Migration notes

Aztec is in full-speed development. Literally every version breaks compatibility with the previous ones. This page attempts to target errors and difficulties you might encounter when upgrading, and how to resolve them.

0.41.0

[Aztec.nr] State variable rework

Aztec.nr state variables have been reworked so that calling private functions in public and vice versa is detected as an error during compilation instead of at runtime. This affects users in a number of ways:

New compile time errors

It used to be that calling a state variable method only available in public from a private function resulted in obscure runtime errors in the form of a failed_is_some assertion.

Incorrect usage of the state variable methods now results in compile time errors. For example, given the following function:

[aztec(public)]

fn

get_decimals ( )

->

pub

u8

{ storage . decimals . read_private ( ) } The compiler will now error out with

Expected type SharedImmutable<_, &mut PrivateContext>, found type SharedImmutableThe key component is the second generic parameter: the compiler expects aPrivateContext (becuseread_private is only available during private execution), but aPublicContext is being used instead (because of the#[aztec(public)] attribute).

Generic parameters inStorage

TheStorage struct (the one marked with#[aztec(storage)]) should now be generic over aContext type, which matches the new generic parameter of all Aztec.nr libraries. This parameter is always the last generic parameter.

This means that, without any additional features, we'd end up with some extra boilerplate when declaring this struct:

[aztec(storage)]

- struct Storage {
- struct Storage {
- nonce_for_burn_approval: PublicMutable,
- nonce_for_burn_approval: PublicMutable,
- portal_address: SharedImmutable,
- portal_address: SharedImmutable,
- approved_action: Map>,
- approved_action: Map, Context>, } Because of this, the#[aztec(storage)] macro has been updated toautomatically inject thisContext generic parameter. The storage declaration does not require any changes.

Removal ofContext

TheContext type no longer exists. End users typically didn't use it, but if imported it needs to be deleted.

[Aztec.nr] View functions and interface navigation

It is now possible to explicitly state a function doesn't perform any state alterations (including storage, logs, nullifiers and/or messages from L2 to L1) with the#[aztec(view)] attribute, similarly to solidity'sview function modifier.

[aztec(public)]

- # [aztec(view)]

  fn get_price(asset_id: Field) -> Asset { storage.assets.at(asset_id).read() } View functions only generate aStaticCallInterface that doesn't include.call or.enqueue methods. Also, the denominationstatic has been completely removed from the interfaces, in favor of the more familiarview

- let price = PriceFeed::at(asset.oracle).get_price(0).static_call(&mut context).price;

- let price = PriceFeed::at(asset.oracle).get_price(0).view(&mut context).price;

[aztec(private)]

fn enqueue_public_get_value_from_child(target_contract: AztecAddress, value: Field) { - StaticChild::at(target_contract).pub_get_value(value).static_enqueue(&mut context); + StaticChild::at(target_contract).pub_get_value(value).enqueue_view(&mut context); } Additionally, the Noir LSP will now honor "go to definitions" requests for contract interfaces (Ctrl+click), taking the user to the original function implementation.

[Aztec.js] Simulate changes

- .simulate()
- now tracks closer the process performed by.send().wait()
- , specifically going through the account contract entrypoint instead of directly calling the intended function.
- wallet.viewTx(...)
- has been renamed towallet.simulateUnconstrained(...)
- to better clarify what it does.

[Aztec.nr] Keys: Token note now stores an owner master nullifying public key hash instead of an owner address

i.e.

struct TokenNote { amount: U128, - owner: AztecAddress, + npk_m_hash: Field, randomness: Field, } Creating a token note and adding it to storage now looks like this:

- let mut note = ValueNote::new(new_value, owner);
- storage.a_private_value.insert(&mut note, true);
- let owner_npk_m_hash = get_npk_m_hash(&mut context, owner);
- let owner_ivpk_m = get_ivpk_m(&mut context, owner);
- let mut note = ValueNote::new(new_value, owner_npk_m_hash);
- storage.a_private_value.insert(&mut note, true, owner_ivpk_m); Computing the nullifier similarly changes to use this master nullifying public key hash.

0.40.0

[Aztec.nr] Debug logging

The functiondebug_log_array_with_prefix has been removed. Usedebug_log_format with{} instead. The special sequence{} will be replaced with the whole array. You can also use{0} ,{1} , ... as usual withdebug_log_format .

- debug_log_array_with_prefix("Prefix", my_array);
- debug_log_format("Prefix {}", my_array);

0.39.0

[Aztec.nr] Mutable delays inSharedMutable

The type signature forSharedMutable changed fromSharedMutable toSharedMutable . The behavior is the same as before, except the delay can now be changed after deployment by callingschedule_delay_change .

[Aztec.nr] get_public_key oracle replaced with get_ivpk_m

When implementing changes according to anew key scheme we had to change oracles. What used to be called encryption public key is now master incoming viewing public key.

- use dep::aztec::oracles::get_public_key::get_public_key;
- use dep::aztec::keys::getters::get_ivpk_m;
- let encryption_pub_key = get_public_key(self.owner);
- let ivpk_m = get_ivpk_m(context, self.owner);

0.38.0

[Aztec.nr] Emitting encrypted logs

Theemit_encrypted_log function is now a context method.

- use dep::aztec::log::emit_encrypted_log;
- use dep::aztec::logs::emit_encrypted_log;
- emit_encrypted_log(context, log1);
- context.emit_encrypted_log(log1);

0.36.0

FieldNote

removed

FieldNote only existed for testing purposes, and was not a note type that should be used in any real application. Its name unfortunately led users to think that it was a note type suitable to store aField value, which it wasn't.

If usingFieldNote , you most likely want to useValueNote instead, which has both randomness for privacy and an owner for proper nullification.

SlowUpdatesTree

replaced forSharedMutable

The oldSlowUpdatesTree contract and libraries have been removed from the codebase, use the newSharedMutable library instead. This will require that you add a global variable specifying a delay in blocks for updates, and replace the slow updates tree state variable withSharedMutable variables.

