CS4455 2025 2nd Year Epic Project

version 1	**19th May 2025**	**Initial release**

Project Specification

Summary

In this project, students are tasked with the design and implementation of a secure, end-to-end encrypted file sharing platform that guarantees confidentiality, integrity, and authenticity of data even in the event of the server being compromised.

Students will build one or more desktop clients that connect to a common back-end server. At a minimum, a client must be created that uses C++ and Qt should use appropriate libraries to do cryptography and secure connectivity.

A second optional client can be created using HTML and Javascript and should use the web crypto API and web security elements. Clients should support the following operations:

- User sign-up, login and password management
- Viewing a list of owned and shared files
- Uploading a file
- Sharing a file with another user after verifying their identity
- Revoking a user's access to a previously shared file
- Downloading a file (owned or shared)
- Deleting an uploaded file

Students will build a server application to handle authentication and to provide an API for the clients to interact with. The back-end can be developed in any language. NodeJS, python or other can be used.

Marks breakdown

Marks will be awarded as follows:

C++	20%
Networking	20%
Cryptography	20%
Ethical hacking	20%
Teamwork and innovation	10%
Commercialisation (ecosystem minor)	10%

Team Names and Members

ID	Surname	First name	user	group
23387076	Kazimierek	Alisia	alisia	chrispp
23391707	Borko	Mia	mia	chrispp
23373326	Olajitan	Oluwajomiloju	jj	chrispp
23381574	ODowd	Ruan	ruan	chrispp
23384549	Jakubowski	Dawid	dawid	fourohfour
23371323	Nelson	Ellice	ellice	fourohfour
23303972	Carson	Jean	jean	fourohfour
23365412	Abidoye	Kelly	kelly	fourohfour
23397179	Phelan	Edmund	edmund	gobblergang
23374039	Doherty	James	jameso	gobblergang
23370858	Kelly	James	jamesk	gobblergang
23382732	Glackin	Ruairi	ruairi	gobblergang
23382163	Jaffray	Andrew	andrew	leftovers
23368276	Frizzell	Oisin	oisinf	leftovers
23372176	Walsh	Patrick	thepa	leftovers
23383372	BowenMac	Tola	tola	leftovers
23369914	Heaphy	Dara	dara	nrmc
23379243	Haq	Naem	naem	nrmc
23365528	Scully	Tiernan	tiernan	nrmc
23363258	Deane	Tomas	tomas	nrmc
23370157	Moody	Daniel	daniel	nrmc
23359285	Roche	Aaron	aaron	packetsniffers
23303816	Przyborska	Justyna	justyna	packetsniffers
23363959	Murphy	Patrick	patrickm	packetsniffers
23367857	Joyce	Thomas	thomas	packetsniffers
23365498	Teehan	Dylan	dylan	rudebhoys
23361433	Sheppard	Frederick	fred	rudebhoys
23369698	Sloggett	Josh	sloggo	rudebhoys
23371455	Somers	Niall	niall	rudebhoys
23362308	Browne	Conor	conor	sql
23366303	Shannon	Cormac	cormac	sql
23357614	Casey	Jack	jack	sql
23362073	Wijayaratne	Ushen	ushen	sql
23359048	Luby	Darragh	darragh	networkninjas
23233761	Vaz	MichelleAndrea	michelle	networkninjas
23070854	Vinayakrishnan	Nandakishore	nanda	networkninjas

Project Server

A cloud server is available for teams to run cloud backends. The server name is GOBBLER.INFO. Each team has their own virtual host (e.g. SQL.GOBBLER.INFO) and group name. Standard development tools have been installed and more can be added as required. MySQL database server is installed and is recommended for persistent data storage.

The server should have adequate resources to support all teams but students are asked to ensure they do not use more than 2GiB of storage in total. If there are any bottlenecks with respect to RAM, CPU or disk then the server spec can be increased but this does add some cost.

Submissions and interviews

Project tracking

A brief weekly status report per team must be submitted to Mark Burkley by 5pm on friday 23rd and friday 30th May. This report should include accomplishments, highlights and lowlights for the week. This report will form part of the teamwork and innovation marks (10%) for the project.

Project submission

Upload a zipped archive of your project's GitHub repository to Brightspace. This archive should include:

- All source code for the project
- A README file with clear instructions on how to install any dependencies, set up the project, and run it

Your submission must also include a cover document (PDF or Markdown) alongside the zipped archive containing the following:

- Group name and project URL
- The full name and student ID of each group member
- The URL of the GitHub repository used for the project
- A breakdown of each member's contributions, including:
    - An estimated percentage of the overall work completed by each person

    - The specific features, components, or tasks each member worked on
- Any additional design summaries, diagrams or explanations requested by the topic-specific requirements below

Submissions must also include a short presentation which describes and justifies the design and implementation of all cryptographic components used in the system. A diagram illustrating the cryptographic architecture is optional but encouraged.

Incomplete submissions or those missing any of the required elements above may be penalized. Late submissions may be subject to penalties.

While this is a group project, grades are awarded individually. Lecturers reserve the right to adjust individual grades up or down based on each student's contribution to the project and their demonstrated understanding of it.

