

# Project: Dropbox

**Design Document Due:** Friday, April 18, 2025 @ 11:59pm EDT (No late days on this part!)

**Implementation Due:** Wednesday, May 7, 2025 @ 11:59pm EDT

**Warning:** This is a team project. If you have not done so already, please fill out the partner assignment form to create your team or ask to be matched to a team, which is available here:

<https://forms.gle/JJtN4RDb17kaLB6TA>

**You must fill out the form by Sunday, March 30 by 11:59pm EDT.** If you do not fill out the form, we cannot match you to a team, which is required!

On Tuesday, April 8, **you will receive a link to clone the stencil repository for your team with this email.** An announcement will be posted on Ed when you should have received it.

<b>1 Specification</b>	<b>2</b>	<b>2.1 Design document . . . . .</b>	<b>3</b>
1.1 Motivation . . . . .	2	2.2 Grading . . . . .	5
1.2 Assignment . . . . .	2	2.3 Submitting your work . . . . .	5
1.2.1 CS1620/CS2660 Requirement	3		
<b>I Design</b>	<b>3</b>	<b>II Implementation</b>	<b>5</b>
<b>2 Assignment</b>	<b>3</b>	2.4 Testing . . . . .	5
		2.5 Grading . . . . .	5
		2.6 Handing In . . . . .	6

## 0 Introduction

After you broke into almost every single technical system at Blue University, the Information Technology office figured that the only way to prevent vulnerabilities in their newest project was to hire you as the lead developer.

In this project, you will implement an API for a secure file storage system, with a focus on creating a secure design. This project will give you experience not only with writing secure software, but equally importantly, experience with *designing*, *planning*, and *critically analyzing* secure software—carefully thinking, threat modeling, and critiquing your own design before you write any code.

This project consists of two components: Part I, where teams will write a detailed design plan for their system and Part II, where you will build your implementation in Python.

### 0.1 Logistics

This is a team project designed to be completed in teams of 2 students. Before the start of the project, you may form a team with someone you know or asked to be matched to a team—we will send out a form on Ed with instructions. Note that both partners are required for fill out the form, only mutual requests will be considered.

**CS1620/CS2660 requirement** CS1620/CS2660 students are required to implement one additional component in their design with additional requirements on **either** performance or sharing functionality (you pick which one).

Since this requirement affects the overall system design, any team with at least one CS1620/CS2660 student must implement the additional CS1620/CS2660 requirement. For more information, see Section 1.2.1.

**Late days** This project has two deadlines: an initial “checkpoint” submission, where you will submit a design document describing your system design, and final submission with your implementation and writeup. After you submit your design document, you will set up a meeting with a TA to discuss your design and plan for implementation. Since this review is time-sensitive with the rest of your implementation, **late days cannot be used for the design document.**

For the final submission, you will submit your completed implementation and an updated design document based on your final design. For this part, late submissions use late days for each team member—that is, submitting one day late separately counts as one late day used by each team member.

# 1 Specification

**Note:** The section below is a quick introduction to the project, but the majority of the specification can be found at the CS1660 Dropbox Wiki at <https://brown-csci1660.github.io/dropbox-wiki>. The online documentation is the canonical source of information on this project, so make sure you read the wiki to learn about the main requirements.

## 1.1 Motivation

When you’re developing software, you naturally trust the systems that you are using—whether it’s your computer, the campus network, the cloud services we use every day—to not act *maliciously*. For instance, when you save files on your local computer or upload a file to, say, Google Drive, you trust that there aren’t any attackers on your machine, that your hard drive won’t start flipping random bits, and so on.

In the real world, these kinds of trust assumptions may not be reasonable or valid. In industry, where companies delegate computation to third-party resources all the time, trusting in one of those companies implies trust in the third-party resources. However, if you’re dealing with particularly sensitive information, it may be unreasonable to keep expanding your circle of trust.

