RAMAIAH
Institute of Technology

Department of
COMPUTER SCIENCE AND
ENGINEERING (CYBER
SECURITY)

Secure Coding in C and C++ Report

On

Zero Knowledge Proof Authentication Using Secure Coding Practices

Submitted in partial fulfilment of the Requirements for the

Academic of 5th Semester – **Secure Coding in C and C++[** CYE551**]**

In

Dept. of CSE (CyberSecurity)

By

Nishanth M
1MS22CY047

Under the guidance
of **SHANKARAMMA**
Assistant professor
CSE (Cyber Security)

M. S. Ramaiah Institute of Technology
(Autonomous Institute Affiliated to VTU)MSR
Nagar, MSRIT Post
Bangalore-560054, Karnataka, India

Table of Contents

1.Introduction

This report demonstrates the implementation of **Zero-Knowledge Proof (ZKP)** authentication and highlights the importance of secure coding practices in ensuring its robustness. By leveraging secure coding techniques, the system mitigates vulnerabilities and provides a highly secure authentication mechanism. This report includes an explanation of the ZKP concept, implementation details, pitfalls, advantages, and suggestions for future improvements.

---

2. Concept Explanation

Zero-Knowledge Proof (ZKP) Basics

- **ZKP** is a cryptographic method where one party (prover) proves to another (verifier) that they know a value (like a password) without revealing the actual value.
- In ZKP, the authenticity is verified through mathematical computations that ensure secure communication while preserving privacy.
- **Example:-** Imagine you're trying to prove you know the password to a vault without saying the password. Instead, you show actions that can only be performed by someone who knows the password. This concept is at the heart of ZKP.
- ZKPs are used in various real-world applications like secure authentication, blockchain, and privacy-preserving systems

Secure Coding Practices in ZKP

The implementation of ZKP authentication in this project includes the following secure coding practices:

1. **Input Validation:** Ensures that all inputs (e.g., prime numbers, generators, passwords) are checked for validity to prevent attacks such as SQL injection or buffer overflow.
2. **Hashing Sensitive Data:** Passwords are hashed before being transmitted, ensuring that plaintext credentials are never exposed.
3. **Use of Cryptographic Libraries:** Reliable cryptographic functions (e.g., secure hashing) are used to prevent weak encryption vulnerabilities.
4. **Modularization:** The code is broken into separate functions for key steps (e.g., prime number generation, hashing, verification), making it easier to test and secure.
5. **Memory Management:** Avoids buffer overflows by using modern C++ constructs and libraries.

3. How Does It Ensure Authenticity?

ZKP ensures authenticity through the following principles:

• **Challenge-Response Mechanism:** The random challenge c ensures that the Prover cannot fake knowledge. If the Prover does not know x, they cannot generate a correct response consistently.

• **No Information Leak:** Only public values (p, g, and y) and a one-time challenge-response pair are exchanged. The Verifier learns nothing about x. 1. Start with the LHS (Verifier's Computation

1. Start with the LHS (Verifier's Computation)

The Verifier computes the Left-Hand Side:

$$\text{LHS} = g^r \cdot y^c \mod p$$

2. Substitute $y = g^x \mod p$

Substitute the definition of y into y^c:

$$\text{LHS} = g^r \cdot (g^x)^c \mod p$$

3. Simplify $(g^x)^c$

Using the power rule $(a^b)^c = a^{b \cdot c}$, we have:

$$(g^x)^c = g^{x \cdot c} \mod p$$

Substituting this back:

$$\text{LHS} = g^r \cdot g^{x \cdot c} \mod p$$

4. Combine the Exponents

Using the rule $a^m \cdot a^n = a^{m+n}$:

$$\text{LHS} = g^{r+x\cdot c} \mod p$$

5. Substitute $r = v - c \cdot x \mod (p-1)$

Now, replace r with the Prover's response $r = v - c \cdot x \mod (p-1)$:

$$\text{LHS} = g^{(v-c\cdot x)+c\cdot x} \mod p$$

Simplify the exponent:

$$\text{LHS} = g^{v} \mod p$$

6. Compare with the RHS

The Prover sent $t = g^v \mod p$ to the Verifier. Thus:

$$\text{LHS} = \text{RHS} = g^{v} \mod p$$

Conclusion

The equation holds because:

- $r = v - c \cdot x \mod (p-1)$ ensures the Prover's response encodes the secret x.
- The LHS combines g^r and y^c, reconstructing g^v without exposing x.

