**Activity 3 Guide: Password Hasing and Cracking**

Table of Contents

[Part 1 - How Hashing Works 3](#_Toc177826169)

[About Hashing 3](#_Toc177826170)

[One Way Street 3](#_Toc177826171)

[Hashing Demonstrated 3](#_Toc177826172)

[Use Case #1 Passwords 6](#_Toc177826173)

[Why use Password Hashing? 7](#_Toc177826174)

[About Rainbow Tables 7](#_Toc177826175)

[User Case #2 Data Verification 8](#_Toc177826176)

[1. File Downloads 8](#_Toc177826177)

[2. Data Transmission 10](#_Toc177826178)

[3. Data Storage and Backup Verification 10](#_Toc177826179)

[5. Version Control in Code Repositories 11](#_Toc177826180)

[Example of a Commit Hash 12](#_Toc177826181)

[6. Digital Signatures and Certificates 12](#_Toc177826182)

[7. Blockchain and Cryptocurrencies 13](#_Toc177826183)

[Observations about Hashing 13](#_Toc177826184)

[Hashing Algorithms Compared 13](#_Toc177826185)

[What are hash “collisions”? 14](#_Toc177826186)

[A collision occurs when two different input string produce the exact hash result. Collisions are theoretically possible since the domain (input) of a hashing function is essentially infinite and the range (output) of a hashing function is finite. However, the range of a hashing function is very large, and collisions are rare. 14](#_Toc177826187)

[Birthday Function Collision Analogy 14](#_Toc177826188)

[Probability of Birthday Collisions 14](#_Toc177826189)

[Table 1 The probability of a collision grows as the size of a group grows, seen in Figure 5. 15](#_Toc177826190)

[**Explanation of Birthday Problem Collision Probabilities** 15](#_Toc177826191)

[**Why Hash Collisions Matter** 15](#_Toc177826192)

[Real-World Results 16](#_Toc177826193)

[Check for Understanding 16](#_Toc177826194)

[Deliverables 18](#_Toc177826195)

[Part 2 - Coding a Password Hashing Program in Java 18](#_Toc177826196)

[Setup a Project 18](#_Toc177826197)

[How to Install the Maven runtime (if necessary) 19](#_Toc177826198)

[Create a new Java project using the Maven archetype. 22](#_Toc177826199)

[Eclipse Instructions for Creating a Maven Project 25](#_Toc177826200)

[Simple Hashing Test 29](#_Toc177826201)

[Message Digest class in Java Varies 32](#_Toc177826202)

[ANSII Colors in Eclipse 33](#_Toc177826203)

[Hashing Algorithms Compared 33](#_Toc177826204)

[Part 3 - Password Cracking Application 34](#_Toc177826205)

[Compare Dictionary Attacks to Brute Force Attacks 34](#_Toc177826206)

[Prefix 37](#_Toc177826207)

[Combine Words 38](#_Toc177826208)

[Calculate Password Security 39](#_Toc177826209)

[What You Learned: Password Hashing and Cracking 42](#_Toc177826210)

[Check for Understanding 43](#_Toc177826211)

[Deliverables: 45](#_Toc177826212)

### ****Introduction:****

In today's digital landscape, protecting sensitive information such as passwords and ensuring the integrity of data is more crucial than ever. Password hashing is a fundamental security practice that prevents attackers from easily reversing exposed or stolen passwords into their original form. Similarly, hashing is used in numerous other areas to guarantee data integrity, from file signatures and network transmissions to version control systems like GitHub and advanced technologies like blockchain.

In this activity, you will learn the basics of password hashing, its real-world applications, and the methods hackers use to crack password hashes. You will write Java programs to explore different hashing algorithms, including MD5, SHA-256, and BCrypt, and experiment with various password cracking techniques. Additionally, you will examine how hashing is used beyond password security—such as verifying file integrity, ensuring data transmission accuracy, and safeguarding commit history in version control systems like Git.

You will also be able to explain how hashing works in blockchain technology, where it provides immutability and security to the chain of data blocks. By the end of this activity, you will not only understand how password hashing works but also appreciate how modern security measures like salting and secure hashing algorithms prevent attacks.

# Part 1 - How Hashing Works

In this part of the activity, you will read about hashing and write a response to some questions.

## About Hashing

Hashing is an algorithmic function that takes a large amount of information as an input and outputs a smaller, fixed-size value, called a hash. Think of it as a way to create a unique *fingerprint* for the data. The hashing process is commonly used for keeping passwords secure and checking data integrity.

## One Way Street

A hash is a one-way function, meaning that it is not reversable. Unlike encryption, there is no inherent information about the original value stored in the hash. The hashing function creates a fixed length result regardless of the size of the input. A single password hash will produce the same length of output as the hash value of an entire book. Theoretically, it is possible, although extremely unlikely, for two pieces of data to produce an identical output, called a collision, from a hash function. Collisions are more likely if the length of the output string is not sufficiently long.

## Hashing Demonstrated

Follow these steps to see a hashing function in action.

1. Navigate to md5hashgenerator.com
2. Enter a string and click “Generate”. You will see an example of a hashed password, shown in Figure 1.

A screenshot of a computer

Description automatically generated

Figure Cardinals password is hashed online.

Here are some other MD5 hashing results.

hashMD5(“qwerty”) = d8578edf8458ce06fbc5bb76a58c5ca4

hashMD5(“superman”) = 84d961568a65073a3bcf0eb216b2a576

hashMD5(“this is a really long password that would be a secure choice 1!@”) = 25974feaded56ce8f9d42606bda6f732

1. Use the MD5 hashing page to process some passwords or phrases that you have used in the past.
2. Copy the resulting hash from a common password and perform a Google search using the MD5 hash you produced.

A screenshot of a web page

Description automatically generated

Figure Search results for an MD5 hash value.

1. You will soon discover that common passwords have been hashed and stored in lists. These are sometimes called Dictionaries or “Rainbow Tables”. They are useful for quickly hacking stolen passwords.

A screenshot of a computer

Description automatically generated

Figure A website that has a dictionary of hash results for common passwords.

## Hashing Use Case #1 Passwords

Hashing passwords is a method used to securely store passwords in a database. Instead of saving the actual password, a hashed version of the password is stored in the database. Here's a step-by-step explanation of how it works:

Figure Process for hashing and storing a password

Figure Process for login and authentication.

1. **User Creates a Password**: When a user creates a password, the system takes the password and processes it through a hashing algorithm.
2. **Hashing the Password**: A hashing algorithm takes the password and transforms it into a fixed-length string of characters, which is the hashed password. This process is one-way, meaning it is extremely difficult to convert the hashed password back to the original password. Common hashing algorithms include SHA-256, BCrypt, and Argon2.

A database table with user accounts with clear text passwords

|  |  |  |
| --- | --- | --- |
| id | username | password |
| 1 | Andy | pass123 |
| 2 | Butch | password456 |
| 3 | Clarissa | letmein789 |

User accounts with hashed passwords

|  |  |  |
| --- | --- | --- |
| id | username | hashedpw |
| 1 | Andy | 9b8769a4a742959a2d0298c36fb70623f2dfacda8436237df08d8dfd5b37374c |
| 2 | Butch | c6ba91b90d922e159893f46c387e5dc1b3dc5c101a5a4522f03b987177a24a91 |
| 3 | Clarissa | 60dff37752c1055b32d100ab3d8d5cd0578d3dd582fb390660e183c884fac62a |

1. **Salting**: To further secure the hashed passwords, a random value called a "salt" is added to the password before hashing. This ensures that even if two users have the same password, their hashed passwords will be different. The salt is usually stored in the database alongside the hashed password.
2. **Storing the Hashed Password**: The hashed password is stored in the database instead of the plain text password.

