Last Updated on 2020-06-06 (draft v0.0.3) based on ONE Wallet Wiki version 2020-05-23 (draft v0.1.4)

Definitions and common notations

Role	**Symbols**
User	$\mathbb{U}$
Client	$\mathbb{C}$
Google Authenticator	$\mathbb{A}$
Smart Contract Wallet	$\mathbb{S}$
(Trusted) Relayer	$\mathbb{R}$
Harmony Blockchain	$\mathbb{H}$
HMAC function in RFC2104 (with hash function h and key k)	H_h^k
Keccak256 Hash Function	h_3
SHA256 Hash Function	h_2
SHA1 Hash Function	h_1
Base32 Encoding	B_{32}
Concatenation of A and B	$A \oplus B$
Concatenation of $A_1, ..., A_n$	$\oplus_{i=1...n} A_i$
Offline Message Transmission (e.g. Air-gapped QR code scan)	$A \overset{\circ}{\rightarrow} B$
Online Message Transmission	$A \rightarrow B$
Sending Message m From A to B	$A \rightarrow B : m$
(Signed by) Private Key of A	$sk(A)$
(Encrypted by) Public Key of A	$pk(A)$
Message Containing A	$m\{A\}$
Message Signed by Secret S	$m\{...\}_S$
Message of Type t	m_t
Transaction of Type t	tx_t
ID of a Transaction of Type t	$id(tx_t)$
Function f on Smart Contract $\mathbb{S}$	$\mathbb{S}_f$
(Enum) Operation of Type t	o_t
Byte 0xff repeated for t times	$(0\text{xff})^t$
Bitwise AND between A and B	$A \wedge B$
Bitwise OR between A and B	$A \vee B$
Shift A by B bits to the left	$A \ll B$
Shift A by B bits to the right	$A \gg B$

Immutable Varibles

Visibility Hierarchy: $\mathbb{H} = \mathbb{S} > \mathbb{C} > \mathbb{A}$, where the relation $A > B$ should be interpreted as: if X is visible at A, it may also be made visible at B for the purpose of security analysis. On the other hand, if X is visible at B, it remains invisible at A.

Role	**Symbols**	Visibility
OTP Seed	k	$\mathbb{A}$
Hash of OTP Seed	k_h	$\mathbb{C}$
OTP Root Hash	r	$\mathbb{S}$
Last Resort Address	$addr_{recovery}$	$\mathbb{S}$
Wallet Effective Time	t_0	$\mathbb{S}$
Wallet Lifespan	T	$\mathbb{S}$
OTP Merkle Tree Height	d	$\mathbb{S}$
Number of OTP Merkle Leaves	n	$\mathbb{S}$
OTP Merkle Tree Leaves (i-th leaf)	L_i^0	$\mathbb{C}$
OTP Merkle Tree Nodes (i-th node at j levels above leaves)	L_i^{-j}	$\mathbb{S}$
Wallet Configuration	G	$\mathbb{S}$
Wallet Address	$addr(\mathbb{S})$	$\mathbb{S}$

Assigned Varibles

Role	**Symbols**	**Visibility**
Daily Spending Limit	$limit_{daily}$	$\mathbb{S}$
Low Transfer Limit	$limit_{low}$	$\mathbb{S}$
Medium Transfer Limit	$limit_{medium}$	$\mathbb{S}$
High Transfer Limit	$limit_{high}$	$\mathbb{S}$

Transient Varibles

Role	**Symbols**	**Used In**
i-th OTP Code	Q_i	$\mathbb{C}$
OTP Code for Time t	Q^t	$\mathbb{A}$, $\mathbb{C}$
Mapping time t to i-th OTP code	$\sigma(t)$	$\mathbb{C}$
Hashed Salted OTP Code	$\bar{Q}$	$\mathbb{C}$, $\mathbb{S}$
Transfer Amount	tr_{amount}	$\mathbb{C}$, $\mathbb{S}$
Transfer Destination	tr_{dest}	$\mathbb{C}$, $\mathbb{S}$
Merkle Branch	$\hat{P}^t$	$\mathbb{C}$
Merkle Path	P^t	$\mathbb{C}$
Projected OTP Merkle Tree Node	$\hat{L}_i^{-j}$	$\mathbb{S}$
Recorded Timestamp Entry	t'	$\mathbb{S}$
Timestamp for Block b	t_b	$\mathbb{S}$
Integer i in 4-byte Big Endian Layout	$\bar{i}$	$\mathbb{C}$

