Limitations

The Aztec Sandbox and the Aztec Smart Contract Library areprototypes , and should be treated as such. They've been released early, to gather feedback on the capabilities of the protocol and user experiences.

What to expect?

- Regular Breaking Changes;
- Missing features;
- Bugs;
- An 'unpolished' UX;
- Missing information.

Why participate?

Front-run the future!

Help shape and define:

- Previously-impossible smart contracts and applications
- Network tooling;
- Network standards;
- Smart contract syntax;
- Educational content;
- Core protocol improvements;

Limitations developers need to know about

- It is a testing environment, it is insecure, unaudited and does not generate any proofs, its only for testing purposes;
- Constructors can not call nor alter public state* The constructor is executed exclusively in private domain, WITHOUT the ability to call public functions or alter public state. This means to set initial storage values, you need to follow a pattern similar toproxies in Ethereum
-
  - , where youinitialize
-
  - the contract with values after it have been deployed, seeinitializer functions
-
  - .
-
  - Beware that what you think of as aview
-
  - could alter state ATM! Notably the account could alter state or re-enter whenever the account contract'sis_valid
-
  - function is called.
- msg_sender
- is currently leaking when doing private -> public calls* Themsg_sender
-
  - will always be set, if you call a public function from the private world, themsg_sender
-
  - will be set to the private caller's address. Seefunction context
-
  - .
- The initialmsg_sender
- is 0, which can be problematic for some contracts, seefunction visibility
- .
- Unencrypted logs don't link to the contract that emitted it, so essentially just a `debug_log`` that you can match values against.
- A note that is created and nullified in the same transaction will still emit an encrypted log.
- A limited amount of new note hashes, nullifiers and calls that are supported by a transaction, seecircuit limitations
- .

Limitations

There are plans to resolve all of the below.

It is not audited

None of the Sandbox code is audited. It's being iterated-on every day. It will not be audited for quite some time.

No Proofs

That's right, the Sandbox doesn't actually generate or verify any zk-SNARKs yet!

The main goal of the Sandbox is to enable developers to experiment with building apps, and hopefully to provide feedback. We want the developer experience to be as fast as possible, much like how Ethereum developers use Ganache or Anvil to get super-fast block times, instead of the slow-but-realistic 12-second block times that they'll encounter in production. A fast Sandbox enables fast testing, which enables developers to iterate quickly.

That's not to say a super-fast proving system isn't being worked on as we speak .

What are the consequences?

By the time mainnet comes around, zk-SNARKs will be needed in order to validate the correctness of every function that's executed on Aztec. In other words, in order for the execution of a function to be registered as part of the Aztec blockchain's history, a proof of correct execution will need to be furnished. Each proof will be an attestation that the rules of a particular function were followed correctly.

But proofs are really only needed as a protection against malicious behavior. The Sandbox is an emulated ecosystem; entirely contained within your laptop, and it follows the network's rules out of the box. So as long as its inner workings aren't tampered-with, it will act 'honestly'. Since you'll be the only person interacting with the Sandbox on your own laptop, and with a healthy assumption that you should be honest with yourself, you won't need proofs when testing.

No Circuits

This is kind-of a repetition of 'No Proofs!' above, but for the sake of clarity, there aren't yet any arithmetic circuits in the Sandbox. We might refer to certain components of the core protocol as being 'circuits', and we might refer to user-defined smart contract functions as being compiled to 'circuits', but the Sandbox doesn't actually contain any circuits yet. Instead, there is code which emulates the logic of a circuit. This is intentional, to make execution of the Sandbox as fast as possible.

Obviously, as development continues, the so-called 'circuits' will actually become proper circuits, and actual proofs will be generated.

What are the consequences?

The Sandbox will execute more quickly. The logic of all 'circuits' is still in place. Smart contract logic will be executed, and core protocol logic will be executed . So invalid transactions will be caught*and rejected.

*Note: some core protocol circuit assertions and constraints still need to be written (see GitHub). This would be bad in an adversarial environment, but the Sandbox is not that. Naturally, proper circuits will need to be written.

No Fees

That's right, there are no L2 network fees yet!

The Sandbox can currently be thought of as a bare-minimum execution layer. We'll be speccing and implementing gas metering and fees soon!

Note: there is still a notion of an L1 fee in the Sandbox, because it uses Anvil to emulate the Ethereum blockchain.

What are the consequences?

Apps won't yet be able to allow for any L2 fee logic. Once fees are introduced, this will cause breaking changes to in-progress apps, which will need to be updated to accommodate the notion of paying network fees for transactions. Clear documentation will be provided.

Basic Keys and Addresses

The way in which keypairs and addresses are currently derived and implemented (inside the Sandbox) is greatly over-simplified, relative to future plans.

