Advanced Text Message Encryption System

Leveraging Fernet and Key Strengthening in Python

-> Agenda

1. **The Solution:** Fernet Cryptography Suite.
2. **Advanced Feature 1:** Secure Key Derivation (PBKDF2).
3. **Advanced Feature 2:** Anti-Replay Protection (TTL).
4. Implementation & Summary.
5. Next Steps & Q&A.

-> The Challenge:

- **The Solution:** Fernet Cryptography Suite.
- **Advanced Feature 1:** Secure Key Derivation (PBKDF2).
- **Advanced Feature 2:** Anti-Replay Protection (TTL).
- Implementation & Summary.
- Next Steps & Q&A.

Presented by: AEGIS

The Secure Core: Fernet

Fernet is an "Opinionated" Specification that combines multiple primitives securely.

Components	Standard Used	Purpose
Confidentiality	**AES-128 (CBC Mode)**	Ensures the data remains secret.
Integritiy Check	**HMAC-SHA256**	Verifies the message content has not been altered in transit.
Transmission Safet	**URL-safe Base64**	Encodes the final output for safe transport in URLs or JSON.
Freshness Check	**Embedded Timestamp**	Used for Time-to-Live (TTL) and replay protection.

Key Requirement: Requires a single, shared 256-bit (32-byte) master key.

Key Strengthening: Password to Key

How we turn a passphrase into an unbreakable 256-bit Fernet Key.

The Standard: PBKDF2HMAC

1. **Passphrase Input:** A User-firendly, memorable sting (e.g., “my_secret\phrase”).
2. **The Salt:** A random, unique 16-byte value is generated. (Must be stored/sent alongside ciphertext, but is not secret).
3. **The Iterations:** The passphrase is computationally hashed **100,000 times.**
   - **Why?** This intentionally makes key derivation slow, rasing the cost/time neede for an attacker to brute-force the password exponentially.
4. **Output:** A Cryptographically stong, 256-bit Fernet key/

Key Takeaway: PBKDF2 defends against dictionary and rainbow-table attacks.

Defending Against Replay Attacks: Time-to-Live (TTL)

Ensuring the message is “fresh” and used only once

The Mechanism

1. **Encryption:** A UTC timestamp is embedded **inside** the encrypted payload.
2. **Transmisson:** The sender specifies a `ttl_minutes` value (e.g., 30 minutes).
3. **Decryption Check:** After integrity is verified, the system checks:

```
Current UTC Time > (Embedded Timestamp + TTL Duration)?
```

Security Benifit

If an attacker intercepts a valid message and tries to "replay" or resend it after the TTL, the Decryptor will **immediately reject the message** (prevents execution of old, valid commands).

Implementation: Python Flow & Security Summary

-> The ‘AdvancedtextEncryptor’ Class Flow

1. **Initialization:** Uses **Passphrase + Salt + PBKDF2** to derive the Fernet Key.
2. **Sender (`encrypt`):** Encrypts message + timestamp. Returns the transport JSON package.
3. **Receiver (`decrypt`):** Re-derives key, decrypts (**HMAC Integrity** check), and checks the **TTL** timestamp.

-> Summary of Security Layers

- **Confidentiality:** AES-128 in CBC mode.
- **Key Strength:** PBKDF2HMAC with 100k iterations.
- **Integrity:** HMAC-SHA256 authentication.
- **Replay Protection:** Time-to-Live (TTL) timestamp check.

Conclusion & Next Steps

Advancement & Future Enhancements

- **Key Management Upgrade:** Implement a **Diffie-Hellman Key Exchange** for secure initial key distribution.
- **Asymmetric Upgrade:** Integrate **RSA** or a public-key system for user identity and key wrapping.
- **Message Persistence:** ntegrate with **Firestore** or a database for secure storage.
- **Performance:** Optimize for streaming large data files > 1MB.

Questions & Discussion

Thank You