SecureDrop

Final Project - COMP.2300

 Made by Will Noonan

TABLE OF CONTENTS

01

Assumptions

02

Code Structure

03

Security

01 ASSUMPTIONS

 For testing purposes, we need to make some basic assumptions. In a real world application, these assumptions would pose security risks and need to be addressed.

ASSUMPTIONS

CERTIFICATES

We need to assume any certificates are signed and distributed by a trusted Certificate Authority (CA)

FILE SECURITY

Any private keys and json files that are used are stored securely. For testing purposes, these files are being stored locally.

PICKLE

Pickle is encrypted separately. For testing purposes, we are using standard pickle functions.

02 STRUCTURE

 To provide a deeper understanding on how SecureDrop works, we will briefly go over the structure of the code.

SERVER

Server Class (server.py)

- Handles multiple client connections to perform various operations based on received messages from a client.
- Has its own key and certificate
- Can be seen as the middle-man between two clients

Instance Variables:

host: Set to 127.0.0.1 (local machine)

port: Set to 52000

key_file: Path to private key file

cert_file: Path to certificate file

socket: Stores SSL-wrapped socket object to accept client connections

connectedClients: List that stores all clients currently connected to the server.

transfer: Handles data transfer between two clients

CLIENT

UserHandler Class (client.py)

- Handles client side operations, such as displaying the main menu,
- Sends messages to the server class to obtain information such as the list of connected clients.
- Represents users on each side of communication

Instance Variables:

host: Set to 127.0.0.1 (local machine)

port: Set to 52000

key_file: Path to private key file

cert_file: Path to certificate file

socket: Stores SSL-wrapped socket object to connect to a server

user: Stores a users name and email

transfer: Stores a list of contacts in contacts.json

03 SECURITY

 Cybersecurity isn’t a choice, but a necessity in the digital landscape. SecureDrop implements a combination of cryptographic methods to provide Confidentiality, Integrity and Availability to its users.