User Accounts with Hashed Passwords and Salt

|  |  |  |  |
| --- | --- | --- | --- |
| id | username | salt\_plus\_pw\_hashed | salt |
| 1 | Andy | 86f11b15acf99ddbc48abf8a33302e2336bfd8cd8a5e2a6e74f330665871c4b5 | 387cbd469f6ef765c2bd1587b7e0b280 |
| 2 | Butch | dba9c30190e16a7da3a6428e97613d979111535a9121f66044e53e1df11a15ef | baf5bc14856c382aede355d1f7ca62ee |
| 3 | Clarissa | 8a9465b430591e4c79ce155c1c5e0c08ced56c7247805503fc7d28c4aa5c0c5e | 689453739d8688834a6887c604c051ed |

1. **Verifying the Password**:
   * When the user logs in, they enter their password.
   * The system retrieves the salt (if used) and the hashed password from the database.
   * The entered password is combined with the salt (if used) and hashed using the same algorithm.
   * The newly hashed password is compared with the stored hashed password. If they match, the password is correct, and the user is authenticated.

### Why use Password Hashing?

1. **Security**: Even if the user table database is stolen, the attacker only gets hashed passwords, which are very difficult to reverse-engineer.
2. **Unique Hashes with Salts**: Adding a salt ensures that even common passwords produce unique hashes.
3. **Prevents Rainbow Table Attacks**: Salting and using secure hashing algorithms make precomputed hash attacks (rainbow tables) ineffective.

## About Rainbow Tables

A rainbow table, sometimes called a password dictionary, is a pre-computed list of hash values for commonly used passwords. Rainbow tables can help hackers crack passwords in seconds if an insecure password is used.

A screenshot of a computer

Description automatically generated

Figure A website that provides pre-computed hashes for all possible passwords in various algorithm formats.

Commonly used passwords are weak because password dictionaries remove the need for intense computing power to find the reverse hash values. There are plenty of commonly used passwords and hashed result lists available online.

A screenshot of a computer

Description automatically generated

Figure Common passwords are notoriously easy to hack.

## Hashing User Case #2 Data Verification

Hashing ensures data integrity and security, particularly when verifying whether data has been corrupted or tampered with during transmission, storage, or retrieval. A **hash function** transforms input data (such as files or messages) into a fixed-size string of characters, often called a **hash value** or **digest**. These hashes can be used to ensure that data has not been altered, as even the slightest change to the input will result in a vastly different hash value.

### 1. ****File Downloads****

When users download files from the internet, especially large files like software packages or operating system images, hashing is often employed to verify the integrity of the downloaded data. The website or file provider will typically publish the hash value (often using algorithms like SHA-256 or MD5) alongside the file. After downloading the file, the user can calculate the hash value of the downloaded file using a hashing utility and compare it to the hash provided by the website. If the values match, the file was downloaded correctly and has not been tampered with.

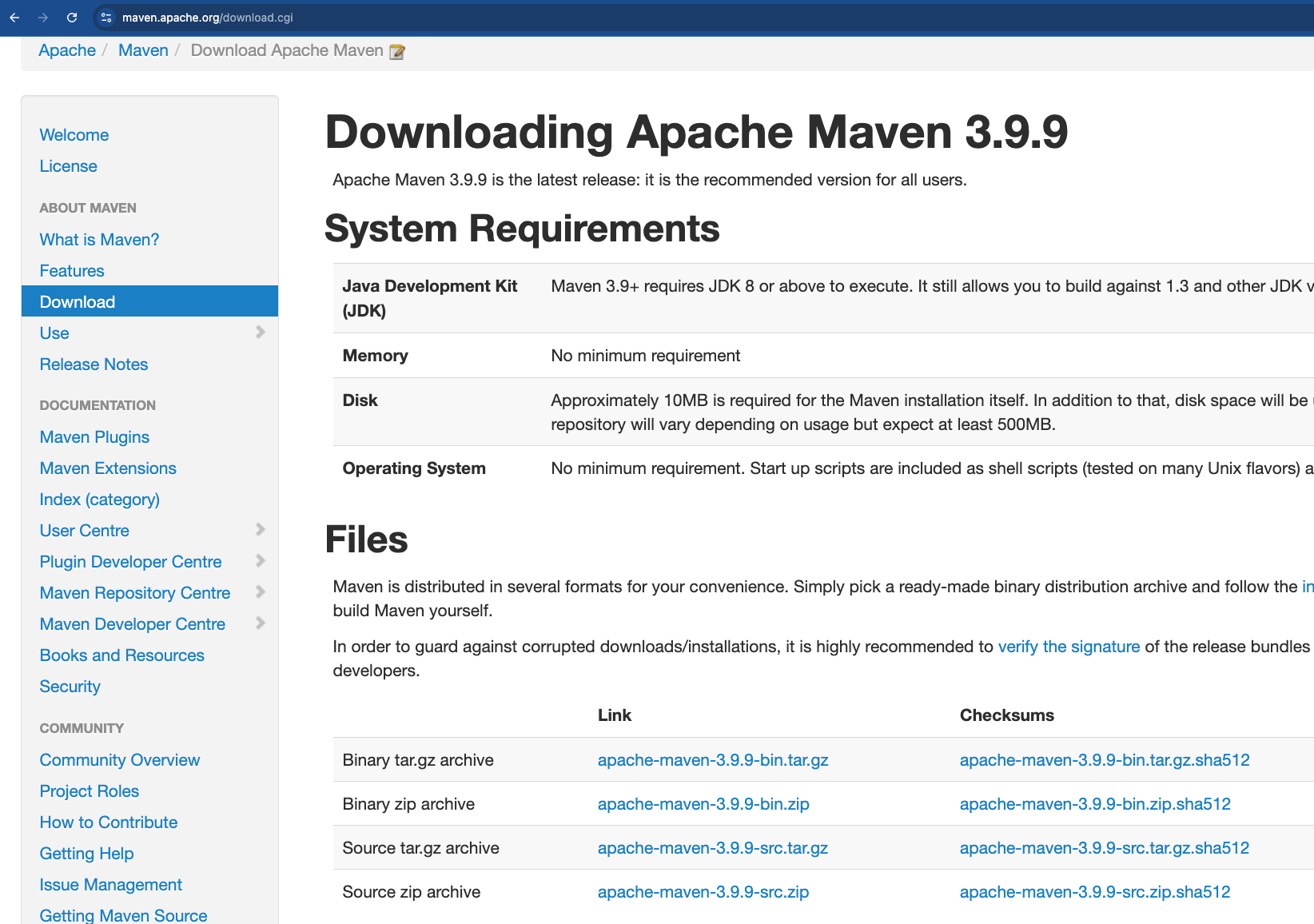


Figure Example of a software package download page thta provides Checksum hash values.

#### Example:

* Website provides a file app.zip along with its hash:

app.zip hash (SHA-256): 5d41402abc4b2a76b9719d911017c592

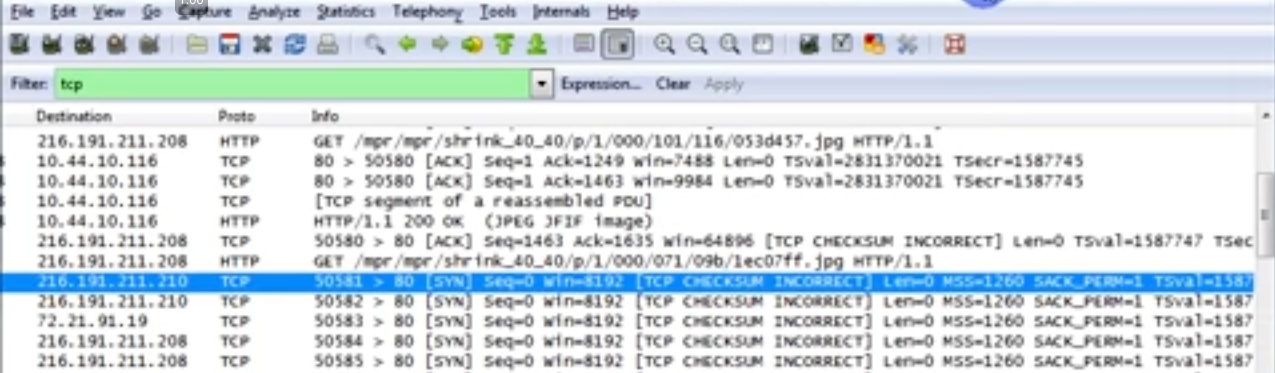
* User downloads the file and runs a hash check:

Hash of downloaded file (SHA-256): 5d41402abc4b2a76b9719d911017c592

* Since the hash values match, the file is verified as unaltered.

