Algorand Smart Contracts at Layer 1

By Silvio Micali

Algorand is the blockchain that enables the creation of frictionless financial assets. We are committed to providing our users the best and easiest-to-use tools to let them issue their financial products. And we do so at Layer 1!

As described in my previous blog post, tools implemented at Layer 1 avoid the complexities, expense, and security risks of smart contracts. Indeed, Layer-1 functionalities have the same security, efficiency, and atomicity as single-payment transactions. Part 1 of our Layer-1 Strategy consisted of creating both *Algorand Standard Assets and Atomic Transactions* (respectively, ASAs and ATs for short), both of which are implemented at Layer 1. Using these functionalities (which were called **fungible tokens and AMPTs in the earlier blog post**), Algorand users can, without relying on smart contracts, issue their own tokens (e.g., a new currency, points in a loyalty program, shares in an asset or a corporation, etc.). Moreover, they can realize any set of transfers, possibly involving multiple types of tokens, via a single transaction, thus preventing misbehaving users from refusing to go through with their transfers after getting what they wanted from other users.

We now want to move to Part 2 of our strategy: Algorand Smart Contracts at Layer 1, ASC1s for short. An ASC1 can be used for a host of valuable applications, including:

- *Post-and-sale transactions.* You enable any other user to purchase an item of yours at a given price, without any further involvement from you.
- *Securitized Loans.* You make a loan, secured by collateral of the borrower, so that you automatically get the collateral if the borrower fails to make payments according to a specified payment schedule.
- *Crowdfunding.* Without any further involvement of yours, you collect funds for a given project if a preset funding target is reached, and automatically return all money otherwise.
- *Accredited-Only Transactions.* You enable only qualified users to execute a certain kind of transaction, without having to approve each transaction separately.
- *Multi Multi-Sig Wallets.* You create a wallet where not only any k out of the n public keys of a given set A can authorize transactions (an ordinary multi-sig wallet), but also any k' of the n' of the public keys of another set B, and so on.

ASC1s can be applied to either individual transactions or accounts, respectively yielding ASC1-transactions and ASC1-accounts.

1. ASC1-TRANSACTIONS

In this section, we describe ASC1-transactions by focusing on the special but crucial case of money transfers (though such transactions can be used much more generally.)

In Algorand, money is associated with accounts. The most basic type of account consists of a public key (of the underlying digital signature scheme). And, the most basic way to transfer an amount of money x from an account A, whose owner is Alice, to another account B,whose owner is Bob, is an ordinary payment. Such a payment (1) is authorized by a direct digital signature of Alice, $SIG_A(A,x,B)$, and (2) is valid if A's balance is greater than or equal to x.

Alice, however, may not want to be the only one allowed to authorize a payment from her account A. For instance, as long as A has sufficient funds, Alice may *also* want to authorize (1) her spouse to transfer an arbitrary amount of Algos from A, (2) any of her children to transfer a maximum of 10 Algos at a time, or (3) her tax accountant to make an arbitrary donation to one of her favorite charities. In the classical financial system, she may accomplish this via some kind of power of attorney. In the Algorand blockchain, she may use an ASC1 to specify which additional public keys are authorized to transfer money from A, how much, and under what conditions.

An ASC1 may consist of an exhaustive list of various possibilities. However, this would be both cumbersome and limiting, because it may be hard for Alice to envisage all possibilities in advance. More generally, an *ASC1* consists of a simple computer program P that determines the conditions required for transfers from A to be valid (in addition to A having a sufficient balance).

Conceptually, an *ASC1-transaction* is a transaction T accompanied by an ASC1 that helps one to establish T's validity. The flow of information for an ASC1-transaction in which Alice allows user X to transfer money from her account A is as follows:

1. Alice chooses an ASC1 P, digitally signs it, and makes both P and $SIG_A(P)$ available.
2. To request posting on the chain a transfer$T$ of x Algos from account A to another account B, user X propagates P, $SIG_A(P)$, and $T=SIG_X(A,x,B)$.
3. P, SIGAP, and T are posted in a block of the Algorand blockchain only after the relevant block proposer and Algorand verifiers check that:
    - $SIG_A(P)$ is the digital signature of A's owner for ASC1 P;
    - $P(T)=TRUE$ — i.e., that T is a valid transfer; and
    - A's balance is greater than x. (Indeed, x Algos are automatically deducted from A's balance once T is posted.)

Remarks

- Alice may make P and $SIG_A(P)$ available in a variety of ways. For instance, she may post them on-chain, off-chain, or hand them directly to the users she wishes to authorize.
- A block may contain several ASC1-transactions, T_1T_2,..., relative to the same P, so long as each T_i is valid and A has a sufficient balance to handle T_i after the transfers T_1,..., T_{i-1} have been made.
- To be more compact, $SIG_A(P)$ may consist of $SIG_A(H(P))$, that is, it may be Alice's digital signature of the (collision resilient) hash of P.

2. ASC1-ACCOUNTS

Alice may use ASC1-transactions to enable certain other users to have (limited or full) access to account A that she continues to own, and from which she may continue to withdraw money. Alice may also wish, however, to create an account to which only certain other users, possibly excluding herself, are given (limited or full) access. For instance, she may want to put some money in escrow so that only Bob, under certain conditions, is capable of withdrawing money. She can accomplish this and other goals via an *ASC1-account*.

