Implementation of a Vault-Based Mutual Authentication Protocol for IoT Devices

Davide Baggio – 2119982

`davide.baggio.1@studenti.unipd.it`

Abstract

As Internet of Things (IoT) devices become more and more prevalent, securing their communication channels is critical. This project presents a Python-based implementation of a lightweight mutual authentication protocol designed for resource-constrained environments. The system utilizes a "Secure Vault" architecture: a synchronized storage of secret keys which enable a challenge-response mechanism. This approach ensures mutual authentication and dynamic session key agreement without transmitting long-term secrets. This work validates the protocol's logic and feasibility through a complete software simulation.

1 Introduction

The increased use of Internet of Things (IoT) devices in recent years has led to many more security challenges, especially in the context of authentication of those devices to their server. The solution proposed by Shah et al. [1] utilizes a Secure Vault which is a collection of secret keys shared between the IoT device and the server. The protocol employs a challenge-response mechanism where the server challenges the device to prove possession of keys at random indices, and vice-versa. This mutual exchange verifies identities and derives a session key for subsequent encrypted communication. In this report, it's provided a Python implementation of this authentication protocol, covering secure vault management, the four-step handshake, and the dynamic vault update process. The code can be found at this repository [2].

2 Authentication Mechanism

This section details the three-way authentication protocol implemented for the communications between IoT devices and server. It relies on a pre-shared "Vault" of random keys and establishing a temporary Session Key (SK) through a 4-message handshake.

2.1 Secure Vaults

A secure vault is a protected storage containing N random keys. In this implementation, `VAULT_SIZE` defines N, and keys are 128-bit blocks.

$$V = \{k_0, k_1, ..., k_{N-1}\} \tag{1}$$

To prevent replay attacks and ensure forward secrecy, the contents of the secure vault are changed after every successful communication session. This rotation ensures that even if a device is compromised later, previous sessions remain secure. The update process utilizes the *HMAC-SHA256* algorithm and consists of three distinct steps:

1. **HMAC Computation:** A hash h is computed using the data exchanged during the session (e.g., sensor readings) as the cryptographic key, and the entire current secure vault ($V_{current}$) as the message.

$$h = \text{HMAC}(SessionData, V_{current}) \tag{2}$$

   The resulting h has a bit-length of k (256 bits for SHA-256).

2. **Vault Partitioning:** To apply the update uniformly, the current secure vault is divided into j equal partitions, each of size k bits. If the total size of the vault is not an exact multiple of k, the final partition is padded with zeros to ensure alignment.

3. **Partition Update via XOR:** Each partition P_i (where $i = 1, \ldots, j$) is updated by XORing it with the computed hash digest h to generate the new partition P_i':

$$P_i' = P_i \oplus h \tag{3}$$

Parameter	Value	Description
VAULT_SIZE	128	Total keys in the Vault (N)
KEY_SIZE_BITS	128	Bit-length of each Vault key
CHALLENGE_INDICES	8	Number of keys per challenge ($|C_1|, |C_2|$)
BLOCK_SIZE	16	AES block size in bytes (128 bits)

Table 1: System Configuration Parameters

The new vault V_{new} is formed by combining all updated partitions P_i'. Both the IoT device and the server perform this operation independently. If their vaults remain synchronized, it serves as a final implicit confirmation of a successful session.

2.2 Authentication Process

The protocol consists of a total of four messages (M_1–M_4).

1. **Connection Request**
   The IoT device initiates the session by sending its ID and a Session ID and it doesn't require encryption.

$$M_1 = \text{Device ID} \parallel \text{Session ID} \qquad (4)$$

2. **Server Challenge**
   The server validates the Device ID by checking its presence in the authorized devices list. It then generates:

   - A random nonce r_1.
   - A challenge C_1 which is a list of random indices pointing to keys in the vault.

$$M_2 = \{C_1, r_1\} \qquad (5)$$

3. **Device Response & Challenge**
   The device receives M_2 and uses indices C_1 to derive key k_1 via XORing of the selected keys. It then generates a random nonce r_2, a session component t_1, and its own challenge indices C_2. It encrypts the payload:

$$M_3 = \text{Enc}(k_1, r_1 \parallel t_1 \parallel \{C_2, r_2\}) \qquad (6)$$

By including r_1, the device proves it could derive k_1 and thus possesses authorized access to the vault.