ATTACKS TO MITIGATE

Password Cracking

Brute Force Attack

Dictionary Attack

Rainbow Table Attack

Data Tampering

Unauthorized Access

Data Modification

Session Attack

Session Hijacking

Network Attacks

Packet Sniffing

Interception Attacks

Other Attacks

Replay Attacks

MITM Attacks

Impersonation Attacks

PKI

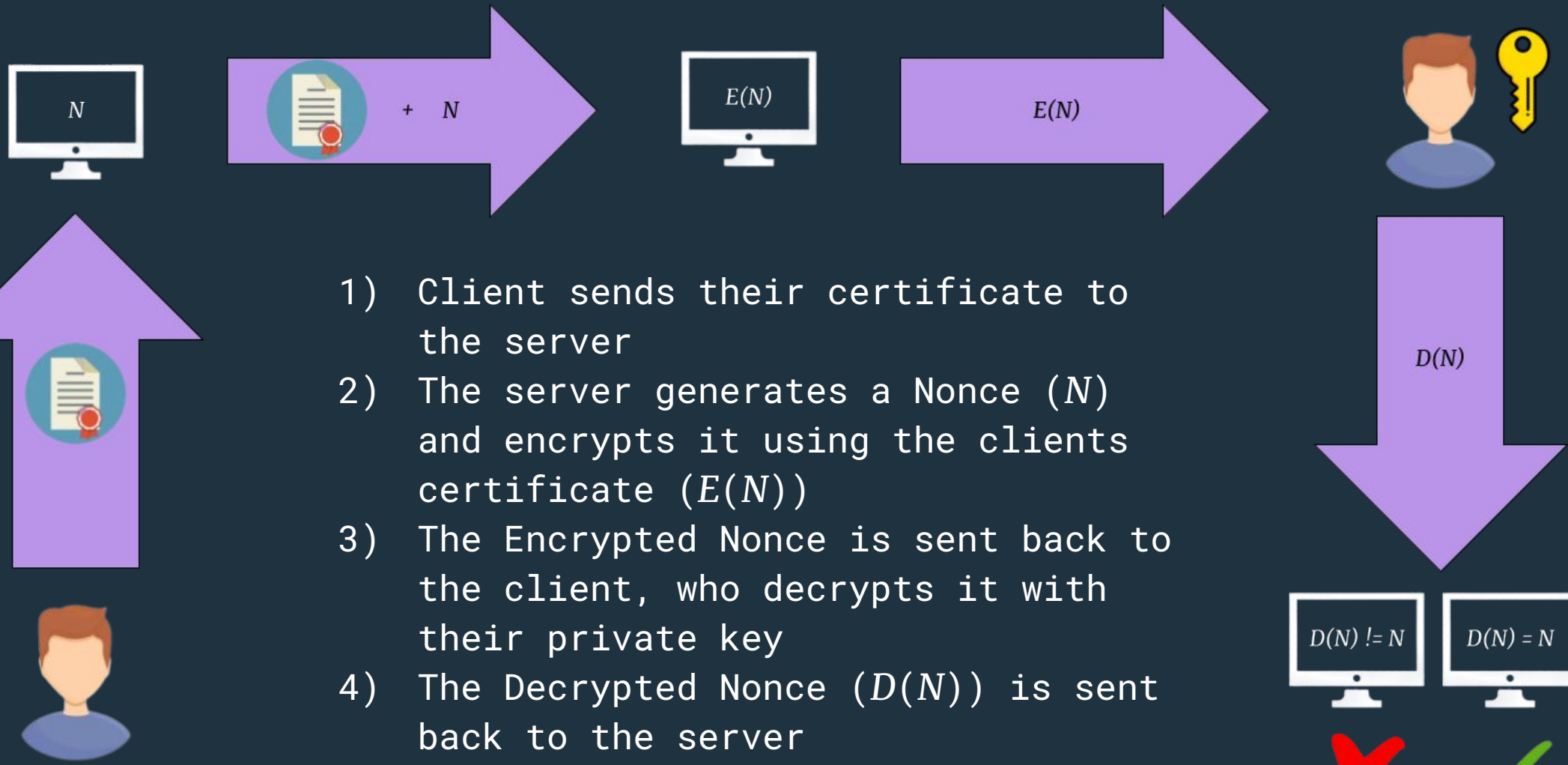
1

 Mutual Authentication

Public Key Infrastructure (PKI) is a set of procedures needed to manage digital certificates and public-key encryption.