By proving LHS = RHS, the Verifier is convinced the Prover knows x, without learning x itself.

Pitfalls

- **Complexity:** Implementing ZKP algorithms can be error-prone, leading to subtle bugs or security loopholes if not thoroughly tested.
- **Cryptographic Failures:** Using weak or outdated hashing algorithms (e.g., MD5) can compromise security.
- **Resource Constraints:** Computational overhead from repeated modular exponentiation may slow down systems with limited resources.

Advantages

- **Enhanced Security:** Prevents vulnerabilities such as brute force and man-in-the-middle attacks.
- **Data Privacy:** Ensures that sensitive data like passwords are never shared in plaintext.
- **Robust Authentication:** ZKP provides a strong mechanism to authenticate users without direct exposure of credentials.
- **Scalability:** Modular and secure code design allows the system to be extended for other cryptographic applications.

Further Improvements

- **Advanced Cryptographic Techniques:** Use zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) for improved efficiency.

- **Integration with Blockchain:** Implement ZKP for secure decentralized identity verification.

- **Optimization of Computations:** Reduce computational overhead by using optimized algorithms for modular arithmetic.

- **Enhanced GUI:** Provide an intuitive user interface to visualize ZKP steps and their security features.

---

2. main.cpp:

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <ctime>
#include <fstream>
#include <windows.h> // For Sleep()

using namespace std;

// Colors
#define RESET   "\033[0m"
#define BOLD    "\033[1m"
#define RED     "\033[31m"
#define GREEN   "\033[32m"
#define YELLOW  "\033[33m"
#define BLUE    "\033[34m"
#define MAGENTA "\033[35m"
#define CYAN    "\033[36m"
#define WHITE   "\033[37m"

// ASCII Art
void displayHeader() {
    cout << MAGENTA << R"(
    __|
|______________________________________________________________________________________
```

```
_________________________________________________________________________________
___|  |__
    __
_________________________________________________________________________________
_________________________________________________________________________________
__    __
      |  |
|  |
      |  |:::::::::::: .  ::::::::::.      :::.      ...    ::::::::::::::: ::
.: .,:::::::::.   :::.:::::::::::::::: .,-:::::   :::. :::::::::::::::    ...
:::.   ::::.|  |
      |  |'`````;;;;;; .;;,.`;;;```.;;;      ;;`;;      ;;     ;;;;;;;;;;;'''',;;
;;,;;;;''''`;;;;,  `;;;;;;;;;;;''''';;;,;;;'`````'   ;;`;;;;;;;;;;;'''';;;
.;;;;;;;.`;;;;,  `;;;|  |
      |  |    .n[['[[[[[/'   `]]nnn]]'      ,[[ '[[,  [['     [[[     [[
,[[[,,,[[[ [[cccc   [[[[[. '[[     [[     [[[[[[         ,[[ '[[,   [[     [[[,[[
\[[,[[[[[. '[[|  |
      |  |  ,$$P" _$$$$,      $$$""       c$$$cc$$$c $$      $$$     $$
"$$$"""$$$ $$""""   $$$ "Y$c$$     $$     $$$$$$         c$$$cc$$$c  $$     $$$$$$,
$$$$$$ "Y$c$$|  |
      |  |,888bo,_"888"88o,   888o          888   888,88    .d888     88,    888
"88o888oo,__ 888    Y88     88,    888`88bo,__,o, 888   888, 88,    888"888,_
_,88P888    Y88|  |
      |  | `""*UMM MMM "MMP"  YMMMb         YMM   """`  "YmmMMMM""     MMM    MMM
YMM""""YUMMMMMM     YM     MMM    MMM  "YUMMMMMP"YMM   """`  MMM    MMM  "YMMMMMP"
MMM      YM|  |
    __|
|________________________________________________________________________________
_________________________________________________________________________________
___|  |__
    __
_________________________________________________________________________________
_________________________________________________________________________________
__    __
      |  |
|  |
    )" << RESET;
}

// System Info
void displaySystemInfo() {
    time_t now = time(0);
    char *dt = ctime(&now);
    cout << YELLOW << "Current Time: " << RESET << dt;
}