Helper Functions

Generate OTP codes given Base32-encoded OTP seed k, starting time t_0 (in seconds), and duration T

```
Function GenOTP(k, t_0, T):
    seed := B_32^{-1}(k)
    t := t_0
    codes := []
    while t < t_0 + T:
        c := (0xff)^8 ∧ (⊕_{j=1...8}[(0xff) ∧ (⌊t/30⌋ ≫ (8 − j))]
        hash := H_{h_1}^{seed}(c)
        append Truncate(hash) into codes
        t ← t + 30
    return codes

Function Truncate(h):
    p := h ∧ (0x0f)
    x_1 := h[p] ∧ (0x7f) ≪ 24
    x_2 := h[p + 1] ∧ (0xff) ≪ 16
    x_3 := h[p + 2] ∧ (0xff) ≪ 8
    x_4 := h[p + 3] ∧ (0xff)
    return x_1 ∨ x_2 ∨ x_3 ∨ x_4
```

1 Creating a Wallet

Triggered by the User at the Client. The User chooses wallet configuration parameters G (such as last-resort address $addr_{recovery}$, daily spending limit $limit_{daily}$, and other things), wallet lifespan T (in seconds, defaults to 1 year), and wallet effective time t_0 (in seconds, defaults to current time)

1.1 Actions by the Client

1. Generate a 20-byte long random string k'
2. Compute OTP Seed $k := B_{32}(k')$, and the hash of OTP Seed $k_h := h_2(k)$
3. Generate a URI link encoding the OTP seed k, using standard time-based OTP configuration (h_1 for hash function, 30-second refresh interval)
4. Create a QR-code from the URI (version 6, byte mode, low error correction)
   (a) If Client is an mobile app, display an auto-setup button deep linked to Google Authenticator app.
5. Without waiting for Client to scan the QR code or click the auto-setup button in previous step:
   (a) Compute the height of OTP Merkle Tree $d := \lceil \log_2(\frac{T}{30}) \rceil$ and the number of OTP Leaves required $n := 2^d$
   (b) Compute all OTPs $Q_{1...n}$ for time interval $[t_0, t_0 + T]$ using Algorithm $GenOTP(k, t_0, T)$
   (c) Compute all OTP Leaves for $i = 1...n$:
   $$L_i^0 = h_2(h_2(k_h \oplus Q_i))$$
   (d) Compute OTP Merkle Tree for $j = 1...d$ and $i = 1...\frac{n}{2^j}$:
   $$L_i^{-j} := h_2(L_{2i-1}^{-j+1} \oplus L_{2i}^{-j+1})$$
   (e) Set OTP Root Hash $r := L_1^{-d}$
6. Wait for User to confirm the completion of the actions in previous step.

7. Inform the selected Relayer to create the wallet:

$$\mathbb{C} \to \mathbb{R} : \{r, t_0, d, T, G\}$$

8. Destroy $Q_{1...n}$ and k. Store $r, k_h, t_0, d, \{L_i^{-j} : \ j = 1...d, \ i = 1...\frac{n}{2^j}\}$
9. Wait for confirmation from Relayer, and retrieve wallet address $addr(\mathbb{S})$ and transaction id $id(tx_{create})$ in the response.
10. Verify $id(tx_{create})$ is a completed transaction. Store $addr(\mathbb{S})$ and $id(tx_{create})$.