### 2. ****Data Transmission****

In network communication, data can be corrupted or altered as it moves between the sender and receiver. To ensure that the data received is the same as the data sent, hashing is commonly used along with checksums or message digests. The sender creates a hash of the data before transmitting it and sends both the data and the hash to the receiver. Upon receiving the data, the receiver hashes the data again and compares the result with the hash sent by the sender. If the hashes match, the data is considered intact.



Checksum error

Figure A TCP packet shows a checksum error. This is a Wireshark packet capture session. The packet will likely be discarded and retransmitted.

#### Example:

* Sender sends the message:

"Hello World" with hash value (MD5): b10a8db164e0754105b7a99be72e3fe5

* Receiver calculates the hash of the received message and gets:

b10a8db164e0754105b7a99be72e3fe5

* The hashes match, so the message was transmitted correctly.

This method is often used in **TCP/IP protocols**, where checksums verify packet integrity in network layers.

### 3. ****Data Storage and Backup Verification****

When data is backed up or stored for long periods of time, it’s important to ensure that it hasn’t been corrupted or altered. Hashing helps detect corruption by periodically hashing stored data and comparing it to the original hash created when the data was first stored. If the hashes differ, the stored data has likely become corrupted and needs to be restored from a backup.

A screenshot of a computer

Description automatically generated

Figure The MySQL CHECKSUM TABLE authors; command provides a hash value that can verify whether a table has been modified or corrupted either to internal errors or tampering.

### 5. ****Version Control in Code Repositories****

In version control systems like Git, every commit is assigned a unique hash value (often called a commit hash) based on the contents of the files in the commit. This hash ensures that every change is uniquely identified and tracked. If a single byte in a file changes, the entire commit hash changes, providing a robust mechanism for tracking changes in the codebase.

A screenshot of a computer

Description automatically generated

Figure A commit id number in a GitHub repository is a shortened form of a full hash value.

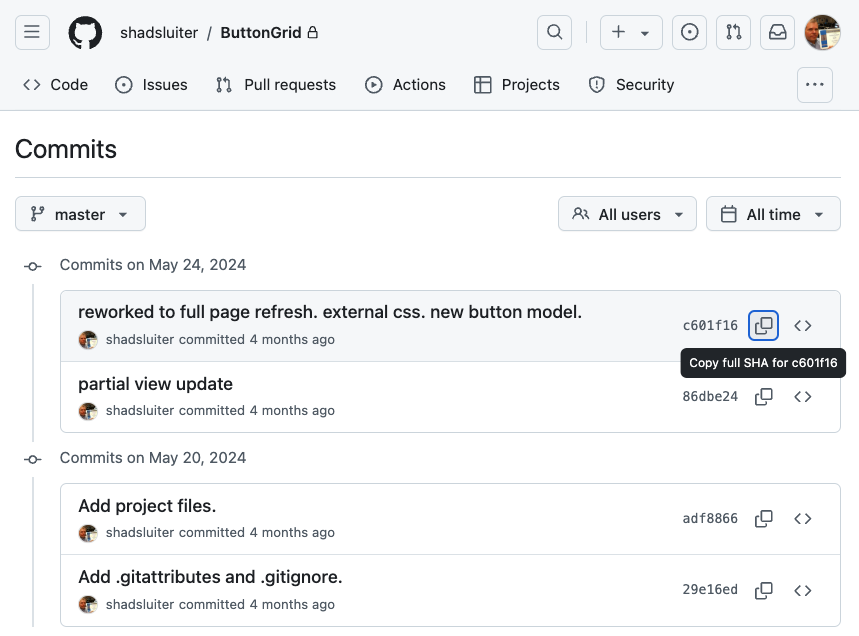


Figure The commits page of a GitHub repository allows the user to copy the full SHA for the commit number.

### Example of a Commit Hash

When you create a commit in Git, the commit hash is automatically generated. For example, after committing a change, you might see a message like this:

[main **d3e5c4f**] Add a new feature

1 file changed, 10 insertions(+)

Here, **d3e5c4f** is the abbreviated version of the commit hash, but the full commit hash looks like this:

d3e5c4f2b0932a5fdfc01977f1a4bde9058e0c7d

You can view the full commit hash with:

git log

This shows a list of commits with their corresponding hashes. The hash value ensures that every change is accurately recorded and identifiable.

### 6. ****Digital Signatures and Certificates****

Digital signatures use a combination of hashing and encryption to verify the authenticity of digital messages or documents. A digital signature is created by hashing the message and encrypting the hash with the sender’s private key. The receiver decrypts the signature using the sender's public key and compares the hash to a newly calculated hash of the message. If they match, the document’s integrity and authenticity are verified.

#### Example:

* Sender creates a digital signature by hashing the document and encrypting the hash.
* Receiver decrypts the signature and compares it with the hash they generate from the document.
* If both hashes match, the document is considered authentic and unaltered.

### 7. ****Blockchain and Cryptocurrencies****

Hashing is used in **blockchain technology**. Every block in a blockchain contains a hash of the previous block, ensuring that any modification to a block would invalidate the entire chain. This hashing chain provides security and integrity for cryptocurrencies like Bitcoin, as altering past transactions would require recalculating the hashes for all subsequent blocks, which is computationally infeasible.

A diagram of a blockchain

Description automatically generated

Figure A blockchain block contains: transaction data, a hash value and the hash of the previous value in the chain.

The next block’s hash is based on the previous block’s hash. If any data in the previous block is altered, the hash changes, invalidating the entire chain.

## Observations about Hashing

1. You should notice that each hashing result from the same algorithm is always the same length. More modern hashing algorithms produce hash outputs much longer, and more secure, than MD5.
2. You should notice that the hash result has no pattern related to the input. A single letter difference in two passwords will produce completely different output values.

## Hashing Algorithms Compared

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Hashing Algorithm** | **Year Introduced** | **Hash Length (bits)** | **Security Level** | **Speed** | **Automatic Salt Support** | **Notes** |
| MD2 | 1989 | 128 | Low | Slow | No | Obsolete, vulnerable to various attacks |
| MD4 | 1990 | 128 | Low | Fast | No | Obsolete, vulnerable to collision attacks |
| MD5 | 1991 | 128 | Low | Fast | No | Vulnerable to collision attacks |
| SHA-0 | 1993 | 160 | Low | Fast | No | Withdrawn due to significant flaws |
| SHA-1 | 1995 | 160 | Low | Fast | No | Deprecated due to collision vulnerabilities |
| SHA-256 | 2001 | 256 | High | Moderate | No | Part of the SHA-2 family |
| SHA-512 | 2001 | 512 | Very High | Slow | No | Part of the SHA-2 family |
| BCrypt | 1999 | Variable | High | Slow | Yes | Adaptive function resistant to brute force |
| SCrypt | 2009 | Variable | Very High | Very Slow | Yes | Memory-intensive, resistant to hardware attacks |
| PBKDF2 | 2000 | Variable | High | Slow | Yes | Part of RSA's PKCS #5, configurable iterations |
| Argon2 | 2015 | Variable | Very High | Slow | Yes | Winner of the Password Hashing Competition |

## What are hash “collisions”?

A collision occurs when two different input string produce the exact hash result. Collisions are theoretically possible since the domain (input) of a hashing function is essentially infinite and the range (output) of a hashing function is finite. However, the range of a hashing function is very large, and collisions are rare.

## Birthday Function Collision Analogy

Let’s imagine a very limited hashing algorithm using birthdays. Imagine we have a room of people, and each person has a birthday that falls on one of the 365 days of the year. If we think of each person's birthday as a piece of input data (the domain) and the days of the year as possible hash values (the range), we can understand how hashing works and why collisions (two people having the same birthday) might occur.

Int day = BirthdayHash(“January 3”); // day is 3

Int day = BirthdayHash(“December 31”); // day is 365

Int day = BirthdayHash(“February 1”); // day is 32

## Probability of Birthday Collisions

The range of the Birthday Function is very limited since there are only 365 days in the year. The probability of at least two people sharing the same birthday increases as the number of people in the group increases. This is known as the birthday problem. Let's see the collision probabilities in a table:

|  |  |
| --- | --- |
| **Number of People** | **Probability of Collision** |
| 2 | 0.27% |
| 5 | 2.71% |
| 10 | 11.7% |
| 15 | 25.3% |
| 20 | 41.1% |

Table 1 The probability of a collision grows as the size of a group grows, seen in Figure 5.

