CS 431: Computer and Network Security

Design Document

Indian Institute of Technology Gandhinagar

Security Guard at IITGN

GitHub: https://github.com/Shr-Agr/CNS_Project_Security_Guard
Group Number: 6
4th September, 2024

Netram Choudhary (Leader)	21110138
Soham Rahatal	21110173
Shreesh Agrawal	21110198
Pranjal	21110160
Ishika Raj	21110081
Twinkle Devda	21110228
Apurva Ochawar	21110142

Security Guard

This project focuses on implementing a *secure log system* designed to monitor and record the state of an institute. This system is built to handle critical information about guests and employees, tracking their movements and activities within various buildings and rooms on the campus.

The secure log system has two core components: **logappend** and **logread**

- **logappend** is responsible for inserting new log entries into the log file.
- **logread** will be used to retrieve and display the information based on the queries of the user. It will be used to view the current state of the institute.

Tools, libraries and APIs

- **Language:** C++
- **Libraries:** Libsodium (NaCl) / OpenSSL (for encryption and authentication), zlib (for compression)

  (Our main aim is to utilize the Libsodium library and its API's mentioned below. But at the same time it is also important to keep a failsafe for the future. So, for that purpose, we have also mentioned OpenSSL library.)
- **Algorithms:** XChaCha20 for encryption, Poly1305 for authentication, and Argon2id for hashing.

  (The reason for choosing XChaCha20 is its good CPU performance over AES which improves the speed of the system, and also its 192 bit nonce, which again improves the security. Also, Poly1305 is designed for efficient and secure software implementations.)
- **APIs:**
  - **crypto_aead_xchacha20poly1305_ietf_*():**
    Offers various functions for secure encryption using the XChaCha20 stream cipher for encrypting data and the Poly1305 message authentication code (MAC) for ensuring integrity and authenticity.

  - **crypto_pwhash_***:
    This provides functions for combining the token with a salt and additional factors to create a protected key through hashing.

Log format

Log File Structure: Each log entry is stored in JSON format, which is then encrypted and validated.

JSON Log Entry Format:

(Plaintext log entry format)

```
{
  "timestamp": <integer>,
  "event_type": "<A|L>",
  "name": "<employee_name_or_guest_name>",
  "room_id": <integer (optional)>
}
```

- **timestamp**: An integer representing the time of the event in seconds since the campus opened.
- **event_type**: Specifies whether the event is an arrival ("A") or departure ("L").
- **name**: The name of the employee or guest involved in the event. Names are case-sensitive and contain only alphabetic characters.
- **room_id** (optional): The room ID where the event took place. Room IDs are non-negative integers. This field is omitted if the event is related to the campus as a whole.

System Operation

- **logappend**:
  - Appends log entries with timestamps, names, and room IDs in a JSON file.
  - Ensures the integrity of the log file by validating timestamps and tokens.
  - The data is encrypted using Libsodium Library (with XChaCha20-Poly1305) and compressed using zlib.
  - Algorithms used for hashing - Argon2id.

- **logread**:
  - It takes the token and the encrypted file, checks its validity and decrypts it using the Libsodium Library (with XChaCha20-Poly1305).
  - It decompresses the decrypted file.
  - It uses a hardcoded string and Argon2id hash to verify the correctness of decrypted data.
  - Verifies the integrity of the log entries by checking the MAC.

    - Reads and processes the data according to the user Query.

- **Hashing module**:
    - Creates cryptographic hashes.
    - These hashes are primarily used for key generation from token.
    - Will use the Argon2id algorithm for hashing.

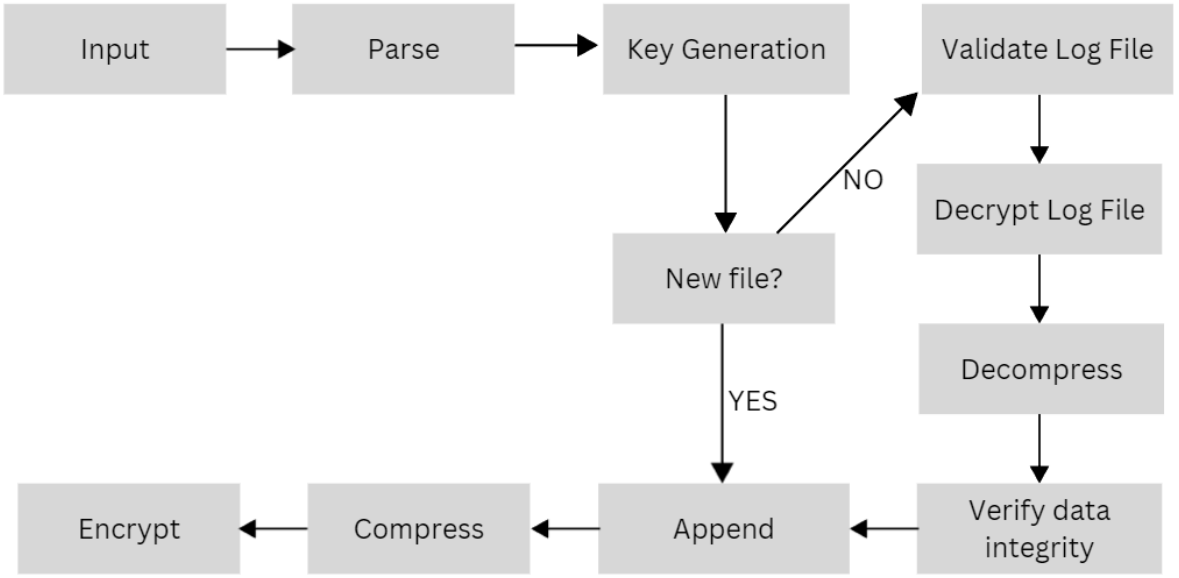
- **Helper functions:**
    - **Function for compression/decompression**:
        - Compresses the data using zlib before it is encrypted.
        - Decompresses the data after it has been decrypted.
    - **Data Integrity**:
        - Creates a MAC for the encrypted log to verify its integrity.
        - Confirms the integrity of the data by verifying its MAC to ensure it hasn't been altered.