Presentation and Interview

Teams will be given an opportunity to present and demonstrate their project at the end of the epic which will be followed by a short panel interview. During the interview, students will be expected to clearly explain, justify, and defend the cryptographic design decisions made in their submission. Students must demonstrate a clear understanding of what they've implemented and submitted. Failure to adequately explain the solution, regardless of whether it's technically correct or not, will result in a loss of marks.

Interviews will take place on friday 6th June between 10am and 5pm. Interviews will be strictly limited to 30 minutes per team. Each team will be given time to present and demonstrate their projects (maximum 10 minutes) followed by questioning by the interview panel (maximum 20 minutes). Every student is expected to understand all the code in the project even if they didn't author it themselves. Marks will be awarded on an Individual basis (not every member of the team will necessarily get the same grade).

Major subject requirements

Students are required to demonstrate learnings from each major subject in this epic. This section lists requirements from each of the major topics.

Network security project requirements

The front end programs should create SSL protected connections to the virtual host. A back-end service running on the server should accept requests and process them. Part of the back-end may be written in C++ with html parsing in NodeJS or python or the entire back-end can be written in one of those languages.

All communication between client and server should be protected by SSL/TLS. Server certificates and identity should be verified. Teams should demonstrate their ability to write code that resolves host names and establishes secure connections. Using low level socket calls with libcrypto and libssl libraries would be impressive but if time is tight then using a library such as libcurl is acceptable.

Cryptography Requirements

The system must guarantee confidentiality, integrity, and authenticity of all shared files using end-to-end encryption. These requirements assume an adversary with full access to the communication network and server infrastructure, but not the client devices. The following security guarantees must be met:

File Encryption

- All files must be encrypted on the client side before uploading using a modern authenticated encryption scheme (e.g., AES-GCM, ChaCha20-Poly1305) with IND-CCA2 security.
- Each file must be encrypted with a fresh encryption key. These keys must be securely generated, stored, and deleted on the client side, using secure mechanisms (e.g., encrypted local storage, OS-provided keychains, or hardware-backed key stores). Knowledge of how to decrypt one file should not allow a user or adversary to decrypt other files.
- The encryption scheme must ensure that adversaries cannot modify the ciphertexts or associated metadata (e.g., filenames, MIME types) in transit or while stored on the server without detection.

Secure File Sharing

- Files must remain encrypted from the sender to the recipient. Neither the server nor any intermediary should be able to decrypt the files in transit or at rest.
- The system must employ cryptographic authentication mechanisms (e.g., digital signatures, certificate-based identity binding) to prevent impersonation of senders or recipients, even if an adversary has access to the server. When sharing a file, the sender and recipient must be able to verify one another's identity.
- When downloading a file, the user must be able to verify that the file has not been modified in transit or in storage.

Password and Key Management

- Passwords used for authentication should be protected from an adversary even if the application or database is compromised. Password strength controls should follow industry standards and best practice (e.g., NIST SP800-63B or similar).
- Users must be able to change their password without losing access to their encrypted files. Authentication credentials should be separate from encryption credentials (e.g., master keys or master passwords).
- Keys must be generated or derived appropriately. If keys are derived from passwords, appropriate key derivation functions must be used.
- Nonces and IVs must be generated securely and used appropriately in accordance with cryptographic best practice. Avoid insecure scenarios (e.g., nonce reuse with ChaCha20-Poly1305, predictable IVs with AES-CBC).

Implementation

- All cryptographic components must provide a minimum of 128 bits of effective security against cryptographic attacks.
- Cryptographic primitives must be sourced from reputable, secure libraries (e.g., OpenSSL, libsodium, the Web Crypto API, or similar).
- The selection of algorithms must be justified based on current, widely accepted cryptographic standards. Avoid deprecated and known-vulnerable algorithms (e.g., RC4, DES, Dual_EC_DRBG, and MD5).
- The selection of parameters for chosen algorithms must be justified with reference to modern standards and best practice. Avoid weak or known-vulnerable configurations for sensitive parameters (e.g., AES-GCM IV length).

C++ Requirements

Essential Requirements for Submitted C++ Code:

Students must illustrate and explain if required their understanding and application of the following C++ concepts within their submitted code:

1. Functions:

- **Function Declaration and Definition:** Demonstrate the ability to declare function prototypes and provide their complete implementations.
- **Call by Value vs. Call by Reference:** Clearly differentiate between passing arguments by value (creating copies) and by reference (allowing modification of the original). Provide examples of both techniques and justify their use.
- **Function Overloading:** Implement multiple functions with the same name but different parameter lists (in terms of number or types of parameters) to perform related operations.
- **Inline Functions and Default Arguments:** Appropriately use the inline keyword for suitable small functions to potentially improve performance. Demonstrate the use of default argument values in function definitions.