Figure The probability of a collision approaches 100% as the size of the group grows.

**Why Hash Collisions Matter**

In hashing, collisions occur when two different pieces of data produce the same hash value. Just like with birthdays, the more items in a collection we have, the higher the chance of a hash collision.

**What is the probability of a MD5 collision?**

MD5 produces a 128-bit hash value. This means there are 2 128 possible different hash values. Using MD5 we need about 1.54×10 19 random items to have a 50% chance of at least one collision.

1.54×10 19 = 15,400,000,000,000,000,000. (15 quintillion)

### Real-World Results

Although the number for a collision probability is extremely low, there have been ways to cause MD5 to produce collisions.

In 1996, researchers found a weakness in the MD5 algorithm that could potentially allow collision attacks, meaning two different inputs could produce the same output. By 2004, they showed that attackers could create these collisions. In 2005, it was demonstrated that attackers could generate colliding x.509 certificates, which are used for SSL secure communications.

In 2008, further research showed that Public Key Infrastructures (PKI), which rely on these certificates, were vulnerable to these attacks. Researchers even created a fake SSL certificate that could make an attacker appear as a trusted root Certificate Authority (CA). Most operating systems come with a collection of trusted CA certificates, some of which use the MD5 algorithm, making them easy targets for attackers to spoof. As a result, MD5 is no longer considered secure.

## Deliverables

There are no deliverables for part 1. However, the review questions will help you prepare for upcoming assessments.

# Part 2 - Coding a Password Hashing Program in Java

In this section we will create a simple program that uses various hashing algorithms on a sample password. We will use the program to demonstrate hashing, and then use it to perform some password cracking.

## Setup a Project

Screen shots will show how to create the project in both VS Code and Eclipse.

VS Code

1. Ensure the following items are installed:
   1. Visual Studio Code
   2. Java Development Kit (JDK)

A black screen with white text

Description automatically generated

Figure Verify that Java is installed with java -version command at the command prompt.

* 1. Maven for Java

A black screen with white text

Description automatically generated

Figure Maven is not installed yet. See instructions below for installing.

* 1. Extension Pack for Java Extension

A screenshot of a computer

Description automatically generated

Figure Extension Pack for Java found in the Extension Marketplace.

## How to Install the Maven runtime (if necessary)

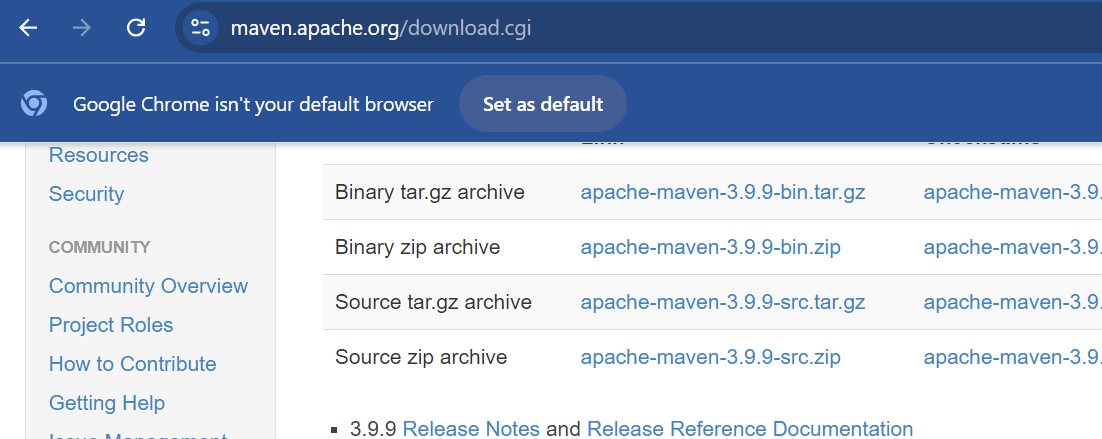


Figure Choose a Binary (executable) of the Apache Maven application.

1. Unzip the archive and move the folder to a location of your choice. In this example, the maven program is placed in c:\Program Files

A screenshot of a computer

Description automatically generated

Figure Maven has been moved to c:\Program Files

1. Add the location of the bin folder to the system Path.

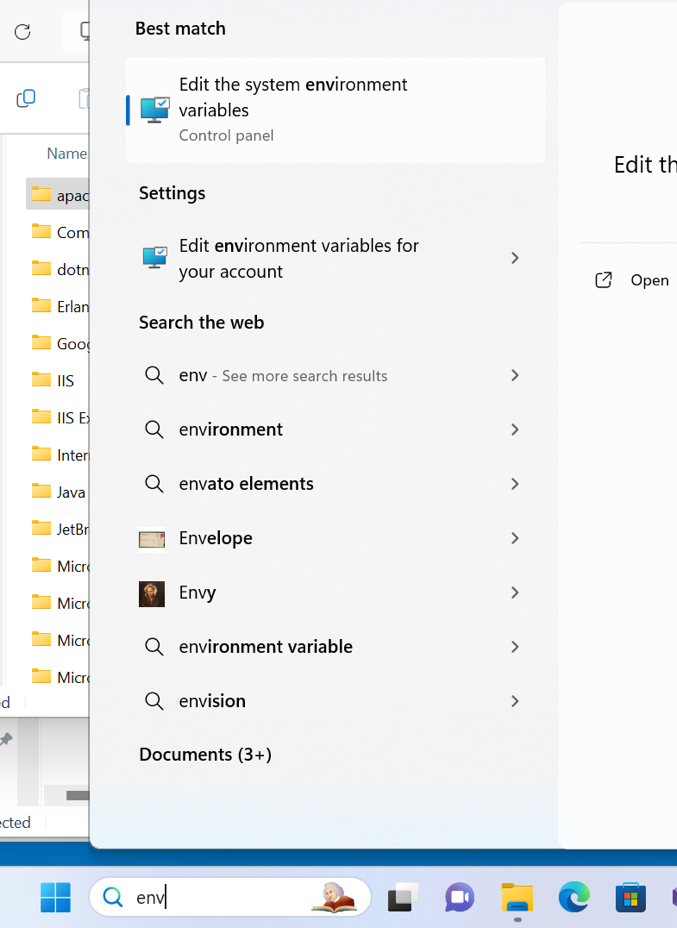


Figure Search for "Environment Varaibles" with "env" search term.

1. Edit the Path variable.

A screenshot of a computer program

Description automatically generated

Figure Path selected. Click Edit button.

A screenshot of a computer

Description automatically generated

Figure All system paths shown. Maven is not included.

A screenshot of a computer

Description automatically generated

Figure Added a new path using the New button.

A screenshot of a computer

Description automatically generated

Figure Browsing to c:\Program Files\apache-maven-3.9.9\bin using the Browse button.

A screenshot of a computer

Description automatically generated

Figure New path has been added to the Paths of this system.

1. Restart the computer for the Path updates to take effect.

## Create a new Java project using the Maven archetype.

1. Create a new Java project for Maven.

A screenshot of a computer

Description automatically generated

Figure Creating a Maven project in VS Code.

1. Choose **maven-archetype-quickstart**, seen in Figure 9 and Figure 10.

A screenshot of a computer

Description automatically generated

Figure Choosing maven-archetype-quickstart in VS Code.

1. Select the newest version the **maven-archetype-quickstart** template.

A blue and white text on a white background

Description automatically generated

Figure VS Code version selection.

A screenshot of a computer

Description automatically generated

Figure Naming the project in VS Code.

A screenshot of a computer

Description automatically generated

Figure Naming the artifact in VS Code.

1. For VS Code, in the terminal, confirm all the default choices for the project, seen in Figure 14 and Figure 15.

A screenshot of a computer program

Description automatically generated

Figure Final step in VS Code console.

A screenshot of a computer

Description automatically generated

Figure Finished project setup in VS Code.

## Eclipse Instructions for Creating a Maven Project

The following figures show the project being created with the Eclipse for Java IDE.

A screenshot of a computer program

Description automatically generated

Figure Creating a new project in Eclipse. File > New > Project.