- **Input parsing module:**
    - Accurately parse the commands (logappend, logread) along with their respective arguments, ensuring that all required fields are present and correctly formatted (e.g., timestamps, user IDs, room numbers)
    - Error handling to catch and respond to invalid or malformed inputs, such as missing arguments or incorrect data types, with clear and informative error messages.
    - Token validation

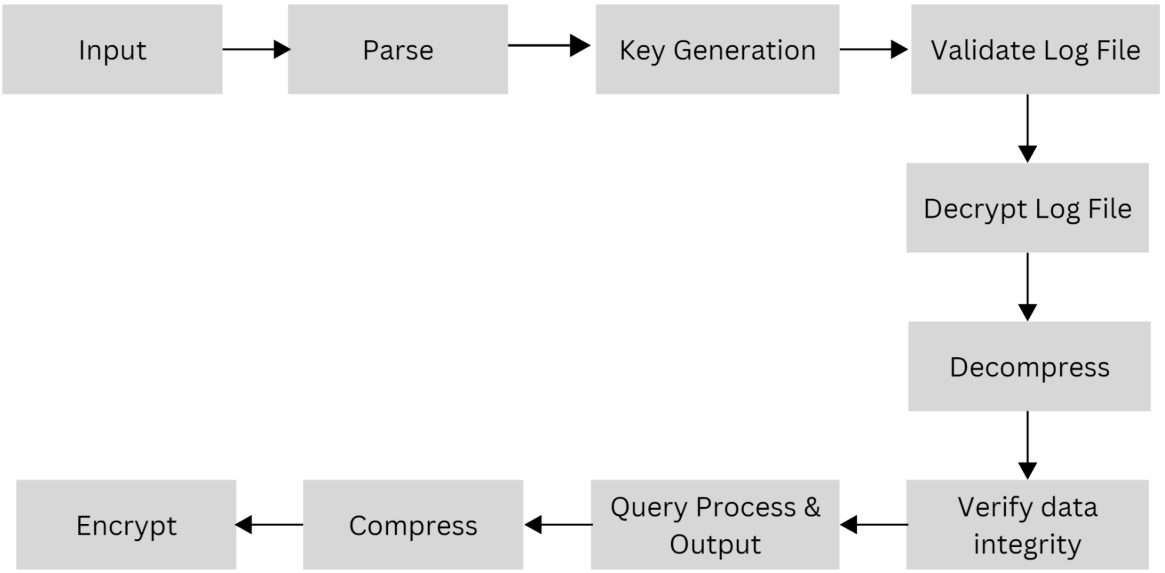
- **Encrypt/ decrypt module:**
    - This module is essential for securing the data via encrypting it.
    - This would use the Libsodium API for encryption and authentication as mentioned above.
    - Serialize JSON Entries: Every log entry is transformed into one JSON string that depicts the complete log file.
    - Compression: Use a compression algorithm like zlib to shrink the file size of the serialized JSON string.
    - Encryption process: Utilize the XChaCha20-Poly1305 algorithm from the Libsodium library to encrypt the entire serialized JSON string.
    - Calculate a Message Authentication Code (MAC) using Poly1305 for the encrypted data in order to verify its integrity.
    - Save the MAC and Encrypted Data: Keep both the encrypted data and the MAC you calculated in the same final log file.

Flowchart

Working of logappend module-



Working of logread module-



Work Distribution

Here is the list of modules that would be required for the project. The list may get updated as the project progresses according to the needs of the project.

Although we will follow a collective approach for the entire project, here is a tentative list of modules being assigned to different persons. We might be changing the modules assigned to some person as the project progresses as per the need.

- Logappend module - Shreesh, Soham, Pranjal
- Logread module - Apurva, Ishika, Twinkle
- Hashing module - Soham, Ishika, Twinkle
- Helper functions - Apurva, Pranjal, Netram
- Input Parsing Module - Twinkle, Ishika, Netram
- Encrypt/Decrypt Module - Apurva, Shreesh, Netram

References

[1]https://crypto.stackexchange.com/questions/43112/poly1305-aes-vs-aes-gcm
[2]https://doc.libsodium.org/secret-key_cryptography/aead/chacha20-poly1305/xchacha20-poly1305_construction
[3]https://crypto.stackexchange.com/questions/91945/xsalsa20poly1305-for-encryption-at-rest
[4]https://security.stackexchange.com/questions/261902/argon2-vs-sha-512-whats-better-in-my-case
[5]https://stackoverflow.com/questions/64220922/bcrypt-vs-argon2-and-their-hashing-algorithms
[6]https://en.wikipedia.org/wiki/Argon2
[7]https://medium.com/@rahul660singh/if-youre-going-to-compress-and-encrypt-a-file-which-do-you-do-first-and-why-1a99cde9c4e3#:~:text=The%20correct%20order%20is%20to,then%20encrypt%20the%20compressed%20data.

END