2. Object-Oriented Programming (OOP):

- **Classes and Objects:** Define classes with data members (attributes) and member functions (methods). Create and utilize objects (instances) of these classes.
- **Access Specifiers:** Implement and correctly apply private, public, and protected access specifiers to control the visibility and accessibility of class members.
- **Constructors and Destructors:** Define and implement constructors (including parameterized constructors) to initialize object state upon creation. Implement a destructor to perform cleanup operations when an object is destroyed.
- **Copy Constructor and Assignment Operator:** Demonstrate the creation and use of a copy constructor to handle object initialization from another object of the same class. Implement or explicitly prevent the use of the assignment operator (operator=) to control object assignment behavior.
- **this Pointer:** Illustrate the use of the this pointer to refer to the current object within a class's member functions.

3. Inheritance:

- **Types of Inheritance:** Implement and demonstrate at least one form of inheritance (single, multiple) by creating derived classes from base classes. Clearly show the relationship between the classes.

- **Use of protected Members in Inheritance:** Demonstrate how protected members of a base class are accessible to derived classes.
- **Constructor and Destructor Behavior in Inheritance:** Illustrate how constructors and destructors of base and derived classes are invoked during object creation and destruction in an inheritance hierarchy.
- **Function Overriding and Base Class Pointers:** Override virtual functions in derived classes to provide specialized implementations. Demonstrate the use of base class pointers (or references) to invoke the appropriate (overridden) function based on the actual object type at runtime.

4. Polymorphism:

- **Compile-time (Function and Operator Overloading):** Illustrate the concept of function overloading. Additionally, demonstrate operator overloading to define custom behavior for operators when used with user-defined types (objects).
- **Run-time (Virtual Functions and Dynamic Dispatch):** Implement virtual functions in base classes and override them in derived classes to achieve runtime polymorphism. Show how the correct version of the virtual function is called through base class pointers or references.
- **Pure Virtual Functions and Abstract Classes:** Define at least one pure virtual function within a base class, making it an abstract class. Demonstrate the inability to instantiate abstract classes and the requirement for derived classes to provide implementations for pure virtual functions.
- **Virtual Destructors:** Implement virtual destructors in base classes within inheritance hierarchies to ensure proper cleanup of derived class objects when deleted through base class pointers.

5. Memory Management and Pointers:

- **Pointers: Declaration, Initialization, and Dereferencing:** Demonstrate the correct syntax for declaring pointer variables of different data types. Show how to initialize pointers with the address of variables using the address-of operator (&) and how to access the value pointed to by a pointer using the dereference operator (*).
- **Pointer Arithmetic:** Illustrate basic pointer arithmetic (incrementing, decrementing pointers) where applicable, understanding its behavior in relation to the size of the data type being pointed to.
- **Pointers and Arrays:** Demonstrate the close relationship between pointers and arrays, showing how array names can be treated as pointers and how pointer arithmetic can be used to access array elements.

- **Dynamic Memory Allocation using new and delete:** Use the new operator to allocate memory dynamically for single objects and arrays, and the delete (or delete[] for arrays) operator to deallocate that memory, preventing memory leaks.
- **Function Pointers:**
    - **Declaration and Usage:** Declare and initialize function pointers to point to functions with specific signatures (return type and parameter list).
    - **Passing Functions as Arguments:** Demonstrate how function pointers can be passed as arguments to other functions (e.g., for implementing callbacks or strategy patterns).
    - **Returning Functions from Functions (Less Common, but Illustrative):** how how functions can return function pointers.
- **Smart Pointers:**
    - **std::unique_ptr:** Demonstrate the creation and use of std::unique_ptr for exclusive ownership of dynamically allocated memory. Show how std::move is used to transfer ownership.
    - **std::shared_ptr:** Demonstrate the creation and use of std::shared_ptr for shared ownership of dynamically allocated memory, highlighting the concept of reference counting.

6. Templates:

- **Function Templates:** Write generic functions using templates that can operate on different data types without the need for explicit overloading for each type.
- **Class Templates:** Define generic classes using templates where the data type of members is parameterized, allowing the creation of type-safe containers or algorithms.

Ethical Hacking Requirements

The implemented solution must be tested and ensure protection against undesired side effects. Students must demonstrate that they have implemented controls and have actively checked for issues around:

- Improper Input Validation
- Broken Authentication
- Broken Access Control
- Cryptographic Issues
- Injection
- Security Misconfiguration

- Sensitive Data Exposure
- Vulnerable Components

Both front end and backend services are expected to be resilient against common vulnerabilities and exploits. Students are encouraged to **include a penetration testing report** in their submission detailing penetration tests executed and their findings.

Enterprise Ecosystems & Funding

Gearoid Kearney (RDI Hub) will lead the Enterprise Ecosystems & Funding minor, which will explore the funding and commercialisation landscape for software ventures. The minor topic will be delivered in person on Wednesday, the 21st, and Thursday, the 28th, within ISE.

Teams will be expected to present an investor pitch and roadmap to commercialise their concept. Teams can present on a future iteration of their concept with additional features (i.e. they do not have to build a full commercial concept).

The Enterprise Ecosystems and Funding element will account for 10% of the EPIC. Presentations will be held on Wednesday, June 4th.

**Note: Attendance at masterclass sessions (21/28th) is highly important. If a team's attendance at these sessions falls below 60%, the opportunity to present on the 4th will be forfeited.*