Ira A. Fulton Schools of Engineering
Arizona State University

CSE 539: Applied Cryptography

Hash Project

Purpose

In this project, you will employ cryptographic libraries to hash strings using the MD5 library. You will gain the experience necessary to use any of the modern hash functions in C#'s hashing library. With an MD5 hashing function, you will learn how to convert a string to bytes and salt that string for more secure hashing. You will also learn how to find collisions in a hash function using a birthday attack.

Objectives

Learners will be able to:

- Justify salting password hashes.
- Employ the Birthday Paradox to find a collision in a hash function.

Technology Requirements

This project requires the knowledge of the following tools and technologies:

- .NET Core 8.0
- Basic Understanding of C#
- Zybooks IDE for editing your code

Directions

Accessing ZyLabs

You will complete and submit your work through zyBooks's zyLabs. Follow the directions to correctly access the provided workspace:

1. Go to the Canvas submission space, "**Submission: Hash Project**"
2. Click the "**Load Submission…in new window**" button.

3. When ready, review the provided code and develop your work where instructed.

Project Description

The use of hashes typically comes with the assumption that they cannot be reversed. As such, they are the perfect fit for storing passwords. Passwords can be verified by hashing the input and comparing them to the stored hash. As hashing is deterministic, the same hash will always be generated. Unlike encryption, hashing is not designed to be reversible. If an attacker obtains the hash of the password, they should be unable to find the original password. However, rainbow tables have been created for popular hashing functions and passwords. Thus, hashed passwords that are part of the rainbow table can be quickly looked up. Furthermore, since hashing is deterministic, two users with the same password would have the same hash. This gives additional information to an attacker.

To mitigate these issues, a salt can be added to the password. Typically, a different salt is used per user, causing users with the same password to have different hashes. Furthermore, salted passwords are much less likely to be found in a rainbow table, increasing the security of the password hashes.

In this exercise, you will employ MD5, which is an outdated hashing algorithm. The goal of this exercise is to find collisions using a birthday attack. As the logic is the same, you will be evaluated using a hashing function with a smaller output space.

First, write a program in C# to find the MD5 hash of a string. Documentation for MD5 in C# is available here: MD5 Class (System.Security.Cryptography)

To convert a string to a byte array, use the Encoding.UTF8.GetBytes method. You can test your code using an online MD5 calculator. Strings converted using that method will produce identical hashes as an MD5 calculator.

A 1 byte salt will be passed to you as a command line argument. You should append this byte to the end of your byte array before hashing.

Example: suppose you are hashing the string “Hello World!” and are passed the salt C5.

> Bytes before hashing: 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 C5
>
> Bytes after hashing: E6 D9 B0 B9 D1 78 B2 40 02 89 EB EA 33 E8 B8 82

You can compare this result to hashing only “Hello World!” to see the difference:

> Bytes before hashing (only “Hello World!”): 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
>
> Bytes after hashing: ED 07 62 87 53 2E 86 36 5E 84 1E 92 BF C5 0D 8C

You will now perform a birthday attack with a function for calculating a hash with salt. As the standard version of MD5 is time-consuming to attack, you should modify the hashing result to make it easier to attack. Instead of using the full hash, only compare the first 5 bytes (in this case, “Hello World!” with the salt C5 hashes to E6 D9 B0 B9 D1).

You may choose the length of the string, but you may only use alphanumeric characters in your string [A-Z][a-z][0-9]. Your program should output two strings that hash to the same value with MD5 and the given salt. These strings should be separated by a comma.

Continuing from the earlier example, suppose that the byte C5 is passed in as the salt. Here is a sample command and output from the program:

> **Command**: dotnet run “C5”
>
> **Sample output**: AQJCMW0DGL,I95ORWB1A7

In this example, both values hash to 2B 68 3B 65 7A when salted with C5.

Note: There will be many different possible solutions. Your answers will be verified by hashing them.

Submission Directions for Project Deliverables

You are given an unlimited attempts to submit your best work. The number of attempts is given to anticipate any submission errors you may have in regards to properly submitting your best work within the deadline (e.g., accidentally submitting the wrong paper). It is not meant for you to receive multiple rounds of feedback and then one (1) final submission. Only your most recent submission will be assessed.

You must submit your Hash Project deliverable through zyBooks’s zyLabs. Carefully review submission directions outlined in this overview document in order to earn credit for your work correctly. Learners may not email or use other means to submit any assignment or project for review, including feedback, and grading.

1. Please use the **program.cs** file provided in your zyBooks workspace to complete your 'Hash Project' program, ensuring that you do not change the class name.
    a. Execute your code by running the below command in the terminal located at the bottom of the IDE.
        i. dotnet run “C5”
2. When you are ready to submit your completed work, click on “**Submit for grading**” located on the bottom left.
3. You will know you have completed the project when feedback appears below the workspace.

4. If needed: to resubmit the project in zyLabs
    a. Edit your work in the provided workspace.
    b. Click “**Submit for grading**” again at the bottom of the screen.

The Hash Project includes one (1) deliverable:

- **Project Code:** Complete and submit your program.cs file through zyLabs.

Your submission will be reviewed by the course team and then, after the due date has passed, your score will be populated from zyBooks into your course grade.

Evaluation

The auto-grader will execute your code by passing a byte (salt) through the command line. The grader will compare the two hash values generated by your code with the expected hash values. If these match, you will receive **100 points**. Otherwise, you will receive **0 points** with corresponding error messages.