- global CHANGE_ROLES_DELAY_BLOCKS = 5;

struct Storage { - slow_update: SharedImmutable, + roles: Map>, } Reading fromSharedMutable is much simpler, all that's required is to callget_current_value_in_public orget_current_value_in_private , depending on the domain.

- let caller_roles = UserFlags::new(U128::from_integer(slow.read_at_pub(context.msg_sender().to_field()).call(&mut context)));
- let caller_roles = storage.roles.at(context.msg_sender()).get_current_value_in_public(); Finally, you can remove all capsule usage on the client code or tests, since those are no longer required when working withSharedMutable .

[Aztec.nr & js] Portal addresses

Deployments have been modified. No longer are portal addresses treated as a special class, being immutably set on creation of a contract. They are no longer passed in differently compared to the other variables and instead should be implemented using usual storage by those who require it. One should use the storage that matches the usecase - likely shared storage to support private and public.

This means that you will likely add the portal as a constructor argument

- fn constructor(token: AztecAddress) {

- storage.token.write(token);
- }
- struct Storage { ...
- portal_address: SharedImmutable,
- }
- fn constructor(token: AztecAddress, portal_address: EthAddress) {
- storage.token.write(token);
- storage.portal_address.initialize(portal_address);

- } And read it from storage whenever needed instead of from the context.

- context.this_portal_address(),

- storage.portal_address.read_public(),

[Aztec.nr] Oracles

Oracleget_nullifier_secret_key was renamed toget_app_nullifier_secret_key andrequest_nullifier_secret_key function on PrivateContext was renamed asrequest_app_nullifier_secret_key .

- let secret = get_nullifier_secret_key(self.owner);
- let secret = get_app_nullifier_secret_key(self.owner);
- let secret = context.request_nullifier_secret_key(self.owner);
- let secret = context.request_app_nullifier_secret_key(self.owner);

[Aztec.nr] Contract interfaces

It is now possible to import contracts on another contracts and use their automatic interfaces to perform calls. The interfaces have the same name as the contract, and are automatically exported. Parameters are automatically serialized (using theSerialize trait) and return values are automatically deserialized (using theDeserializetrait). Serialize and Deserialize methods have to conform to the standard ACVM serialization schema for the interface to work!

1. Only fixed length types are supported
2. All numeric types become Fields
3. Strings become arrays of Fields, one per char
4. Arrays become arrays of Fields following rules 2 and 3

5. Structs become arrays of Fields, with every item defined in the same order as they are in Noir code, following rules 2, 3, 4 and 5 (recursive)

6. context.call_public_function(

7. storage.gas_token_address.read_private(),
8. FunctionSelector::from_signature("pay_fee(Field)"),
9. [42]

10. **);**

11. context.call_public_function(
12. storage.gas_token_address.read_private(),
13. FunctionSelector::from_signature("pay_fee(Field)"),
14. [42]

15. **);**

16. let _ = context.call_private_function(
17. storage.subscription_token_address.read_private(),
18. FunctionSelector::from_signature("transfer((Field),(Field),Field,Field)"),
19. [
20. context.msg_sender().to_field(),
21. storage.subscription_recipient_address.read_private().to_field(),
22. storage.subscription_price.read_private(),
23. nonce
24.]
25.);
26. use dep::gas_token::GasToken;
27. use dep::token::Token; +
28. ...

29. // Public call from public land
30. GasToken::at(storage.gas_token_address.read_private()).pay_fee(42).call(&mut context);
31. // Public call from private land
32. GasToken::at(storage.gas_token_address.read_private()).pay_fee(42).enqueue(&mut context);
33. // Private call from private land
34. Token::at(asset).transfer(context.msg_sender(), storage.subscription_recipient_address.read_private(), amount, nonce).call(&mut context); It is also possible to use these automatic interfaces from the local contract, and thus enqueue public calls from private without having to rely on low levelcontext calls.

[Aztec.nr] Rename max block number setter

Therequest_max_block_number function has been renamed toset_tx_max_block_number to better reflect that it is not a getter, and that the setting is transaction-wide.

- context.request_max_block_number(value);
- context.set_tx_max_block_number(value);

[Aztec.nr] Get portal address

Theget_portal_address oracle was removed. If you need to get the portal address of SomeContract, add the following methods to it

[aztec(private)]

fn get_portal_address() -> EthAddress { context.this_portal_address() }

[aztec(public)]

fn get_portal_address_public() -> EthAddress { context.this_portal_address() } and change the call toget_portal_address

- let portal_address = get_portal_address(contract_address);
- let portal_address = SomeContract::at(contract_address).get_portal_address().call(&mut context);

[Aztec.nr] Required gas limits for public-to-public calls

When calling a public function from another public function using thecall_public_function method, you must now specify how much gas you're allocating to the nested call. This will later allow you to limit the amount of gas consumed by the nested call, and handle any out of gas errors.

Note that gas limits are not yet enforced. For now, it is suggested you usedep::aztec::context::gas::GasOpts::default() which will forward all available gas.

- use dep::aztec::context::gas::GasOpts;

- context.call_public_function(target_contract, target_selector, args);

- context.call_public_function(target_contract, target_selector, args, GasOpts::default()); Note that this is not required when enqueuing a public function from a private one, since top-level enqueued public functions will always consume all gas available for the transaction, as it is not possible to handle any out-of-gas errors.

[Aztec.nr] Emitting unencrypted logs

Theemit_unencrypted_logs function is now a context method.

- use dep::aztec::log::emit_unencrypted_log;

- use dep::aztec::log::emit_unencrypted_log_from_private;

- emit_unencrypted_log(context, log1);

- emit_unencrypted_log_from_private(context, log2);
- context.emit_unencrypted_log(log1);
- context.emit_unencrypted_log(log2);

0.33

[Aztec.nr] Storage struct annotation

The storage struct now identified by the annotation#[aztec(storage)] , instead of having to rely on it being calledStorage .

