Lesson 12

Running a [starknet node](https://twitter.com/emilianobonassi/status/1503005250999005185)

Starknet Fees (https://medium.com/starkware/starknet-alpha-0-8-0-16e046e0f94b)

FEE STRUCTURE ON 0.8.0

"On 0.8.0, fees will be collected according to the computational complexity alone, while StarkWare will still subsidize L1 communication cost.
We will update the fee mechanism to include these L1 operation and communication costs over the next few weeks.
The payment will be collected atomically with the transaction execution on StarkNet L2. See the fees documentation for an in-depth description."

RUGPULL ON STARKNET

https://twitter.com/thiscrypto_life/status/1503440185681657856

Plonkish protocols

(fflonk, turbo PLONK, ultra PLONK, plonkup, and recently plonky2.)

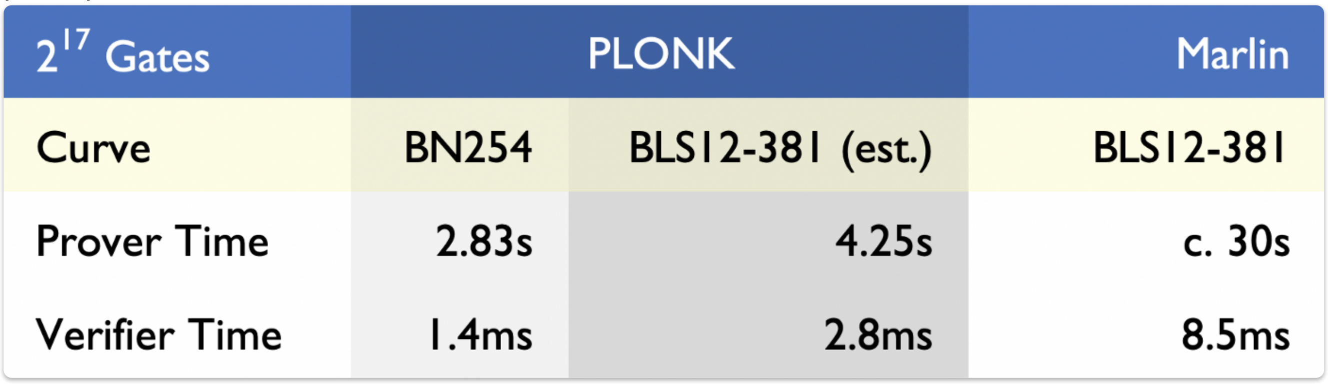
Before PLONK

Early SNARK implementations such as Groth16 depend on a common reference string, this is a large set of points on an elliptic curve.
Whilst these numbers are created out of randomness, internally the numbers in this list have strong algebraic relationships to one another. These relationships are used as short-cuts for the complex mathematics required to create proofs.
Knowledge of the randomness could give an attacker the ability to create false proofs.

A trusted-setup procedure generates a set of elliptic curve points $G, G \cdot s, G \cdot s^2 \ldots . G \cdot s^n$, as well as $G2 \cdot s$, where G and $G2$ are the generators of two elliptic curve groups and s is a secret that is forgotten once the procedure is finished (note that there is a multi-party version of this setup, which is secure as long as at least one of the participants forgets their share of the secret).
The Aztec reference string looks like
$(x.[1], x^2.[1], \ldots, x^{1\,0\,0\,6\,6\,3\,9\,6}.[1])$

A problem remains that if you change your program and introduce a new circuit you require a fresh trusted setup.

In January 2019 Mary Maller and Sean Bowe et al released SONIC that has a universal setup, with just one setup, it could validate any conceivable circuit (up to a predefined level of complexity).
This was unfortunately not very efficient, PLONK managed to optimise the process to make the proof process feasible.



2^{17} Gates	PLONK		Marlin
Curve	BN254	BLS12-381 (est.)	BLS12-381
Prover Time	2.83s	4.25s	c. 30s
Verifier Time	1.4ms	2.8ms	8.5ms

PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge
Also see Understanding PLONK

Trusted Setup

This is still needed, but it is a "universal and updateable" trusted setup.