They're so over-simplified that they're known to be insecure. Other features have been prioritized so-far in Sandbox development.

What are the consequences?

This will impact the kinds of apps that you can build with the Sandbox, as it is today:

- The management of keys when designing account contracts and wallets will be affected.
- The keys used when generating nullifiers will be affected. (Although the machinery relating to nullifiers is mostly abstracted away from developers who useAztec.nr
-).* In particular the current, over-simplified key derivation scheme is known to beinsecure
-
    - :* Currently, the same nullifier secret key is used byevery
-
    -
        - smart contract on the network. This would enable malicious apps to trivially emit a user's nullifier secret key to the world!
-
    -
        - In future, there are detailed plans to 'silo' a nullifier key per contract address (and per user), to fix this obvious vulnerability.
- The keys used when encrypting and decrypting logs will be affected.* In particular the current, over-simplified key derivation scheme is known to beinsecure
-
    - :* Currently, a user's nullifier secret key is the same as their encryption secret key. And as stated above, this would enable malicious apps to trivially emit a user's secret key to the world!
-
    - In future there are also plans to have incoming and outgoing viewing keys, inspired byZCash Sapling
-
    - .
-
    - If developers wish to design apps which incorporate certain auditability patterns, the current over-simplification of keys might not be sufficient.

Please open new discussions ondiscourse or open issues ongithub , if you have requirements that aren't-yet being met by the Sandbox's current key derivation scheme.

It's not-yet decentralized

It's an emulated blockchain entirely contained within your own laptop! It's centralized by design! As for deploying this all to mainnet, a decentralized sequencer selection and prover selection protocols are stillbeing discussed . There are plans for decentralized testnets in 2024.

You can't read mutable public state from a private function

Private smart contract functions won't be able to read mutable public state yet. We have someideas for how to solve this, and will look to implement something very soon.

What are the consequences?

Reading public state from a private contract will be a common pattern. For example, it's needed if you want to maintain a public whitelist/blacklist, but prove you are/aren't on that blacklist privately. This will be a high priority, coming soon.

No delegatecalls

A contract can't perform a delegatecall yet (if ever). Delegatecalls are quite a controversial feature of the EVM.

No privacy-preserving queries to nodes

Ethereum has a notion of a 'full node' which keeps-up with the blockchain and stores the full chain state. Many users don't wish to run full nodes, so rely on 3rd-party 'full-node-as-a-service' infrastructure providers, who service blockchain queries from their users.

This pattern is likely to develop in Aztec as well, except there's a problem: privacy. If a privacy-seeking user makes a query to a 3rd-party 'full node', that user might leak data about who they are, or about their historical network activity, or about their future intentions. One solution to this problem is "always run a full node", but pragmatically, not everyone will. To protect less-advanced users' privacy, research is underway to explore how a privacy-seeking user may request and receive data from a 3rd-party node without revealing what that data is, nor who is making the request.

No private data authentication

Private data should not be returned to an app, unless the user authorizes such access to the app. An authorization layer is not-yet in place.

What are the consequences?

Any app can request and receive any private user data relating to any other private app. Obviously this sounds bad. But the Sandbox is a sandbox, and no meaningful value or credentials should be stored there; only test values and test credentials.

An auth layer will be added in due course.

No bytecode validation

Bytecode should not be executed, unless the Sandbox has validated that the user's intentions (the function signature and contract address) match the bytecode.

What are the consequences?

Without such 'bytecode validation', if the incorrect bytecode is executed, and that bytecode is malicious, it could read private data from some other contract and emit that private data to the world. Obviously this would be bad in production. But the Sandbox is a sandbox, and no meaningful value or credentials should be stored there; only test values and test credentials.

There are plans to add bytecode validation soon.

Insecure hashes

Currently, Pedersen hashes are being used pretty-much everywhere. To any cryptographers reading this, don't panic. A thorough review of which hashes to use in which part of the protocol will be conducted soon.

Additionally, domain separation of hashes needs some review.

What are the consequences?

Collisions and other hash-related attacks might be possible in the Sandbox. Obviously that would be bad in production. But it's unlikely to cause problems at this early stage of testing.

msg_sender

is leaked when making a private -> public call

There areongoing discussions around how to address this.

What are the consequences?

When a private function makes a call to a public function, themsg_sender of the calling function will be given to the public world. Most critically, this includes if themsg_sender is an account contract. This will be patched in the near future, but unfortunately, app developers might need to 'overlook' this privacy leakage until then, with the assumption that it will be fixed. But note, one possible 'patch' might be to setmsg_sender to0 for all private -> public calls. This might cause breaking changes to your public functions, if they rely on readingmsg_sender . There are patterns to work around this, but they wouldn't be pretty, and we won't go into details until a solution is chosen. Sorry about this, and thanks for your patience whilst we work this out :)

New Privacy Standards are required

There are manypatterns which can leak privacy, even on Aztec. Standards haven't been developed yet, to encourage best practices when designing private smart contracts.

