Project:
Simulated Encrypted File System (SEFS)

Omar Chowdhury

Motivation

- Traditionally files are stored in the disk in plaintext.
- If the disk gets stolen by a perpetrator, he can access all the data in the disk.
- Disk containing sensitive personal information getting stolen by hackers are very common.

A Possible Defense (Encrypted File Systems)

- **Defense**: encrypt the files using some semantically secure encryption scheme.
- No one should be access/change the file's contents without proper credentials.
- An individual with proper credentials should be able to perform all the necessary operations on the encrypted file.
- An encrypted file system (in short, EFS) can support such operations.
- **Example**: Solaris, Windows NT, and Linux support EFS.

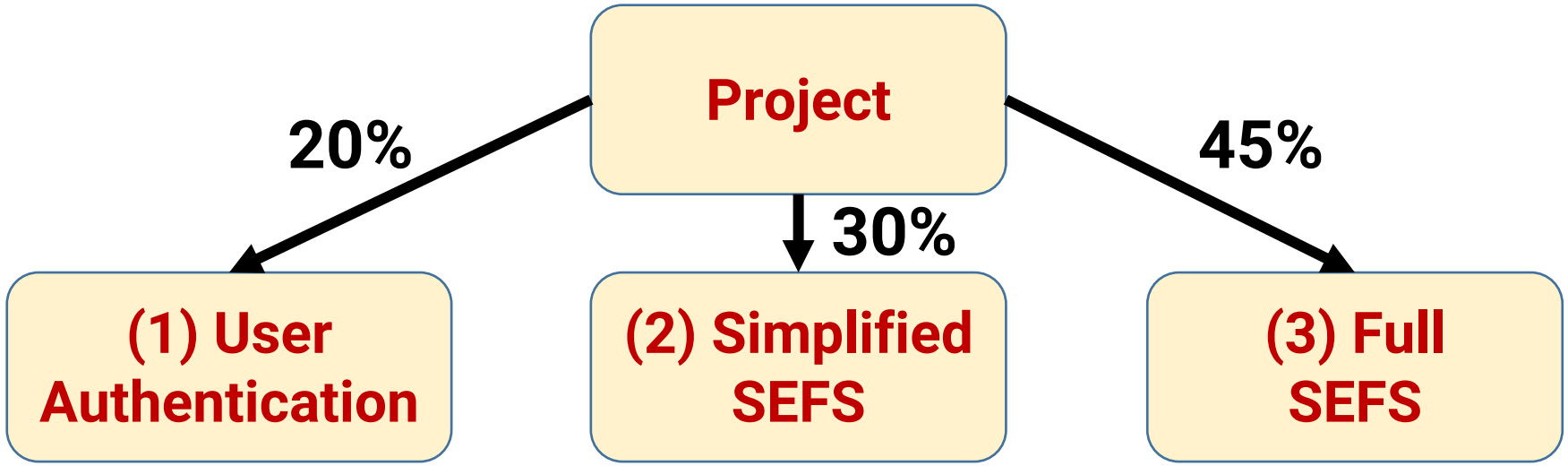
Goal of this Project

- **Goal**: Implement a simulated version of EES
- **Ta**
  D
  us
  ther
- W
- Additionally, we are trying something new this semester. To increase the communication between your classmates we want the projects to be inter-operable.