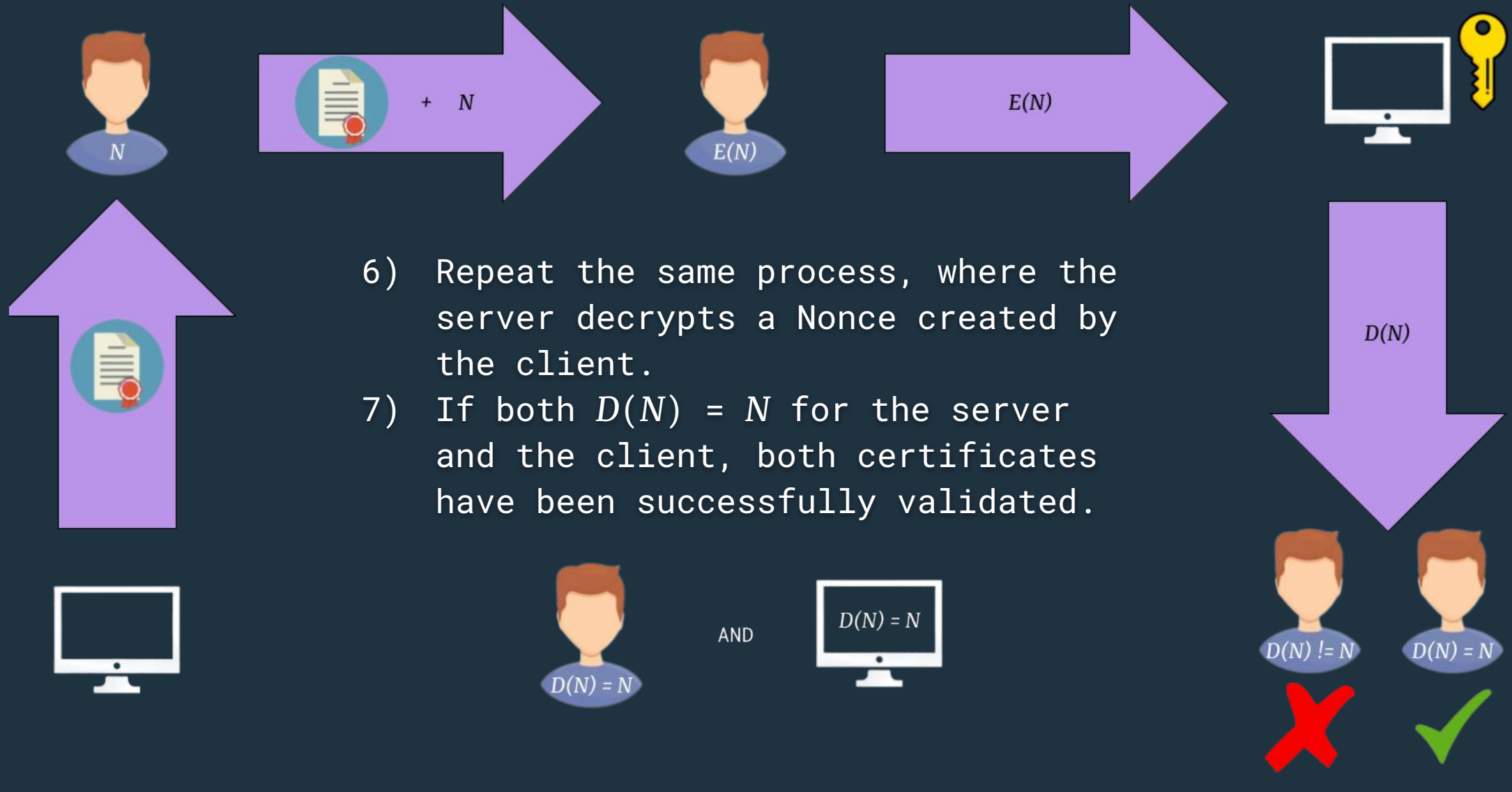
In the context of SecureDrop, we use PKI upon a user logging in for mutual authentication between a server and a client.

```
def processClientMessages(self, conn)
```



1) Client sends their certificate to the server
2) The server generates a Nonce (N) and encrypts it using the clients certificate ($E(N)$)
3) The Encrypted Nonce is sent back to the client, who decrypts it with their private key
4) The Decrypted Nonce ($D(N)$) is sent back to the server
5) If $D(N) = N$, the client has been successfully authenticated

```
def startApp_handle(self)
```



6) Repeat the same process, where the server decrypts a Nonce created by the client.
7) If both $D(N) = N$ for the server and the client, both certificates have been successfully validated.

MUTUAL AUTHENTICATION

Confidentiality: Uses encryption to protect data from unauthorized access.

Integrity: Mutual authentication verifies the identity of the server and client.

CIA TRIAD

MITIGATIONS

 Mutual Authentication

Impersonation Attacks: Attackers cannot impersonate the server or client without having access to their private keys.

Replay Attacks: Since nonces are used in conjunction with Mutual Authentication, it can prevent replay attacks where an attacker tries to replay a previous communication with the server.

PASSWORD STORAGE

2

 Salted Hashing

When a new user is created, we store their password using the bcrypt hashing algorithm, and generate a unique salt for each user.

