Components Interactions

In the previous section the components of the system were introduced. For each key feature of CAPE (mint, transfer, freeze, …), a sequence diagram describes how the components interact with each other. Note that while some details have been omitted for clarity, the figures below are faithful representations of what actually happens in the deployed system.

Mint

Alice mints 50 000alice coins for herself. If Alice wants to mint some tokens, she first needs to create a mint transaction (1). This transaction specifies who the minter is, who will receive the minted tokens, the amount of minted tokens and the asset type (denoted asalice_coin) of the tokens. Once the transaction is created, Alice's wallet will send it to the Relayer (2), along with the memo that contains the asset record in its encrypted form. Then (3) the Relayer will take the transaction and the memo and put them into a CAP block (in future versions of CAPE, the Relayer will pack several transactions into a block). This CAP block is submitted to the CAPE contract (4). If the block is valid, the CAPE contract will emit some event (5) containing the block and a list of additional items that will be discussed below. For now this list is empty. In the meanwhile the EQS constantly monitors the CAPE contract for new events, when such an event is emitted, the EQS forwards it to all the subscribed participants (6,7,8,9). Wallets when receiving new block event try to fetch the memos contained in the block and decrypt their content (10,11). If the decryption is successful the corresponding asset records are stored locally for further use (12).

Transfer

Alice sends 1000 units of some asset to Bob. The Auditor (or viewer) sees some of the information of the CAP transaction based on the policy of the type of the asset transferred. If Alice wants to transfer some assets to Bob, she first needs to get the encryption public key corresponding to Bob's address (1,2). This encryption public key will be needed to compute the receiver memo so that Bob can recover her asset records once the transfer transaction will have been processed by the CAPE contract. Building the transfer transaction requires Alice to provide her own address, Bob's address, the kind of asset and the amount to be transferred (3). The output of this computation will be the transfer transaction and two memos. One for Alice corresponding to the change and one for Bob. As with theMint feature, the CAP transaction and memos will be sent to the Relayer (4), packed into a block (5) and forwarded to the CAPE contract (6). Once the block is validated a new block event is emitted (7) and all the subscribed participants are notified (8,9,10,11). Alice and Bob respectively fetch the memos , decrypt them (12 resp 14) and update their local state accordingly (13 resp 15). Finally, an Auditor (or viewer) is able to view some of the fields of the transaction according to the policy of the asset being transferred. Such information can be stored locally for further analysis (step 16).

Freeze

The Freezer freezes some asset records belonging to Bob. Bob is not able to spend those records anymore. In some situations a special participant, the Freezer, may want to freeze some asset records. To do so, she will create afreeze transaction that specifies which asset record is to be frozen (1). As described above, this transaction will be sent to the Relayer (2) which will forward it to the CAPE contract under the form of a block (3,4). Once the transaction is validated by the CAPE contract, a new block event is emitted (5) and forwarded to the subscribed participants of the EQS (6,7,8,9). Now if Bob wants to spend his record by creating a transfer transaction (10,11,12,13), the CAPE contract will reject it as it is not possible to build a valid transfer transaction containing some frozen inputs (14). Note that in the current implementation the Relayer does not validate the transaction.

Sponsor, Wrap and Unwrap

Alice sponsors a new asset based on the USDC ERC20 token. Bob can now wrap USDC tokens into this new asset wrapper. At some point Bob can convert back (a.k.a unwrap) those asset records into the original ERC20 USDC tokens. CAPE allows not only to manipulate native assets, that is assets that are minted inside the system, but also ERC20 tokens. To achieve this, a user (e.g. Alice) can sponsor a CAPE asset which consists of providing a mapping between an ERC20 token (e.g. USDC) and an asset type definition. This mapping is received and stored by the CAPE contract (1,2).

Once the mapping is defined, any user like Bob can convert some ERC20 tokens (USDC) into some asset records that can be used inside the system. In practice Bob will deposit the ERC20 tokens and provide the corresponding asset record to the CAPE contract (3). The CAPE contract will verify the consistency between the ERC20 tokens transfers and the record (in particular amounts should match). Once the verification is successful, the CAPE contract will emit some event about the new wrapped asset record (4), and this event will be forwarded by the EQS (5,6). At this stage the asset record is not yet available for Bob, but placed in the queue of pending deposits inside the CAPE contract. Only when a new block is submitted and successfully validated (7,8,…,14) will Bob be able to be in condition to store his asset record (representing some ERC20 tokens) for further use (15).

At some point Bob may want to redeem his wrapped asset record for ERC20 tokens. This process is called unwrapping. To do so, Bob will create a burn transaction which is a transfer transaction of the wrapped asset record to some null address. This transaction also provides the Ethereum address to which the ERC20 tokens are sent. In the diagram Bob wants to unwrap his asset records using Eve's Ethereum address, to send them to Eve (16). The transaction after being sent to the Relayer (17) and forwarded to the CAPE contract (18) triggers an ERC20 token transfer corresponding to the record's data to Eve’s ethereum address (19).