A screenshot of a project

Description automatically generated

Figure Selecting the Maven project type using Eclipse.

A screenshot of a project

Description automatically generated

Figure Choosing maven-archetype-quickstart in Eclipse.

A screenshot of a project

Description automatically generated

Figure Choosing Group ID, Artifact ID, Version in Eclipse.

1. In Eclipse, confirm the choices in the console.

A screenshot of a computer

Description automatically generated

Figure Final setup in Eclipse console.

A screenshot of a computer

Description automatically generated

Figure Finished project in Eclipse.

1. Update the Java compile version from 1.7 to whatever version is installed on your computer.

A screenshot of a computer code

Description automatically generated

Figure Version 17 is the target Java runtime on this computer.

1. Add a dependency in the POM file to support the bCrypt algorithm.

A screenshot of a computer program

Description automatically generated

Figure Segment from the <dependencies> section of the POM.XML file.

<dependencies>

<dependency>

<groupId>org.mindrot</groupId>

<artifactId>jbcrypt</artifactId>

<version>0.4</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

</dependencies>

1. Verify that Maven correctly downloads the jbcrypt-0.4.jar file

A screenshot of a computer

Description automatically generated

Figure Maven dependencies include jbcrypt as seen in Visual Studio Code..

A screenshot of a computer

Description automatically generated

Figure Maven Dependencies includes jbcrypt as seen in Eclipse.

Eclipse may not update the dependencies automatically. If the dependency is not added automatically, you can add the dependency using a right-click > Maven > Add Dependency process.

A screenshot of a computer

Description automatically generated

Figure In Eclipse, adding a dependency in the pom file using the right-click method.

## Simple Hashing Test

1. Add the following program to App.java

package com.shadsluiter;

// your package name here.

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Scanner;

public class App

{

public static void main( String[] args )

{

Scanner scanner = new Scanner(System.in);

System.out.print("Please enter a password: ");

String password = scanner.nextLine();

String hash = hashWithAlgorithm(password, "MD5");

printWithColor("MD5" + ": ", hash);

scanner.close();

}

private static void printWithColor(String string, String string2) {

// print first string in red

System.out.print("\u001B[31m" + string + " ");

// print second string in blue

System.out.println("\u001B[34m" + string2);

// Reset color

System.out.print("\u001B[0m");

}

private static String hashWithAlgorithm(String password, String algorithm) {

try {

// messagedigest is part of java.security package.

// Supports algorithms like MD5, SHA-1, SHA-256, SHA-512.

// Does not support bcrypt

MessageDigest md = MessageDigest.getInstance(algorithm);

byte[] hashBytes = md.digest(password.getBytes());

StringBuilder sb = new StringBuilder();

for (byte b : hashBytes) {

sb.append(String.format("%02x", b));

}

return sb.toString();

} catch (NoSuchAlgorithmException e) {

return "Algorithm not found: " + algorithm;

}

}

}



Figure The new version of the program runs hashes on an entire list of algorithms.

A computer screen shot of a program code

Description automatically generated

Figure Run a timed test on each algorithm by executing the hash 10000 times!

A screenshot of a computer code

Description automatically generated

Figure Hash with bcrypt is done seperately using the "Mindrot" dependency.

A computer screen shot of a program

Description automatically generated

Figure Hash with bcrypt is a separate helper method since bcrypt is not supported by MessageDigest

1. Run the program again. Note the time required to run each hash 10000 times is similar across all hashing algorithms. Contrast this result with the much longer time for the BCrypt output.

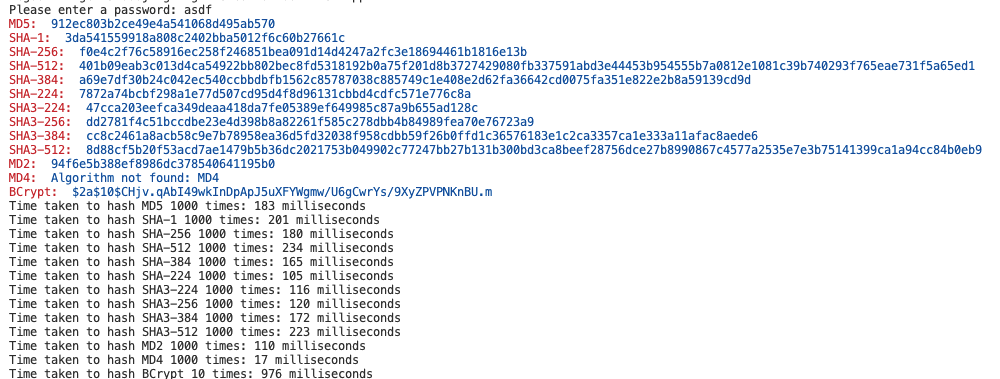


Figure Results of speed tests. Your results will vary depending on the CPU of the computer. The BCrypt algorithm stands apart from the others due to its longer time to compute a hash value.

## Message Digest class in Java Varies

The availability of hashing algorithms can differ between Java versions. For example, some algorithms, particularly newer ones like SHA-224 or SHA3-\* (SHA-3), were added in later versions of Java.

* **SHA-3 algorithms** (SHA3-224, SHA3-256, SHA3-384, SHA3-512) were introduced in **Java 9**. If one of your computers is running Java 8 or earlier, it won't support SHA-3 algorithms, and you'll get a NoSuchAlgorithmException.
* Algorithms like MD4, SHA-224, or MD2 might not be available in certain versions of Java or may be disabled due to security concerns in more recent versions.

## ANSII Colors in Eclipse

The color codes used in this program may not display correctly in Eclipse terminal. You can either update the version of Eclipse or search the Eclipse Marketplace for **ansii color printing** options.



1. Take a screen shot of the application running at this point.
2. Paste the image into a Word document.
3. Put a caption below the image explaining what is being demonstrated.

**Key Differences Between BCrypt and Other Algorithms**

**Speed:**

**MD5, SHA-1, SHA-256, etc**.: These algorithms are designed to be very fast. For example, MD5 can hash the password "asdf" 10000 times in around 204 milliseconds. Their speed is a critical characteristic in applications where fast hashing is necessary, such as checksums and data integrity checks.

**BCrypt**: In contrast, BCrypt is intentionally slow. Hashing a password 10 times takes around 966 milliseconds. BCrypt is specifically designed for hashing passwords. It incorporates a salt to protect against rainbow table attacks and is designed to be slow to thwart brute-force attacks. Its computational expense can be adjusted with a cost factor.

**Security:**

**MD5, SHA-1**: These algorithms are no longer considered secure for cryptographic purposes due to vulnerabilities such as collision attacks.

**SHA-256, SHA-512**: These are more secure than MD5 and SHA-1, but still not ideal for password hashing due to their speed.

**BCrypt**: BCrypt is designed to be secure for password hashing. Its slowness and salt incorporation make it resistant to brute-force attacks and precomputed attacks like rainbow tables.

## Hashing Algorithms Compared

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Hash Length** | **Speed (ms for 1000 hashes)** | **Security Level** | **Notes** |
| MD5 | 128 bits | 204 | Low | Fast, vulnerable to collisions |
| SHA-1 | 160 bits | 206 | Low | Deprecated, vulnerable to collisions |
| SHA-256 | 256 bits | 169 | High | Secure, widely used |
| SHA-512 | 512 bits | 227 | Very High | Very secure |
| SHA-384 | 384 bits | 173 | High | Balanced security |
| SHA-224 | 224 bits | 104 | Moderate | Shorter hash length |
| SHA3-224 | 224 bits | 111 | Moderate | Part of SHA-3 family |
| SHA3-256 | 256 bits | 126 | High | Secure, different structure |
| SHA3-384 | 384 bits | 164 | High | Secure, longer hash length |
| SHA3-512 | 512 bits | 212 | Very High | Most secure among SHA-3 |
| MD2 | 128 bits | 108 | Low | Obsolete, slow |
| MD4 | 128 bits | 56 | Low | Obsolete, insecure |
| BCrypt | Variable | 966 (10 times) | Very High | Slow, designed for password hashing |

# Part 3 - Password Cracking Application

In this section we are going to use a Java program to attempt to guess passwords based on their hash value. At the heart of the program, the same hashing process and hashing algorithms are used as the previous example. The difference is that this program will attempt to guess the original password from which a hash is generated, seen in Figure 24. The program will try a dictionary attack (rainbow table) first, followed by a brute-force attack.

