Matthew King, Joe Canero, Gary Patricelli, Dan Thielke, Ray Clyne
Dr. Monisha Pulimood
CSC 260-02
17 October 2012

Collaborative Project Stage II: Requirements and Analysis

Security and Encryption:

Encryption

In C++, the logical exclusive or (XOR) is a useful operator for performing encryption and decryption. The operator is the circumflex ^ and is performed on bytes.
This is useful because the program can store an array of characters to be subjected to the XOR operation. The result of the logical XOR will be stored and used for when the encrypted array needs to be decrypted. The XOR is a simple, and relatively inexpensive, way to encrypt where the data can be easily decrypted.

Encrypting XOR example:

The char 's' will be encrypted once using 'x' as the encryption key.

Char 's' (int value 115)	01110011
Char 'x' (int value 120)	01111000 XOR
	00001011 = 11

Decrypting XOR example:

The byte 00001011 will be subjected to XOR on the same character, x.

00001011
01111000 XOR
01110011 = 115 = char 's'

It does not take much to make this algorithm more complex: simple mathematical operations can serve to obfuscate the original identity of the to-be-encrypted number even more. Another option is to use more logical operators; one must be careful about their use: to decrypt properly, the operators should be used in the reverse sequence.

Backup and Recovery:

Most software systems, especially a password manager, must have files outside the systems in which they store application data outside working memory, which stores the systems while they run. However, when the user shuts down the systems, memory returns the allocated space for the systems back to the operating system. So, there must be files kept in storage to maintain the data needed to run the system again in the future. Nyan Security's Password Manager will implement a simple, yet robust solution.

On the system's first start-up on a machine, NSPM will prompt the user for a master account username and password. The system will then generate an ASCII text file, with a filename matching the original account username and with the file extension ".nyn." NSPM will store this file in the same OS directory as its executable. Whenever the user enters any online account information into the system while it is running, the system will encrypt the data and write it to this text file. Upon all subsequent initializations of NSPM, it will prompt the user for

his/her account data, it will open the appropriate file in an input stream, and it will load all the user data into memory. Once the user data is loaded in an object-oriented fashion, the system will be able to decrypt and search the data. The user may then initialize any of the use-cases listed elsewhere in this document.

Potential Legal Issues:

In order to avoid dealing with patent/copyright issues with the wrong person/company, the password manager must take on a different aspect from the other managers that have already been patented/released. The following programs are examples to keep in mind during the creation of another password manager that may potentially be released to the public.
KeePass (Windows) – Stores passwords in one database, locked with a master key. The database is free, open source, and encrypted via AES/Twofish.

Roboform (Windows, Mac, online) – One of the earliest password managers available, fills web forms with login info and other relevant information. Allows for password synchronization, password generation, credit card storage and bookmark synchronization. Roboform also has versions available for smart phones and USB devices.

1Password (Mac, Windows) – Originally a Mac-oriented manager, 1Password provides browser plugins to fill in passwords and information for the user. It also has an indicator for password strength, lists of logins, and tags.

Portable password manager (patent) – a software application for login management residing on portable device which can be connected to a computerized terminal, said portable device include memory means, wherein said software application include: means for password managing, monitoring means for identifying login scenarios, interception means for identifying and recording new login data and means for providing login data to login challenges based on prerecorded data stored on said portable device memory.