One solution is to simply avoid outsourcing any resources and maintain physical and technical control over every system you use. However, this may not be cost-efficient or practical. For example, it may not be feasible for a company to maintain physical control over a secured data center, and so that company might outsource their cloud storage to a third-party provider. However, it may be feasible for that company to access a small, secure amount of trusted storage space, and somehow devise a way to combine both types of trusted and untrusted storage to create an overall secure system. In this project, you will architect a secure system that explores how to do this.

## 1.2 Assignment

You will write a client for a file storage service. The client must implement eight operations—`CreateUser`, `AuthenticateUser`, `UploadFile`, `DownloadFile`, `AppendFile`, `ShareFile`, `ReceiveFile`, `RevokeFile`.

Users of the client will provide their username and password to authenticate themselves to the service. Once authenticated, users will use the client to upload new files (and identify them with a filename chosen by the user) and download previously uploaded files from the server. Users may also modify the contents of their files by uploading a file with the same name as another file previously uploaded. Users will also use the client to *share* files with other users (as well as revoke permissions from previously shared files) and download files shared to them by other users.

To implement this functionality, your client will have access to two servers: `dataserver`, which is an *untrusted* data storage server which can store arbitrary data; and `keyserver`, which is a *trusted* public key server.

Using only these two servers, you will implement the functions above in such a way that your client ensures *confidentiality* and *integrity* of files that are stored on the server.

Finally, users are not guaranteed to be online between invocations of calls (nor are they guaranteed to be using the same machine!), and thus your client must be *stateless*. This means that your client may not rely on local storage or global variables to provide its security guarantees—if the client is restarted, it must be able to pick up where it left off given only a username and password.

### 1.2.1 CS1620/CS2660 Requirement

CS1620/CS2660 students must implement **one** of the following extra requirements:

- **Efficient file updates:** This adds a performance requirement to `UploadFile` to replace components of files more efficiently. This requirement is detailed in this section of the wiki. For further information, also see the notes/recording for the project gearup.
- **Delegated sharing:** Students who want an extra challenge can implement a more advanced version of file sharing where users inherit the ability to share file from the owner. This component is more open-ended—for details, see the description in this section of the wiki.

In general, your implementation for these requirements is more open-ended than the rest of the project. As you build your design, your writeup should describe in detail how your implementation meets the requirements while preserving confidentiality and integrity, and discuss any tradeoffs you considered as you were developing your design. We will grade your design mostly based on manual review of your code and your design document.

## Part I

# Design

## 2 Assignment

To begin your project, you will plan out your client implementation and submit a detailed design document. This project is more open-ended than our other projects so you have a chance to do some critical thinking about designing secure systems. In terms of learning goals for this course, we're not just interested in your ability to pick a strong hash function or avoid path escaping vulnerabilities—we want you to think about how you'll store data and use the cryptographic tools available to you in order to build a secure system.

In addition, once you submit your design document, you will meet with a TA to discuss your work before you begin your implementation. Spending time on your design now will make your implementation easier later!

**Collaboration** We encourage you to do lots of brainstorming with your partner, and consider many possible designs. At all stages of your project, we encourage you to talk with other teams about design tradeoffs and possible attacks—so long your design and all the code you write is your own.

### 2.1 Design document

Your design document should contain two components: an overview of how your system works, and a more concrete security analysis that shows how your system prevents certain attacks.

While there is no official length requirement, a complete design is typically on the order of 4 pages (+ any number of pages for diagrams). You can use any format you like (paragraphs, bullets, etc.), so long as your design includes the components listed here in a way we can understand.

**System design** Your system overview should explain your overall system and the design decisions you made such that a fellow CS1660/CS2660 student could implement your design just by reading your design document. The goal is to be concise, while providing enough detail to understand how your implementation works.

At minimum, your design should address the following:

- How does your design create/authenticate users?
- How are files stored on the data server? How does your design allow `AppendFile` to meet its efficiency requirements?
- How are files are shared with other users?
- How can shared files revoked, and how does it make sure a revoked user can no longer use the file?
- (CS1620/CS2660 only) How does your design allow implement efficient updates or delegated sharing?