Figure Process of cracking used in this program.

### Compare Dictionary Attacks to Brute Force Attacks

|  |  |  |
| --- | --- | --- |
|  | Dictionary Attack | Brute Force Attack |
| Method | A dictionary attack uses a list of potential passwords, known as a dictionary. This list typically contains common passwords, words from the dictionary, and phrases. | A brute force attack tries every possible combination of characters until the correct one is found. This can include letters (uppercase and lowercase), numbers, and special characters. |
| Efficiency | It's more efficient than a brute force attack because it targets passwords that are likely to be used by people. Some dictionaries are collections of a few thousand common words while other dictionaries can contain millions. Even the largest dictionaries are many times faster than brute force methods. | It is much less efficient than a dictionary attack because it must go through all possible combinations, which can be extremely time-consuming, especially for long and complex passwords. An 8-letter password using lower case letters only contains 268 combinations! 208,827,064,576 (208 billion) |
| Limitations | It is limited to the passwords within the dictionary. If the actual password is not in the list, the attack will fail. | While theoretically guaranteed to find the password, the time required can be impractically long for complex passwords, making it infeasible in many cases. |
| Usage | Dictionary attacks are often used when the attacker has some knowledge or guess about the possible passwords, such as common passwords or passwords based on user information. | Brute force attacks are used when the dictionary attack fails to find a password. There is no knowledge about the password or the password is expected to be very complex and not easily guessed. |

1. Download and open the CST-407-RS-T3.Hashing.zip file. It is provided as topic resources.
2. Unzip / extract the file to a separate folder.
3. Open the application and inspect the contents.
4. You should see a list of txt files used by the program, shown in Figure 25. These are the dictionary lists that the program will use in a variety of combinations.

A screenshot of a computer code

Description automatically generated

Figure HashUtil.java contains a dictionary method that utilizes the txt files by combing combinations of words into password guesses.

1. Run the program with a “Dictionary Attack”

A screenshot of a computer

Description automatically generated

Figure Results of cracking the **ilovesuperman** password by using the dictionary attack. **ilove** is the first word. **superman** is the second word. The dictionary tried combining words in its hacking attempt.

1. Enter a string of characters for a password to crack. Simple and short passwords are easier to crack.
2. Enter an algorithm name. Press ENTER to use the MD5 (default) algorithm or type the name of another algorithm from the list.
3. Press ENTER to crack the hash value you just created. You can also copy and paste a hash value from another source.
4. Enter an algorithm name again. Press ENTER to use MD5 as the default choice.
5. The program will look through its dictionary of precomputed hashed values. If the password is found within the dictionary, it will be cracked almost immediately.
6. Run the program as a “Brute Force” attack.

A screenshot of a computer program

Description automatically generated

Figure Results of cracking **zell9** password using brute force attacks.

## Prefix

The program contains a list of prefixes and suffixes.

A screenshot of a computer

Description automatically generated

Figure Text files in the application are used to join word segments into password guesses.

1. Choose a prefix from the prefix.txt file and combine it with any English word. In the example shown in Figure 28, **ihatesuperman** is the secret password.

A screenshot of a computer

Description automatically generated

Figure The prefix "ihate" is joined with "superman" and the password is hacked very quickly.

## Combine Words

The next step in the process is to join all English words together in all combinations. According to the output shown in Figures 29 through 31 there are over 3 billion passwords that can be formed using this method.

A screenshot of a computer

Description automatically generated

Figure angelfire is a password made of two English words. It is hacked rather quickly using the dictionary.

A screen shot of a computer

Description automatically generated

Figure anti (prefix) cheating and password (suffix) is found using the dictionary.

A screenshot of a computer screen

Description automatically generated

Figure 59 **ukhdep**, a six letter random password is not found in the dictionary. The final attempt to reverse the hash is through brute-force attempts at all combinations of letters.

**Experiment with Password Security**

1. Choose several passwords to hash. Submit the hash value to the cracking portion of the program as shown above.
2. For each hash, experiment with prefixes, suffixes, combining words and random strings of letters.

## Part 4 - Calculate Password Security

1. Complete the following chart. One of the rows has been done as an example.
2. Try to determine what a “safe” password length is considering the speed of today’s computing power.
3. Compare the advantages of using longer passwords such as *thisismyverylongpasswordusinglowercaseletters* to using shorter passwords with more complex sets of alphabets (^A3xf9x\*&#@\_.

Here are three example alphabets

* abcdefghijklmnopqrstuvwxyz has 26 letters
* abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ has 52 letters
* abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!@#$%^&\*() has 72 letters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Password Length | Speed of App (adjust for your CPU) | Alphabet Length | Number of total combinations in pw string | Time to calculate all possible password combinations |
| 5 | 4,000,000 guess / sec | 26 | 26 \* 26 \* 26 \* 26 \* 26  = 26 ^ 5 |  |
| 6 |  | 26 |  |  |
| 7 |  | 26 |  |  |
| 8 |  | 26 |  |  |
| 9 |  | 26 |  |  |
| 10 | 4000000 | 26 | 26 ^ 10 = 141167095653376 | 141167095653376 / 4,000,000 = 35291773 seconds  35291773 / 60 = 588196 minutes  588196 / 60 = 9803 hours  9803 / 24 = 408 days  408 / 365 = 1.1 years |
| 11 |  | 26 |  |  |
| 12 |  | 26 |  |  |
| Use the bottom half of the chart to calculate some larger alphabets. | | | | |
|  |  | 52 |  |  |
|  |  | 72 |  |  |

You could also create this chart in Excel.

A screenshot of a spreadsheet

Description automatically generated

Figure Excel spreadsheet used to estimate the time to crack passwords of various lengths.

Here are some graphs that show the exponential growth of password complexity.

Figure Estimated time to calculate all hashes. X axis is password length. Y axis is years to complete.

Figure Estimated time to calculate all hashes. X axis is password length. Y axis is years to complete.

# Part 5 – Demonstrate Hashing in a Block Chain

## Block Chain Demo

In the next steps you will create the following block chain demonstration.

A screenshot of a computer program

Description automatically generated

Figure A series of actions are registered in a blockchain, making it impossible to tamper with previous transaction data without recomputing the entire chain.

## About the Block Chain Demo

This program demonstrates the basic principles of how a **blockchain** works by simulating the creation and linking of blocks in a simplified blockchain structure. Each block contains data, a unique identifier (ID), a hash of its own data, and a reference to the hash of the previous block (known as the **previous hash**). Additionally, each block records the exact time it was created, giving a timestamp to the block.

## Steps to Create the Block Chain Demo

1. Create a new Java project using **maven-archetype-quickstart**.
2. Create a **Block.java** class in the project.

A screenshot of a computer

Description automatically generated

Figure Block.java is added to the project.

1. Use the following code for App.java

package com.shadsluiter;

import java.util.ArrayList;

import java.util.List;

public class App

{

public static void main( String[] args )

{

List<Block> blockchain = new ArrayList<Block>();

Block genesisBlock = new Block(0, "Genesis Block", "0");

System.out.println("Genesis Block created!");

System.out.println(genesisBlock.toString());

blockchain.add(genesisBlock);

boolean finished = false;

while (!finished) {

System.out.println("Enter the data for the next block:");

String data = System.console().readLine();

Block newBlock = new Block(genesisBlock.getId() + 1, data, genesisBlock.getHash());

genesisBlock = newBlock;

System.out.println("Block added to the blockchain!");

blockchain.add(genesisBlock);

printBlocks(blockchain);

System.out.println("Would you like to add another block? (y/n)");

String response = System.console().readLine();

if (!response.equals("y")) {

finished = true;

}

}

}

private static void printBlocks(List<Block> blockchain) {

// Print the blockchain

System.out.println("Blockchain:");

for (Block block : blockchain) {

System.out.println(block.toString());

}

}

}

1. Use the following code for Block.java

package com.shadsluiter;

import java.security.MessageDigest;

import java.sql.Date;

import java.text.DateFormat;