- There is one single trusted setup for the whole scheme after which you can use the scheme with any program (up to some maximum size chosen when making the setup).
- There is a way for multiple parties to participate in the trusted setup such that it is secure as long as any one of them is honest, and this multi-party procedure is fully sequential:

Polynomial Commitments

The transformations in PLONK are similar to the ones we saw in the lesson about R1CS and QAPs

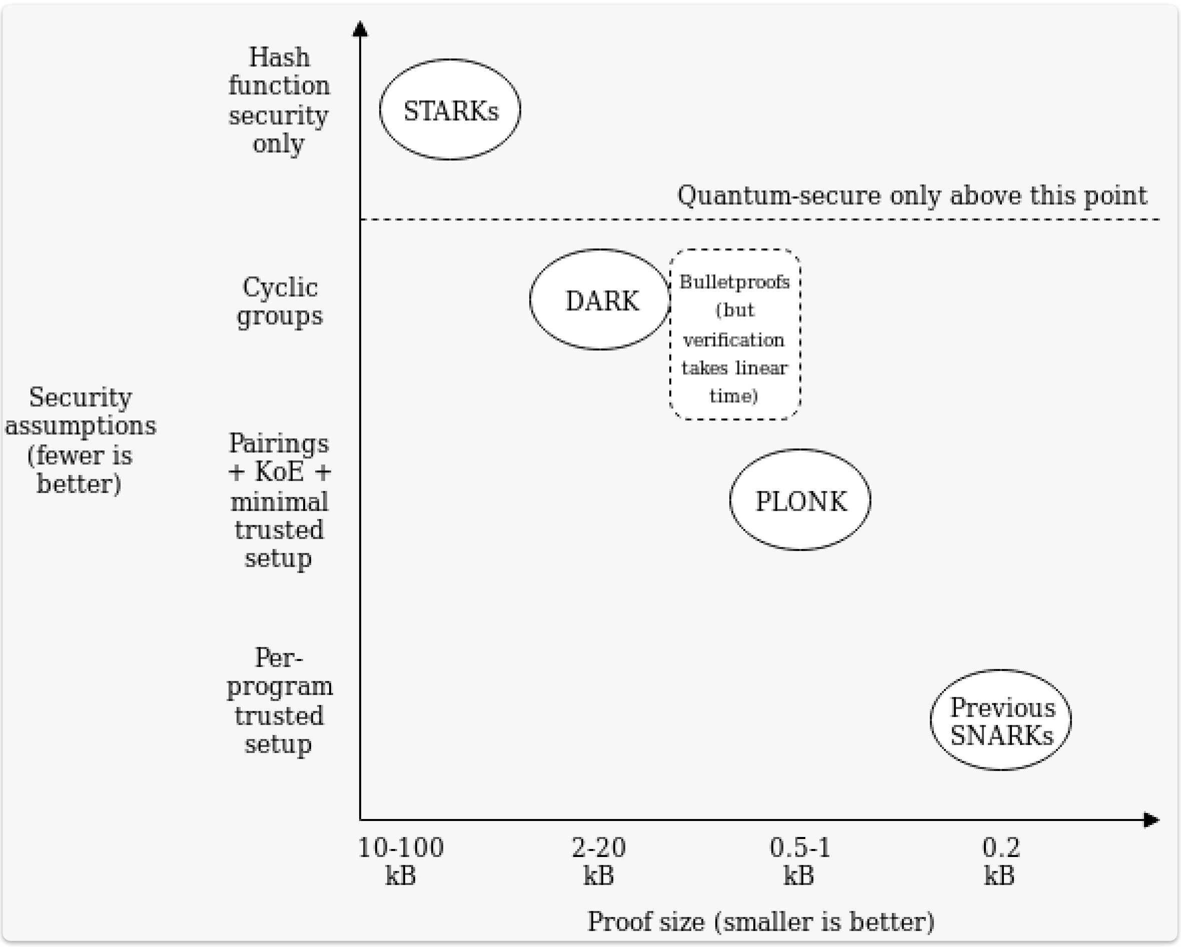
A problem we face is that we can construct polynomials to represent our contraints, but these end up being being quite big.

A polynomial commitment is a short object that "represents" a polynomial, and allows you to verify evaluations of that polynomial, without needing to actually contain all of the data in the polynomial.