Communication does not imply copying each other's code ☺

Logistics

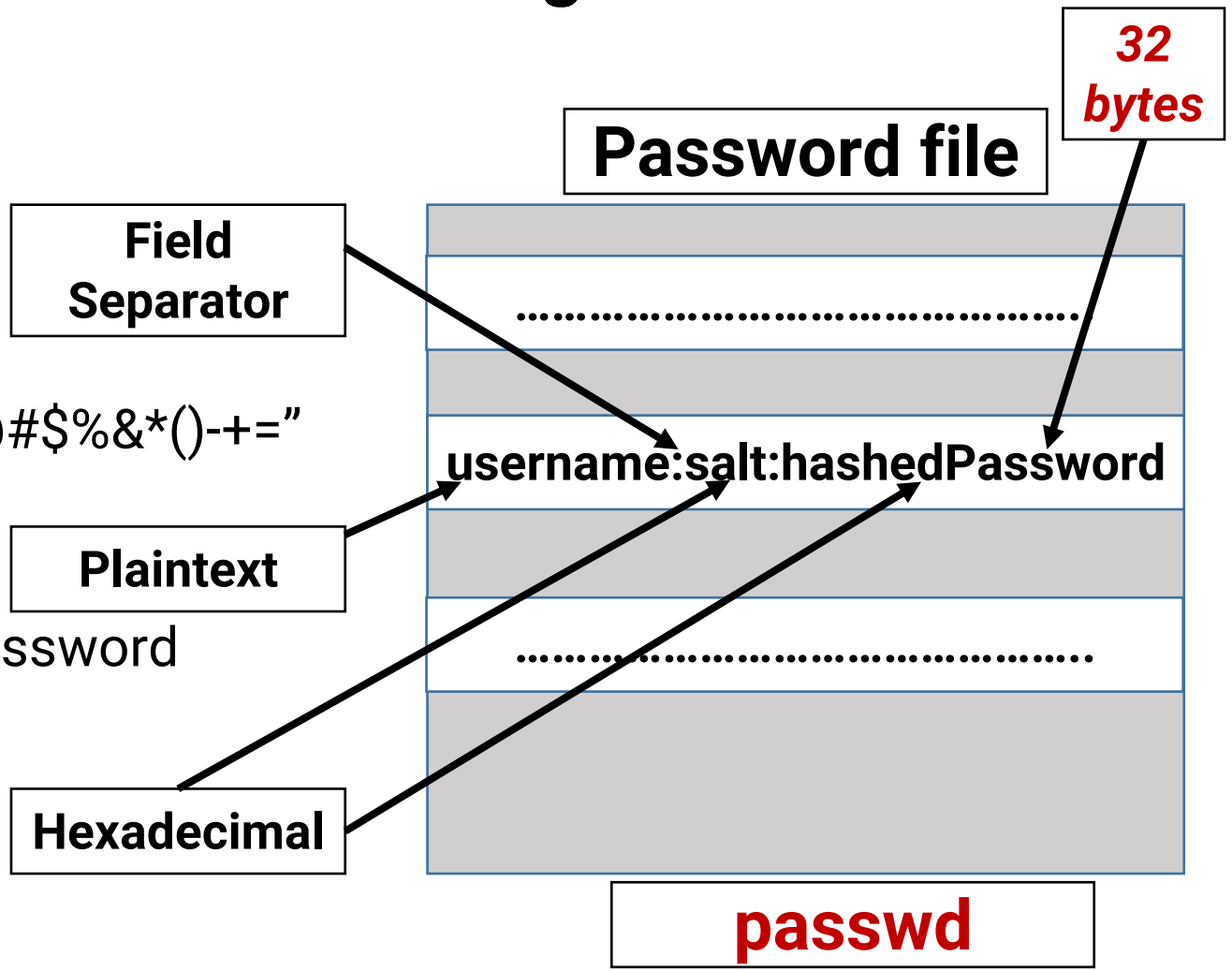
- **Team**: You can work in a team of consisting of (maximum) ***two*** members.



- **Inter-operability**: 5% of the total project points.