The hashed password and salt are stored in user.json.

```
def createUser(self)
```

1) When creating a new user, we collect the name, email and password of the user.
2) The password is entered twice. If they match, we hash it using the bcrypt.hashpw() function.
   a) This takes in the password to be hashed (as a bytes object) and a salt (generated by bcrypt.gensalt()).

bcrypt.gensalt()

Encode Password to bytes (UTF-8)

BCrypt HASH

Decode

Hashed Password

user.json

salt = os.random(16)
Encode to b64 format

Salt

def createUser(self)

1) When creating a new user, we collect the name, email and password of the user.
2) The password is entered twice. If they match, we hash it using the bcrypt.hashpw() function.
   a) This takes in the password to be hashed (as a bytes object) and a salt (generated by bcrypt.gensalt()).

bcrypt.gensalt()

Encode Password to bytes (UTF-8)

BCrypt HASH

Decode

Hashed Password

user.json

salt = os.random(16)
Encode to b64 format

Salt

3) A separate salt is generated using os.random(16), which generates 16 random bytes.
4) This is encoded to base64 format, and is used in addition to the salt used by the bcrypt hashing function.

def createUser(self)

bcrypt.gensalt()

Encode Password to bytes (UTF-8)

BCrypt HASH

Decode

Hashed Password

user.json

salt = os.random(16)
Encode to b64 format

Salt

1) When creating a new user, we collect the name, email and password of the user.
2) The password is entered twice. If they match, we hash it using the bcrypt.hashpw() function.
   a) This takes in the password to be hashed (as a bytes object) and a salt (generated by bcrypt.gensalt()).

3) A separate salt is generated using os.random(16), which generates 16 random bytes.
4) This is encoded to base64 format, and is used in addition to the salt used by the bcrypt hashing function.

5) Finally, store in user.json
6) We do not store Plain Text Password in memory.

SALTED HASHING

Confidentiality: Even if someone gains access to user.json, they can’t see the actual password. Using bcrypt ensures a high level of confidentiality to users.

CIA TRIAD

MITIGATIONS

 Salted Hashing

Brute Force and Dictionary Attacks: The bcrypt algorithm is designed to be computationally slow and intensive to compute, which helps deter brute force and dictionary attacks.

Rainbow Table Attacks: Generating a unique salt for each user ensures their hashed passwords will be different even if they have the same password.

CONTACT STORAGE

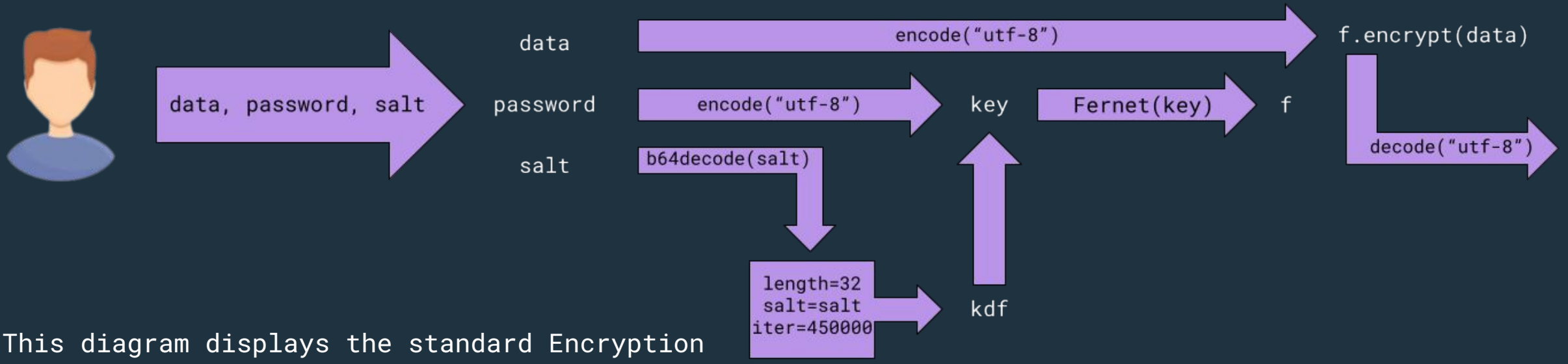
3

 Encryption/Decryption

When adding and listing contacts, it’s essential to ensure there are no vulnerabilities when communicating between the client and server.

Leaking personal information to an attacker can lead to spam or worse to a user’s contacts.

def Encrypt(self, data, passwd, salt)



This diagram displays the standard Encryption method used to encrypt objects such as user information, contact lists, and more.