// Loading Animation
```

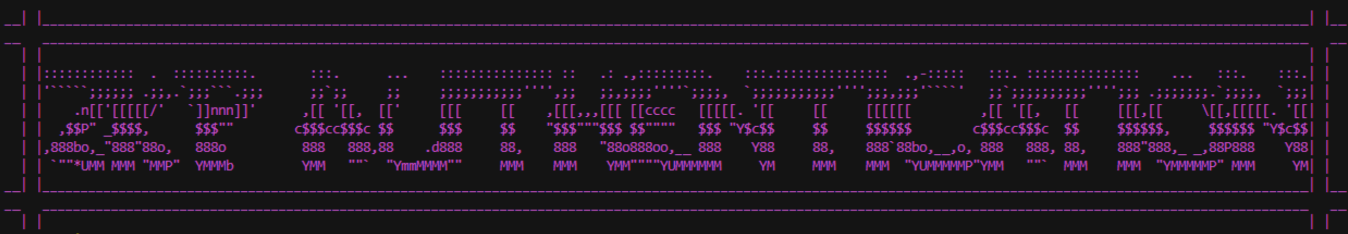
```
void loadingAnimation(const string &message) {
    cout << GREEN << message << RESET;
    for (int i = 0; i < 5; ++i) {
        cout << ".";
        cout.flush();
        Sleep(300); // 300ms
    }
    cout << endl;
}

// Progress Bar
void progressBar(const string &task) {
    cout << task << ": [";
    for (int i = 0; i <= 50; i++) {
        cout << "#";
        cout.flush();
        Sleep(50); // 50ms
    }
    cout << "] Done!" << endl;
}

// Logging System
void logMessage(const string &message) {
    ofstream logFile("project_log.txt", ios::app);
    logFile << message << endl;
    logFile.close();
    cout << YELLOW << "[LOG] " << RESET << message << endl;
}

// Function to execute Python scripts
void executePythonScript(const string &scriptName) {
    loadingAnimation("Running Python script");
    string command = "python " + scriptName; // Ensure Python is in PATH
    int result = system(command.c_str());
    if (result != 0) {
        cout << RED << "Failed to execute Python script: " << scriptName << RESET
<< endl;
    } else {
        logMessage("Successfully executed Python script: " + scriptName);
    }
}

// Function to execute C++ programs
void executeCppProgram(const string &programName) {
    loadingAnimation("Running C++ program");
    string command = programName; // Use "./" for Linux/Mac
    int result = system(command.c_str());
```

```
    if (result != 0) {
        cout << RED << "Failed to execute C++ program: " << programName << RESET
<< endl;
    } else {
        logMessage("Successfully executed C++ program: " + programName);
    }
}

// Easter Egg
void easterEgg() {
    cout << GREEN << R"(
         YOU FOUND AN EASTER EGG!
           ,     ,
          /(     )`
         (  \   /  )
         )   o o   (
        (    '='    )
         \         /
          \_______/
    )" << RESET << endl;
}

// Main Menu
int main() {
    displayHeader();
    displaySystemInfo();
    int choice;

    while (true) {
        cout << "\n" << CYAN << "Select a phase to execute:" << RESET << endl;
        cout << BOLD << GREEN
             << "1. Phase 1: Connection Check (Python)\n"
             << "2. Phase 2: Prime Number Generation (C++)\n"
             << "3. Phase 3: Registration (Python)\n"
             << "4. Phase 4: Compute y (C++)\n"
             << "5. Phase 5: Clear Memory (C++)\n"
             << "6. Phase 6: Check Connections (Python)\n"
             << "7. Phase 7: Server Send (C++)\n"
             << "8. Phase 8: Login (Python)\n"
             << "9. Phase 9: Client Receive (C++)\n"
             << "10. Phase 10: Compute r (C++)\n"
             << "11. Phase 11: Verification (C++)\n"
             << "12. EXIT\n"
             << RESET;
        cout << "Enter your choice: ";
        cin >> choice;
```

```
        switch (choice) {
            case 1:
                executePythonScript("phase1.py");
                break;
            case 2:
                executeCppProgram("phase2");
                break;
            case 3:
                executePythonScript("phase3_registration.py");
                break;
            case 4:
                executeCppProgram("phase4");
                break;
            case 5:
                executeCppProgram("phase5");
                break;
            case 6:
                executePythonScript("phase6_connections.py");
                break;
            case 7:
                executeCppProgram("phase7");
                break;
            case 8:
                executePythonScript("phase8_login.py");
                break;
            case 9:
                executeCppProgram("phase9");
                break;
            case 10:
                executeCppProgram("phase10");
                break;
            case 11:
                executeCppProgram("phase11");
                break;
            case 12:
                logMessage("User exited the program.");
                cout << GREEN << "Goodbye!" << RESET << endl;
                return 0;
            case 42:  // Easter egg
                easterEgg();
                break;
            default:
                cout << RED << "Invalid choice. Please select a valid option." <<
RESET << endl;
        }
    }
```