What are the consequences?

For example, until community standards are developed to reduce the uniqueness of'Tx Fingerprints' app developers might accidentally forfeit some function privacy.

Circuit limitations

Upper limits on function outputs and tx outputs

Due to the rigidity of zk-SNARK circuits, there are upper bounds on the amount of computation a circuit can perform, and on the amount of data that can be passed into and out of a function.

Blockchain developers are no stranger to restrictive computational environments. Ethereum has gas limits, local variable stack limits, call stack limits, contract deployment size limits, log size limits, etc. Here are the current constants:

```
constants // "PER CALL" CONSTANTS global MAX_NEW_NOTE_HASHES_PER_CALL :

u64

=

16 ; global MAX_NEW_NULLIFIERS_PER_CALL :

u64

=

16 ; global MAX_PRIVATE_CALL_STACK_LENGTH_PER_CALL :

u64

=

4 ; global MAX_PUBLIC_CALL_STACK_LENGTH_PER_CALL :

u64

=

4 ; global MAX_NEW_L2_TO_L1_MSGS_PER_CALL :

u64

=

2 ; global MAX_PUBLIC_DATA_UPDATE_REQUESTS_PER_CALL :

u64

=

16 ; global MAX_PUBLIC_DATA_READS_PER_CALL :

u64

=

16 ; global MAX_NOTE_HASH_READ_REQUESTS_PER_CALL :

u64

=

32 ; global MAX_NULLIFIER_READ_REQUESTS_PER_CALL :

u64

=

2 ;

// Change it to a larger value when there's a seperate reset circuit. global
MAX_NULLIFIER_NON_EXISTENT_READ_REQUESTS_PER_CALL :

u64

=

2 ; global MAX_NULLIFIER_KEY_VALIDATION_REQUESTS_PER_CALL :

u64

=

1 ;
```

```
// "PER TRANSACTION" CONSTANTS global MAX_NEW_NOTE_HASHES_PER_TX :
u64
=
64 ; global MAX_NON_REVERTIBLE_NOTE_HASHES_PER_TX :
u64
=
8 ; global MAX_REVERTIBLE_NOTE_HASHES_PER_TX :
u64
=
56 ;
global MAX_NEW_NULLIFIERS_PER_TX :
u64
=
64 ; global MAX_NON_REVERTIBLE_NULLIFIERS_PER_TX :
u64
=
8 ; global MAX_REVERTIBLE_NULLIFIERS_PER_TX :
u64
=
56 ;
global MAX_PRIVATE_CALL_STACK_LENGTH_PER_TX :
u64
=
8 ;
global MAX_PUBLIC_CALL_STACK_LENGTH_PER_TX :
u64
=
8 ; global MAX_NON_REVERTIBLE_PUBLIC_CALL_STACK_LENGTH_PER_TX :
u64
=
3 ; global MAX_REVERTIBLE_PUBLIC_CALL_STACK_LENGTH_PER_TX :
u64
=
5 ;
global MAX_PUBLIC_DATA_UPDATE_REQUESTS_PER_TX :
u64
=
32 ; global MAX_NON_REVERTIBLE_PUBLIC_DATA_UPDATE_REQUESTS_PER_TX :
u64
=
16 ; global MAX_REVERTIBLE_PUBLIC_DATA_UPDATE_REQUESTS_PER_TX :
u64
=
16 ;
global MAX_PUBLIC_DATA_READS_PER_TX :
u64
=
32 ; global MAX_NON_REVERTIBLE_PUBLIC_DATA_READS_PER_TX :
u64
=
16 ; global MAX_REVERTIBLE_PUBLIC_DATA_READS_PER_TX :
u64
=
16 ;
global MAX_NEW_L2_TO_L1_MSGS_PER_TX :
u64
=
2 ; global MAX_NOTE_HASH_READ_REQUESTS_PER_TX :
u64
=
128 ; global MAX_NULLIFIER_READ_REQUESTS_PER_TX :
u64
=
8 ;
// Change it to a larger value when there's a seperate reset circuit. global
MAX_NULLIFIER_NON_EXISTENT_READ_REQUESTS_PER_TX :
u64
=
8 ; global MAX_NULLIFIER_KEY_VALIDATION_REQUESTS_PER_TX :
u64
=
4 ; global NUM_ENCRYPTED_LOGS_HASHES_PER_TX :
u64
=
1 ; global NUM_UNENCRYPTED_LOGS_HASHES_PER_TX :
```

u64

=

1 ; Source code: noir-projects/noir-protocol-circuits/crates/types/src/constants.nr#L24-L68

What are the consequences?