1.2 Actions by the User

1. Wait for Client to display QR Code or show auto-setup button
2. Setup the Authenticator by scanning the QR Code or click the auto-step button:

$$\mathbb{C} \overset{\circ}{\to} \mathbb{A} : \{k\}$$

3. (Optional) backup the Authenticator setup QR Code

$$\mathbb{C} \overset{\circ}{\to} \mathbb{U} : \{k\}$$

1.3 Actions by the Authenticator

1. Setup a new OTP code entry as per standard process based on OTP seed k_h

1.4 Actions by the Relayer

1. Wait for message $m_{create}\{r, t_0, d, T, G\}$ from Client
2. Send transaction $tx_{create} := \mathbb{S}_{new}(r, t_0, d, T, G)$ to Harmony blockchain and sign:

$$\mathbb{R} \to \mathbb{H} : \{tx_{create}\}_{sk(\mathbb{R})}$$

   to create a new wallet $\mathbb{S}$

3. Wait for confirmation from $\mathbb{H}$. Obtain transaction id $id(tx_{create})$ and smart contract wallet's address $addr(\mathbb{S})$
4. Return $id(tx_{create})$ to Client:

$$\mathbb{R} \to \mathbb{C} : \{addr(\mathbb{S}), id(tx_{create})\}$$

2 Transferring Funds (Simple)

This process is triggered by the User at the Client. The User chooses the amount tr_{amount} and the destination address tr_{dest}. Here, we assume completing the transfer would not exceed the daily spending limit set by the wallet, and tr_{amount} is below a limit that would require Composable Authentication.

2.1 Actions by the User

1. Obtain the current OTP code on the Authenticator: $\mathbb{A} \overset{\circ}{\to} \mathbb{U} : \{Q^t\}$ where t is current time, $Q^t := Q_{\sigma(t)}$, and $\sigma(t) := \lfloor \frac{t-t_0}{30} \rfloor$
2. Confirm transfer by providing the OTP code to Client: $\mathbb{U} \overset{\circ}{\to} \mathbb{C} : \{Q^t\}$

2.2 Actions by the Client

1. Wait for current OTP code Q^t from User. Compute $\bar{Q}^t := h_2(k_h \oplus Q^t)$
2. Denote $t' := \sigma(t)$. Construct Merkle Branch:

$$\hat{P}^t := \{\begin{cases} L_{t''}^{-j}, L_{t''+1}^{-j} & t'' \equiv 1 \mod 2 \\ L_{t''-1}^{-j}, L_{t''}^{-j} & t'' \equiv 0 \mod 2 \end{cases} : \; t'' := \frac{t'}{2^j}, \; j = 1...d\}$$

3. Construct Merkle Path:

$$P^t := \{L_i^{-j} : L_{2*i-1}^{-j+1} \notin \hat{P}^t \;\wedge\; L_{2*i}^{-j+1} \notin \hat{P}^t \;\wedge\; (i \neq t', j = 0)\}$$

   - i.e. remove a node if its child is also in Merkle Branch, and remove the leaf corresponding to current OTP

4. Construct Merkle Proof $(P^t, \bar{Q}^t))$ and message for commit $m_{commit} := h_3(P^t, \bar{Q}^t, tr_{amount}, tr_{dest})$
5. Send commit message to Relayer:

$$\mathbb{C} \to \mathbb{R} : \{m_{commit}, addr(\mathbb{S})\}$$

6. Wait for confirmation from Relayer and retrieve transaction id $id(tx_{commit})$ in the response.
7. Verify $id(tx_{transfer})$ is finalized and wait for 1 more block (2 seconds)
8. Send reveal message to Relayer:

$$\mathbb{C} \to \mathbb{R} : \{m_{reveal}, addr(\mathbb{S})\}$$

   where $m_{reveal} := \{P^t, \bar{Q}^t, tr_{amount}, tr_{dest}\}$

9. Wait for confirmation from Relayer and retrieve transaction id $id(tx_{reveal})$ in the response.
10. Verify $id(tx_{reveal})$ is finalized

