EFFICIENT SMART CONTRACTS AT SCALE: Algorand's Stateful TEAL Contracts

By Silvio Micali

Smart contracts are one of the most beautiful and powerful gifts of the blockchain to the world. But they are also technically very challenging. Traditional smart contracts are solely implemented at layer 2 and are slow, expensive, and fragile. By contrast, Algorand's smart contracts are very much non-traditional, efficient, and secure, and are implemented both at layer 1 and at layer 2.

Algorand's layer-1 smart contracts are executed at the very consensus layer, the most secure layer in any blockchain, without slowing down block production in the least. These smart contracts are thus as efficient and secure as ordinary payments. They are also called TEAL contracts, because they are written in a special language: *TEAL,* short for *Transaction Execution Approval Language*.

Algorand's layer-1 smart-contract platform has two components, namely:

1. *Stateless TEAL contracts.*
   This technology was released last November (2019) and has been discussed in my earlier blog.

2. *Stateful TEAL contracts.*
   This technology is being released this August (2020) and is the subject of this blog.

The next and final chapter in Algorand's smart-contract platform will consist of its *Smart^2 Contracts*. As discussed in my latest blog, such contract operates at level 2, but in fundamentally novel ways. This technology will be deployed next year.

Before describing Stateful TEAL, it is best to quickly recall its powerful "predecessor."

0. Stateless TEAL

Stateless TEAL contracts are very efficient logic programs *designed to either approve or deny a transaction at the time it is submitted*. They consist of an at most 1KB-long sequence of some 30 basic instructions, such as COMPARE (X,Y), where X and Y are integers, and VERIFY(X), where X is a digital signature. They are stateless, because they only access information available within the transaction itself, which is one of the reasons they are so efficient.

Stateless TEAL technology is very powerful, as it allows one to configure highly tailored restrictions on transactions and groups of transactions so as to remove high-cost intermediaries and to add unprecedented levels of transparency to often opaque economic activities. For example, with Stateless TEAL, you can post an item for sale along with a possible list of what you may accept in return. Having done this, no further interaction from you is needed: you might as well leave for vacation! The contract will ensure that your item will be automatically sold to the first user who accepts your offer. If your conditions are not fully met, no trade will happen, and your item will remain unsold. At the same time, the other party is protected too. Indeed, the trade is "atomic." Whoever the second party might be, both of you are guaranteed to get what you respectively want. If not, the status quo is maintained.

Additional examples of stateless TEAL contracts include collateralized loans, escrow systems, sets of complex and interdependent payments, and much more.

1. Stateful TEAL

Stateless TEAL contracts are great. But: what if you need a contract to act conditioned on information that is not a part of incoming transactions? For these applications, we have developed stateful TEAL programs.

As the name suggests, such programs maintain state and are written in TEAL. In fact, TEAL has been enriched with a few new instructions, not only to properly handle state information, but also to handle Merkle trees and other valuable functionalities.

THE CHALLENGE

Stateful Teal contracts must respect a fundamental Algorand tenet: ensuring that each block continues to be generated in a few seconds, with immediate transaction finality.

Guaranteeing such a performance is not easy, particularly if user costs must be kept low.

A FIRST, NAÏVE APPROACH

Storing state information directly on the blockchain is certainly the easiest approach, but hardly the most efficient one. In this approach, the state information of an application that runs for a long time would be dispersed over widely separated blocks, making it be impossible, at layer 1, to retrieve it without significantly slowing down block production. Furthermore, data, once written on the blockchain, cannot be changed, but efficiently updating state information requires over-writing. Simulating such over-writing by appending newer data will make this first approach enormously inefficient.

A SECOND, LOW-CONCURRENCY APPROACH

In another approach, an application may store its state information off-chain, while keeping only a commitment to (for example the Merkle hash of) this information on the chain itself, in order to guarantee its integrity. In this approach, changing a piece of data from, say, x to y, requires presenting a proof of the value of x, relative to its previous commitment, and generating and storing a new commitment that reflects the replacement of x with y.