- struct Storage {
- ...
- }

- # [aztec(storage)]

- struct MyStorageStruct {
- ...
- }

[Aztec.js] Storage layout and note info

Storage layout and note information are now exposed in the TS contract artifact

- const note = new Note([new Fr(mintAmount), secretHash]);
- const pendingShieldStorageSlot = new Fr(5n); // storage slot for pending_shields
- const noteTypeId = new Fr(84114971101151129711410111011678111116101n); // note type id for TransparentNote
- const extendedNote = new ExtendedNote(
- note,
- admin.address,
- token.address,
- pendingShieldStorageSlot,
- noteTypeId,
- receipt.txHash,
-);
- await pxe.addNote(extendedNote);
- const note = new Note([new Fr(mintAmount), secretHash]);
- const extendedNote = new ExtendedNote(
- note,
- admin.address,
- token.address,
- TokenContract.storage.pending_shields.slot,
- TokenContract.notes.TransparentNote.id,
- receipt.txHash,
-);
- await pxe.addNote(extendedNote);

[Aztec.nr] rand oracle is now called unsafe_rand

oracle::rand::rand has been renamed tooracle::unsafe_rand::unsafe_rand . This change was made to communicate that we do not constrain the value in circuit and instead we just trust our PXE.

- let random_value = rand();
- let random_value = unsafe_rand();

[AztecJS] Simulate and get return values for ANY call and introducingprove()

Historically it have been possible to "view"unconstrained functions to simulate them and get the return values, but not forpublic norprivate functions. This has lead to a lot of bad code where we have the same function implemented thrice, once inprivate , once inpublic and once inunconstrained . It is not possible to callsimulate on any call to get the return values! However, beware that it currently always returns a Field array of size 4 for private and public. This will change to become similar to the return values of theunconstrained functions with proper return types.

- # [aztec(private)]

- fn get_shared_immutable_constrained_private() -> pub Leader {
- storage.shared_immutable.read_private()

- **}**

- unconstrained fn get_shared_immutable() -> pub Leader {
- storage.shared_immutable.read_public()

- }

- # [aztec(private)]

- fn get_shared_immutable_private() -> pub Leader {
- storage.shared_immutable.read_private()

- }

- const returnValues = await contract.methods.get_shared_immutable().view();

- const returnValues = await contract.methods.get_shared_immutable_private().simulate(); await expect(
- asset.withWallet(wallets[1]).methods.update_admin(newAdminAddress).simulate()).rejects.toThrow(
- asset.withWallet(wallets[1]).methods.update_admin(newAdminAddress).prove()).rejects.toThrow("Assertion failed: caller is not admin 'caller_roles.is_admin'",);

0.31.0

[Aztec.nr] Public storage historical read API improvement

history::public_value_inclusion::prove_public_value_inclusion has been renamed tohistory::public_storage::public_storage_historical_read , and its API changed slightly. Instead of receiving avalue parameter it now returns the historical value stored at that slot.

If you were using an oracle to get the value to pass toprove_public_value_inclusion , drop the oracle and use the return value frompublic_storage_historical_read instead:

- let value = read_storage();
- prove_public_value_inclusion(value, storage_slot, contract_address, context);

- let value = public_storage_historical_read(storage_slot, contract_address, context); If you were proving historical existence of a value you got via some other constrained means, perform an assertion against the return value ofpublic_storage_historical_read instead:

- prove_public_value_inclusion(value, storage_slot, contract_address, context);

- assert(public_storage_historical_read(storage_slot, contract_address, context) == value);

0.30.0

[AztecJS] Simplify authwit syntax

- const messageHash = computeAuthWitMessageHash(accounts[1].address, action.request());
- await wallets[0].setPublicAuth(messageHash, true).send().wait();
- await wallets[0].setPublicAuthWit({ caller: accounts[1].address, action }, true).send().wait(); const action = asset .withWallet(wallets[1]) .methods.unshield(accounts[0].address, accounts[1].address, amount, nonce); -const messageHash = computeAuthWitMessageHash(accounts[1].address, action.request()); -const witness = await wallets[0].createAuthWitness(messageHash); +const witness = await wallets[0].createAuthWit({ caller: accounts[1].address, action }); await wallets[1].addAuthWitness(witness); Also note some of the naming changes:setPublicAuth ->setPublicAuthWit createAuthWitness ->createAuthWit

[Aztec.nr] Automatic NoteInterface implementation and selector changes

Implementing a note required a fair amount of boilerplate code, which has been substituted by the#[aztec(note)] attribute.

- # [aztec(note)]

  struct AddressNote { address: AztecAddress, owner: AztecAddress, randomness: Field, header: NoteHeader }

impl NoteInterface for AddressNote { - fn serialize_content(self) -> [Field; ADDRESS_NOTE_LEN]{ - [self.address.to_field(), self.owner.to_field(), self.randomness] - } - - fn deserialize_content(serialized_note: [Field; ADDRESS_NOTE_LEN]) -> Self { - AddressNote { - address: AztecAddress::from_field(serialized_note[0]), - owner: AztecAddress::from_field(serialized_note[1]), - randomness: serialized_note[2], - header: NoteHeader::empty(), - } - } - - fn compute_note_content_hash(self) -> Field { - pedersen_hash(self.serialize_content(), 0) - } - fn compute_nullifier(self, context: &mut PrivateContext) -> Field { let note_hash_for_nullify = compute_note_hash_for_consumption(self); let secret = context.request_nullifier_secret_key(self.owner); pedersen_hash([note_hash_for_nullify, secret.low, secret.high,],0) }

fn compute_nullifier_without_context(self) -> Field { let note_hash_for_nullify = compute_note_hash_for_consumption(self); let secret = get_nullifier_secret_key(self.owner); pedersen_hash([note_hash_for_nullify, secret.low, secret.high,],0) }

