

Vault Key Algorithm: RSA

This document describes the asymmetric encryption schema based on RSA keys implemented in VESVault (<https://vesvault.com>), and available through VESVault public APIs (<https://ves.host>).

REST API and libVES access:

- *algo*: string "RSA"
- *publicKey*: PEM encoded RSA public key (SPKI)
- *privateKey*: PEM encoded encrypted RSA private key (PKCS #8).
Recommended default symmetric algorithm for PKCS #8: AES-256-CBC
- Recommended default RSA modulus length: 2048 bits

Vault Entries for RSA Vault Keys deploy the following implementation of RSA OAEP encryption:

Encrypting a Vault Entry data:

If the length of the plaintext P exceeds $ML - 48$, where ML is the modulus byte length of the Vault Key V

- Generate a random 32-byte K , and a 12-byte $/V$
- Produce a padded plaintext PP :

$$PP = \{PL\} \parallel \{P\} \parallel \{\text{ignored padding } (PL \text{ bytes})\}$$

where PL is the padding length byte (0 .. 255), recommended value is to align PP to the next 32-byte boundary

- Encrypt the padded plaintext PP with AES-256-GCM, using the key K and the $/V$, result in ciphertext C and 16-byte GMAC value G
- Encrypt concatenated ($\{K\} \parallel \{/V\}$) with $\text{pub}(V)$ using OAEP padding, result in ciphertext CK , of fixed length ML
- Generate the Vault Entry structure:

$$D = \{CK\} \parallel \{C\} \parallel \{16\text{-byte } G\}$$

where " \parallel " denotes concatenation.

Otherwise

- Encrypt $\{P\}$ with $\text{pub}(V)$ using OAEP padding, result in ciphertext data D of fixed length ML

The result D is to be passed as Base64 encoded *encData* of the Vault Entry.

Decrypting a Vault Entry data:

If the length of the Base64 decoded data D exceeds ML

- Parse the ciphertext data D :

```
{CK (ML bytes)} || {C} || {G (16 bytes)}
```

- Decrypt CK with $\text{priv}(V)$ using OAEP padding, result in

```
{K (32 bytes)} || {IV (12 bytes)}
```

- Decrypt C using AES-256-GCM with the key K , and IV , result in the padded plaintext PP
- Validate the GMAC value G , return an error if not valid
- Restore the plaintext P by stripping padding from PP :

```
{PL (1 byte)} || {P} || {ignored padding (PL bytes)}
```

- Return P

Otherwise

- Decrypt the ciphertext D with $\text{priv}(V)$ using OAEP padding, return the resulting plaintext P