Part 1 – User Authentication using Passwords

- Username:
  - Allowed characters: “a-zA-Z0-9”
  - Length: >5 and <32
- Password:
  - Allowed characters: “a-zA-Z0-9@#$%&*()-+=”
  - Length: >8 and <32
- Salt:
  - Randomly generated for each password
  - Length 32 bytes
- Hashing algorithm:
  - **PKCS5_PBKDF2_HMAC_SHA1**



Part 1 – Functionalities

Password file

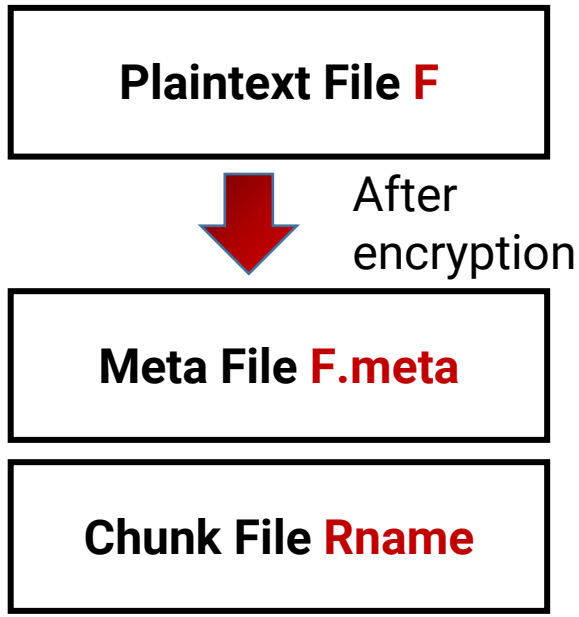
Functions developed in this part of the project for checking user authentication will be used in the next two parts of the project.

change_user_password(u,p,p_n,pFile)

passwd

Part 2 – Simplified SEFS

- Simplified SEFS
  - Master key: Randomly generated, 128 bit
  - Master IV: Randomly generated, **128 bit**
- A sample master key file will be given to you which contains the **binary representation** of a key and IV.
- A sample key and IV loading program is given to you.
- A sample random key and IV generator program is given to you.