- fn set_header(&mut self, header: NoteHeader) {
- self.header = header;

- **}**

- fn get_header(note: Self) -> NoteHeader {
- note.header
- }

fn broadcast(self, context: &mut PrivateContext, slot: Field) { let encryption_pub_key = get_public_key(self.owner); emit_encrypted_log(context, (*context).this_address(), slot, Self::get_note_type_id(), encryption_pub_key, self.serialize_content(),); }

- fn get_note_type_id() -> Field {
- 6510010011410111511578111116101
- } } Automatic note (de)serialization implementation also means it is now easier to filter notes usingNoteGetterOptions.select via the::properties() helper:

Before:

let options =

NoteGetterOptions :: new ( ) . select (0 , amount ,

Option :: none ( )) . select (1 , owner . to_field ( ) ,

Option :: none ( )) . set_limit (1) ; After:

let options =

NoteGetterOptions :: new ( ) . select (ValueNote :: properties ( ) . value , amount ,

Option :: none ( )) . select (ValueNote :: properties ( ) . owner , owner . to_field ( ) ,

Option :: none ( )) . set_limit (1) ; The helper returns a metadata struct that looks like this (if autogenerated)

ValueNoteProperties

{ value :

PropertySelector

{ index :

0 , offset :

0 , length :

32

} , owner :

PropertySelector

{ index :

1 , offset :

0 , length :

32

} , randomness :

PropertySelector

{ index :

2 , offset :

0 , length :

32

} , } It can also be used for the.sort method.

0.27.0

initializer

macro replacesconstructor

Before this version, every contract was required to have exactly oneconstructor private function, that was used for deployment. We have now removed this requirement, and madeconstructor a function like any other.

To signal that a function can be used toinitialize a contract, you must now decorate it with the#[aztec(initializer)] attribute. Initializers are regular functions that set an "initialized" flag (a nullifier) for the contract. A contract can only be initialized once, and contract functions can only be called after the contract has been initialized, much like a constructor. However, if a contract defines no initializers, it can be called at any time. Additionally, you can define as many initializer functions in a contract as you want, both private and public.

To migrate from current code, simply add an initializer attribute to your constructor functions.

- **[aztec(initializer)]**

[aztec(private)]

fn constructor() { ... } If your private constructor was used to just call a public internal initializer, then remove the private constructor and flag the public function as initializer. And if your private constructor was an empty one, just remove it.

0.25.0

[Aztec.nr] Static calls

It is now possible to perform static calls from both public and private functions. Static calls forbid any modification to the state, including L2->L1 messages or log generation. Once a static context is set through a static all, every subsequent call will also be treated as static via context propagation.

context . static_call_private_function (targetContractAddress , targetSelector , args) ;

context . static_call_public_function (targetContractAddress , targetSelector , args) ;

[Aztec.nr] Introduction toprelude

A newprelude module to include common Aztec modules and types. This simplifies dependency syntax. For example:

use

dep :: aztec :: protocol_types :: address :: AztecAddress ; use

dep :: aztec :: { context :: { PrivateContext ,

Context } ,

note :: { note_header :: NoteHeader , utils as note_utils } , state_vars :: Map } ; Becomes:

use

dep :: aztec :: prelude :: { AztecAddress ,

NoteHeader ,

PrivateContext ,

Map } ; use

dep :: aztec :: context :: Context ; use

dep :: aztec :: notes :: utils as note_utils ; This will be further simplified in future versions (See4496 for further details).

The prelude consists of

prelude use

dep :: protocol_types :: { address :: { AztecAddress ,

EthAddress } ,

abis :: function_selector :: FunctionSelector , traits :: { Serialize ,

Deserialize } } ; use

crate :: { state_vars :: { map :: Map ,

private_immutable :: PrivateImmutable ,

private_mutable :: PrivateMutable , public_immutable :: PublicImmutable ,

public_mutable :: PublicMutable ,

private_set :: PrivateSet , shared_immutable :: SharedImmutable ,

shared_mutable :: SharedMutable ,

storage :: Storable } , context :: { PrivateContext ,

PackedReturns ,

FunctionReturns } , note :: { note_header :: NoteHeader ,

note_interface :: NoteInterface ,

note_getter_options :: NoteGetterOptions , note_viewer_options :: NoteViewerOptions , utils :: compute_note_hash_and_nullifier as utils_compute_note_hash_and_nullifier } } ; Source code: noir-projects/aztec-nr/aztec/src/prelude.nr#L1-L19

internal

is now a macro

Theinternal keyword is now removed from Noir, and is replaced by anaztec(internal) attribute in the function. The resulting behavior is exactly the same: these functions will only be callable from within the same contract.

Before:

[aztec(private)]

internal fn

double (input :

Field)

->

Field

{ input *

2 } After:

[aztec(private)]

[aztec(internal)]

fn

double (input :

Field)

->

Field

{ input *

2 }

[Aztec.nr] No SafeU120 anymore

Noir now have overflow checks by default. So we don't need SafeU120 like libraries anymore.

You can replace it withU128 instead

Before:

SafeU120::new(0) Now:

U128::from_integer(0)

[Aztec.nr]compute_note_hash_and_nullifier

is now autogenerated

Historically developers have been required to include acompute_note_hash_and_nullifier function in each of their contracts. This function is now automatically generated, and all instances of it in contract code can be safely removed.

It is possible to provide a user-defined implementation, in which case auto-generation will be skipped (though there are no known use cases for this).

[Aztec.nr] Updated naming of state variable wrappers

We have decided to change the naming of our state variable wrappers because the naming was not clear. The changes are as follows:

1. Singleton
2. ->PrivateMutable
3. ImmutableSingleton
4. ->PrivateImmutable
5. StablePublicState
6. ->SharedImmutable
7. PublicState
8. ->PublicMutable

This is the meaning of "private", "public" and "shared": Private: read (R) and write (W) from private, not accessible from public Public: not accessible from private, R/W from public Shared: R from private, R/W from public

Note:SlowUpdates will be renamed toSharedMutable once the implementation is ready.

