Finding Partial Hash Collisions by Brute Force Parallel Programming

Vincent Chiriaco  
Dept. of Computer Science  
University of North Alabama  
Florence, AL 35632, U.S.A.

Rebecca Thayil  
Dept. of Physics  
Bryn Mawr College  
Bryn Mawr, PA 19010, U.S.A.

Aubrey Franzen  
Dept. of Computer Science  
Northern Kentucky University  
Highland Heights, KY 41099, U.S.A.

Xiaowen Zhang  
Dept. of Computer Science  
College of Staten Island / CUNY  
Staten Island, NY 10314, U.S.A.

Abstract—A hash function maps an arbitrary length of (longer) message into a fixed length of shorter string, called message digest. Inevitably there will be a lot of different messages being hashed to the same or similar digest. We call this collision or partial collision. By utilizing multiple processors from the CUNY High Performance Computing Center’s facility, we locate partial collisions for MD5 and SHA-1 by brute force parallel programming in C with MPI library. The brute force method of finding a second preimage collision entails systematically computing all of the permutations, digests, and Hamming distances of the target preimage. We explore varying size target strings and the number of processors allocation and examine the effect these variables have on finding partial collisions. The results show that for the same message space the search time for the partial collisions is roughly halved for each doubling of the number of processors; and the longer the message is the better partial collisions are produced.

Keywords—Partial hash collision; brute force; MD5; SHA-1; high performance computing; parallel programming, MPI.

I. INTRODUCTION

Because of its various critical applications, a cryptographic hash function is called cryptographer’s Swiss Army knife. For example, in digital signature, a hash function is required to reduce a long document to be signed to a short, fixed-length message digest. Then we use the private key to sign (encrypt) the message digest. Hash functions are used to design secret sharing schemes [2], stronger hash functions [1]. This research entails exploring the brute force method of finding partial collisions as second preimages of target strings with the cryptographic hash functions MD5 and SHA-1. Both hash functions pre-process the arbitrary length of message into multiple blocks of 512-bits, and then use Merkle-Damgard construction iteratively compress each block. MD5 generates a 128-bit digest while SHA-1 generates a 160-bit digest.

Other than brute force, there is no known better algorithms for finding second preimage collisions for hash functions [4]. A collision arises when two different inputs share the same digest; whereas, a partial or almost-collision is when the digests are similar. This similarity is measured by Hamming distance. As the Hamming distance approaches zero, the partial collision becomes closer to a collision. This research focuses on searching for the best five partial collisions with the target strings when using MD5 and SHA-1 hash functions.

A. Cryptographic Hash Functions

A hash function [5, 6] compresses a message of arbitrary length into a short, fixed-length output string. This fixed-length string is called a message digest, hash, hash value, or fingerprint. There are three basic properties for a hash function: preimage, second preimage, and collision resistances. Preimage resistance states that given hash $y = h(x)$ for input $x$, it is infeasible to find the preimage $x$. This property also makes a hash function a one-way function. Second preimage resistance states that given hash $y = h(x)$ and input $x$ it is infeasible to find a second input $x' \neq x$ such that $h(x') = y$. Collision resistance states that it is infeasible to find two distinct inputs $x$ and $x'$, such that $h(x) = h(x')$. Collision resistance is the strongest property. A hash function that is collision resistant is also second preimage resistant; and second preimage resistance implies preimage resistance [4].

B. Research Goals and Environment

We implement a brute force collision machine that uses multiple processors and parallel programming to test for hashing speed and testing the second preimage resistance property by finding best partial collisions in which the hash of the second message has the smallest Hamming distance with the hash of the target message. We build two versions of collision machines to test time effectiveness for using multiple processors. The version 1 takes in a “target” character string, hashes it, and then generates each permutation of the target string before hashing it and comparing it to the target hash for the best partial collisions. The version 2 takes in a “target” hex string (a bit string), hashes it, and hashes every increment of the starting hex string until entire message space is traversed, comparing the hash to the target hash each time. Both versions work for MD5 and SHA-1.
Using Message Passing Interface (MPI) library in C language, we parallelize our program and generalized it so that version 1 could run with any number of processors and version 2 could run with any $2^n$ number of processors. We ran the programs with different sizes of target strings with different number of processors. Then we collect the times that the programs run for measuring the performance.