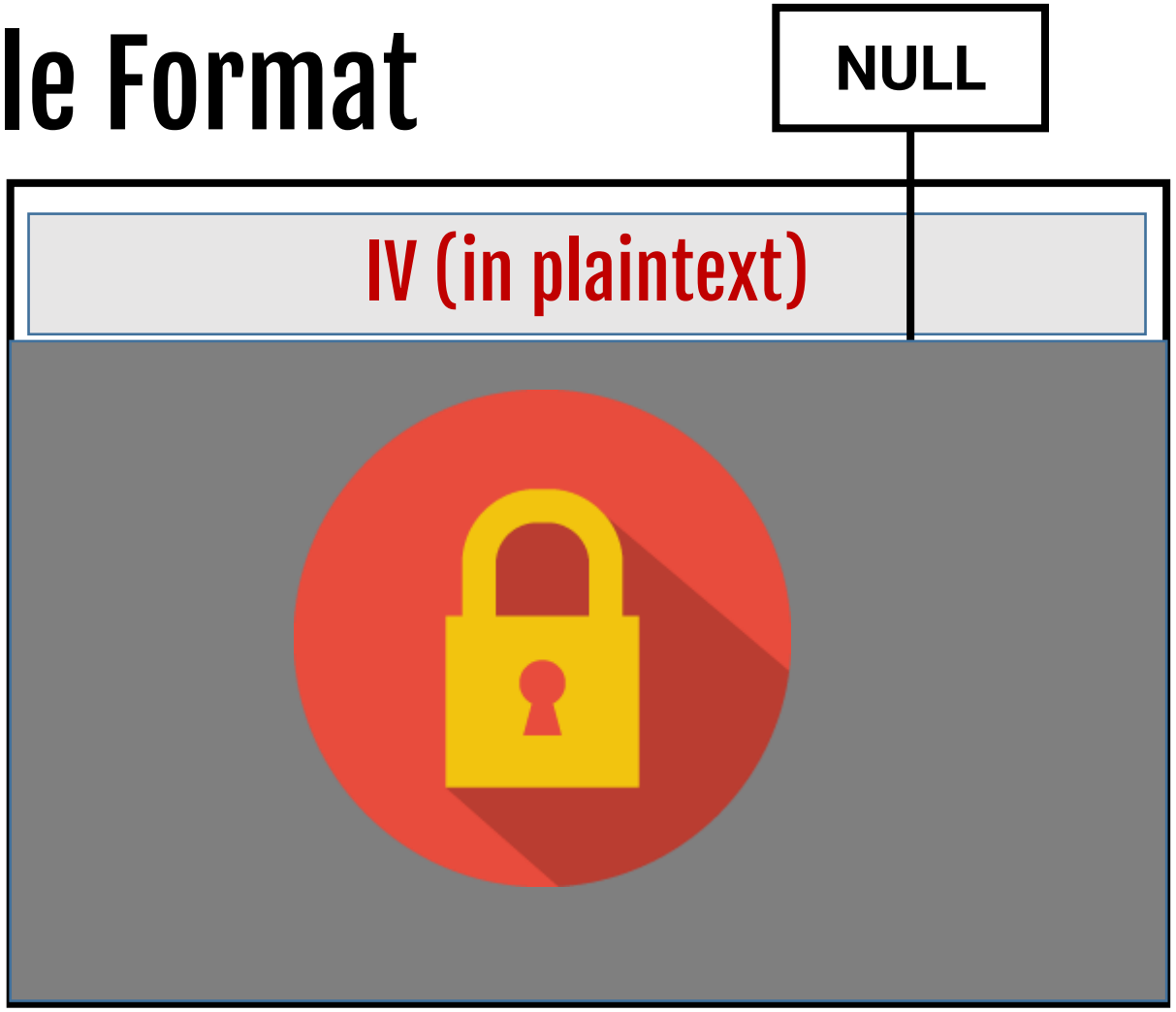


Chunk file –

- Name can contain only alphanumeric characters
- File name length maximum 20 characters.