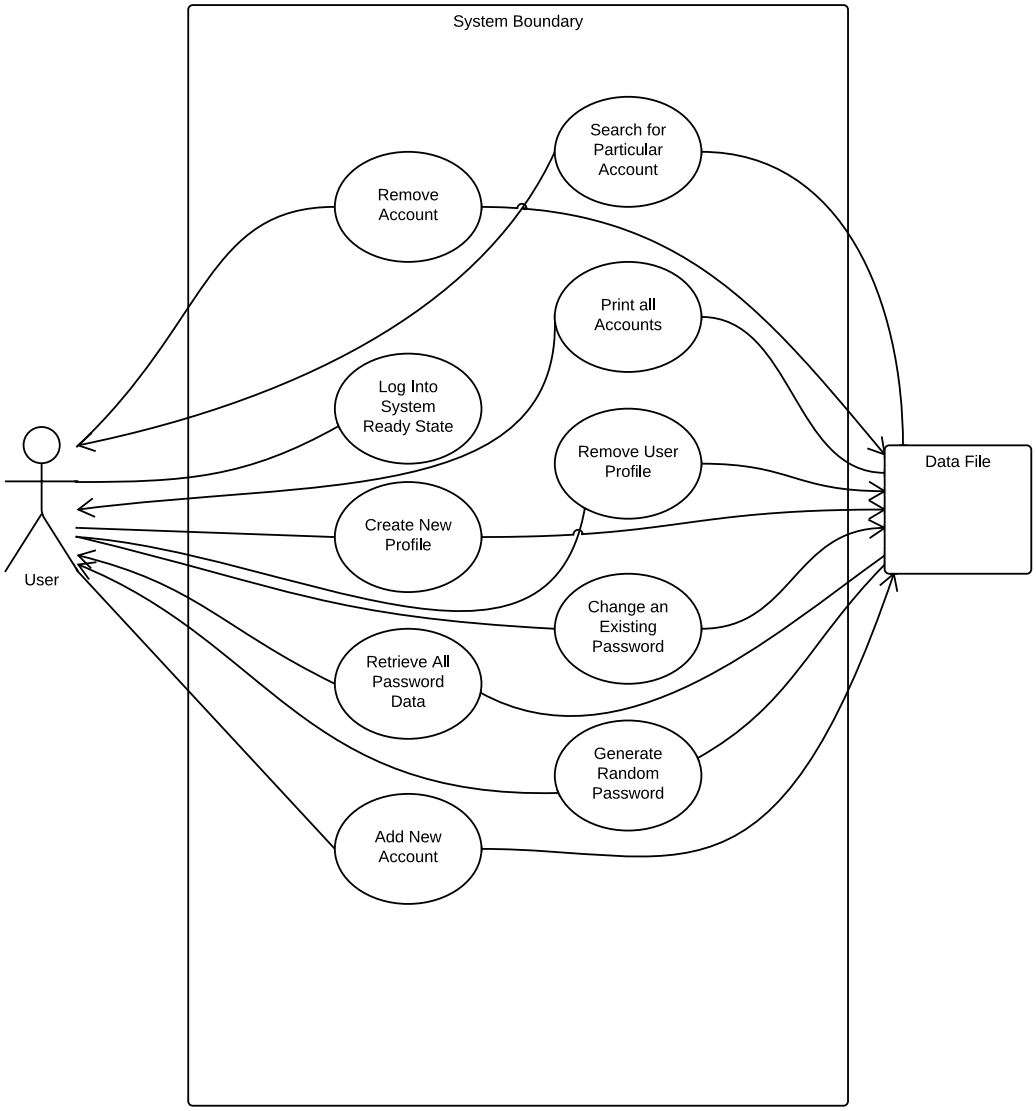
IBM (patent) - A convenient and secure system/method for access to any number of password-protected computer applications, web sites and forms without forcing the user to remember all passwords or circumventing the inherent security of such password-protection schemes. An existing password field on a device display is overlaid with password wallet pop-up field, which allows a wallet "master" key to unlock the wallet. An application-specific and/or user-specific password is automatically retrieved from the wallet and entered into the password field with no other user action required.

Multi-Domain Computer Password Management (patent) – Software including data and computer-executable instructions tangibly encoded on said media, said software interacting with said hardware to provide multiple password-protectable domains including a first domain and a second domain, said first domain having a multi-domain password manager for determining whether a password candidate is valid for said first and second domains, and, if so, submitting said password candidate to said second domain, and, if not, not submitting said password candidate to said second domain.