/\*

\* Block class

\* Each block represents a single block in the blockchain

\*

\*/

public class Block {

private int id; // sequence number of the block

private String data; // The data stored in the block

private String previousHash; // The hash of the previous block

private String hash; // The hash of the block data

// time of the block creation. showing date and time

private Date timestamp;

public Block(int id, String data, String previousHash) {

this.id = id;

this.data = data;

this.previousHash = previousHash;

this.hash = calculateHash();

// format to show the time when the block was created

this.timestamp = new Date(System.currentTimeMillis());

}

/\*

\* calculateHash method

\* Use the SHA-256 algorithm to calculate the hash of the block data

\*/

public String calculateHash() {

try {

// Combine block data with the previous hash

String input = data + previousHash;

// Create a SHA-256 digest

MessageDigest digest = MessageDigest.getInstance("SHA-256");

// Calculate the hash of the input

byte[] hashBytes = digest.digest(input.getBytes("UTF-8"));

// Convert the byte[] into a hex string

return bytesToHex(hashBytes);

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

/\*

\* Helper method to convert byte array to hex string

\*/

private String bytesToHex(byte[] hashBytes) {

StringBuilder sb = new StringBuilder();

for (byte b : hashBytes) {

sb.append(String.format("%02x", b));

}

return sb.toString();

}

public int getId() {

return id;

}

public String getData() {

return data;

}

public String getPreviousHash() {

return previousHash;

}

public String getHash() {

return hash;

}

public void setId(int id) {

this.id = id;

}

public void setData(String data) {

this.data = data;

}

public void setPreviousHash(String previousHash) {

this.previousHash = previousHash;

}

public void setHash(String hash) {

this.hash = hash;

}

/\*

\* toString method creates a box with the block information

\*/

@Override

public String toString() {

// Define ANSI color codes

final String RESET = "\033[0m";

final String RED = "\033[31m";

final String GREEN = "\033[32m";

final String YELLOW = "\033[33m";

final String BLUE = "\033[34m";

final String MAGENTA = "\033[35m";

// Define the block content with color

String line1 = RED + "Block ID: " + id + RESET;

String line2 = GREEN + "Block data: " + data + RESET;

String line3 = YELLOW + "Block hash: " + hash + RESET;

String line4 = BLUE + "Block previous hash: " + previousHash + RESET;

DateFormat dateFormat = DateFormat.getDateTimeInstance();

String output = dateFormat.format(timestamp);

String line5 = MAGENTA + "Block timestamp: " + output + RESET;

// Calculate the maximum length of content, ignoring ANSI codes

int maxLength = Math.max(Math.max(visibleLength(line1), visibleLength(line2)), Math.max(visibleLength(line3), visibleLength(line4)));

// Create padding to make the content align inside the box

String border = "+-" + "-".repeat(maxLength) + "-+";

String paddedLine1 = "| " + padRight(line1, maxLength) + " |";

String paddedLine2 = "| " + padRight(line2, maxLength) + " |";

String paddedLine3 = "| " + padRight(line3, maxLength) + " |";

String paddedLine4 = "| " + padRight(line4, maxLength) + " |";

String paddedLine5 = "| " + padRight(line5, maxLength) + " |";

String connector = " ".repeat(maxLength / 2) + "|" + "\n" +

" ".repeat(maxLength / 2) + "V";

// Return the box with dynamic padding and colored lines

return border + "\n" +

paddedLine1 + "\n" +

paddedLine2 + "\n" +

paddedLine3 + "\n" +

paddedLine4 + "\n" +

paddedLine5 + "\n" +

border + "\n" +

connector;

}

// Helper method to calculate the visible length of a string without ANSI codes

private int visibleLength(String str) {

// strip out the ANSI color codes and return the length using regex pattern that matches ANSI color codes

return str.replaceAll("\\u001B\\[[;\\d]\*m", "").length();

}

// Helper method to pad the content with spaces to the right, based on visible length

private String padRight(String str, int totalLength) {

int visibleLen = visibleLength(str);

return str + " ".repeat(totalLength - visibleLen);

}

}

The program implements the following concepts of a blockchain:

1. **Block Structure**:
   * Each block in the blockchain contains the following fields:
     + **Block ID**: A sequence number that uniquely identifies the block in the blockchain.
     + **Data**: The content stored in the block, provided by the user (e.g., transaction details or arbitrary text).
     + **Hash**: A cryptographic hash (using the SHA-256 algorithm) generated from the block's data and the hash of the previous block. This ensures data integrity and links the blocks together.
     + **Previous Hash**: The hash of the previous block in the chain, ensuring the linkage between blocks.
     + **Timestamp**: The date and time when the block was created, allowing the tracking of when each block was added.
2. **Genesis Block**:
   * The **Genesis Block** is the first block created in the blockchain. It doesn't reference any previous block, so its **previous hash** is set to "0". This block serves as the foundation of the blockchain.
3. **Adding Blocks**:
   * After creating the Genesis Block, the user is prompted to enter data for subsequent blocks. Each new block references the **hash** of the previously added block, linking them together to form a chain.
   * The program continues adding blocks until the user decides to stop by entering n when prompted.
4. **Hashing**:
   * The program uses the **SHA-256** algorithm to create a cryptographic hash for each block. The hash acts as a unique fingerprint of the block's data, and even the smallest change in the data will result in a completely different hash.
   * This ensures that any tampering with a block's data is easily detectable because it would break the chain of hashes.
5. **Blockchain Integrity**:
   * The use of the **previous hash** field ensures that each block is cryptographically linked to the previous one. This chain of hashes guarantees the integrity of the entire blockchain: if any block is altered, it will invalidate the subsequent blocks because the hashes will no longer match
6. Compile and run the program.
7. Create several blocks.
8. Rerun the program. If you enter the exact same block data, the hash values should repeat from the previous time you ran the program.



1. Take a screen shot of the application running at this point.
2. Paste the image into a Word document.
3. Put a caption below the image explaining what is being demonstrated.

## What improvements are needed to make this a “real” Blockchain application?

This program provides a basic simulation of how a blockchain works, but it is significantly simplified compared to a real-world blockchain. Here's a breakdown of the **differences** and **what is missing** when compared to a fully functioning blockchain like Bitcoin or Ethereum:

**1. Lack of Distributed Consensus**

* **Real Blockchain**: In a real blockchain, there are multiple participants (nodes) in a decentralized network. Each node holds a copy of the blockchain, and consensus mechanisms (e.g., Proof of Work or Proof of Stake) are used to agree on the state of the blockchain.
* **In This Program**: This program runs locally on a single machine, with no network or consensus mechanism. There's no way for multiple participants to validate or agree on the contents of the blockchain.

**2. No Mining or Proof of Work**

* **Real Blockchain**: In many blockchains (like Bitcoin), new blocks are added through a process called mining, which requires solving complex cryptographic puzzles. This process is called **Proof of Work** and ensures that adding a block requires significant computational effort, securing the network from tampering.
* **In This Program**: Blocks are added without any computational cost. There's no concept of mining or Proof of Work, so anyone can add blocks freely. There is no resistance to tampering or malicious activity.

**3. No Peer-to-Peer Network**

* **Real Blockchain**: A real blockchain operates over a **peer-to-peer (P2P)** network where nodes communicate with each other to share and validate new blocks and transactions.
* **In This Program**: The program runs as a standalone application with no network connectivity. There is no communication between different instances of the program or between users.

**4. No Transactions**

* **Real Blockchain**: In a real blockchain, blocks typically contain **transactions**, which record the transfer of assets, such as cryptocurrency, tokens, or smart contract executions. Transactions are signed by the participants involved to ensure authenticity.
* **In This Program**: The blocks only contain arbitrary data entered by the user. There are no actual transactions, signatures, or assets being transferred.

**5. No Cryptographic Signatures**

* **Real Blockchain**: In a real blockchain, participants sign transactions with their **private keys** using cryptographic signatures (e.g., ECDSA). This ensures that only the owner of a private key can authorize transactions.
* **In This Program**: There is no concept of participants, private keys, or cryptographic signatures. Any user can enter data and add blocks without any authentication or authorization.