The programs were first run locally on a PC using OpenMPI library in Cygwin64 environment. Once syntax and logical errors were fixed and accuracy was achieved, the code was uploaded and ran on CUNY High Performance Computing Center’s cluster called Penzias. Penzias is a 1152 core cluster with each core having 4 gigabytes of memory.

II. IMPLEMENTATION AND RESULTS

A. Implementation Algorithms

The main.c program invokes functions from the source code of MD5 [7] and SHA-1 [3] to perform hash all string permutations and search for top 5 partial colliding strings with the target. The pseudocode structure of main.c is as follows:

- Declare and initialize all necessary variables
- MPI_Init, MPI_Comm_rank, MPI_Comm_size
- Set initial start string for every processor
- While(next string permutation exists)
  - Hash the next string
  - If (Hamming dist < top 5 string’s)
    - Insert it to top 5 string
- MPI_Gather: root collects top 5 strings from all processors
- Root sorts all collected strings by their distances
- Output final top 5 strings and their hashes and distances
- MPI_Finalize

In addition we define a Tuple structure that contains a string, its hash, and Hamming distance to the target hash, and other utility functions.

B. Results

We used larger target strings (up to 13 character and 10 bytes hex) and ran them with 16 or more processors to see what partial collisions were found. Inputting larger strings leads to a greater chance of finding better partial collisions.

Due to the space limit, we only display hex strings performances for MD5 and SHA-1 in Figures 1 and 2.

Two collision machines were created to find second preimage partial collisions of MD5 and SHA-1 using the brute force method. Each program has functionality to assess the processing speed of each processor. The smallest Hamming distances between found strings of 7, 8, 9 characters and the target string "abcedfg" "abcedfgh" and "abedefgh" are around 40 for 7, and 30 for 8 and 9 in MD5; the message spaces are 7!, 8! and 9!, respectively. For hex bit-strings of 4 bytes, the smallest Hamming distances from the target bit-string $FFFFFFFE$’s hash is around 40 for SHA-1. The trend of execution time is a gradual decrease by about half for each increase in the number of processors.

II. IMPLEMENTATION AND RESULTS

A. Implementation Algorithms

The main.c program invokes functions from the source code of MD5 [7] and SHA-1 [3] to perform hash all string permutations and search for top 5 partial colliding strings with the target. The pseudocode structure of main.c is as follows:

- Declare and initialize all necessary variables
- MPI_Init, MPI_Comm_rank, MPI_Comm_size
- Set initial start string for every processor
- While(next string permutation exists)
  - Hash the next string
  - If (Hamming dist < top 5 string’s)
    - Insert it to top 5 string
- MPI_Gather: root collects top 5 strings from all processors
- Root sorts all collected strings by their distances
- Output final top 5 strings and their hashes and distances
- MPI_Finalize

In addition we define a Tuple structure that contains a string, its hash, and Hamming distance to the target hash, and other utility functions.

B. Results

We used larger target strings (up to 13 character and 10 bytes hex) and ran them with 16 or more processors to see what partial collisions were found. Inputting larger strings leads to a greater chance of finding better partial collisions.

Due to the space limit, we only display hex strings performances for MD5 and SHA-1 in Figures 1 and 2.

Two collision machines were created to find second preimage partial collisions of MD5 and SHA-1 using the brute force method. Each program has functionality to assess the processing speed of each processor. The smallest Hamming distances between found strings of 7, 8, 9 characters and the target string "abcedfg" "abcedfgh" and "abedefgh" are around 40 for 7, and 30 for 8 and 9 in MD5; the message spaces are 7!, 8! and 9!, respectively. For hex bit-strings of 4 bytes, the smallest Hamming distances from the target bit-string $FFFFFFFE$’s hash is around 40 for SHA-1. The trend of execution time is a gradual decrease by about half for each increase in the number of processors.

ACKNOWLEDGMENT

College of Staten Island/CUNY REU Site sponsored by NSF/DoD Grant 1359266; CUNY HPCC operated by CSI, funded, in part, by grants from New York State and City, and NSF Grants CNS-0958379, CNS-0855217 and ACI 1126113.

REFERENCES