[Aztec.nr] Authwit updates

Authentication Witnesses have been updates such that they are now cancellable and scoped to a specific consumer. This means that theauthwit nullifier must be emitted from the account contract, which require changes to the interface. Namely, theassert_current_call_valid_authwit_public andassert_current_call_valid_authwit inauth.nr willNO LONGER emit a nullifier. Instead it will call aspend_*_authwit function in the account contract - which will emit the nullifier and perform a few checks. This means that theis_valid functions have been removed to not confuse it for a non-mutating function (static). Furthermore, thecaller parameter of the "authwits" have been moved "further out" such that the account contract can use it in validation, allowing scoped approvals from the account POV. For most contracts, this won't be changing much, but for the account contract, it will require a few changes.

Before:

[aztec(public)]

fn

is_valid_public (message_hash :

Field)

->

Field

{ let actions =

AccountActions :: public (& mut context ,

ACCOUNT_ACTIONS_STORAGE_SLOT , is_valid_impl) ; actions . is_valid_public (message_hash) }

[aztec(private)]

fn

is_valid (message_hash :

Field)

->

Field

{ let actions =

AccountActions :: private (& mut context ,

ACCOUNT_ACTIONS_STORAGE_SLOT , is_valid_impl) ; actions . is_valid (message_hash) } After:

[aztec(private)]

fn

spend_private_authwit (inner_hash :

Field)

->

Field

{ let actions =

AccountActions :: private (& mut context ,

ACCOUNT_ACTIONS_STORAGE_SLOT , is_valid_impl) ; actions . spend_private_authwit (inner_hash) }

[aztec(public)]

fn

spend_public_authwit (inner_hash :

Field)

->

Field

{ let actions =

AccountActions :: public (& mut context ,

ACCOUNT_ACTIONS_STORAGE_SLOT , is_valid_impl) ; actions . spend_public_authwit (inner_hash) }

0.24.0

Introduce Note Type IDs

Note Type IDs are a new feature which enable contracts to have multipleMap s with different underlying note types, something that was not possible before. This is done almost without any user intervention, though some minor changes are required.

The mandatorycompute_note_hash_and_nullifier now has a fifth parameternote_type_id . Use this instead ofstorage_slot to determine which deserialization function to use.

Before:

unconstrained fn

compute_note_hash_and_nullifier (contract_address :

AztecAddress , nonce :

Field , storage_slot :

Field , preimage :

[Field ;

TOKEN_NOTE_LEN])

->

pub

[Field ;

4]

{ let note_header =

NoteHeader :: new (contract_address , nonce , storage_slot) ;

if

(storage_slot == storage . pending_shields . get_storage_slot ( ))

{ note_utils :: compute_note_hash_and_nullifier (TransparentNote :: deserialize_content , note_header , preimage) }

else

if

(note_type_id == storage . slow_update . get_storage_slot ( ))

{ note_utils :: compute_note_hash_and_nullifier (FieldNote :: deserialize_content , note_header , preimage) }

else

{ note_utils :: compute_note_hash_and_nullifier (TokenNote :: deserialize_content , note_header , preimage) } Now:

unconstrained fn

compute_note_hash_and_nullifier (contract_address :

AztecAddress , nonce :

Field , storage_slot :

Field , note_type_id :

Field , preimage :

[Field ;

TOKEN_NOTE_LEN])

->

pub

[Field ;

4]

{ let note_header =

NoteHeader :: new (contract_address , nonce , storage_slot) ;

if

(note_type_id ==

TransparentNote :: get_note_type_id ( ))

{ note_utils :: compute_note_hash_and_nullifier (TransparentNote :: deserialize_content , note_header , preimage) }

else

if

(note_type_id ==

FieldNote :: get_note_type_id ( ))

{ note_utils :: compute_note_hash_and_nullifier (FieldNote :: deserialize_content , note_header , preimage) }

else

{ note_utils :: compute_note_hash_and_nullifier (TokenNote :: deserialize_content , note_header , preimage) }
TheNoteInterface trait now has an additionalget_note_type_id() function. This implementation will be autogenerated in the future, but for now providing any unique ID will suffice. The suggested way to do it is by running the Python command shown in the comment below:

impl

NoteInterface < N

for

MyCustomNote

{ fn

get_note_type_id ( )

->

Field

{ // python -c "print(int(''.join(str(ord(c)) for c in 'MyCustomNote')))" 771216711711511611110978111116101 } }

[js] Importing contracts in JS

@aztec/noir-contracts is now@aztec/noir-contracts.js . You'll need to update your package.json & imports.

Before:

import

{ TokenContract }

from

"@aztec/noir-contracts/Token" ; Now:

import

{ TokenContract }

from

"@aztec/noir-contracts.js/Token" ;

[Aztec.nr] aztec-nr contracts location change in Nargo.toml

Aztec contracts are now moved outside of theyarn-project folder and intonoir-projects , so you need to update your imports.

Before:

easy_private_token_contract

{ git =

"https://github.com/AztecProtocol/aztec-packages/" , tag = "v0.23.0" , directory =

"yarn-project/noir-contracts/contracts/easy_private_token_contract" } Now, update theyarn-project folder fornoir-projects :

easy_private_token_contract

{ git =

"https://github.com/AztecProtocol/aztec-packages/" , tag = "v0.24.0" , directory =

"noir-projects/noir-contracts/contracts/easy_private_token_contract" }

0.22.0

Note::compute_note_hash

renamed toNote::compute_note_content_hash

Thecompute_note_hash function in of theNote trait has been renamed tocompute_note_content_hash to avoid being confused with the actual note hash.