2.3 Actions by the Authenticator

1. Generate OTP code Q^t as per standard process for current time t.

2.4 Actions by the Relayer

1. Wait for message $\{m_{commit}, addr(\mathbb{S})\}$ from Client
   (a) Send transaction $tx_{commit} := \{\mathbb{S}_{commit}(m_{commit})\}$ to contract $addr(\mathbb{S})$ on Harmony Blockchain:

$$\mathbb{R} \to \mathbb{H} \to \mathbb{S} : \{tx_{commit}\}_{sk(\mathbb{R})}$$

   (b) Wait for confirmation from $\mathbb{H}$ and obtain transaction id $id(tx_{commit})$
   (c) Return $id(tx_{commit})$ to Client: $\mathbb{R} \to \mathbb{C} : \{id(tx_{commit})\}$
2. Simutaneously, wait for message $\{m_{reveal}\{P^t, \bar{Q}^t, tr_{amount}, tr_{dest}\}, addr(\mathbb{S})\}$ from Client
   (a) Send transaction

$$tx_{reveal} := \{\mathbb{S}_{reveal}(P^t, \bar{Q}^t, tr_{amount}, tr_{dest})\}$$

   to $addr(\mathbb{S})$ on Harmony Blockchain:

$$\mathbb{R} \to \mathbb{H} \to \mathbb{S} : \{tx_{reveal}\}_{sk(\mathbb{R})}$$

   (b) Wait for confirmation from $\mathbb{H}$ and obtain transaction id $id(tx_{reveal})$
   (c) Return $id(tx_{reveal})$ to Client:

$$\mathbb{R} \to \mathbb{C} : \{id(tx_{reveal})\}$$

2.5 Actions by the Smart Contract

1. On invocation of $\mathbb{S}_{commit}\{m_{commit}\}$:
   (a) Add a map entry $m_{commit} \to t_b$ to commit-table of $\mathbb{S}$, where t_b is the current block's timestamp in seconds
2. On invocation of $\mathbb{S}_{reveal}\{P^t, Q^t, tr_{amount}, tr_{dest}\}$:
   (a) Lookup from commit-table at $\mathbb{S}$ for map entry
   $$t'_b := h_3(P^t, \bar{Q}^t, tr_{amount}, tr_{dest})$$
      i. If t_b does not exist, exit and emit error (rejection: transaction not committed)
      ii. If $t'_b < t_b - 30$ (where t_b is the current block's timestamp in seconds), exit and emit error (rejection: commit is too old)
   (b) Verify that
   $$\sum_{i,j:L_i^{-j+1} \in \bar{P}^t} 2^j \zeta(i) = \lfloor \frac{t'_b - t_0}{30} \rfloor$$
   where
   $$\zeta(i) = \begin{cases} 0 & i = 2m+1 \\ 1 & i = 2m \end{cases} \quad m \in \mathbb{Z}^+$$
   (c) Set
   $$\bar{P}^t := \{ \begin{cases} h_2(\bar{Q}^t) & j = 0, i = \sigma(t) \\ \hat{L}_i^{-j} & \text{otherwise} \end{cases} : \hat{L}_i^{-j} \in P^t \}$$
   Compute recursively:
   $$\hat{L}_i^{-j} := h_2(\hat{L}_{2i-1}^{-j+1} \oplus \hat{L}_{2i}^{-j+1})$$
   (d) Verify that $\hat{L}_1^{-d} = r$. If not successful, exit and emit error (rejection: bad proof)
   (e) Verify that the current balance of $\mathbb{S}$ is not less than tr_{amount}. Otherwise, exit and emit error (rejection: insufficient funds)
   (f) Verify that the total amount transferred for the current 24-hour window is not greater than $limit_{daily} - tr_{amount}$. Otherwise, exit and emit error (rejection: exceeds daily limit)
   (g) Proceed with sending tr_{amount} from $\mathbb{S}$ to tr_{dest}

3 Recovering Wallet

This process is triggered by the User at the Client.

3.1 From Authenticator

Here, it is assumed all information on the Client is lost.