```
    return 0;
}
```

```
PS C:\Users\Nishanth\Documents\GitHub\Zero-Knowledge-Proof-Authentication> c:\Users\Nishanth\Documents\GitHub\Zero-Knowledge-Proof-Authentication\main.exe
```

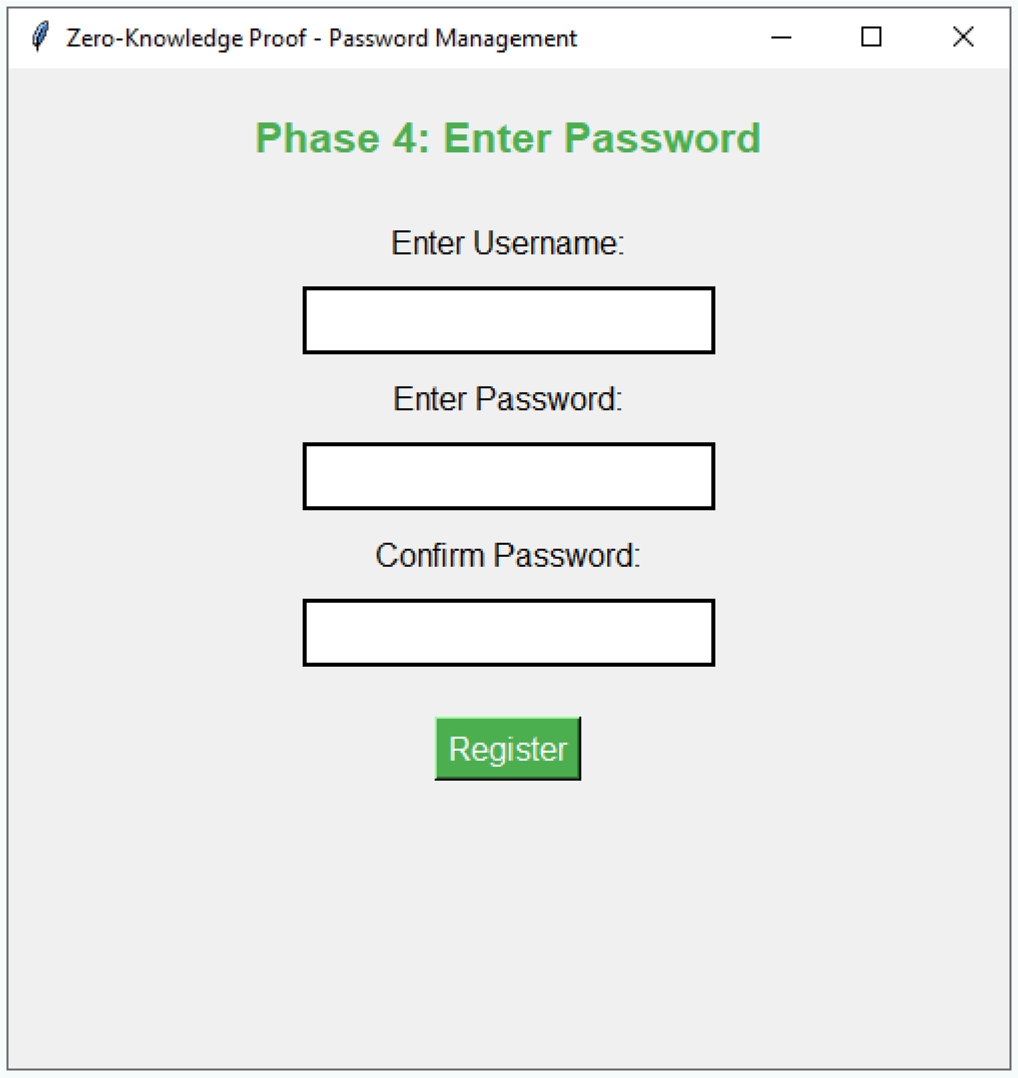