Alice sets up such an account by choosing an ASC1 P, and by transferring an initial amount ia of Algos (from one or more of her accounts) to the account $H(P)$. The program P determines which users can transfer money from account $H(P)$ and which transfers are valid.

Above, H is the SHA-512_256 hash function, and thus $H(P)$ is a 32-byte value, like any other address in the Algorand system. Syntactically, therefore, block proposers and Algorand verifiers may not realize that $H(P)$ is an ASC1-account instead of an ordinary account. The Algorand blockchain is permissionless and Alice—or anyone else—can introduce a new account by transferring to it some money from an existing account of hers. Only later, when presented with the ASC1 P, might users other than Alice realize that $H(P)$ is an ASC1-account.

Conceptually, the flow of information relative to an ASC1-account is as follows:

1. Alice chooses an ASC1 P, transfers some initial amount of Algos to $H(P)$, and makes P available.
2. To request posting on the chain a transfer T of x Algos from account $H(P)$ to another account B, a user propagates P and T.
3. P and T are posted in a block of the Algorand blockchain only after the relevant block proposer and Algorand verifiers check that:
    - $H(P)$ is the hash of the ASC1 P;
    - $P(T)=TRUE$ —i.e., that T is a valid transfer; and
    - $H(P)$'s balance is greater than x.

Remarks

- A block may contain several transactions, $T_1,T_2,\ldots,$ relative to the same ASC1-account $H(P)$, so long as each T_i is valid and $H(P)$ has a sufficient balance to handle T_i after the transfers $T_1,\ldots,T_{i-1}$ have been made.
- Recall that the hash function H is collision resilient. That is, it is computationally infeasible to find two strings a and b such that $H(a)=H(b)$. Accordingly, in setting up an ASC1-account $H(P)$, Alice is guaranteed that no one can find a different ASC1 P' capable of determining which transactions are valid for $H(P)$.

3. WRITING ASC1S IN TEAL

ASC1s are written in a special language: the *Transaction Execution Authorization Language*, TEAL for short. TEAL consists of 30 basic instructions, including

- $ADD(x,y)$, the binary operation that, given two integers x and y returns $a+b$;
- $HASH(x)$, the operation that, given a string x, returns the SHA-256 hash of x; and
- $SIG\ VER(pk,m,s)$, the predicate that returns TRUE if and only if s is the digital signature of a message m relative to a public key pk.

An ASC1 P consists of a sequence of TEAL operations. There are no "loops", no "go-back-to" instructions, and no retained states. If P consists of k TEAL operations, then each execution of P produces an output by performing no more than these same k operations in the same order. (TEAL does support a forward branch operation, so some operations may be skipped in some executions.)

Moreover, every ASC1 is at most 1KB-long and may include at most 10 SIG VER operations. (Indeed, signature verification is by far the most expensive operation. All other operations require a negligible amount of computation.)

These restrictions ensure that ASC1s can be handled at Layer 1, without slowing down the process of block generation.

4. THE POWER OF ASC1S IN LAYER 1

Like Algorand's atomic transactions (ATs), ASC1-transactions and ASC1-accounts provide a simple type of smart contracts. Yet, in combination with ATs, they succeed in implementing, at Layer 1, the majority of the applications requiring smart contracts today. Below, we provide just a simple example that showcases this power.

Post-and-sale transactions

Recall that a *post-and-sale transaction* enables any user to purchase an item of yours, at a price you choose, without any further involvement from you. At any time in which no one has yet bought your item, you can withdraw it from the sale.

Also, recall that Algorand's Atomic Transaction feature allows a group of transactions to be executed truly atomically. For example, assume a group that consists of just two transactions T_1 and T_2, then $AT(T_1, T_2)$ guarantees that either T_1 and T_2 are both executed or neither is executed. Taking advantage of this novel feature of Algorand's Layer 1, you can implement post-and-sale transactions as follows. This said, here is a way to implement a post-and-sale transaction via an ASC1-account. (Other implementations may not require ASC1-accounts.)

INTUITIVE DESCRIPTION

Put the asset for sale in an ASC1-account specifying two rules for withdrawal.

First, to allow anyone to buy the item, your ASC1 approves an Atomic Transaction $AT(T_1, T_2)$, where T_1 is any transaction transferring the asset from the ASC1-account to any other account, and T_2 is a payment to an address you specify for at least your asking price.

Second, your ASC1 includes a "cancel" clause, which, as long as the ASC1-account still owns the asset, allows you to move it away to any account of your choice. You do so by specifying, in the ASC1, a public key pk (whose private key only you know) and allowing a transaction to move the asset out of the ASC1-account if that transaction includes a signature s, such that $SIG\ VER(pk,cancel,s)$ returns true.

REMARK

If you have multiple identical items (represented by fungible tokens), you can post them all for sale with a single signed ASC1, which buyers can use until the supply runs out. In other words, Algorand's post-and-sale transaction allows you to implement a simple ICO.

STAY TUNED!

Over the following two weeks, we shall describe how to use Atomic Transactions and ASC1s to build additional and powerful applications, such as securitized loans and cross-chains swaps.