This way of proceeding, however, does not allow for much concurrency. Consider two transactions each of whose execution is requested at a given point in time. The first transaction requires changing a piece of the state from x to y, thus presenting a proof of the value of x relative to the latest commitment of the state information. The second requires changing another piece of state from u to v. Each of these transactions may be valid, not only individually, but also if it executed after the other has been executed. In this second approach, however, whichever of the two transactions is executed first will delay the execution of the other! Assume that the commitment of current state of the application is C, and that the first transaction is executed first. Then, after presenting a proof, relative to C, of the correctness of the value x, the first transaction (a) off chain, changes x with y, and (b) on chain, changes the commitment to the application's state from C to C'. Thus, once the second transaction arrives, it will not be executed because its proof of the correctness of the value u is no longer valid. Indeed, such a proof remains relative to the previous commitment C, rather than being relative to the new commitment C'!

If scalability were not a goal, this second approach might, with additional effort, be extended to cope with a few transactions per second. But it would fail to handle hundreds of transactions per second. The second approach is unacceptable because it cannot handle concurrency at scale. We need a different solution.

ALGORAND'S APPROACH

Algorand solves these problems by storing the application's state in the *accounts* of the relevant parties: namely, the creator of the application and the users who have chosen to participate.

A first reason for choosing this approach is that, in Algorand, accounts are extremely efficiently generated by their owners, efficiently and transparently consulted by everyone, and altogether cheap to operate. So much so that Algorand does not impose any fee to maintain an account. The only requirement is to keep a very modest balance of 0.1 Algos. If an account owner chooses to create an application, then an additional balance of 0.1 Algos is required, enabling the account to maintain up to 64 pairs (x, y), where both x and y are 64-byte long strings. Each additional pair increases the required balance by 0.05 Algos.[1]

[1] Actually, the balance increases may be of just 0.035 Algos, if the value stored is an integer rather than an arbitrary 64-byte string.

Typically, x specifies a public key and y an amount of a given fungible token (or a specific non-fungible token) controlled by key x. Nevertheless, no restrictions are imposed on either x or y. Thus, in a stateful TEAL application, (x, y) is a *variable-value* pair: that is, x is the name of a variable, and y its current content. These variables and contents store the application's state information. Specifically, the account of the application creator stores the application's *global state*, and the accounts of all users who have opted in collectively store the application's *local state*.

Note that anyone can access the information stored in any given account. However, only the application can modify its own global and local states. This is no different from the way that a user's account stores the number of Algos she owns, but the user herself cannot modify (e.g., increase) the number of her own Algos!

CONCURRENCY

Also note that the Algorand approach allows for a great amount of concurrency. Indeed, two valid transactions are free to modify the application's internal state legitimately without one blocking the other. Such concurrency is indeed a main advantage of sharding the application state information among the accounts of all its users, an advantage that becomes increasingly significant as the number of users and transactions grows.

RESOURCES, EFFICIENCIES, AND COSTS

A stateful TEAL application does not have an arbitrarily large state. As mentioned, the application creator's account can store up to 64 variable-value pairs of global state. And each opting-in account can store up to 16 variable-value pairs of local state. These storage limitations are necessary to guarantee that Algorand's stateful TEAL contracts do not slow down block generation. In fact, this approach allows Algorand to handle, *at layer 1!,* one thousand stateful TEAL transactions per second, no matter how large the number of opting-in users may be.

Although the amount of local state per account may appear small, note that the total local state information grows linearly with the number of opting-in users. Thus, it will be big if there are many users, making it plausible that the amount of available state information may be adequate in many applications. In any case, once more, the proof is in the pudding: as we shall see in section 4, stateful TEAL can successfully handle many crucial applications.

As for costs, as already mentioned, an opting-in account does not pay any fee to store its portion of local state. Rather, it is required to keep an additional balance of 0.05 Algos for each variable-value pair of local state it is required to keep. Such additional balance requirement is immediately lifted the moment the account opts out of the application, by posting a proper transaction on the chain.

In sum, stateful TEAL is a major addition to Algorand's layer-1 technology and enjoys the scale and the great economic efficiency that Algorand users are accustomed to.

2. Examples of Stateful TEAL Applications

Stateful TEAL contracts can efficiently and securely handle all kinds of applications. Below are just a few ones.

Auctions

Stateful TEAL enables a variety of auctions. Let us consider, for example, a Dutch auction.

Suppose I have N new tokens to sell via a Dutch auction. Then, I create a new stateful TEAL application, with three global state variables:

(a) the available number of tokens, N,
(b) the current price P, initialized to some starting price, and
(c) the total committed money, CM, initially set to 0.