```
    Current Time: Mon Dec 23 20:03:24 2024

Select a phase to execute:
1. Phase 1: Connection Check (Python)
2. Phase 2: Prime Number Generation (C++)
3. Phase 3: Registration (Python)
4. Phase 4: Compute y (C++)
5. Phase 5: Clear Memory (C++)
6. Phase 6: Check Connections (Python)
7. Phase 7: Server Send (C++)
8. Phase 8: Login (Python)
9. Phase 9: Client Receive (C++)
10. Phase 10: Compute r (C++)
11. Phase 11: Verification (C++)
12. EXIT
Enter your choice:
```

```
Select a phase to execute:
1. Phase 1: Connection Check (Python)
2. Phase 2: Prime Number Generation (C++)
3. Phase 3: Registration (Python)
4. Phase 4: Compute y (C++)
5. Phase 5: Clear Memory (C++)
6. Phase 6: Check Connections (Python)
7. Phase 7: Server Send (C++)
8. Phase 8: Login (Python)
9. Phase 9: Client Receive (C++)
10. Phase 10: Compute r (C++)
11. Phase 11: Verification (C++)
12. EXIT
Enter your choice: 2
Running C++ program.....

=== Prime Number Generation ===
Do you want to (e)nter your own number or (g)et a system-suggested prime? (e/g): g
The system suggests the prime number: 4027
Do you accept this prime number? (y/n): y
Prime number 4027 has been selected.
Prime number 4027 has been saved to prime.txt.

=== Phase 3: Generator Selection ===
Explanation:
A generator (g) is a number such that all elements in the group (Z_p*) can be generated as powers of g modulo p.
It must satisfy: 1 < g < 4026
Enter a generator (g): 1000
Generator 1000 has been selected.
[LOG] Successfully executed C++ program: phase2
```

```cpp
#include <iostream>
#include <fstream>
#include <string>
```

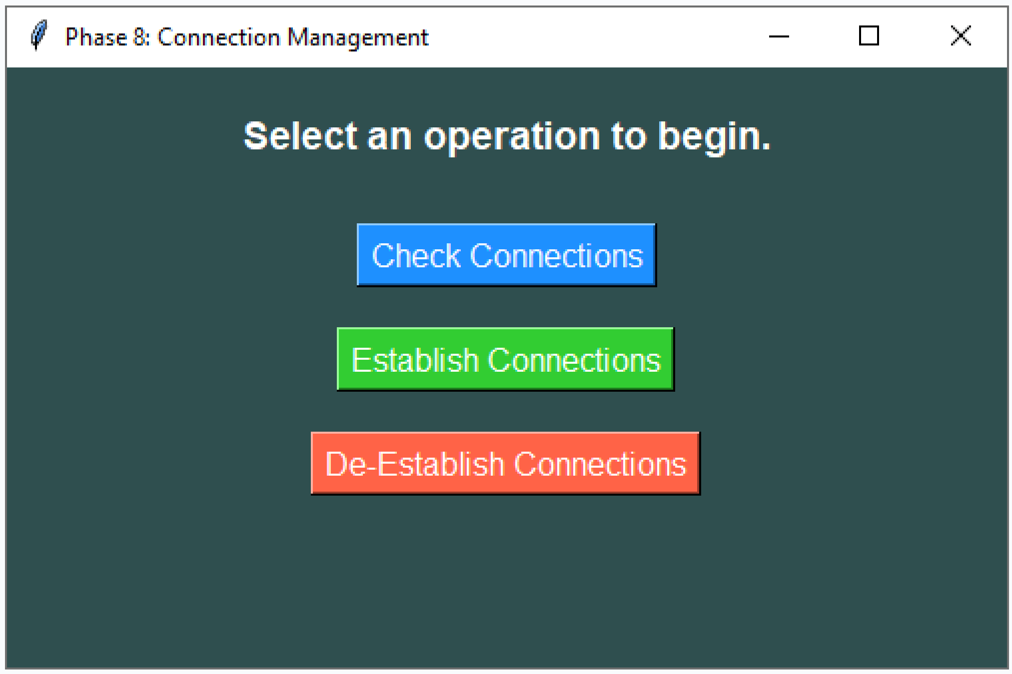
```
// Function to read values from a file
std::string read_from_file(const std::string& filename) {
    std::ifstream file(filename);
    std::string value;

