How to emit an event

Events in Aztec work similarly to Ethereum events in the sense that they are a way for contracts to communicate with the outside world. They are emitted by contracts and stored inside each instance of an AztecNode.

info Aztec events are currently represented as raw data and are not ABI encoded. ABI encoded events are a feature that will be added in the future. Unlike on Ethereum, there are 2 types of events supported by Aztec:encrypted andunencrypted .

Encrypted Events

Register a recipient

Encrypted events can only be emitted by private functions and are encrypted using a public key of a recipient. For this reason it is necessary to register a recipient in the Private Execution Environment (PXE) before encrypting the events for them. Recipients can be registered using the Aztec CLI or Aztec.js:

- Aztec CLI
- Aztec.js

aztec-cli register-recipient --address 0x147392a39e593189902458f4303bc6e0a39128c5a1c1612f76527a162d36d529 --public-key 0x26e193aef4f83c70651485b5526c6d01a36d763223ab24efd1f9ff91b394ac0c20ad99d0ef669dc0dde8d5f5996c63105de8e15c2c87d8260b9e6f02f72af622 --partial-address 0x200e9a6c2d2e8352012e51c6637659713d336405c29386c7c4ac56779ab54fa7 const aztecAddress = AztecAddress . fromString ("0x147392a39e593189902458f4303bc6e0a39128c5a1c1612f76527a162d36d529") ; const publicKey = Point . fromString ("0x26e193aef4f83c70651485b5526c6d01a36d763223ab24efd1f9ff91b394ac0c20ad99d0ef669dc0dde8d5f5996c63105de8e15c2c87d8260b9e6f02f72af622") ; const partialAddress = Fr . fromString ("0x200e9a6c2d2e8352012e51c6637659713d336405c29386c7c4ac56779ab54fa7") ;

const completeAddress = CompleteAddress . create (aztecAddress , publicKey , partialKey) ; await pxe . registerRecipient (completeAddress) ; info If a note recipient is one of the accounts inside the PXE, we don't need to register it as a recipient because we already have the public key available. You can register a recipient as shownhere

At this point the Sandbox only enables the emitting of encrypted note preimages through encrypted events. In the future we will allow emitting arbitrary information. (If you currently emit arbitrary information, PXE will fail to decrypt, process and store this data, so it will not be queryable).

Import library

To emit encrypted logs first import theemit_encrypted_log utility function which wraps anoracle :

encrypted_import use

dep :: aztec :: log :: emit_encrypted_log ;Source code: noir-projects/aztec-nr/address-note/src/address_note.nr#L1-L3

Call emit_encrypted_log

After importing, you can call the function:

encrypted emit_encrypted_log (context , (* context) . this_address ( ) , slot , Self :: get_note_type_id ( ) , encryption_pub_key , self . serialize_content ( ) ,) ; Source code: noir-projects/aztec-nr/address-note/src/address_note.nr#L74-L83

Successfully process the encrypted event

One of the functions of the PXE is constantly loading encrypted logs from theAztecNode and decrypting them. When new encrypted logs are obtained, the PXE will try to decrypt them using the private encryption key of all the accounts registered inside PXE. If the decryption is successful, the PXE will store the decrypted note inside a database. If the decryption fails, the specific log will be discarded.

For the PXE to successfully process the decrypted note we need to compute the note's 'note hash' and 'nullifier'. Aztec.nr enables smart contract developers to design custom notes, meaning developers can also customize how a note's note hash and nullifier should be computed. Because of this customizability, and because there will be a potentially-unlimited number of smart contracts deployed to Aztec, an PXE needs to be 'taught' how to compute the custom note hashes and nullifiers for a particular contract. This is done by a function calledcompute_note_hash_and_nullifier , which is automatically injected into every contract when compiled.

Unencrypted Events

Unencrypted events are events which can be read by anyone. They can be emitted by both public and private functions.

danger * Emitting unencrypted events from private function is a significant privacy leak and it should be considered by the developer whether it is acceptable. * Unencrypted events are currentlyNOT * linked to the contract emitting them, so it is practically adebug_log * . * :::

Import library

To emit unencrypted logs first import theemit_unencrypted_log utility function inside your contract:

unencrypted_import use

dep :: aztec :: prelude :: emit_unencrypted_log ;Source code: noir-projects/noir-contracts/contracts/test_contract/src/main.nr#L12-L14

Call emit_unencrypted_log

After importing, you can call the function:

emit_unencrypted emit_unencrypted_log (& mut context , value) ;Source code: noir-projects/noir-contracts/contracts/test_contract/src/main.nr#L253-L255

Querying the unencrypted event

Once emitted, unencrypted events are stored in AztecNode and can be queried by anyone:

- Aztec CLI

- Aztec.js

aztec-cli get-logs --fromBlock 5 get_logs // Get the unencrypted logs from the last block const fromBlock =

await pxe . getBlockNumber ( ) ; const logFilter =

{ fromBlock , toBlock : fromBlock +

1 , } ; const unencryptedLogs =

(await pxe . getUnencryptedLogs (logFilter)) . logs Source code: yarn-project/end-to-end/src/fixtures/utils.ts#L461-L469 Get logs functionality provides a variety of filtering options. To display them run:

aztec-cli get-logs --help

Costs

All event data is pushed to Ethereum as calldata by the sequencer and for this reason the cost of emitting an event is non-trivial.

In the Sandbox, an encrypted note has a fixed overhead of 4 field elements (to broadcast an ephemeral public key, a contract address, and a storage slot); plus a variable number of field elements depending on the type of note being emitted.

AValueNote , for example, currently uses 3 fields elements (plus the fixed overhead of 4). That's roughly7 * 32 = 224 bytes of information.

value-note-def struct ValueNote { value: Field, owner: AztecAddress, randomness: Field, header: NoteHeader, }Source code: noir-projects/aztec-nr/value-note/src/value_note.nr#L10-L17 * There are plans to compress encrypted note data further. * There are plans to adopt EIP-4844 blobs to reduce the cost of data submission further. Edit this page

Previous How to write an accounts contract Next Oracle Functions