That is, if someone gives you a commitment c representing $P(x)$, they can give you a proof that can convince you, for some specific z, what the value of $P(z)$ is.

There is a further mathematical result that says that, over a sufficiently big field, if certain kinds of equations (chosen before z is known) about polynomials evaluated at a random z are true, those same equations are true about the whole polynomial as well.
For example,
if $P(z) \cdot Q(z) + R(z) = S(z) + 5$, where z is a specific point
then we know that it's overwhelmingly likely that
$P(x) \cdot Q(x) + R(x) = S(x) + 5$ in general.

PLONK uses Kate commitments based on trusted setup and elliptic curve pairings, but these can be swapped out with other schemes, such as FRI (which would turn PLONK into a kind of STARK



This means the arithmetization - the process for converting a program into a set of polynomial equations can be the same in a number of schemes.
If this kind of scheme becomes widely adopted, we can thus expect rapid progress in improving shared arithmetization techniques.

Introduction to Aztec

Aztec give some useful definitions of Privacy, Anonymity and Confidentiality

***Privacy:** all aspects of a transaction remain hidden from the public or third parties.*

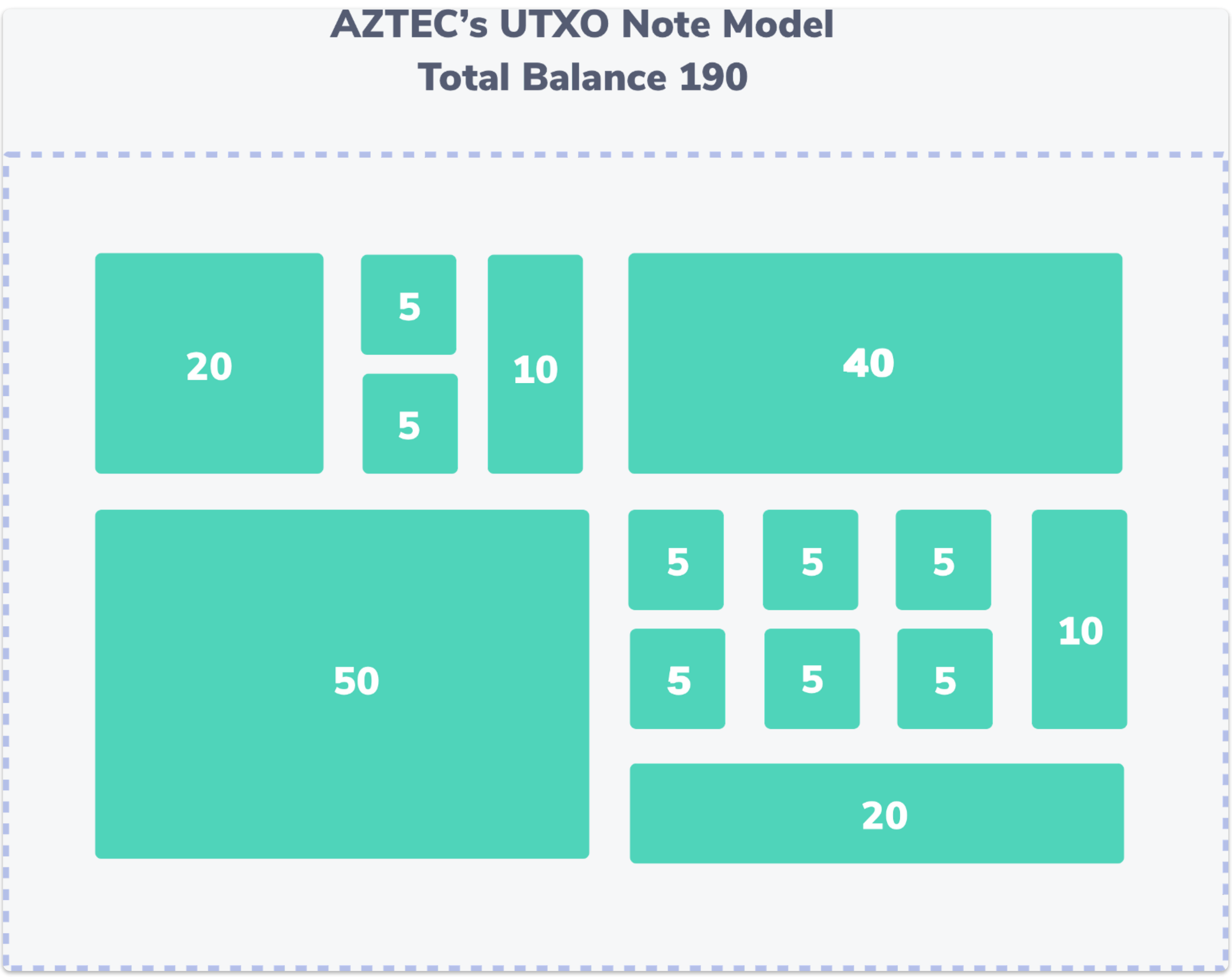
***Confidentiality:** the inputs and outputs of a transaction are hidden from the public but the transaction parties remain public.*