Before:

impl

NoteInterface

for

CardNote

{ fn

compute_note_hash (self)

->

Field

{ pedersen_hash ([self . owner . to_field ( ) ,] ,

0) } Now:

impl

NoteInterface

for

CardNote

{ fn

compute_note_content_hash (self)

->

Field

{ pedersen_hash ([self . owner . to_field ( ) ,] ,

0) }

Introducecompute_note_hash_for_consumption

andcompute_note_hash_for_insertion

Makes a split in logic for note hash computation for consumption and insertion. This is to avoid confusion between the two, and to make it clear that the note hash for consumption is different from the note hash for insertion (sometimes).

compute_note_hash_for_consumption replacescompute_note_hash_for_read_or_nullify .compute_note_hash_for_insertion is new, and mainly used in `lifecycle.nr``

Note::serialize_content

andNote::deserialize_content added to `NoteInterface

TheNoteInterface have been extended to includeserialize_content anddeserialize_content functions. This is to convey the difference between serializing the full note, and just the content. This change allows you to also add aserialize function to support passing in a complete note to a function.

Before:

impl

Serialize < ADDRESS_NOTE_LEN

for

AddressNote

{ fn

serialize (self)

->

[Field ;

ADDRESS_NOTE_LEN] { [self . address . to_field ( ) ,

self . owner . to_field ( ) ,

self . randomness] } } impl

Deserialize < ADDRESS_NOTE_LEN

for

AddressNote

{ fn

deserialize (serialized_note :

[Field ;

ADDRESS_NOTE_LEN])

->

Self

{ AddressNote

{ address :

AztecAddress :: from_field (serialized_note [0]) , owner :

AztecAddress :: from_field (serialized_note [1]) , randomness : serialized_note [2] , header :

NoteHeader :: empty ( ) , } } Now

impl

NoteInterface < ADDRESS_NOTE_LEN

for

AddressNote

{ fn

serialize_content (self)

->

[Field ;

ADDRESS_NOTE_LEN] { [self . address . to_field ( ) ,

self . owner . to_field ( ) ,

self . randomness] }

fn

deserialize_content (serialized_note :

[Field ;

ADDRESS_NOTE_LEN])

->

Self

{ AddressNote

{ address :

AztecAddress :: from_field (serialized_note [0]) , owner :

AztecAddress :: from_field (serialized_note [1]) , randomness : serialized_note [2] , header :

NoteHeader :: empty ( ) , } } ... }

[Aztec.nr] No storage.init() andSerialize

,Deserialize ,NoteInterface as Traits, removal of SerializationMethods and SERIALIZED_LEN

Storage definition and initialization has been simplified. Previously:

struct

Storage

{ leader :

PublicState < Leader ,

LEADER_SERIALIZED_LEN

, legendary_card :

Singleton < CardNote ,

CARD_NOTE_LEN

, profiles :

```
Map < AztecAddress ,
Singleton < CardNote ,
CARD_NOTE_LEN
                 , test :
Set < CardNote ,
CARD_NOTE_LEN
        , imm_singleton :
PrivateImmutable < CardNote ,
CARD_NOTE_LEN
        , }
impl
Storage
{ fn
init ( context :
Context )
->
Self
{ Storage
{ leader :
PublicMutable :: new ( context , 1 , LeaderSerializationMethods , ) , legendary_card :
PrivateMutable :: new ( context ,
2 ,
CardNoteMethods ) , profiles :
Map :: new ( context , 3 , | context , slot |
{ PrivateMutable :: new ( context , slot ,
CardNoteMethods ) } , ) , test :
Set :: new ( context ,
4 ,
CardNoteMethods ) , imm_singleton :
PrivateImmutable :: new ( context ,
4 ,
CardNoteMethods ) , } } } Now:
struct
Storage
{ leader :
PublicMutable < Leader
        , legendary_card :
```

Singleton < CardNote

, profiles :

Map < AztecAddress ,

Singleton < CardNote

, test :

Set < CardNote

, imm_singleton :

PrivateImmutable < CardNote

, } For this to work, Notes must implement Serialize, Deserialize and NoteInterface Traits. Previously:

use

dep :: aztec :: protocol_types :: address :: AztecAddress ; use

dep :: aztec :: { note :: { note_header :: NoteHeader , note_interface :: NoteInterface , utils :: compute_note_hash_for_read_or_nullify , } , oracle :: { nullifier_key :: get_nullifier_secret_key , get_public_key :: get_public_key , } , log :: emit_encrypted_log , hash :: pedersen_hash , context :: PrivateContext , } ;

// Shows how to create a custom note

global CARD_NOTE_LEN :

Field

=

1 ;

impl

CardNote

{ pub

fn

new (owner :

AztecAddress)

->

Self

{ CardNote

{ owner , } }

pub

fn

serialize (self)

->

[Field ;

CARD_NOTE_LEN]