4. **Server Response**
   The server decrypts M_3 using k_1, verifies r_1 and then derives k_2 via XORing of indices C_2. After this, it generates a session component t_2. The response is encrypted using a key derived from k_2 and t_1:

$$M_4 = \text{Enc}(k_2 \oplus t_1, r_2 \parallel t_2) \qquad (7)$$

5. **Session Key Agreement**
   Both parties compute the final Session Key:

$$SK = t_1 \oplus t_2 \qquad (8)$$

3 System Setup

3.1 Requirements

The system was developed using Python3 and the libraries utilized are as follows:

- `PyCryptodome`: Provides cryptographic functionalities including AES encryption in ECB mode.

- `secrets`: Used as a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) for nonces ($r1, r2$) and session components ($t1, t2$).

- `hmac` & `hashlib`: Used for the vault update mechanism (SHA-256).

3.2 Class Definitions

The implementation is structured into three primary classes:

- `SecureVault`: Manages the storage of keys, the XOR-based key derivation from indices, and the HMAC-based update logic.

- `IoTEntity`: A base class providing common encryption/decryption methods.

- `IoTDevice`/`IoTServer`: Implement the specific state machine logic for generating and processing M_1–M_4 messages.

3.3 Configuration Parameters

The security and performance of the simulation are governed by the configuration constants defined in `config.py`. Table 1 summarizes the parameters used for this implementation.

4 Security Analysis

The software simulation demonstrates following security properties:

- **Authentication**: Mutual authentication is achieved because k_1 and k_2 can only be derived by a party possessing the correct Vault. The exchange of nonces r_1 and r_2 prevents replay attacks.

- **Confidentiality**: The components of the session key (t_1, t_2) are never sent in plaintext. t_1 is encrypted with k_1, and t_2 is encrypted with $k_2 \oplus t_1$. Even if an attacker captures the traffic, they cannot reconstruct SK without the Vault.

- **Forward Secrecy**: The Vault Update mechanism ensures that if a device is physically compromised and the Vault is dumped, it cannot be used to decrypt past sessions because the keys have rotated.

The paper by Shah et al. [1] also discusses resistance against the following specific attacks:

- **Man-in-the-Middle (MITM) Attacks**: The protocol ensures that an attacker cannot hijack the session because all post-authentication messages are authenticated using a session key t. This key is derived from random numbers t_1 and t_2, which are exchanged exclusively via encrypted messages. Since the encryption keys for these messages are part of the secure vault (secretly shared between the server and IoT device), a MITM attacker cannot retrieve t or modify the traffic.

- **Next Password Prediction**: To ensure forward secrecy, the secure vault values change after every session. Under the *Random Oracle Model*, the HMAC function acts as a random oracle that generates a random output based on the previous vault and exchanged data. By XORing this random output with the previous vault values (applying the *One-Time Pad theorem*), the new vault becomes statistically random. This ensures that even if an adversary predicts or retrieves part of the current vault, they cannot predict the values of the next secure vault.

- **Side-Channel Attacks**: Standard single-password systems are vulnerable to side-channel attacks (e.g., power or memory analysis) that retrieve the AES encryption key. In this protocol, the AES key used for the challenge-response (k_1 or k_2) is a combination of multiple vault keys XORed together. Even if an attacker retrieves the transient AES key via a side channel, it is impossible to reverse the XOR operation to recover the individual keys stored in the vault. Consequently, an attacker cannot clone the IoT device or inject false messages.

- **Denial of Service (DoS)**: The system is designed to prevent resource exhaustion crashes caused by flooding attacks. The architecture ensures that the server does not assign significant resources to a connection request before the authentication phase is complete, making the protocol resilient to DoS attempts.

5 Conclusion

In this project, the vault-based mutual authentication protocol proposed by Shah et al. [1] has been successfully implemented and simulated. By leveraging a synchronized Secure Vault of random keys, the system demonstrates how robust security can be achieved in resource-constrained IoT environments without relying on computationally expensive public-key infrastructure.

The software simulation confirmed the correctness of the three-way handshake logic, proving that both the IoT device and the server can mutually verify identities and independently derive an identical session key. Furthermore, the implementation of the HMAC-based vault rotation mechanism ensures forward secrecy, significantly mitigating the risks of dictionary and replay attacks.

This work serves as a practical proof-of-concept for a lightweight, scalable, and dynamic authentication framework suitable for modern IoT ecosystems.

References

[1] T. Shah and S. Venkatesan, "Authentication of iot device and iot server using secure vaults," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 819–824, 2018.

[2] D. Baggio, "Iot secure vault authentication." `https://github.com/dvdbaggio/IoT-Secure-Vault-Authentication`, February 2026.