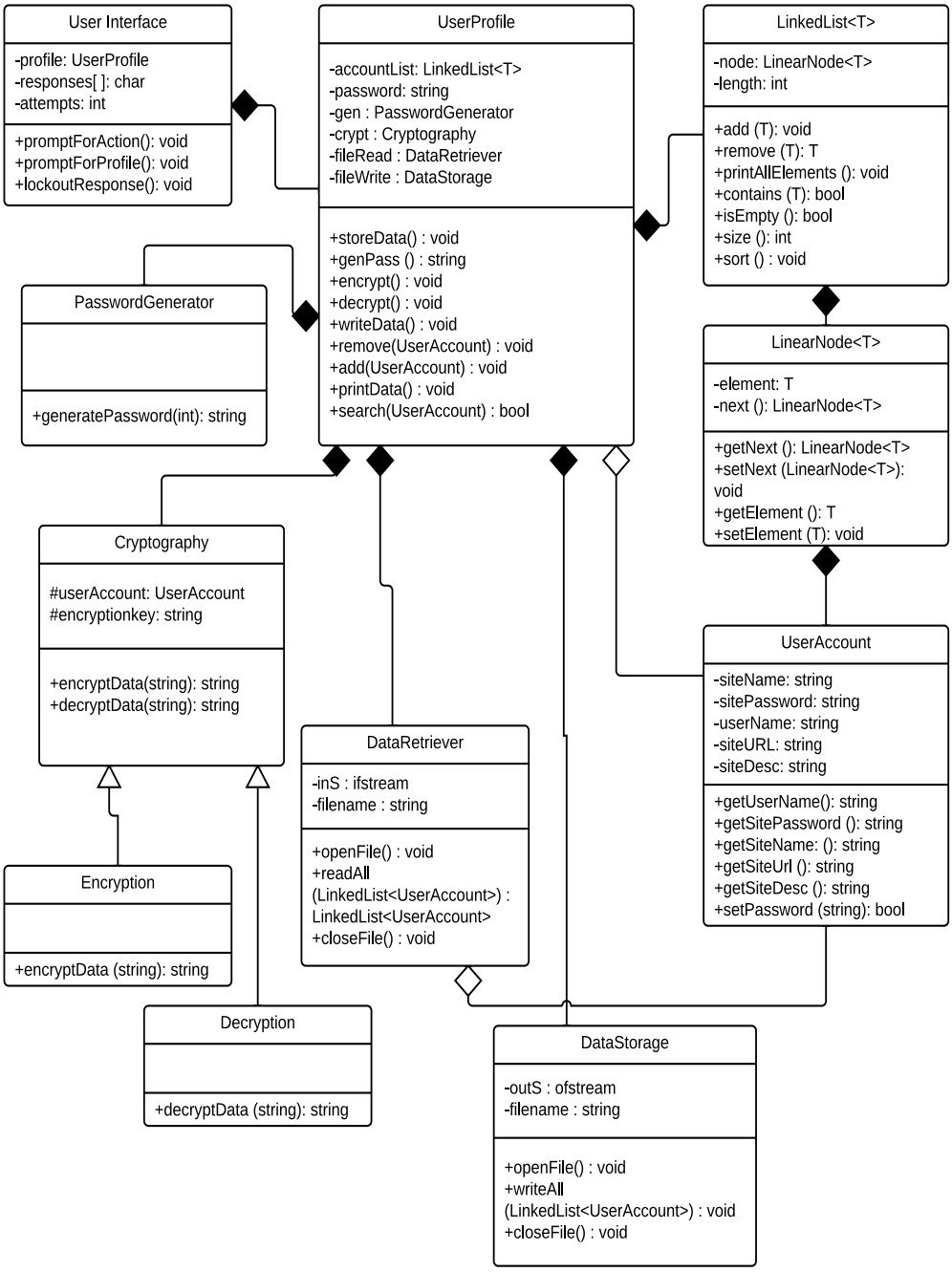
Possible Applications of the System:

Given enough time, the password manager program can have several extra applications that users may utilize. If possible, we could use the NPAPI system to create a browser plugin for our program, allowing users to access our work from Internet applications (e.g. Chrome). Many problems with passwords nowadays are that hackers are attempting to break them and steal other people's information and data. There will be a great use for data wiping in this program, which would discard the original user's data should an unauthorized user attempt to break the master password for the original's accounts. A mobile application for smartphones should also be feasible, allowing for the user to travel without being required to bring the program itself. Another way to help protect the user is to give the option for a multitude of differing encryption algorithms. The user will have the option to choose one among those algorithms in order to protect their password (whichever one they feel is the safest). Last, but not least, the program can be given the ability to mass export or import data, expanding the area of work in which the user can perform personal tasks. For a list of required applications of the system, see the use-case descriptions and diagram given below in this document. The system will run on a command-line interface.

System Use Case Diagram:



UML Analysis Class Diagram:



USE CASE: New User creates new User Profile.

Iteration: 1st

Primary Actor: New User

Goal in context: To efficiently allow the new user to log into the system and set up a new user profile by choosing a master password and username.

Preconditions:
- System is successfully opened.

Trigger: Robust password storage is required so a user decides to try our solution.

Scenario:
1. User opens the system.
2. User chooses to set up a new profile
3. User chooses a username and master password
4. User confirms the user profile

Exceptions:
1. A user profile with the chosen name already exists
2. Username entered is not of desired length
3. Master password is not of desired length

Priority: Extremely high priority; the system is useless without this functionality.

When available: First increment.

Frequency of use: Low to moderate frequency.

Channel to actor: CLI prompt from User Interface class in conjunction with User Profile class

Secondary Actors:
- Data Storage

Channels to secondary actors:
Appropriate interfacing methods.

Open Issues:
1. How will existing User Profiles be determined and checked?