    // Check if file exists and can be opened
    if (file.is_open()) {
        std::getline(file, value);  // Assuming one line per file
        file.close();
    } else {
        std::cerr << "Error: File not found or could not be opened: " << filename
<< std::endl;
        return "";  // Return an empty string to indicate failure
    }

    return value;
}

int main() {
    // Read values from the respective files
    std::string g = read_from_file("generator.txt");
    std::string y = read_from_file("y.txt");
    std::string p = read_from_file("prime.txt");

    // Check if any of the files failed to open
    if (g.empty() || y.empty() || p.empty()) {
        std::cerr << "Error: One or more required files are missing or could not
be read." << std::endl;
        return 1;  // Exit the program with an error code
    }

    // Simulate sending values to the server by printing them
    std::cout << "Sending values to server:" << std::endl;
    std::cout << "g (generator): " << g << std::endl;
    std::cout << "y (computed value): " << y << std::endl;
    std::cout << "p (prime number): " << p << std::endl;

    // Optionally, simulate a "server response" by printing what the server might
expect
    std::cout << "\nServer would now receive these values." << std::endl;

    return 0;
}
```

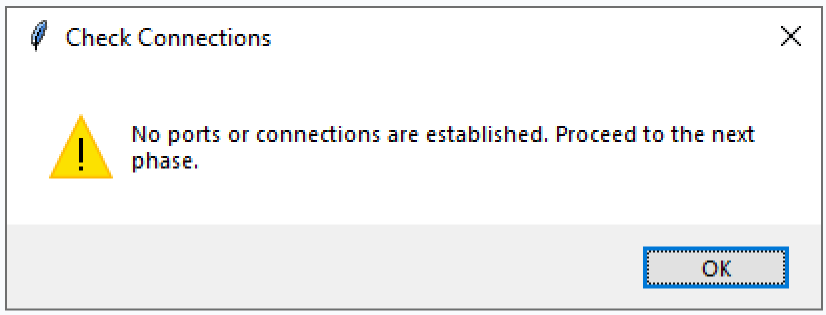


```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>  // For rand() and srand()
#include <ctime>    // For time() (used to seed the random number generator)

using namespace std;

// Color codes for aesthetic output
const string RESET = "\033[0m";       // Reset color
const string GREEN = "\033[32m";      // Success messages
const string RED = "\033[31m";        // Error messages
const string BLUE = "\033[34m";       // Information messages
```

```
// Function to read an integer value from a file
int read_int_from_file(const std::string& filename) {
    std::ifstream file(filename);
    int value = 0;

    // Check if file exists and can be opened
    if (file.is_open()) {
        file >> value;
        if (!file.fail()) {
            file.close();
            return value;
        } else {
            cerr << RED << "Error: Failed to read integer from file: " << filename
<< RESET << endl;
            file.close();
            return -1;  // Return -1 to indicate an error
        }
    } else {
        cerr << RED << "Error: File not found or could not be opened: " <<
filename << RESET << endl;
        return -1;  // Return -1 to indicate an error
    }
}

// Function to write an integer value to a file
void write_int_to_file(const std::string& filename, int value) {
    std::ofstream file(filename);

    if (file.is_open()) {
        file << value << std::endl;  // Write the value followed by a newline
        file.close();
        cout << GREEN << "Value " << value << " has been written to " << filename
<< RESET << endl;
    } else {
        cerr << RED << "Error: Unable to open file " << filename << " for
writing." << RESET << endl;
    }
}

int main() {
    // Read the necessary values from files
    int x = read_int_from_file("hashed_password.txt");
    int c = read_int_from_file("c.txt");  // Assuming the challenge value is
stored in "c.txt"
    int v = read_int_from_file("v.txt");  // Assuming the random value is stored
in "v.txt"
    int p = read_int_from_file("prime.txt");
```

```
    // Check for errors in reading any of the values
    if (x == -1 || c == -1 || v == -1 || p == -1) {
        cerr << RED << "Error: Failed to read one or more necessary values from
files." << RESET << endl;
        return 1;  // Exit the program if any value is missing
    }

    // Compute r using the formula r = (x + c * v) % (p - 1)
    int r = (v - c * x % (p - 1) + (p - 1)) % (p - 1);

    // Print the computed r
    cout << BLUE << "\nClient has computed r:" << RESET << endl;
    cout << "r = " << r << endl;

    // Write the value of r to r.txt
    write_int_to_file("r.txt", r);