3.1.1 Actions by the User

1. Go to Google Authenticator and click "... - Export Accounts".
2. Select the entry corresponding to the wallet and click "export"
3. Scan the displayed QR code using the new Client: $\mathbb{A} \overset{\circ}{\to} \mathbb{C} : \{k\}$
   (a) If the new Client is running on a device without cameras,
      i. Save the QR code as an image and transmit the image offline to the device running the Client.
      ii. Select the image using the file-selection prompt on the Client.

3.1.2 Actions by the Client

1. Wait for the User to trasnmit the QR code containing the OTP seed k
2. Compute the hash of OTP Seed $k_h := h_2(k)$.
3. Regenerate all information on the Client using the procedure in Section 1.1 Step 5.

3.2 From Client

In this case, it is assumed all information on the Authenticator is lost.

3.2.1 Actions by the User

1. Go to Client and click "I lost my authenticator. Transfer all my funds to the last resort address"
   (a) If last resort address is not set when the wallet was created, the user may choose a new address $addr'_{recover}$ to transfer the funds. However, the transfer would be subject to daily spending limit.

3.2.2 Actions by the Client

1. Check whether the wallet $\mathbb{S}$ has a last resort address $addr_{recover}$
2. If $addr_{recover}$ does not exist, set $tr_{address} := addr'_{recover}$ (provided by the User)
   (a) Initiate the transfer protocol in Section 2.2.
   (b) Brute-force current OTP code Q^t by enumerating for $0 \leq i < 10^6$, find i such that
   $$h_2(h_2(k_h \oplus \bar{i})) = L^0_{\sigma(t)}$$
   where:
   - $t$is current timestamp
   - $\sigma(t)$ is defined in Section 2.1 Step 1
   - $\bar{i}$ is the 4-byte big-endian representation of i, i.e.: $\bar{i} := (0\text{xff})^4 \wedge (\oplus_{j=1...4}[(0\text{xff})^1 \wedge (i \gg (4-j))]$

   (c) Continue with the protocol from Section 2.2 Step 2 using the brute-forced OTP code, the transfer address $tr_{address}$, and maximum transfer amount subject to daily limit $tr_{amount} := limit_{daily}$.
3. If $addr_{recover}$ does exist:
   (a) Brute force current OTP code Q^t as described above.
   (b) Initiate the transfer protocol in Section 2.2 using the brute-forced OTP code, but with following variations:
      i. The commit message in Step 4 is redefined as
      $$m_{commit} := h_3(P^t, \bar{Q}^t, o_{reocvery})$$
      where $o_{reocvery}$ is an enum value indicating this is an recovery operation
      ii. The reveal message in Step 8 is redefined as
      $$m_{reveal} := \{P^t, \bar{Q}^t, o_{reocvery}\}$$

3.2.3 Actions by the Relayer

The Relayer acts the same way as defined in Section 2.4 including the additional support for recovery operation as described above.

3.2.4 Actions by the Smart Contract

The Smart Contract follows the same protocol as described in Section 2.5, but with following variations:

1. On invocation of $\mathbb{S}_{reveal}\{P^t, Q^t, o_{recovery}\}$:
   (a) Follow the same protocol as described in Section 2.5 Step (a) to (d).
   (b) Transfer all remaining balance in $\mathbb{S}$ to $addr_{recover}$

4 Guardians

TODO. See design in Wiki Section Guardian and subsequent discussion in Composable Authenetication [Link]

4.1 Add Guardian

4.2 Remove Guardian

4.3 Recovering Wallet From Guardian

4.4 Confirm Spending Limit Adjustment

4.5 Confirm a Pending Operation as Composable Authentication Factor

5 Composable Authentication

TODO. See design in Wiki Section Composable Authenetication: [Link] and a previous discussion: [Link]

5.1 Activating Private Key Signature Authentication

5.2 Activating HOTP Authentication

5.3 Recovering Wallet From Composable Authentication

5.4 Confirm Spending Limit Adjustment

5.5 Confirm a Pending Operation