When you write anAztec.nr function , there will be upper bounds on the following:

- The number of public state reads and writes;
- The number of note reads and nullifications;
- The number of new notes that may be created;
- The number of encrypted logs that may be emitted;
- The number of unencrypted logs that may be emitted;
- The number of L1->L2 messages that may be consumed;
- The number of L2->L1 messages that may be submitted to L1;
- The number of private function calls;
- The number of public function calls that may be enqueued;

Not only are there limits on aper function basis, there are also limits on aper transaction basis.

In particular, theseper-transaction limits will limit transaction call stack depths in the Sandbox. That means if a function call results in a cascade of nested function calls, and each of those function calls outputs lots of state reads and writes, or logs (etc.), then all of that accumulated output data might exceed the per-transaction limits that we currently have. This would cause such transactions to fail.

There are plans to relax all of this rigidity, by providing many 'sizes' ofkernel circuit , and introducing a 'bus' to ferry varying lengths of data between kernel iterations. But that'll all take some time.

In the mean time , if you encounter a per-transaction limit when testing, and you're feeling adventurous, you could 'hack' the Sandbox to increase the limits. See here (TODO: link) for a guide.However , the limits cannot be increased indefinitely. So although we do anticipate that we'll be able to increase them a little bit, don't go mad and provide yourself with 1 million state transitions per transaction. That would be as unrealistic as artificially increasing Ethereum gas limits to 1 trillion.

Circuits Processing Order Differs from Execution Order

Each function call is represented by a circuit with a dedicated zero-knowledge proof of its execution. Theprivate kernel circuit is in charge of stitching all these proofs together to produce a zero-knowledge proof that the whole execution of all function calls within a transaction is correct. In doing so, the processing order differs from the execution order. Firstly, the private kernel has to handle one function call in its entirety at a time because a zk proof cannot be verified partially. This property alone makes it impossible for the ordering of kernel circuit validation to match the order in which the functions of the transaction were executed. Secondly, the private kernel processes function calls in a stack-based order, i.e., after having processed a function call, it processes all direct child function calls in an order which is the reverse of the execution order.

Note that there is no plan to change this in the future.

Example

Let us assume that the main function namedf 1 f_1 f 1 is calling in orderf 2 f_2 f 2 ,f 3 f_3 f 3 (which callsf 5 f_5 f 5 followed byf 6 f_6 f 6), andf 4 f_4 f 4 .

Call Dependency:

f 1 ⟶ f 2 f_1 \longrightarrow f_2 f 1

⟶ f 2 ,f 3 f_3 f 3 ,f 4 f_4 f 4 f 3 ⟶ f 5 f_3 \longrightarrow f_5 f 3

⟶ f 5 ,f 6 f_6 f 6 Execution Order:

f 1 f_1 f 1 ,f 2 f_2 f 2 ,f 3 f_3 f 3 ,f 5 f_5 f 5 ,f 6 f_6 f 6 ,f 4 f_4 f 4 Private Kernel Processing Order:

f 1 f_1 f 1 ,f 4 f_4 f 4 ,f 3 f_3 f 3 ,f 6 f_6 f 6 ,f 5 f_5 f 5 ,f 2 f_2 f 2

What are the consequences?

Transaction output elements such as notes in encrypted logs, note hashes (commitments), nullifiers might be ordered differently than the one expected by the execution.

Chopped Transient Notes are still Emitted in Logs

A note which is created and nullified during the very same transaction is called transient. Such a note is chopped by theprivate kernel circuit and is never stored in any persistent data tree.

For the time being, such chopped notes are still emitted through encrypted logs (which is the communication channel to transmit notes). When a log containing a chopped note is processed, a warning will be logged about a decrypted note which does not exist in data tree. Weimproved error logging to help identify such an occurrence. However, this might be a source of confusion. This issue is tracked in ticket#1641 .

Note Terminology: Note Commitments and Note Hashes

The notes or UTXOs in Aztec need to be compressed before they are added to the trees. To do so, we need to hash all the data inside a note using a collision-resistant hash function. Currently, we use Pedersen hash (using lookup tables) to compress note data. The compressed note data is referred to as "note commitments" in our architecture. However, note commitments are referred to as "note hashes" in aztec-noir code. Be mindful of that fact that note commitments and note hashes mean the same thing. Note that we only mean to talk about terminology here and in no way one should infer security/cryptographic properties (e.g., hiding, binding) based on the name. Namely, notes come with different flavours of security properties depending on the use case.

There's more

See theGitHub issues for all known bugs fixes and features currently being worked on.Edit this page

Previous Privacy Considerations Next Migration notes