***Anonymity:** the inputs and outputs of a transaction are public but the transaction graph is obscured from one transaction to the next, preventing the identification of the transaction parties.*

From Aztec Documentation
See also [yellow paper] (https://hackmd.io/@aztec-network/ByzgNxBfd)

The AZTEC protocol was created to enable privacy on public blockchains. It enables logical checks to be performed on encrypted values without the underlying values being revealed to the blockchain.
The inputs and outputs of a transaction are encrypted using a series of zero-knowledge proofs and homomorphic encryption, yet the blockchain can still test the logical correctness of these encrypted statements.

AZTEC follows a UTXO model similar to that of Bitcoin and ZCash.
The core of any AZTEC transaction is a ***Note.*** The state of notes are managed by a ***Note Registry*** for any given asset.
The AZTEC protocol does not represent 'value' like a traditional balance, which maps owners to how much they own.
The user's balance of any AZTEC asset is made up of the sum of all of the valid notes their address owns in a given ***Note Registry***.



Note Details

A note contains the following public information:

- An AZTEC commitment: an encrypted representation of how much 'value' the note holds
- An Ethereum address of the note's owner

A note has the following private information

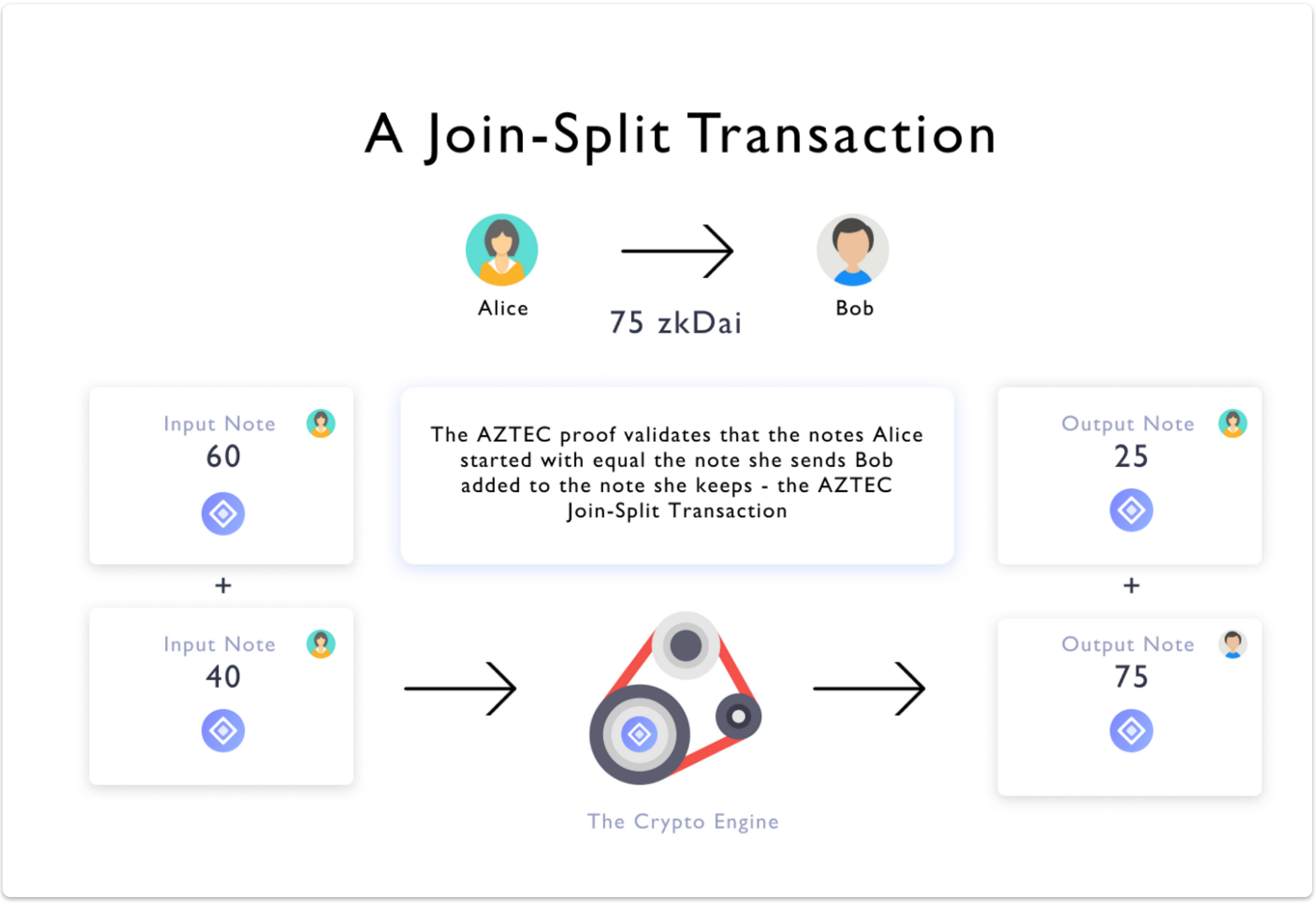
- The value of the note

- The note’s ***viewing key***. Knowledge of the viewing key enables a person to decrypt the note (but not spend it)
  One owner can have multiple notes.

A digital asset that conforms to the AZTEC protocol will contain a **note registry**, which allows a smart contract to recover the public information of every **unspent** note that currently exists.

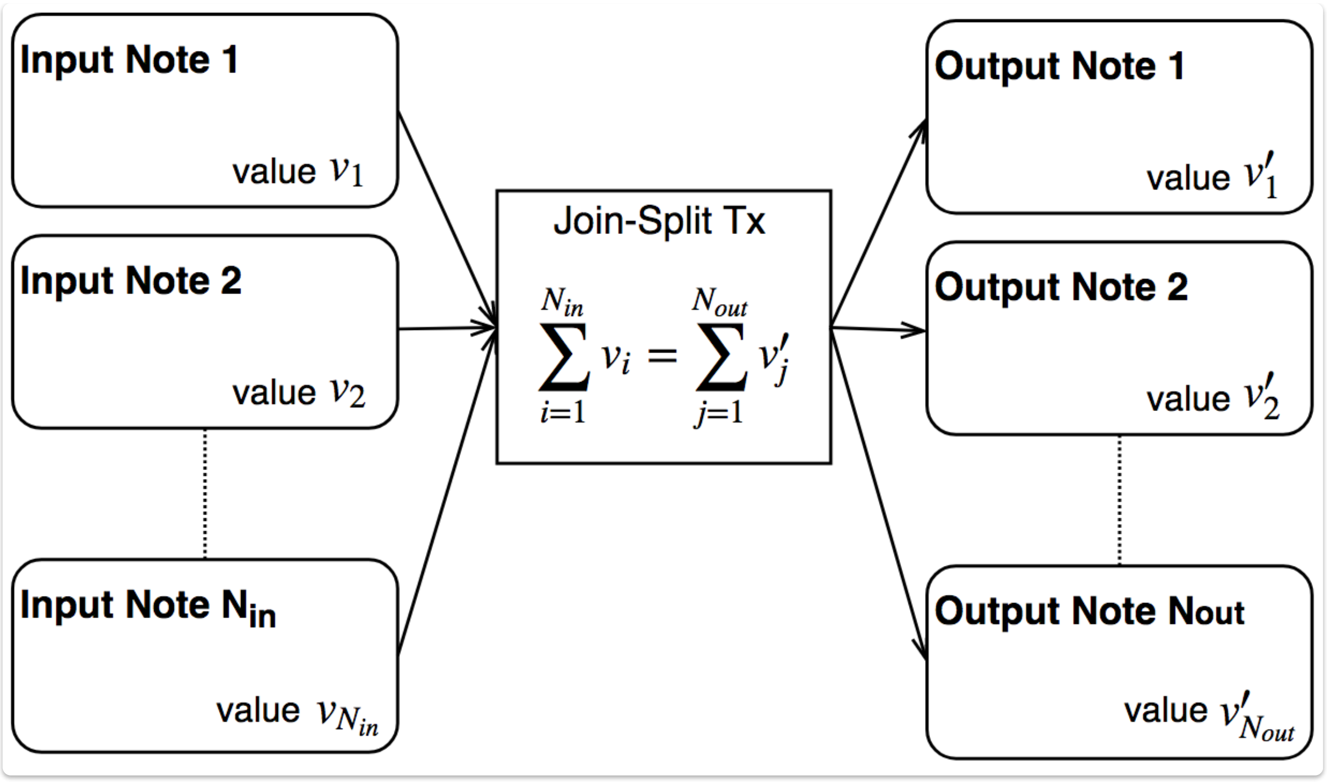
An AZTEC note owner can ‘spend’ their notes in a join-split style confidential transaction.
In this transaction, the note owner will destroy some unspent AZTEC notes they own. In their place, they will create a set of new notes. The sum of the *values* of the new notes must be equal to the sum of the *values* of the old notes, plus a *public* commitment