**6. No Difficulty Adjustment**

* **Real Blockchain**: Many blockchains (like Bitcoin) have a concept of **difficulty adjustment** for mining. This ensures that blocks are added at a consistent rate, even as computational power increases or decreases.
* **In This Program**: There is no difficulty mechanism. Blocks are simply added as soon as the user provides data, without any timing or effort-based restrictions.

**7. No Incentives or Rewards**

* **Real Blockchain**: In blockchains like Bitcoin, miners are incentivized to secure the network by earning **block rewards** (newly minted cryptocurrency) and transaction fees.
* **In This Program**: There are no incentives or rewards for adding blocks. Blocks are added without any cost or benefit.

**9. No Forking or Chain Reorganization**

* **Real Blockchain**: In a decentralized blockchain, multiple blocks may be mined simultaneously, causing a **fork**. Consensus mechanisms decide which chain to continue building on, and invalid blocks are discarded.
* **In This Program**: The program follows a single chain with no possibility of forking, since there is no decentralization or competitive block creation.

**10. No Smart Contracts**

* **Real Blockchain**: Advanced blockchains like Ethereum support **smart contracts**, which are programmable contracts that execute automatically when certain conditions are met.
* **In This Program**: The program only supports adding simple user data to blocks. There are no programmable smart contracts or automated execution of rules.

**11. No Security Measures Against Tampering**

* **Real Blockchain**: In a real blockchain, if someone tries to tamper with a block, they would have to re-mine not just that block but also all subsequent blocks due to the cryptographic linkage (i.e., chain of hashes). This makes tampering computationally impractical.
* **In This Program**: While hashes are used to link blocks, there are no security mechanisms to prevent someone from manually modifying block data and recalculating the hashes. The blockchain is fully mutable in this program.

**12. No Real-World Applications**

* **Real Blockchain**: Blockchains are used for various real-world applications such as cryptocurrencies (Bitcoin, Ethereum), decentralized finance (DeFi), supply chain tracking, voting systems, and more.
* **In This Program**: The blocks are simple containers for user-input data with no specific real-world application.

# What You Learned: Password Hashing and Cracking

In this lesson, we explored the fundamentals of password hashing, cracking techniques, and the wide-ranging applications of hashing in modern technology. Here are the key takeaways:

1. **Understanding Hashing Basics:**
   * Hashing is a one-way algorithmic function that transforms data of any length into a fixed-length output (hash), ensuring data integrity by creating a unique fingerprint for the input.
   * Hashing is used extensively to store passwords securely, verify file signatures, check data transmission integrity, and prevent tampering in version control systems like Git.
2. **Why Hash Collisions Matter:**
   * A hash collision occurs when two different inputs produce the same hash value. Although rare in modern, well-designed algorithms, collisions can compromise data integrity. The Birthday Problem analogy helps explain how the probability of collisions increases as the number of inputs grows, even if the output range is large.
3. **Comparison of Hashing Algorithms:**
   * We compared legacy algorithms such as MD5 and SHA-1, which are no longer considered secure, with more modern and secure options like SHA-256 and BCrypt.
   * BCrypt stands out for password hashing because it is specifically designed to be slow and resistant to brute-force attacks, incorporating a "salt" to further secure the password.
4. **Salting and Rainbow Tables:**
   * Salting adds a unique random value to each password before hashing to prevent two identical passwords from producing the same hash. This makes rainbow tables, which are precomputed lists of hashes, ineffective.
   * Without salting, attackers can use rainbow tables to quickly crack common passwords.
5. **Password Cracking Techniques:**
   * We experimented with dictionary attacks, which use precompiled lists of common passwords, and brute-force attacks, which try every possible character combination to guess a password. The experiments showed how using complex and long passwords significantly increases the difficulty of cracking them.
6. **Beyond Password Security:**
   * **File Signatures:** Hashes are used to verify file integrity during downloads. Users can compare the hash value of a downloaded file with the hash published by the file provider to ensure that the file has not been tampered with.
   * **Network Transmission:** Hashes are used in communication protocols like TCP to verify that transmitted data arrives intact by matching hash values generated at the sender's and receiver's ends.
   * **GitHub and Version Control:** Hashes are critical to tracking changes in repositories. Every commit in Git is assigned a unique hash (commit hash), ensuring that even the smallest change to a file is recorded and cannot be tampered with without detection.
7. **Blockchain Technology:**
   * Hashing is an important part of blockchain technology, where each block in the chain contains a hash of its data and the hash of the previous block. This creates an immutable record that guarantees the integrity of the entire blockchain. Any attempt to modify a block's data invalidates all subsequent blocks, making tampering computationally impractical.
8. **Best Practices for Password Security:**
   * Secure password storage involves using modern hashing algorithms like BCrypt and incorporating salts to protect against attacks. Passwords should be long and complex, using a combination of letters, numbers, and symbols to resist both dictionary and brute-force attacks.

## Check for Understanding

Although these questions are not graded, they will help you prepare for upcoming assessments.

1. **What is the primary purpose of hashing?**

* A) To encrypt data for secure transmission
* B) To create a unique fingerprint for data to ensure integrity
* C) To compress data for faster processing
* D) To reverse data for future access **Answer: B**

1. **How does a hashing function handle inputs of different lengths?**

* A) It produces a variable-length output
* B) It produces a fixed-length output regardless of input size
* C) It increases the output size based on input size
* D) It truncates the input to fit a fixed length **Answer: B**

1. **What is a hash collision?**

* A) When two different inputs produce the same hash value
* B) When a hash function produces an error
* C) When two hash outputs have identical values
* D) When two inputs are reversed into each other **Answer: A**

1. **Why is the Birthday Problem analogy useful in explaining hashing?**

* A) It shows the probability of multiple people sharing the same name
* B) It illustrates how likely it is for two different inputs to produce the same hash
* C) It explains the probability of cracking a hash
* D) It demonstrates why hashing is faster than encryption **Answer: B**

1. **Why is MD5 no longer considered secure for hashing?**

* A) It is too slow for modern systems
* B) It produces short hash values that are easy to crack
* C) It is vulnerable to collision attacks
* D) It can be easily reversed **Answer: C**

1. **What is a rainbow table?**

* A) A tool used to encrypt data
* B) A precomputed list of hash values used to quickly crack passwords
* C) A database of all possible hash values
* D) A tool for reversing a hash value **Answer: B**

1. **What is the role of salting in password hashing?**

* A) To speed up the hashing process
* B) To add security by ensuring identical passwords produce different hashes
* C) To ensure that hash values can be reversed
* D) To compress the data into a smaller output **Answer: B**

1. **Which hashing algorithm is considered highly secure and resistant to brute-force attacks?**

* A) MD5
* B) SHA-1
* C) BCrypt
* D) MD4 **Answer: C**

1. **Why is it important to use secure hashing algorithms for password storage?**

* A) To allow easy decryption when needed
* B) To prevent unauthorized access by making it difficult to reverse-engineer the password
* C) To store the passwords in a smaller format
* D) To allow users to recover passwords if forgotten **Answer: B**

1. **How is hashing used in version control systems like Git?**

* A) To store file content in an encrypted format
* B) To create a unique identifier for each commit, ensuring changes cannot be modified without detection
* C) To ensure that passwords stored in repositories are secure
* D) To verify that all files in a repository have the same name **Answer: B**

1. **What is the purpose of hashing in blockchain technology?**

* A) To encrypt the data in each block
* B) To create a secure link between blocks by storing the hash of the previous block
* C) To ensure that all blocks contain the same amount of data
* D) To make the blockchain easier to search **Answer: B**

1. **Which of the following demonstrates a use case for hashing in network transmissions?**

* A) Encrypting the message to protect against eavesdropping
* B) Creating a value to verify that the data received is the same as the data sent
* C) Compressing the message for faster transmission
* D) Storing a backup copy of the message **Answer: B**

# Deliverables:

1. Submit a Microsoft Word document with screenshots of the applications being run. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.
2. In the same document, write a summary of the key concepts that were demonstrated in this lesson including:
   1. Why hashing is a better choice for password security than encryption.
   2. Other than passwords, how is hashing used to improve data security?
   3. What insights did you learn about password strength?
3. In the same document, include the password cracking time estimate table. Include your findings regarding the best practices that users should follow when creating a password.
4. Convert the Word document to PDF.
5. Attach the PDF document. Multiple files can be uploaded with an assignment.