2. What is the minimum length required for a username/master password?

3. Will a combination of digits/characters/symbols be required in the master password?

USE CASE: Existing User desires to retrieve all password data

Iteration: 1st

Primary Actor: Existing User

Goal in context: To allow the existing user to view/print password data for all accounts stored in the system. The data must be returned securely.

Preconditions:
- System is successfully opened.
- Existing User is logged in successfully
- Existing User has password data stored

Trigger: The Existing User decides to retrieve select password data stored in the system.

Scenario:
1. User opens the system.
2. User logs in with appropriate username and master password.
3. User elects to view all password data.
4. User can decide which passwords to receive or view.

Exceptions:
1. Username is incorrect.
2. Master password is incorrect.
3. No password data is stored.

Priority: Very high priority; being able to retrieve what has been stored is an integral aspect of storage, be it of digital or physical objects.

When available: First increment.

Frequency of use: High to very high.

Channel to actor: CLI

Secondary Actors:
- Data Retriever

Channels to secondary actors:
Appropriate interfacing methods.

Open Issues:
1. What information will be included with returned data (e.g. application name, url, etc.)?

2. How will the user be able to cycle through his list of accounts efficiently and knowledgeably?

USE CASE: Existing User logs into System Ready state.

Iteration: 1st

Primary Actor: Existing User

Goal in context: The Existing User is able to log into the system in preparation for processing; that is, the Existing User will have the system on standby for specific needs.

Preconditions:
- System is successfully opened.
- Existing User is logged in successfully

Trigger: The Existing User foresees use of stored passwords.

Scenario:
1. User opens the system.
2. User logs in with appropriate username and master password.

Exceptions:
1. Username is incorrect.
2. Master password is incorrect.

Priority: Very high priority; Existing Users must be able to log in without necessarily initiating processing.

When available: First increment.

Frequency of use: High to very high frequency.

Channel to actor: CLI

Secondary Actors:
- Data Retriever

Channels to secondary actors:
Appropriate interfacing methods

Open Issues:
1. Will there exist a time limit on how long the Existing User can remain in the System Ready state without executing system functionality?
2. How will the length of inactivity be monitored?
3. Is it feasible to incorporate such functionality?

USE CASE: Existing User adds a User Account

Iteration: 1st

Primary Actor: Existing User

Goal in context: The Existing User is able to log into the system with the goal of adding a new User Account. The User Account will store essential application-related data.

Preconditions:
- System is successfully opened.
- Existing User is logged in successfully

Trigger: The Existing User signs up for a new application online or elsewhere and requires storage of password, site name, etc.

Scenario:
1. User opens the system.
2. User logs in with appropriate username and master password.
3. User chooses to add an account.
4. User enters the account name.
5. User decides whether they want to enter a password or have one generated.
6. User saves the account.

Exceptions:
1. Username is incorrect.
2. Master password is incorrect.

Priority: Very high priority—having a password storage application sans capacity for password storage would be pointless.

When available: First increment.

Frequency of use: High to very high frequency.

Channel to actor: CLI

Secondary Actors:
- Data Retriever
- Data Storage
- Encrypt
- Password Generator

Channels to secondary actors:
Appropriate interfacing methods

Open Issues:
1. Will the system check for adding of an existing account?

USE CASE: Existing User removes a User Account

Iteration: 1st

Primary Actor: Existing User

Goal in context: The Existing User is able to log into the system with the goal of removing an existing User Account.

Preconditions:
- System is successfully opened.
- Existing User is logged in successfully.
- The system stores at least one User Account

Trigger: The Existing User terminates subscription to a service that requires a password.

Scenario:
1. User opens the system.
2. User logs in with appropriate username and master password.
3. User chooses to remove an account.
4. User chooses the account that will be removed.
5. User confirms and removes the account.

Exceptions:
1. Username is incorrect.
2. Master password is incorrect.
3. User tries to remove a nonexistent User Account

Priority: High priority.

When available: First-second increment.

Frequency of use: Moderate frequency.

Channel to actor: CLI

Secondary Actors:
- Data Storage

Channels to secondary actors:
Appropriate interfacing methods

Open Issues:
1. Will the system ask if the Existing User is sure he/she wants to remove the User Account?

USE CASE: Existing User removes a User Profile

Iteration: 1st

Primary Actor: Existing User

Goal in context: The Existing User is able to log into the system with the goal of removing his/her User Profile.

Preconditions:

- System is successfully opened.
- Existing User is logged in successfully.

Trigger: The Existing User wants to terminate his/her User Profile within out system.

