()
{
	collisionAttack();
	preimageAttack();

	return 0;
}
```