    // Optionally, simulate sending r to the server
    cout << BLUE << "\nClient will send r to the server next." << RESET << endl;

    return 0;
}
```





```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include <string>
#include <cstdlib>  // For rand() and srand()
#include <ctime>    // For time() (used to seed the random number generator)

using namespace std;

// Color codes for aesthetic output
const string RESET = "\033[0m";        // Reset color
const string GREEN = "\033[32m";       // Success messages
const string RED = "\033[31m";         // Error messages
const string BLUE = "\033[34m";        // Information messages

// Function to read an integer value from a file
int read_int_from_file(const string& filename) {
    ifstream file(filename);
    int value = 0;
```

```
    // Check if file exists and can be opened
    if (file.is_open()) {
        file >> value;
        if (!file.fail()) {
            file.close();
            return value;
        } else {
            cerr << RED << "Error: Invalid data in file: " << filename << RESET <<
endl;
            file.close();
            return -1;  // Return -1 to indicate an error
        }
    } else {
        cerr << RED << "Error: File not found or could not be opened: " <<
filename << RESET << endl;
        return -1;  // Return -1 to indicate an error
    }
}

// Function to compute the modular exponentiation (base^exp % mod)
int modular_pow(int base, int exp, int mod) {
    int result = 1;
    base = base % mod;  // In case base is larger than mod

    while (exp > 0) {
        if (exp % 2 == 1) {  // If exp is odd
            result = (result * base) % mod;
        }
        exp = exp >> 1;  // exp = exp / 2
        base = (base * base) % mod;  // base = base^2 % mod
    }

    return result;
}

int main() {
    // Read the necessary values from files
    int g = read_int_from_file("generator.txt");
    int p = read_int_from_file("prime.txt");
    int y = read_int_from_file("y.txt");
    int c = read_int_from_file("c.txt");
    int v = read_int_from_file("v.txt");
    int r = read_int_from_file("r.txt");

    // Check for errors in reading any of the values
    if (g == -1 || p == -1 || y == -1 || c == -1 || v == -1 || r == -1) {
```

```
        cerr << RED << "Error: Failed to read one or more necessary values from 
files." << RESET << endl;
        return 1;  // Exit the program if any value is missing
    }

    // Compute t = g^r % p
    int t = modular_pow(g, v, p);

    // Compute result = (y * v^c) % p
    int v_c = modular_pow(v, c, p);  // Compute v^c % p
    int result = (modular_pow(g, r, p) * modular_pow(y, c, p)) % p;

    // Print the results
    cout << BLUE << "\nServer is verifying the response..." << RESET << endl;
    cout << "Computed t = " << t << endl;
    cout << "Expected result = " << result << endl;

    // Verification step
    if (t == result) {
        cout << GREEN << "\nVerification successful: Client's response is valid." 
<< RESET << endl;
    } else {
        cout << RED << "\nVerification failed: Client's response is invalid." << 
RESET << endl;
    }

    return 0;
}
```

```
PS C:\Users\Nishanth\Documents\GitHub\Zero-Knowledge-Proof-Authentication\cpp codes> ./a
Enter your username: Nish
Enter your password: nish123
Computed x for correct password: 578
Generated prime p: 4027
Generated generator g: 3999

Enter your username for verification: Nish
Enter your password for verification: nish123
Computed x for wrong password: 578

====== Results ======
Username:                Nish
Password:                nish123
Wrong Username:          Nish
Wrong Password:          nish123
Prime (p):               4027
Generator (g):           3999
x (hashed correct password):    578
x (hashed wrong password):      578
y = g^x mod p:           702
Random v:                1354
Random challenge c:      138
r = (v - c * wrong_x) mod (p-1):        2110
t = g^v mod p:           966
Verification result:     966
Client has proven the knowledge of password.
PS C:\Users\Nishanth\Documents\GitHub\Zero-Knowledge-Proof-Authentication\cpp codes>
```

```
PS C:\Users\Nishanth\Documents\GitHub\Zero-Knowledge-Proof-Authentication\cpp codes> ./a
Enter your username: Nish
Enter your password: nish123
Computed x for correct password: 578
Generated prime p: 4027
Generated generator g: 3999

Enter your username for verification: Nish
Enter your password for verification: abc
Computed x for wrong password: 778

