Key Security Considerations for Developers

When working with keys for nodes in an AVS, it is essential to consider the security aspects associated with key access and decryption. Keys should be encrypted either using a password or passphrase, understanding the unique security concerns posed by different access layers is crucial. By proactively addressing these concerns, you can enhance the overall security and integrity of the keys within your system:

- Prompt for the passphrase and store it in memory:
- In this scenario, the input must remain hidden to prevent the secret phrase from being stored in the terminal session or used buffer. Attackers might search for this secret in the buffer history. The key should not be stored locally or remotely unless encrypted via the AVS's proprietary methods.
- Request the path to a file containing the passphrase:
- Here, buffer vulnerability issues are absent unless the secret is printed or logged. However, an attacker with access to the machine running the AVS could potentially access this file.
- Retrieve the key remotely:
- Encrypting the validator key offers markedly improved protection when the decryption passphrase is stored remotely. Since the passphrase is not located within the validator client's storage, obtaining an unencrypted key from on-disk data becomes impossible. Instead, an attacker would need to execute considerably more advanced attacks, such as extracting the decrypted key from memory or impersonating the validator client process to receive the decryption key.
- Nonetheless, despite the increased difficulty, a sophisticated attack could still potentially acquire the validator key. Moreover, the user may inadvertently sign undesirable messages.
- Utilize remote signers:
- Employing remote signers involves delegating the signing process to an external service or device, which can offer additional security layers. The users are responsible for the availability and security of the remote signers, however, it is crucial to establish secure communication channels and verify the trustworthiness of the remote signer to prevent unauthorized access or tampering.

Supporting both local and remote signer methods is a good practice.

A good choice for a remote signer isWeb3signer :

- Open-source signing service developed under the Apache 2.0 license, developed by Consensys, and written in Java.
- Capable of signing on multiple platforms using private keys stored in an external vault, or encrypted on a disk.
- Can sign payloads using secp256k1 and BLS12-381 signing keys (AWS HSM can't at the moment, spring 2023).
- Web3Signer uses REST APIs, and all the major Ethereum Consensus clients support it.

Key Management Recommendation for Developers

The AVS can implement a feasible and sufficient method of loading the keys. This is asking for a path to a keystore folder. This keystore needs to follow some structure that AVS knows how to read. Currentlyeigenlayer-cli supports creation of encrypted ecdsa and bn254 keys in theweb3 secret storage format.

note By keys, we refer to any kind of secret, either in plain text or encrypted. The path to this keystore folder can be provided via an environment variable or argument. Previous Key Management Next Multisig Governance