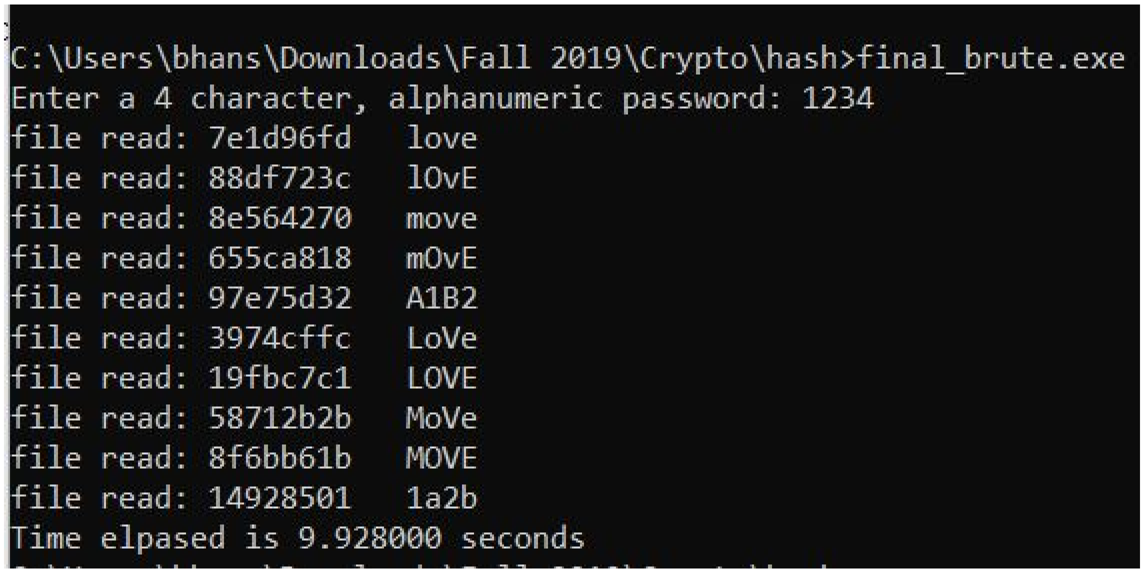
Arizona State University

CSE 539 Applied Cryptography Project-3

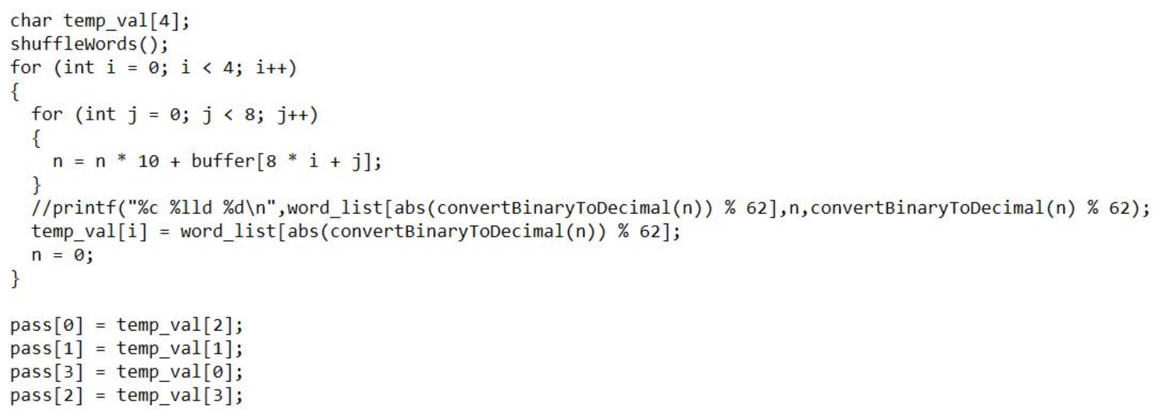
CHIRAG BHANSALI (1215185491)
SHIVANK TIWARI (1217351382)

For the brute force, we first precomputed all the combinations of 4 character long strings/ passwords, which took us about 2 hours to complete.
We stored all the passwords in a text file and then in the brute force C program we, started reading each password and computed the hash of it and compare it with the given 10 hashes, And after we have found out all the 10hashes we stop the execution process, which takes 9.928 seconds.
The initial input for the brute force program is just for running the application since we modified the given code.



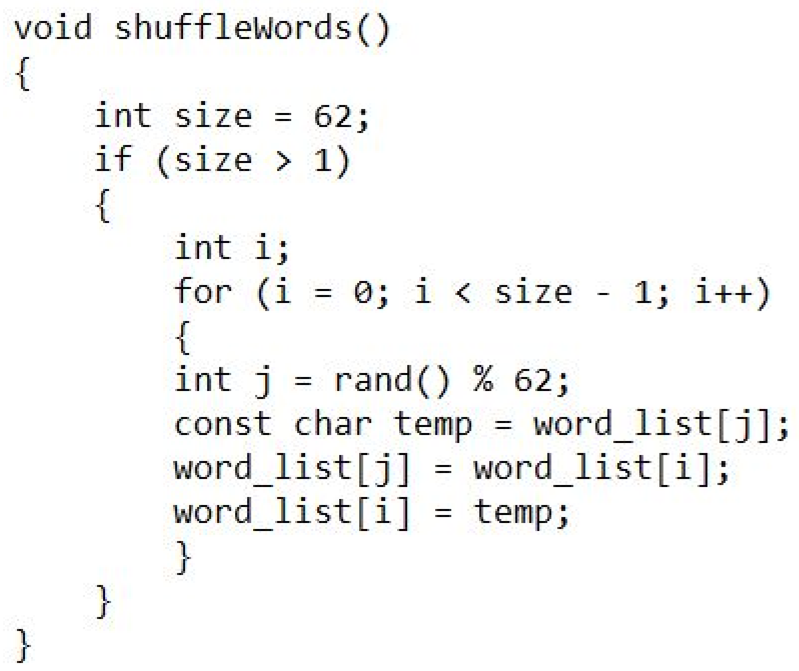
```
C:\Users\bhans\Downloads\Fall 2019\Crypto\hash>final_brute.exe
Enter a 4 character, alphanumeric password: 1234
file read: 7e1d96fd   love
file read: 88df723c   lOvE
file read: 8e564270   move
file read: 655ca818   mOvE
file read: 97e75d32   A1B2
file read: 3974cffc   LoVe
file read: 19fbc7c1   LOVE
file read: 58712b2b   MoVe
file read: 8f6bb61b   MOVE
file read: 14928501   1a2b
Time elpased is 9.928000 seconds
```