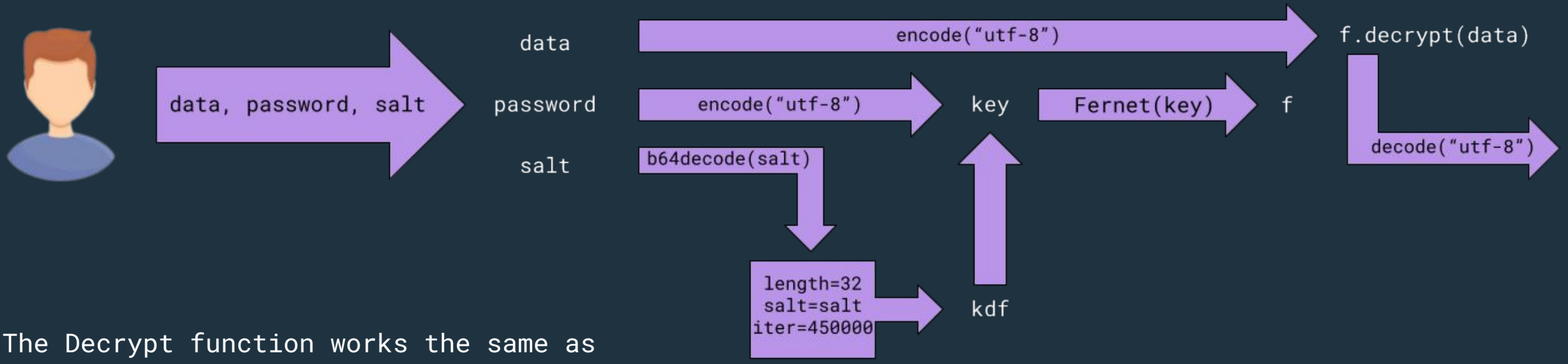
Using a KDF derived from the users password helps protect against brute force attacks given the high number of iterations we are using.

The KDF is set up using the PBKDF2 method with HMAC and SHA256.

Notes:

- We are using Fernet, a symmetric encryption algorithm
- The passwd argument is used to encrypt the data. The salt argument is used to add randomness to our encryption
- The users password is encrypted, and is generally considered safe to store in memory.

def Decrypt(self, data, passwd, salt)



The Decrypt function works the same as Encrypt, except we are using Fernet's (f) decrypt() method.

ADDING CONTACTS

 Your contacts are stored safely.

When adding a contact, their name and email is encrypted and stored in contacts.json. This data is never transmitted over the network in plain text.

 We can detect changes to your contact list.

We can check to see if the contact has already been added. If the entered Email or entered name matches a stored contact, we can detect this change.

 Our server is updated each time you add a new contact.

Each time a new contact is added or contact information is changed, the server is updated with a new contact list.

```
def listContacts_handle(self)
```