At periodic intervals (e.g., every 50 blocks), my application lowers the current price P by a given amount (e.g. 0.1 Algos), until some reserve price is reached or the entire supply of tokens can be sold based on the current bids.

Every opting-in user is able to submit a bid at the current price, because P is part of the global state and is therefore public. Specifically, a user's call to my application (1) specifies a bid consisting of an amount of Algos, A, and a maximum price, B, and (2) transfers A Algos to an escrow account. By these actions, the user shows her commitment to use up to A Algos to buy tokens at a maximum price of B Algos per token (or, of course, at a lower price). The user-chosen price B is typically equal to the current price P, but could be higher.

In response to her call, upon verifying the transfer of A Algos and making sure that B is at least equal to the current price P, my application

- stores the numerical value A into the user's local state, and
- increases by A the global-state variable CM.

Note that the reason that the user needs to specify her maximum price per share B is that, due to asynchrony, the price P may become lower by the time the user's bid enters the blockchain.

Of course, a user can post multiple bids. In fact, up to 16 bids, if she is using a single account. If she wants to add more bids, she can open additional accounts and post all the bids she wants.

This bidding process continues until the ratio CM/P becomes at least N. At that point, the application stops lowering the current price, and the clearing price C coincides with the price P of the last bid.

Once the bidding ends, something that can easily be detected from the global state, every user who previously submitted a bid can now invoke the application again. Avoiding "corner cases" that can be properly handled anyway, the post-bidding activity works as follows. For each of her bids, the application determines, based on her local state and the global state, whether the bid won her some tokens or not. In the latter case, the A Algos of her bid are immediately freed from escrow and returned to her account. In the former case, the application computes how many tokens the user has actually won, at the clearing price, due to her winning bid. (In absence of ties and other corner cases, the bid wins A/C tokens for its user.) This number of tokens will be immediately transferred from my account to hers, and the A Algos she originally put in escrow are immediately released and transferred to my account.

NOTE: Algorand's ability to handle auctions at layer 1 was indeed one of the main reasons for the Government of the Marshall Islands to select Algorand for its CBDC.

Security Tokens

Algorand Standard Assets (ASAs) enable a user to create tokens with specialized manager, freeze, reserve, and clawback addresses. ASAs solve the majority of security-token use cases without needing to write any contract code. However, your use case may require more than ASAs. For instance, you may want to launch a security token requiring customized restrictions on who can trade the token with whom.

Algorand's stateful TEAL contracts allow you to do just that. At the highest level, the contract's global state may specify the general parameters of the token, such as the manager and freeze addresses, the matrix of authorization groups, and how these groups can trade with each other while the assigned authorization group of each account is stored in the account's own local state.

NOTE: Algorand's stateful TEAL ability to handle such complex functionalities at layer 1 is indeed one of the main reasons for which Republic, a consortium of hundreds of thousands of accredited investors, has chosen Algorand to launch its own security token.

Crowdsourcing

A user can use Algorand's stateful TEAL account in order to build her *gofundme* application. If the fund goal is met, the application creator will be allowed to claim the funds. Otherwise, the funds are returned to the donors.
In this example, the application uses the global state to keep track of the fund goal, the running total of donations, as well as start and end times. When an opting-in user donates, her donation amount is recorded in her local state. A separate, stateless contract holds the total donations, with logic that allows for spending based on the aforementioned conditions.

Decentralized Exchanges

Atomic transfers, a major prior example of Algorand's *stateless* TEAL technology, enable network users to trade specific assets they own in a most secure and efficient way, without relying on any third parties. In order to use an atomic transfer, however, you must know who is willing to sell what and for what price. But what if you do not have such knowledge?

Stateful TEAL applications can fill this gap by acting as an order book. A user can post an order to buy or sell which is stored in their local storage for that order book application. The application's logic ensures that transfers can only occur if both the buyer's and the seller's conditions are met. A separate, stateless TEAL contract authorizes the transfer from a user account if her own sell/buy conditions and those of the other user's are met.

Transparent Banking

Algorand freely allows users to store Algo and other asset balances. But that stored value is going to waste just sitting there. Here is where an Algorand banking application can provide value for both the asset owners and prospective businesses or for individuals looking for loans or financial aid.