Scenario:

1. User opens the system.
2. User logs in with appropriate username and master password.
3. User chooses to remove his/her User Profile.
4. User confirms and removes his/her User Profile.

Exceptions:

4. Username is incorrect.
5. Master password is incorrect.
6. User tries to remove a User Profile that isn't theirs.

Priority: Moderate priority.

When available: First-second increment.

Frequency of use: Low frequency.

Channel to actor: CLI

Secondary Actors:

- Data Storage

Channels to secondary actors:
Appropriate interfacing methods

Open Issues:

1. Will the Existing User be required to enter their password before removing their User Profile?

USE CASE: Existing User generates a random password.

Iteration: 1st

Primary Actor: Existing User

Goal in context: The Existing User is able to log into the system with the goal of generating a randomized password.

Preconditions:
- System is successfully opened.
- Existing User is logged in successfully.

Trigger: The Existing User wants to generate a random password for increased security.

Scenario:
1. User opens the system.
2. User logs in with appropriate username and master password.
3. User chooses to generate a random password.
4. Random password is displayed.

Exceptions:
1. Username is incorrect.
2. Master password is incorrect.

Priority: Moderate priority.

When available: First-second increment.

Frequency of use: Moderate frequency.

Channel to actor: CLI

Secondary Actors:
- Password Generator
- Data Storage

Channels to secondary actors:
Appropriate interfacing methods

Open Issues:
1. Will random passwords be of uniform length?
2. Will the Existing User be able to choose parameters for password generation?

USE CASE: Existing User changes a stored password.

Iteration: 1st

Primary Actor: Existing User

Goal in context: The Existing User is able to log into the system with the goal of changing a stored password.

Preconditions:
- System is successfully opened.
- Existing User is logged in successfully.
- Specific User Account is opened successfully.

Trigger: The Existing User decides to change the stored account for a specific application.

Scenario:
1. User opens the system.
2. User logs in with appropriate username and master password.
3. User chooses to change a stored password.
4. User chooses a User Account
5. User chooses to input a password or have one generated.
6. User confirms password change.

Exceptions:
1. Username is incorrect.
2. Master password is incorrect.
3. User Account does not exist.
4. User-input password is incorrect.

Priority: High priority.

When available: First-second increment.

Frequency of use: Moderate to high frequency.

Channel to actor: CLI

Secondary Actors:
- Data Retriever
- Data Storage
- Password Generator

Channels to secondary actors: Appropriate interfacing methods

Open Issues: None as of now.

USE CASE: Existing User decides to print all accounts.

Iteration: 1st

Primary Actor: Existing User

Goal in context: The Existing User is able to log into the system with the goal of printing the data for each stored User Account.

Preconditions:
- System is successfully opened.
- Existing User is logged in successfully.
- At least one User Account exists.

Trigger: The Existing User wants to view the data for all User Accounts so the Existing User can review and make edits outside of the system.

Scenario:
1. User opens the system.
2. User logs in with appropriate username and master password.
3. User chooses to print all stored User Accounts.
4. Accounts can be printed to the screen or to a data file.

Exceptions:
1. Username is incorrect.
2. Master password is incorrect.
3. User Account(s) does not exist.
4. Error opening output file.

Priority: Moderate priority.

When available: First-second increment.

Frequency of use: Moderate frequency.

Channel to actor: CLI

Secondary Actors:
- Data Storage
- Data Retriever

Channels to secondary actors:
Appropriate interfacing methods

Open Issues:
1. Does printing to a file violate the purpose of our password storage system?

USE CASE: Existing User searches for a specific User Account.

Iteration: 1st

Primary Actor: Existing User

Goal in context: The Existing User is able to log into the system with the goal of searching for a specific User Account stored in the system.

Preconditions:
- System is successfully opened.
- Existing User is logged in successfully.
- At least one User Account exists.

Trigger: The Existing User is unable to remember if a User Account(s) exists or would like to view a specific one directly.

Scenario:
1. User opens the system.
2. User logs in with appropriate username and master password.
3. User chooses to search for a specific User Account.
4. User is prompted for the desired search field (e.g. account name, date created, etc.).
5. User enters search terms.
6. Found User Accounts are displayed and able to be accessed.

Exceptions:
1. Username is incorrect.
2. Master password is incorrect.
3. Existing User has no User Accounts

Priority: Moderate priority.

When available: First-second increment.

Frequency of use: Moderate frequency.

Channel to actor: CLI

Secondary Actors:
- Data Retriever

Channels to secondary actors:
Appropriate interfacing methods

Open Issues:
1. What type of searching algorithm will be used?
2. How many search fields should be used?