{ [self . owner . to_field ( )] }

pub

fn

```
deserialize ( serialized_note :
[ Field ;
CARD_NOTE_LEN ] )
->
Self
{ CardNote
{ owner :
AztecAddress :: from_field ( serialized_note [ 1 ] ) , } }
pub
fn
compute_note_hash ( self )
->
Field
{ pedersen_hash ( [ self . owner . to_field ( ) , ] , 0 ) }
pub
fn
compute_nullifier ( self , context :
& mut
PrivateContext )
->
Field
{ let note_hash_for_nullify =
compute_note_hash_for_read_or_nullify ( CardNoteMethods ,
self ) ; let secret = context . request_nullifier_secret_key ( self . owner ) ; pedersen_hash ( [ note_hash_for_nullify , secret . high , secret . low , ] , 0 ) }
pub
fn
compute_nullifier_without_context ( self )
->
Field
{ let note_hash_for_nullify =
compute_note_hash_for_read_or_nullify ( CardNoteMethods ,
self ) ; let secret =
get_nullifier_secret_key ( self . owner ) ; pedersen_hash ( [ note_hash_for_nullify , secret . high , secret . low , ] , 0 ) }
pub
fn
set_header ( & mut
self , header :
```

```
NoteHeader )

{ self . header = header ; }

// Broadcasts the note as an encrypted log on L1. pub

fn

broadcast ( self , context :

& mut

PrivateContext , slot :

Field )

{ let encryption_pub_key =

get_public_key ( self . owner ) ; emit_encrypted_log ( context , ( * context ) . this_address ( ) , slot , encryption_pub_key ,
self . serialize ( ) , ) ; } }

fn

deserialize ( serialized_note :

[ Field ;

CARD_NOTE_LEN ] )

->

CardNote

{ CardNote :: deserialize ( serialized_note ) }

fn

serialize ( note :

CardNote )

->

[ Field ;

CARD_NOTE_LEN ]

{ note . serialize ( ) }

fn

compute_note_hash ( note :

CardNote )

->

Field

{ note . compute_note_hash ( ) }

fn

compute_nullifier ( note :

CardNote , context :

& mut

PrivateContext )

->

Field
```

{ note . compute_nullifier (context) }

fn

compute_nullifier_without_context (note :

CardNote)

->

Field

{ note . compute_nullifier_without_context ( ) }

fn

get_header (note :

CardNote)

->

NoteHeader

{ note . header }

fn

set_header (note :

& mut

CardNote , header :

NoteHeader)

{ note . set_header (header) }

// Broadcasts the note as an encrypted log on L1. fn

broadcast (context :

& mut

PrivateContext , slot :

Field , note :

CardNote)

{ note . broadcast (context , slot) ; }

global CardNoteMethods

=

NoteInterface

{ deserialize , serialize , compute_note_hash , compute_nullifier , compute_nullifier_without_context , get_header , set_header , broadcast , } ; Now:

use

dep :: aztec :: { note :: { note_header :: NoteHeader , note_interface :: NoteInterface , utils :: compute_note_hash_for_read_or_nullify , } , oracle :: { nullifier_key :: get_nullifier_secret_key , get_public_key :: get_public_key , } , log :: emit_encrypted_log , hash :: pedersen_hash , context :: PrivateContext , protocol_types :: { address :: AztecAddress , traits :: { Serialize ,

Deserialize ,

Empty } } } ;

// Shows how to create a custom note

```
global CARD_NOTE_LEN :

Field

=

1 ;

impl

CardNote

{ pub

fn

new ( owner :

AztecAddress )

->

Self

{ CardNote

{ owner , } } }

impl

NoteInterface

for

CardNote

{ fn

compute_note_content_hash ( self )

->

Field

{ pedersen_hash ( [ self . owner . to_field ( ) , ] , 0 ) }

fn

compute_nullifier ( self , context :

& mut

PrivateContext )

->

Field

{ let note_hash_for_nullify =

compute_note_hash_for_read_or_nullify ( self ) ; let secret = context . request_nullifier_secret_key ( self . owner ) ;
pedersen_hash ( [ note_hash_for_nullify , secret . high , secret . low , ] , 0 ) }

fn

compute_nullifier_without_context ( self )

->

Field

{ let note_hash_for_nullify =

compute_note_hash_for_read_or_nullify ( self ) ; let secret =
get_nullifier_secret_key ( self . owner ) ; pedersen_hash ( [ note_hash_for_nullify , secret . high , secret . low , ] , 0 ) }

fn

set_header ( & mut

self , header :

NoteHeader )

{ self . header = header ; }

fn

get_header ( note :

CardNote )

->

NoteHeader

{ note . header }

fn

serialize_content ( self )

->

[ Field ;

CARD_NOTE_LEN ] { [ self . owner . to_field ( ) ] }

fn

deserialize_content ( serialized_note :

[ Field ;

CARD_NOTE_LEN ] )

->

Self

{ AddressNote

{ owner :

AztecAddress :: from_field ( serialized_note [ 0 ] ) , header :

NoteHeader :: empty ( ) , } }

// Broadcasts the note as an encrypted log on L1. fn

broadcast ( self , context :

& mut

PrivateContext , slot :

Field )

{ let encryption_pub_key =

get_public_key ( self . owner ) ; emit_encrypted_log ( context , ( * context ) . this_address ( ) , slot , encryption_pub_key ,
self . serialize ( ) , ) ; } }
```
Public state must implement Serialize and Deserialize traits.

It is still possible to manually implement the storage initialization (for custom storage wrappers or internal types that don't implement the required traits). For the above example, theimpl Storage section would look like this:

```
impl
```

Storage

{ fn

init (context :

Context)

->

Self

{ Storage

{ leader :

PublicMutable :: new (context , 1) , legendary_card :

PrivateMutable :: new (context ,

2) , profiles :

Map :: new (context , 3 , | context , slot |

{ PrivateMutable :: new (context , slot) } ,) , test :

Set :: new (context ,

4) , imm_singleton :

PrivateImmutable :: new (context ,

4) , } } }

0.20.0

[Aztec.nr] Changes toNoteInterface

1. Changingcompute_nullifier()
2. tocompute_nullifier(private_context: PrivateContext)
3. This API is invoked for nullifier generation within private functions. When using a secret key for nullifier creation, retrieve it through:
4. private_context.request_nullifier_secret_key(account_address)
5. The private context will generate a request for the kernel circuit to validate that the secret key does belong to the account.
6. Before:
7. pub
8. fn
9. compute_nullifier
10. (
11. self
12.)
13. ->
14. Field
15. {
16. let
17. secret
18. =
19. oracle
20. .
21. get_secret_key
22. (
23. self
24. .
25. owner
26.)
27. ;
28. pedersen_hash