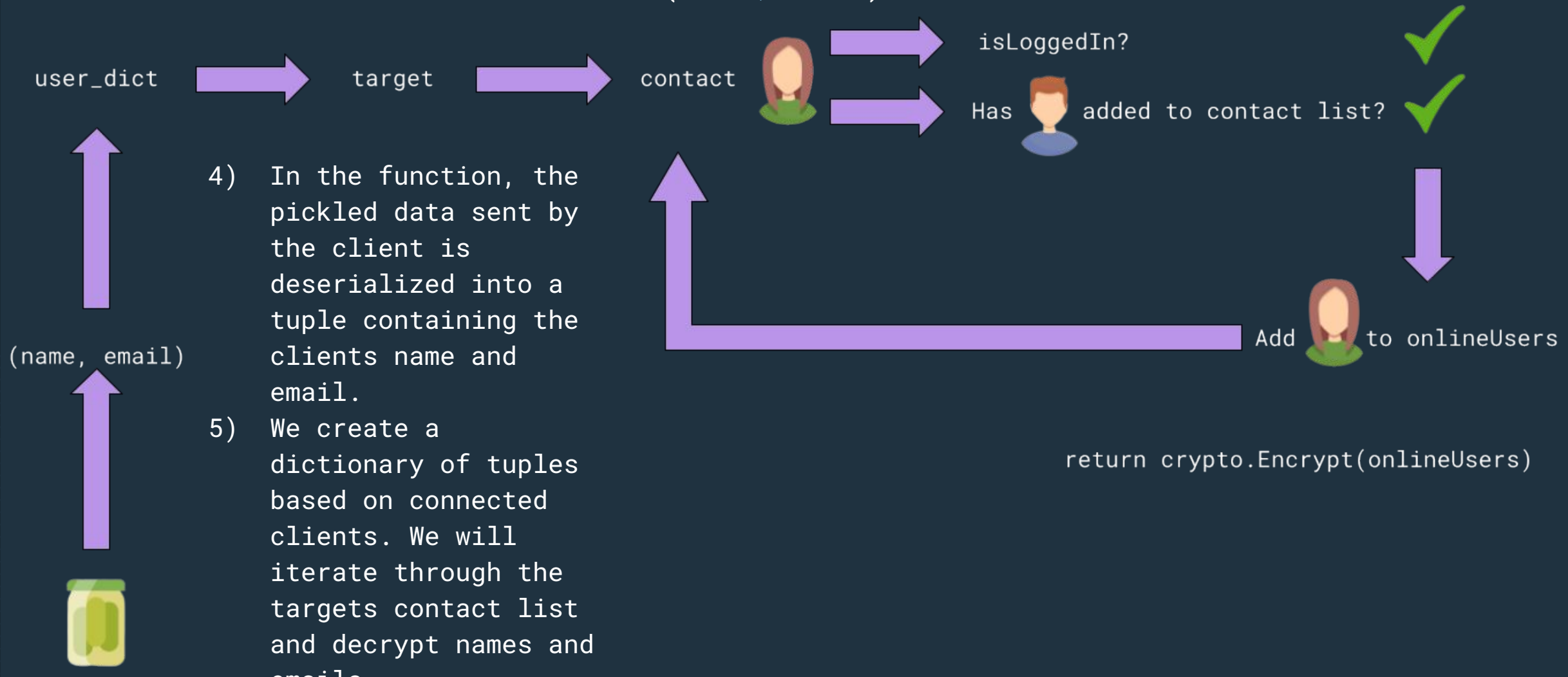
`self.listConnectedClients(data)`

Encrypted list of connected clients

Decrypted list of connected clients

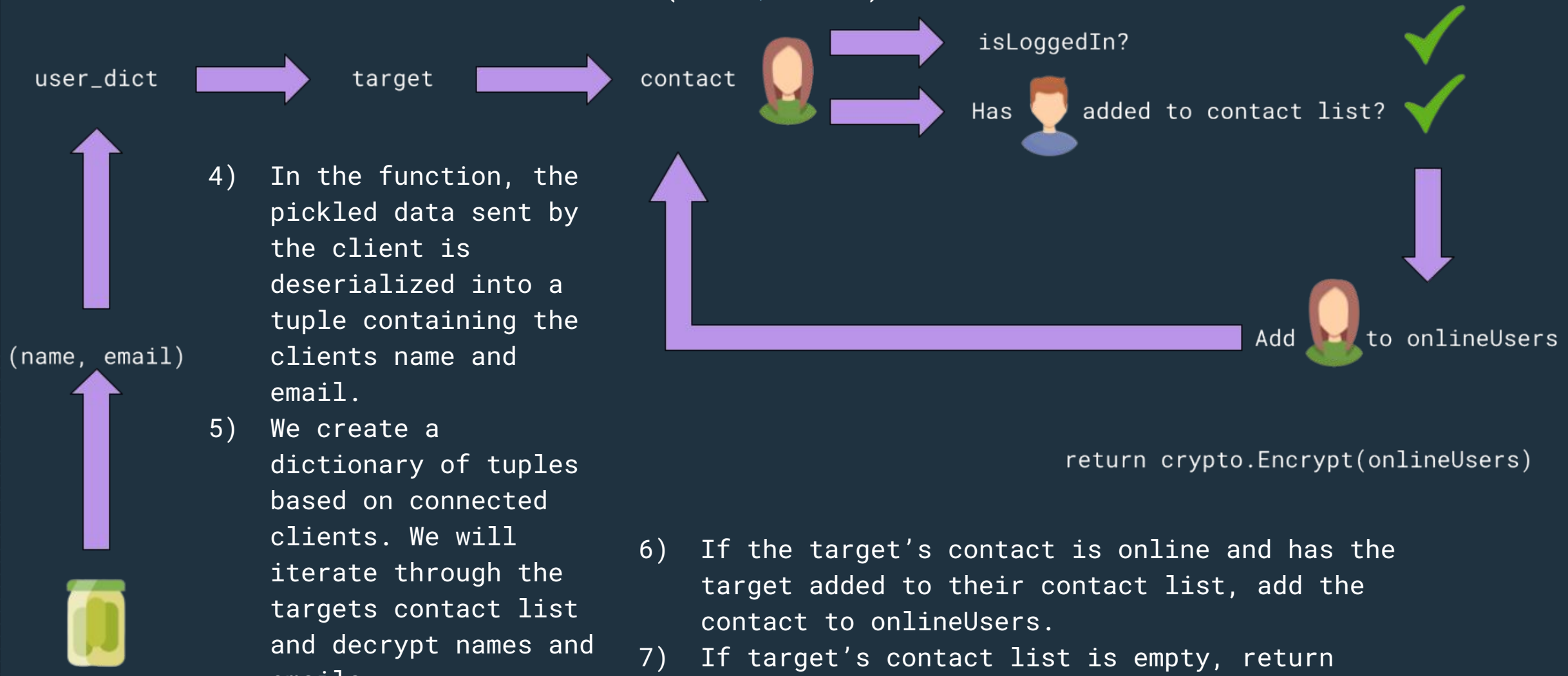
1) The client sends a message “LIST CONTACTS” to the server. This makes the server ready to perform its operations to return the list of connected contacts
2) The client then sends the pickled (serialized) data of their user object containing their name and email to the server.
3) The server receives this request, and calls listConnectedClients with the pickled data.

```
def listConnectedClients(self, data)
```



4) In the function, the pickled data sent by the client is deserialized into a tuple containing the clients name and email.
5) We create a dictionary of tuples based on connected clients. We will iterate through the targets contact list and decrypt names and emails.

```
def listConnectedClients(self, data)
```



4) In the function, the pickled data sent by the client is deserialized into a tuple containing the clients name and email.
5) We create a dictionary of tuples based on connected clients. We will iterate through the targets contact list and decrypt names and emails.

```
return crypto.Encrypt(onlineUsers)
```

6) If the target's contact is online and has the target added to their contact list, add the contact to onlineUsers.
7) If target's contact list is empty, return Encrypt(onlineUsers).

```
def listContacts_handle(self)
```

```
self.listConnectedClients(data)
```

Encrypted list of connected clients

Decrypted list of connected clients

8) The encrypted list of contacts is sent back to the client who can decrypt it using the Decrypt Function.

CONTACT STORAGE

Confidentiality: All contact data is encrypted. There is no way for someone to read contact data as plain text.

Integrity: Detects any tampering of contact data or hashes.

Availability: Contact data is only available to authorized users.

CIA TRIAD

MITIGATIONS

 Encryption/Decryption

Packet Sniffing: Contact list is encrypted when getting a list of connected contacts. This means an attacker sniffing packets cannot get contact information.