When describing your design, you should focus specifically on how you will implement the operations in the Client API, as described in the wiki. You do not need to consider any networking-related components (such as how file data is transferred from client-to-server, or how to store files in a real filesystem or database)—while these components would be important in a real system, they are not part of the version we are building here.

**Security analysis** To give you practice thinking about *how* your system is secure, you should describe **four** possible attacks that an adversary might attempt and how your design protects against the attack. For your initial design document, we will grade your security analysis only on completion: you should make an attempt to help you work out potential issues with your design, and then you will submit a more concrete version with your final design at the end of the project.

Your attacks should describe a concrete set of steps that an adversary may take based on our Client API and security definitions. For example:

*After user A shares a file with user B and later revokes access, user B may try to call `ReceiveFile` again to regain access to the file. To prevent a revoked user from regaining access this way, our design...*

To describe how your design prevents an attack, you should reference specific components or your design. For example:

*The adversary (eg. Eve) cannot decrypt data structure X because it is encrypted with <name of some key referenced in your design>, which can only be obtained by Alice<sup>1</sup>.*

Some more notes on how your security analysis should work:

- The attacks you describe must have a security consequence and cannot simply be a bug. That is, your proposed attacks must result in an attacker breaking confidentiality or integrity guarantees or executing an unprivileged action.
- In your final submission, you should write tests to check if your implementation is vulnerable to the attack you describe, if possible. If you don't believe this is feasible, please describe what *would* be required to test for this attack in your document.

---

<sup>1</sup>When we read your document, it must be clear to us *why* Alice is the only user that can obtain this key (either from your security analysis or the design overview).

## 2.2 Grading

You will submit your design document twice. The initial draft of your design document is due at the Design due date (**Friday, April 18, 2025 @ 11:59pm EDT**) and will be graded primarily on completion for each major component and will count for 5% of the total grade for the project. After submitting your design document, we will provide instructions on how to sign up for a meeting with the TAs to discuss your design and receive feedback. Since we have a limited time to schedule your meetings, **no late days may be used on your initial design document submission.**

For your final submission (**Wednesday, May 7, 2025 @ 11:59pm EDT**), you *must* submit an updated version of your Design document that reflects any changes to your implementation design. This version of the document should reflect your final design and discussion about your security analysis. Your final writeup and analysis is worth 35% of your grade for the project.

## 2.3 Submitting your work

You should hand in your design document as a PDF on Gradescope. Only **one** partner needs to submit a PDF, then you can use the team assignment dropdown in Gradescope to select your partner. By default, we will grade only submissions that any listed with both team members—if you have any issues with this process, please let us know.

At the top of all of the pages of your document (i.e. in the header), you should *clearly mark whether or not the team includes at least one CS1620/CS2660 student* (which means you must complete the CS1620/CS2660 requirements for the design document).

## Part II

# Implementation

After submitting your design document, you will implement your client design in Python. The CS1660 Dropbox Wiki details most of the technical requirements for this part of the project, but here we detail some logistics about the implementation.

**Note:** Looking for a stencil link? You will receive a link in the email confirming your team assignment. An announcement will be posted on Ed when you should have received it.

## 2.4 Testing

We strongly encourage you to write unit tests for your client application. Your tests should verify correct functionality of the client, correct handling of erroneous inputs, and any security problems. Each test should be defined in a separate function.

We have provided a suite of functionality tests in the file `test_client.py`, which comprises most of the tests run by our autograder. We strongly encourage you to write more tests to check the functionality of your program and to test out possible attacks. We'll provide more guidance on how to test your work during the Gearup.

## 2.5 Grading

Your final client implementation and tests are worth approximately 60% of the final grade for this project. Your implementation will be manually reviewed for functionality in combination with any autograder tests

and your design document. The exact weighting of the autograder tests is subject to change, and will be supplemented with manual review of your code.

## 2.6 Handing In

You will hand in your implementation code and testing code on Gradescope. Only **one** member of your team needs to submit—after submitting, you can add your partner to your submission. We will provide more details on how to submit your work as we get closer to the deadline.