

VESrecovery Scramble Algorithm: RDX1.2

RDX1.2 is distributed scrambling algorithm, intended for using with Viral Encrypted Security (VES), and based on the considerations mentioned in <u>VES Whitepaper</u>.

RDX1.2 converts a Recovery Secret into a set of Recovery Tokens with a specified level of redundancy. The algorithm is cryptographically similar to <u>Shamir's Secret</u> <u>Sharing</u>, although it uses a different mathematical approach.

Algorithm version tag: "RDX1.2".

Scrambling the Recovery Secret:

Algorithm Inputs:

- R: (binary) The Recovery Secret.
- *n*: (int) Required number of tokens to achieve VESrecovery.
- N: (int) Total number of friends.

Assertions:

- $n \ge 1$ (n = 1 is a degenerate case, unsecure, highly discouraged).
- $N \geq n$.

Process:

- Generate a vector of bases b_i , $1 \leq i \leq N$.
- Produce the matrix of coefficients $C_{i,j} = b_i^{j-1}$, $1 \le i \le n$, $1 \le j \le n$.
- Generate an intermediate vector U_j :

 $U_1 = R$ $U_j = random(length(R)), 1 < j \leq n.$

• Generate the vector of variables V_j , $1 \le j \le n$:

```
V_j = AES256\_CBC\_encrypt(plaintext: U_j, key: U_{j+1}) || 0x01,
1 \le j < n
V_n = U_n || 0x01
```

(" | | " denotes concatenation).

The resulting values V_j are to be treated as little-endian signed integers of unlimited length (the last byte is the MSB, the highest bit of the last byte is the sign). The 0×01 byte is appended to mitigate denormalization in case MSB equals 0×00 or $0 \times ff$).



Multiply the matrix of coefficients C_{i,j} by the vector of variables V_j to produce the vector of tokens T_i.

Algorithm Outputs:

- T_i (array(binary)) Recovery tokens, $1 \leq i \leq N$.
- Metadata for each Token, to be passed to each Vault Item:
 - o v (string) Algorithm version tag, "RDX1.2".
 - o n (int)
 - o b (int) Base value for the Token, b_i .

Reconstructing the Recovery Secret:

(The input data are collected from tokens with matching algorithm version v, assert same value of n for each token).

Algorithm Inputs:

- *n*: (int) Required number of tokens to achieve VESrecovery
- T_i : array((binary)) The Recovery Tokens, $1 \leq i \leq m$.
- b_i (array(int)) Base values, corresponding to each T_i .

Assertions:

- $n \geq 1$.
- b_i are all distinct values.
- *m* ≥ *n*.

Process:

- Create a vector a_k as a subset of b_i , $1 \leq k \leq n$.
- Produce a square matrix $C_{k,j} = a_k^{j-1}, 1 \le k \le n, 1 \le j \le n$.
- Divide the vector of tokens T_i by the matrix C_{k,j}, using Gauss-Jordan reduction, or some other linear algebra approach. If the division is not possible fail.
 Resulting vector of variables V_j.
- Assert the normalized little-endian binary value of V_j ends with 0x01 byte.
 Otherwise fail.
- Reconstruct the intermediate vector U_j , $1 \leq j \leq n$.



```
U_n = (V_j // 0 \times 01)

U_i = AES256\_CBC\_decrypt(ciphertext: (V_j // 0 \times 01), key: U_{j+1}),

1 \le j < n
```

("//" denotes truncation of a trailing byte sequence)

• If any of the above steps fails – return to the first step and try a different subset a_k of b_i , until the possibilities are exhausted.

Algorithm Output:

• $R = U_1$ (binary) The Recovery Secret.

Additional considerations:

- If the resulting value R is found to be not a valid Recovery Secret try a different subset ak of bi, until the possibilities are exhausted.
- If the valid *R* is found and *m* > *n* foul check can be performed. Scramble the reconstructed vector of variables *V_j* for the supplied vector of bases *b_i*, using the corresponding steps of the Scrambling process, and compare resulting tokens to the supplied values *T_i*. If any mismatches are found, the corresponding friends can be flagged as foul.