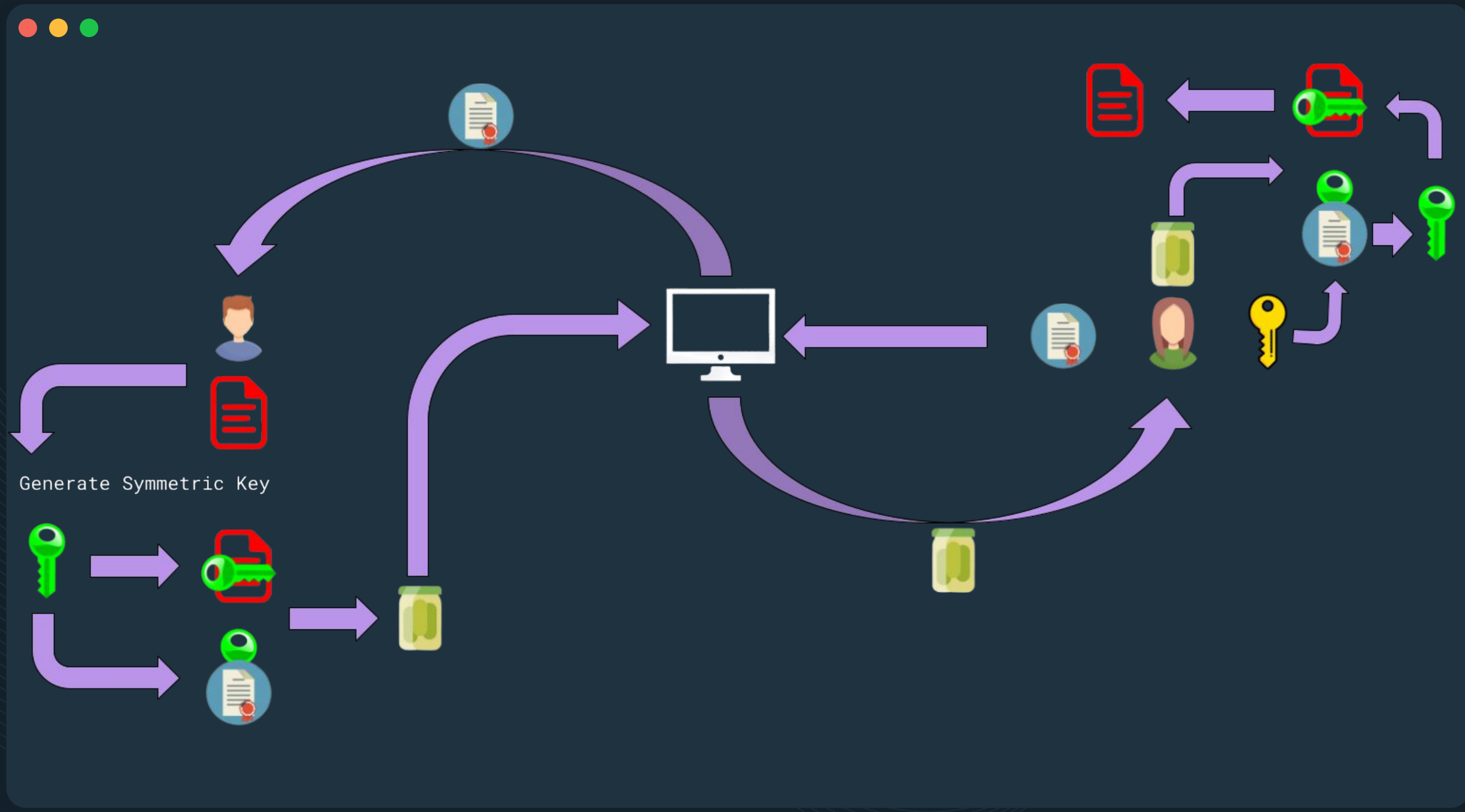
Data Tampering/Unauthorized Access: We mitigate these threats by detecting any changes to contact information, and ensure only authorized users can decrypt data with the Decrypt function.

SECURE TRANSFER

4

 Secure File Transfer

By using a combination of Asymmetric and Symmetric Encryption, SecureDrop is able to send large files both efficiently and securely.



CRYPTO METHOD

Confidentiality: All sent files are encrypted, ensuring confidentiality. Files can only be decrypted using a private key.

Integrity: File data is not modified during transfer and encryption.

Availability: Because data is broken into bytes, any file type can be securely sent.

CIA TRIAD

MITIGATIONS

 Secure File Transfer

MITM/Interception Attacks: An attacker who intercepts a file is not able to read it. If the file is modified, it will be detected during decryption.