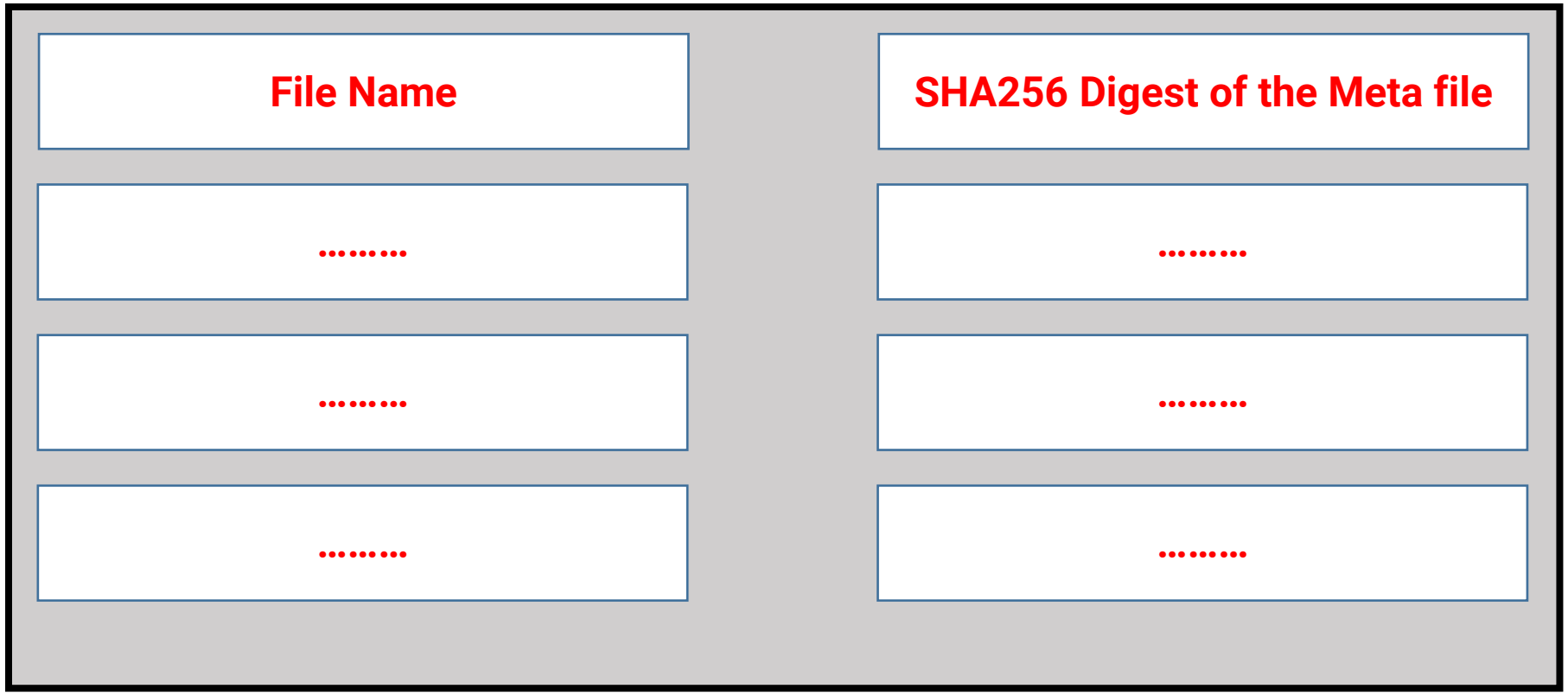
Part 2 – File Format

Meta file format



Chunk file format

Master File List (Simplified SEFS Integrity Protection)



Part 2 – Functionality

- create_file(u,p,filename)
- delete_file(u,p,filename)
- encrypt_file(u,p,filename)
- decrypt_file(u,p,filename,pfilename)
- read_from_file(u,p,filename,position,len)
- write_to_file(u,p,filename,position,newcontent)
- file_size(u,p,filename)
- file_integrity_check(u,p,filename)
- system_health_check()