29. (
30. [
31. self
32. .
33. value
34. ,
35. secret
36. .
37. low
38. ,
39. secret
40. .
41. high
42. ,
43.]
44.)
45. }
46. Now:
47. pub
48. fn
49. compute_nullifier
50. (
51. self
52. ,
53. context
54. :
55. &
56. mut
57. PrivateContext
58.)
59. ->
60. Field
61. {
62. let
63. secret
64. =
65. context
66. .
67. request_nullifier_secret_key
68. (
69. self
70. .
71. owner
72.)
73. ;
74. pedersen_hash
75. (
76. [
77. self
78. .
79. value
80. ,
81. secret
82. .
83. low
84. ,
85. secret
86. .
87. high
88. ,
89.]
90.)
91. }
92. New APIcompute_nullifier_without_context()
93. .
94. This API is used within unconstrained functions where the private context is not available, and using an unverified nullifier key won't affect the network or other users. For example, it's used incompute_note_hash_and_nullifier()
95. to compute values for the user's own notes.

96. pub
97. fn
98. compute_nullifier_without_context
99. (
100. self
101.)
102. ->
103. Field
104. {
105. let
106. secret
107. =
108. oracle
109. .
110. get_nullifier_secret_key
111. (
112. self
113. .
114. owner
115.)
116. ;
117. pedersen_hash
118. (
119. [
120. self
121. .
122. value
123. ,
124. secret
125. .
126. low
127. ,
128. secret
129. .
130. high
131. ,
132.]
133.)
134. }
135. Note that theget_secret_key
136. oracle API has been renamed toget_nullifier_secret_key
137. .

0.18.0

[Aztec.nr] Removeprotocol_types

from Nargo.toml

Theprotocol_types package is now being reexported fromaztec . It can be accessed throughdep::aztec::protocol_types .

aztec

=

{

git

"https://github.com/AztecProtocol/aztec-packages/" ,

tag

"aztec-packages-v0.41.0" ,

directory

"yarn-project/aztec-nr/aztec"

}

[Aztec.nr] key type definition in Map

TheMap class now requires defining the key type in its declaration whichmust implement theToField trait.

Before:

struct

Storage

{ balances :

Map < PublicMutable < Field ,

FIELD_SERIALIZED_LEN

}

let user_balance = balances . at (owner . to_field ( )) Now:

struct

Storage

{ balances :

Map < AztecAddress ,

PublicState < Field ,

FIELD_SERIALIZED_LEN

}

let user_balance = balances . at (owner)

[js] Updated function names

- waitForSandbox
- renamed towaitForPXE
- in@aztec/aztec.js
- getSandboxAccountsWallets
- renamed togetInitialTestAccountsWallets
- in@aztec/accounts/testing

0.17.0

[js] New@aztec/accounts

package

Before:

import

{ getSchnorrAccount }

from

"@aztec/aztec.js" ;

// previously you would get the default accounts from theaztec.js package: Now, import them from the new package@aztec/accounts import

{ getSchnorrAccount }

from

"@aztec/accounts" ;

Typed Addresses

Address fields in Aztec.nr now is of typeAztecAddress as opposed toField

Before:

unconstrained fn

compute_note_hash_and_nullifier (contract_address :

Field , nonce :

Field , storage_slot :

Field , serialized_note :

[Field ;

VALUE_NOTE_LEN])

->

[Field ;

4]

{ let note_header =

NoteHeader :: new (_address , nonce , storage_slot) ; ... Now:

unconstrained fn

compute_note_hash_and_nullifier (contract_address :

AztecAddress , nonce :

Field , storage_slot :

Field , serialized_note :

[Field ;

VALUE_NOTE_LEN])

->

pub

[Field ;

4]

{ let note_header =

NoteHeader :: new (contract_address , nonce , storage_slot) ; Similarly, there are changes when using aztec.js to call functions.

To parse aAztecAddress to BigInt, use.inner Before:

const tokenBigInt =

await bridge . methods . token ( ) . simulate ( ) ; Now:

const tokenBigInt =

(await bridge . methods . token ( ) . simulate ( )) . inner ;

[Aztec.nr] Addprotocol_types

to Nargo.toml

aztec

=

{

git

"https://github.com/AztecProtocol/aztec-packages/" ,

tag

"aztec-packages-v0.41.0" ,

directory

"yarn-project/aztec-nr/aztec"

} protocol_types

=

{

git

"https://github.com/AztecProtocol/aztec-packages/" ,

tag

"aztec-packages-v0.41.0" ,

directory

"yarn-project/noir-protocol-circuits/crates/types" }

[Aztec.nr] moving compute_address func to AztecAddress

Before:

let calculated_address =

compute_address (pub_key_x , pub_key_y , partial_address) ; Now:

let calculated_address =

AztecAddress :: compute (pub_key_x , pub_key_y , partial_address) ;

[Aztec.nr] movingcompute_selector

to FunctionSelector

Before:

let selector =

compute_selector ("_initialize((Field))") ; Now:

let selector =

FunctionSelector :: from_signature ("_initialize((Field))") ;

[js] Importing contracts in JS

Contracts are now imported from a file with the type's name.

Before:

import

{ TokenContract }

from

"@aztec/noir-contracts/types" ; Now:

import

{ TokenContract }

from

"@aztec/noir-contracts/Token" ;

[Aztec.nr] Aztec example contracts location change in Nargo.toml

Aztec contracts are now moved outside of thesrc folder, so you need to update your imports.

Before:

easy_private_token_contract

{ git =

"https://github.com/AztecProtocol/aztec-packages/" , tag = "v0.16.9" , directory =

"noir-projects/noir-contracts/contracts/easy_private_token_contract" } Now, just remove thesrc folder,:

easy_private_token_contract

{ git =

"https://github.com/AztecProtocol/aztec-packages/" , tag = "v0.17.0" , directory =

"noir-projects/noir-contracts/contracts/easy_private_token_contract" } Edit this page

Previous PXE Next Aztec Connect Sunset