For the rainbow table, we took the dimensions as 62 * 238328.
We took 62 different passwords and ran the chaining process for 238328 times for each password, and computed the final hashes, we stored both the final hashes and the 62 passwords in a text file.
For the reduction function, we converted the 32-bit long binary into 4 ASCII characters using the below logic.



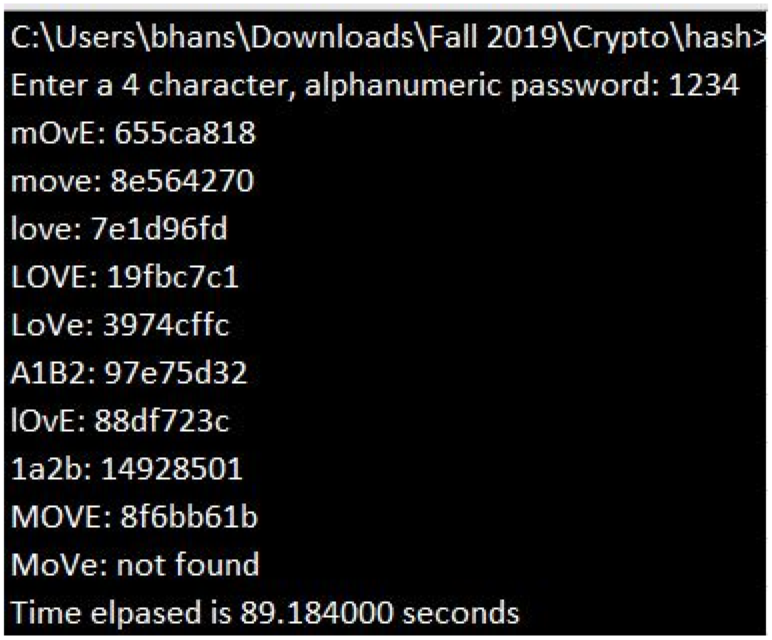
```
char temp_val[4];
shuffleWords();
for (int i = 0; i < 4; i++)
{
  for (int j = 0; j < 8; j++)
  {
    n = n * 10 + buffer[8 * i + j];
  }
  //printf("%c %lld %d\n",word_list[abs(convertBinaryToDecimal(n)) % 62],n,convertBinaryToDecimal(n) % 62);
  temp_val[i] = word_list[abs(convertBinaryToDecimal(n)) % 62];
  n = 0;
}

pass[0] = temp_val[2];
pass[1] = temp_val[1];
pass[3] = temp_val[0];
pass[2] = temp_val[3];
```

We are converting each 8bit long binary to its Integer value and then taking a modulus 62 to map it with the word list containing the 62 characters (A-Z, a-z, and 0-9)
For each iteration, we are shuffling the word list so that we could possibly avoid collision of character, below is the logic for the same.
And for creating a new password for the next iteration, we choose to map the 0 index of pass with 2nd, 1 with 1, 3 with 0 and 2 with 3.



```
void shuffleWords()
{
    int size = 62;
    if (size > 1)
    {
        int i;
        for (i = 0; i < size - 1; i++)
        {
        int j = rand() % 62;
        const char temp = word_list[j];
        word_list[j] = word_list[i];
        word_list[i] = temp;
        }
    }
}
```

For the password finding part, we took hashes 1 by 1 and with the help of reduction function, found the hash value which is there in the text file and the position in the file, then using the position, we computed the chain until the required hash is not found, we ran the chaining for 62 power 3 (238328) times and if no match found till then we print “not found”.



```
C:\Users\bhans\Downloads\Fall 2019\Crypto\hash>
Enter a 4 character, alphanumeric password: 1234
mOvE: 655ca818
move: 8e564270
love: 7e1d96fd
LOVE: 19fbc7c1
LoVe: 3974cffc
A1B2: 97e75d32
lOvE: 88df723c
1a2b: 14928501
MOVE: 8f6bb61b
MoVe: not found
Time elpased is 89.184000 seconds
```

Shortcoming: We found that in the rainbow table, we had to try various combinations before selecting the final combos for reduction function and in many cases, we failed to find 1 or 2 passwords, thus making it completely random/ probabilistic, depending on the reduction and shuffling function so created in the program.

Brute force was able to crack the password but the precomputation took time, but if a professional hacker is using the same, its a one-time effort, for generating each combo.