====== Results ======
Username:                Nish
Password:                nish123
Wrong Username:          Nish
Wrong Password:          abc
Prime (p):               4027
Generator (g):           3999
x (hashed correct password):    578
x (hashed wrong password):      778
y = g^x mod p:           702
Random v:                1354
Random challenge c:      138
r = (v - c * wrong_x) mod (p-1):        2692
t = g^v mod p:           966
Verification result:     1695
Client has not proven the knowledge of password.
PS C:\Users\Nishanth\Documents\GitHub\Zero-Knowledge-Proof-Authentication\cpp codes>
```

Secure Coding Practices Applied in the Project

Below are the secure coding practices applied in the project setup, ensuring it is resistant to common vulnerabilities:

1 Modularization

Practice: The project is broken into multiple phases, each handling a specific task (e.g., key generation, hashing, authentication).
Why it's secure: Modularization prevents monolithic code that's harder to debug and secure. Each phase can be independently verified and updated, minimizing risks of unintended side effects.

2 Use of Cryptographic Techniques

Practice: Secure password hashing and modular arithmetic are used (e.g., $g^x \bmod p$).
Why it's secure:

- Passwords are **never stored in plaintext**; only their hashed versions are used.
- The use of large prime numbers p and secure generators g ensures cryptographic strength, making brute-force attacks computationally infeasible.

3 Zero-Knowledge Proof (ZKP) Design

Practice: The algorithm ensures that sensitive information (e.g., the password x) is never transmitted directly.
Why it's secure:

- Even if an attacker intercepts communications, they cannot reconstruct x, thanks to modular arithmetic and random challenges.

4 Secure Randomness

Practice: The use of random challenges c, private values v, and primes ensures non-repetition in cryptographic operations.
Why it's secure:

- Randomized values prevent replay attacks and ensure that the proof is unique for every interaction.

5 Defense Against Replay Attacks

Practice: Random values (like c) are generated fresh for each verification attempt.
Why it's secure: If an attacker tries to replay previously captured messages, the verification process will fail because the challenges and responses won't align.

6 Safe Handling of File Operations

Practice: File reading and writing operations are error-checked (e.g., verifying file existence and handling `FileNotFoundError`).
Why it's secure:

- Prevents crashes or undefined behavior if files like `status.txt` are missing or corrupted.
- Helps mitigate **path traversal attacks** by using hardcoded file paths and limited file operations.

7 Use of Python Subprocess for External Programs

Practice: The `subprocess` module is used to invoke external programs (e.g., C++ binaries) in a controlled way.
Why it's secure: This approach prevents shell injection vulnerabilities by separating the command and arguments safely (e.g., `subprocess.Popen(["./program", "arg"])`).

8 Input Validation

Practice: User inputs (e.g., phase choices in `main.cpp`) are validated for correctness.
Why it's secure:

- Reduces the risk of undefined behavior or improper program execution caused by invalid inputs.

9 Error Handling

Practice: Proper error handling ensures the program gracefully handles issues like invalid inputs, missing files, or failed subprocess executions.
Why it's secure:

- Prevents crashes and potential exploits caused by unhandled exceptions.

10 Least Privilege

Practice: The code follows the principle of least privilege, where only necessary actions are performed for each phase.
Why it's secure:

- Reduces the attack surface and ensures no unnecessary sensitive operations are exposed.

11 Explicitly Compiled C++ Programs

Practice: Each C++ phase is precompiled into binaries (e.g., `connections.exe`, `phase4.exe`).
Why it's secure:

- Precompiled code minimizes the risk of source code tampering and ensures consistent behavior during execution.

12 GUI Feedback

Practice: The GUI provides clear feedback on the status of each phase (e.g., connection status, password hashing, etc.).
Why it's secure:

- Helps users quickly detect anomalies (e.g., if connections are not established or a phase fails).

13 Avoidance of Hardcoded Secrets

Practice: No secrets (e.g., passwords or keys) are hardcoded in the source code.
Why it's secure:

- Protects against reverse-engineering attacks that aim to extract sensitive information.

Conclusion

This report outlines the **Zero-Knowledge Proof Authentication system** with secure coding practices to prevent vulnerabilities. Key takeaways include:

1. **Zero Knowledge Proof:** Provides robust authentication by verifying knowledge without revealing sensitive data.
2. **Secure Coding Practices:** Protects against common vulnerabilities like weak encryption, buffer overflow, and invalid inputs.
3. **Future Potential:** Integration with modern technologies like blockchain can further enhance ZKP's applications.

By adhering to secure coding principles, the ZKP authentication system ensures strong security while maintaining flexibility for future advancements.