In essence, a user decides to hand over a set of funds to a banking application they trust, thanks to the full transparency around the application that the blockchain provides. The banking application keeps track of the held balance in the user's local storage, and deposits interest payments to that user for holding her money. Separately, the bank could store the total held value and its reserve amount in a global variable, which could be tied to a separate, stateless account that holds the bank's reserve. The logic of both the stateful and stateless smart contracts ensures that (1) the bank cannot go below a certain minimum in its reserves, and (2) the users can withdraw funds (with proper restrictions) freely. Everything is on the blockchain, safe and transparent.

Conditional Asset Sales

Imagine having the ability to enforce that any sale of a particular asset requires a 5% commission back to the asset creator, or perhaps to a government entity as some type of tax. This condition is difficult to enforce off the blockchain since we must rely solely on the honesty of sellers and buyers, and on our legal institutions, which are not always as speedy as desired. On Algorand, we can bolster this application with smart-contract logic that secures such honesty. Specifically, I can create an asset, frozen by default, that can only be unfrozen and transferred when grouped with a set of transactions that ensure that a commission payment is also sent. In particular, one of those group transactions is a call to the governing application that, using its global state to specify the asset ID and any conditions required for transferring the asset, can check that those conditions are indeed met. The transaction group would also include transactions to unfreeze the asset, prior to checking the conditions, and then to refreeze the asset after the transfer.

3. PyTEAL: Writing Stateful TEAL Contracts in Python

Writing smart contracts can be difficult. The code has to approve or reject transactions carefully, and may implement meticulous dApps. It is thus much easier to write smart contracts in Python rather than using TEAL opcodes, much like it is easier to write in a high level language rather than assembly. For these reasons, Algorand has developed PyTEAL.

THE ORIGINAL PyTEAL

Earlier this year, Algorand introduced PyTEAL, python bindings for its stateless smart contracts platform. Instead of dealing with basic arithmetic and condition operations, pushing and popping variables from the stack, PyTEAL allows the developer to create objects representing local variables, and run operations such as arithmetics, comparisons, signature verification, and hashing on them. The python code then compiles into TEAL. Essentially, PyTEAL gives the developer the *full power* of TEAL at the *full speed* of Algorand.

THE NEW PyTEAL

Today, we extend PyTEAL to stateful smart contracts. Thus, developers can keep focus on the application's logic while PyTEAL simplifies access to the application's local and global state. Essentially, PyTEAL provides a simple Pythonic API that allows to Put, Get, and Delete to the smart contract's state. Under the covers, PyTEAL translates these commands to the TEAL opcodes that implement them. Therefore, it allows writing sophisticated applications in a much simpler way and with comparatively few lines of code. As one example, we've used PyTEAL to create a smart contract implementing security tokens. This example is available online: https://github.com/jasonpaulos/pyteal/blob/logicsig-v2-backup/examples/security_token.py

WHY TO LOVE PyTEAL

PyTeal is great:

- Modularity - Easily organize smart contract code into logical sections using a popular, well-known language.
- Interoperability - As its own Python library, use it alongside other Python libraries and leverage their capabilities for more productive coding.
- Simplicity - Easily import data across multiple smart contracts in a single development environment.

In Sum

In sum, stateful TEAL and the new PyTEAL extension make Algorand's layer-1 smart contracts not only more powerful and efficient than ever, but also easier to use.

So, do not walk, but *run* to use them!

SILVIO MICALI | Founder, Algorand

Silvio Micali has been on the faculty at MIT, Electrical Engineering and Computer Science Department, since 1983. Silvio's research interests are cryptography, zero knowledge, pseudorandom generation, secure protocols, and mechanism design and blockchain. In particular, Silvio is the co-inventor of probabilistic encryption, Zero-Knowledge Proofs, Verifiable Random Functions and many of the protocols that are the foundations of modern cryptography.

In 2017, Silvio founded Algorand, a fully decentralized, secure, and scalable blockchain which provides a common platform for building products and services for a borderless economy. At Algorand, Silvio oversees all research, including theory, security and crypto finance.

Silvio is the recipient of the Turing Award (in computer science), of the Gödel Prize (in theoretical computer science) and the RSA prize (in cryptography). He is a member of the National Academy of Sciences, the National Academy of Engineering, the American Academy of Arts and Sciences and Accademia dei Lincei.

Silvio has received his Laurea in Mathematics from the University of Rome, and his PhD in Computer Science from the University of California at Berkeley.