Returns:
OKAY -> 1
ERROR -> -1

Returns:
OKAY -> char *
ERROR -> NULL

Master Key and IV

Part 2 – Read Operation

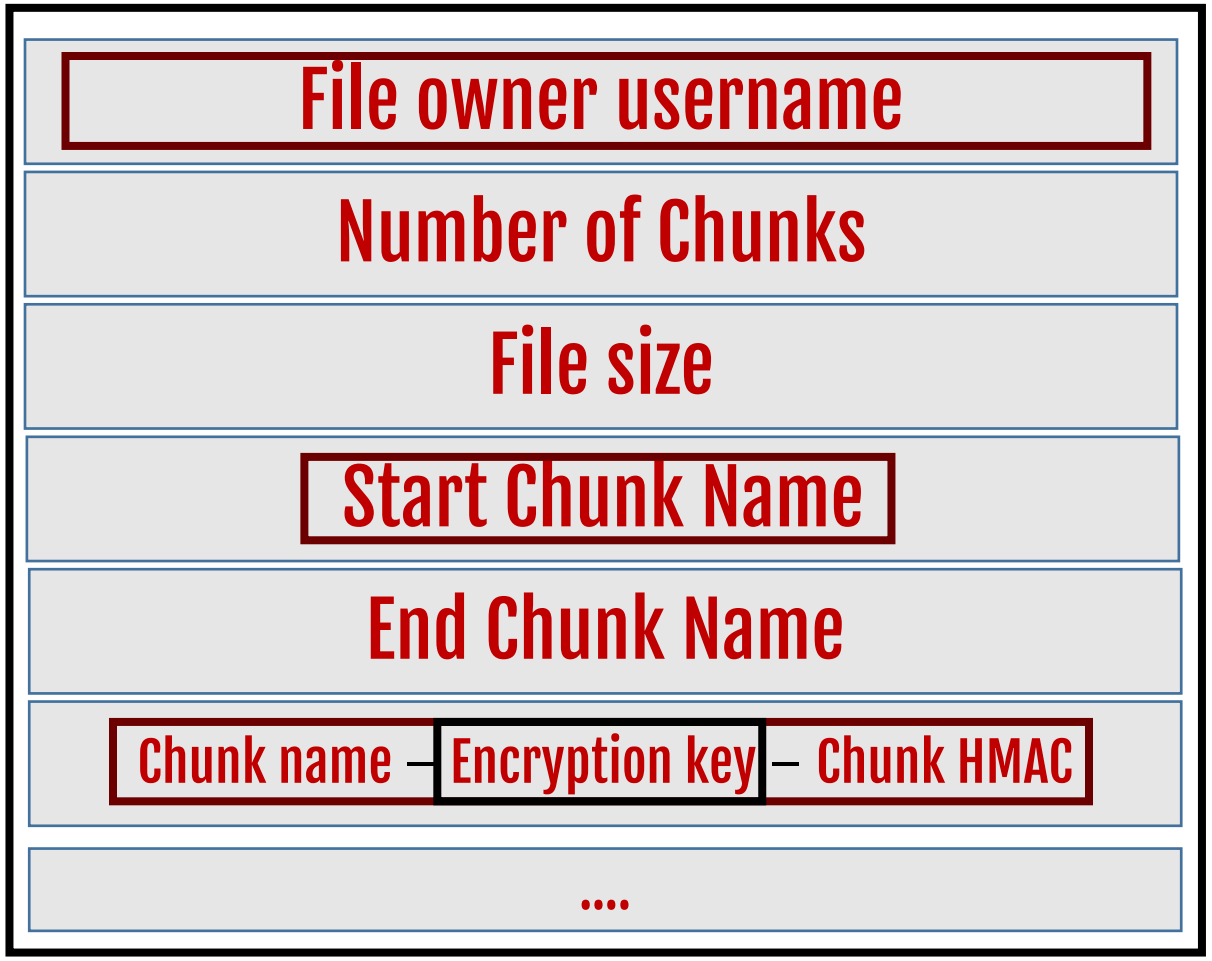
Meta file format

Chunk file format

Full SEFS

- Generalization of the simplified SEFS.
- Each chunk can hold at most 1024 bytes of plaintext data.
- Each plaintext file can be divided into multiple encrypted chunk files.
- If a file has less than 1024 bytes of data, you are required to pad it with ASCII character 0 to make it 1024 bytes.
- **Space restriction**: You are required to use the minimum number of chunk files for storing each plaintext file
- **Example**: If you have a chunk containing 512 bytes of data and the user wants to write 200 bytes to the end of the chunk, you cannot create a new chunk and instead have to write into that chunk.

Part 2 – Full SEFS Read Operation



Meta file format

Chunk file format

Potential Pitfalls

- **Memory leaks** – a lot of the operations of the project require pointer manipulation, make sure to free the pointer after usage
- **File operations** – file operations in C is complicated, you cannot write in the middle of a file without overwriting the content. You have to manually move the following content and then write something
- **Error checking** – a lot of errors can potentially happen during the operation and it is paramount that you do handle these errors. *Do not assume inputs are well-formed*. Perform input validation when applicable.

Different parameters

- username
  - a-zA-Z0-9
  - Length >= 6 and < 32
- Password
  - a-zA-Z0-9@#$%&*()-+=
  - Length >= 9 and < 32
- Password salt
  - Randomly generated
  - 32 bytes
- Master key 128 bits
- Master IV **128** bits
- Chunk keys 128 bits, randomly generated
- For encryption use, AES in the CTR mode
- Chunk IVs 128 bits, randomly generated
- Chunk names are randomly generated and cannot have space character in it
- For padding use the ASCII character 0
- For hash mac, use HMAC with EVP_sha256()
- For digest, use SHA256
- For password hash, use **PKCS5_PBKDF2_HMAC_SHA1** with iteration value **20000**

Questions

- If you do not understand any specifics, please do not make your own assumptions rather confirm with me.
- Making arbitrary, easy to implement assumptions will surely ensure you losing 5% of the inter-operability.
- Direct any questions related to the project to me through piazza, email (ochowdhu@purdue.edu), or drop by my office during office hours (LWSN 2142 R, Thursday 11:30am - 12:30pm)