A Join Split transaction
From (https://medium.com/aztec-protocol/aztec-how-the-ceremony-works-5c23a54e2dd9)


A Join-Split Transaction
Alice
75 zkDai
Bob
Input Note
60
+
Input Note
40
The AZTEC proof validates that the notes Alice started with equal the note she sends Bob added to the note she keeps - the AZTEC Join-Split Transaction
The Crypto Engine
Output Note
25
+
Output Note
75

Suppose Alice has a cluster of notes summing to 100zkDAI, and wants to send 75 zkDAI to Bob — we’ll handle this much like change in a shop.

Alice takes notes of size 60 zkDAI and 40 zkDAI (these are now called input notes — she needs both, because neither can cover the 75 zkDAI she’s wiring to Bob), and she will create two output notes — 75 zkDAI for Bob, 25 zkDAI as change.
In general



Note this is exactly how the UTXO model works— but AZTEC transactions need to be confidential. And Ethereum needs to validate them
i.e. check that $60 + 40 = 75 + 25$ in our example.

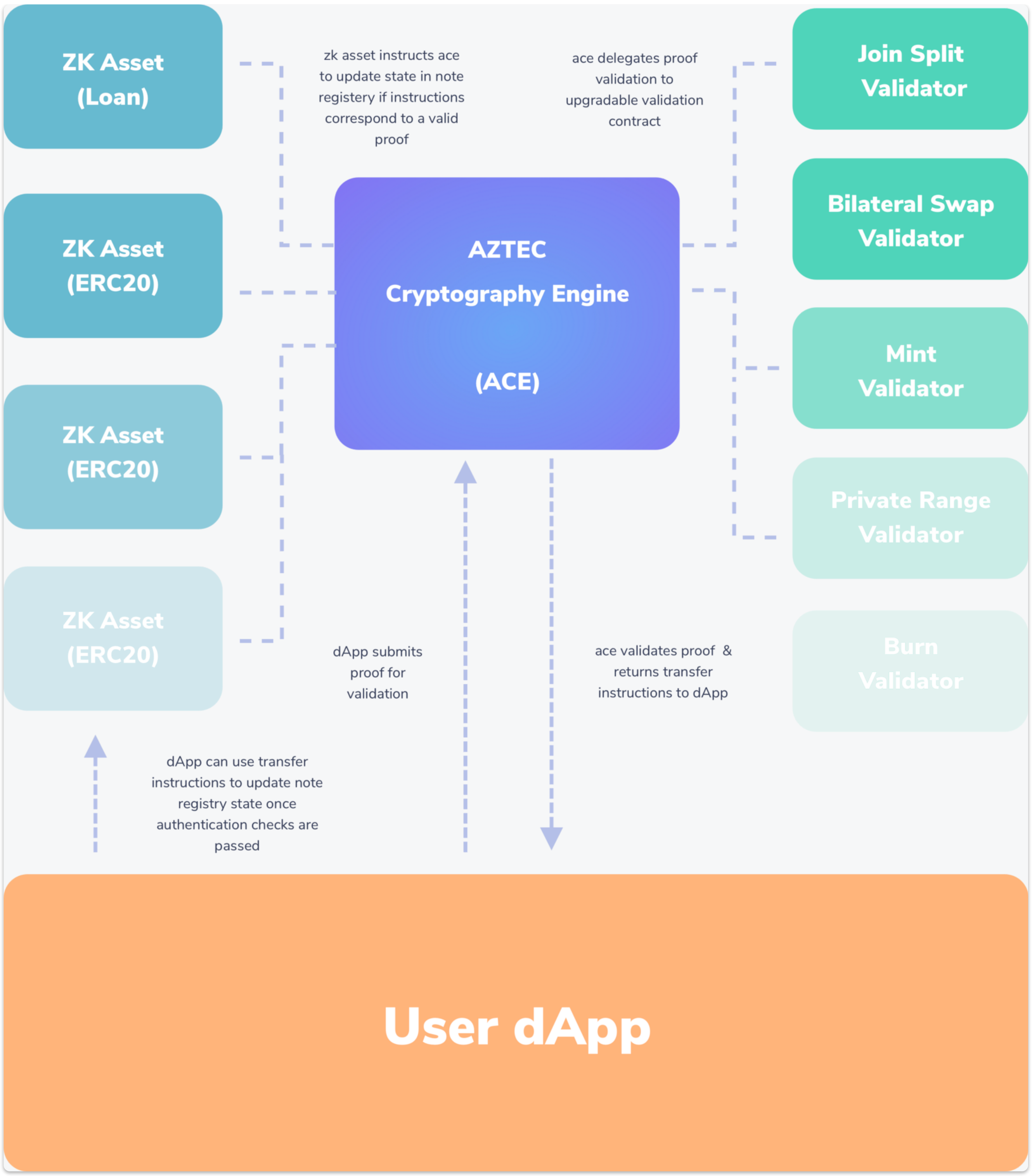
This is possible using the homomorphic additive property of elliptic curves

Alice would create an AZTEC zero-knowledge proof that proves this relationship in zero-knowledge (i.e. Alice does not reveal to anybody how much the notes are actually worth, just that the balancing relationship holds).

The AZTEC token smart contract will then validate this zero-knowledge proof, destroy Alice's input notes and then create the output notes in its note registry.

When Alice is creating Bob's notes, she constructs note viewing keys that Bob will be able to identify, via a non-interactive secret-sharing protocol.
Bob is dependent on Alice to act 'trustfully' in this regard and not provide viewing keys that can be decoded by observers. This is already implicitly required—after all Alice could broadcast to the world how much she is sending Bob if she did not want the transaction to be confidential.

To achieve interoperability with other DApps, all AZTEC assets share a common trusted setup and their state is managed by a single smart contract, the AZTEC Cryptography Engine or ACE


ZK Asset (Loan)
ZK Asset (ERC20)
ZK Asset (ERC20)
ZK Asset (ERC20)
zk asset instructs ace to update state in note registery if instructions correspond to a valid proof
ace delegates proof validation to upgradable validation contract
AZTEC Cryptography Engine (ACE)
Join Split Validator
Bilateral Swap Validator
Mint Validator
Private Range Validator
Burn Validator
dApp submits proof for validation
ace validates proof & returns transfer instructions to dApp
dApp can use transfer instructions to update note registry state once authentication checks are passed
User dApp

Example Confidential Transaction
https://etherscan.io/tx/0xf9a101682c637f7741f281c858527d17036f4df284b7064bd1ca44531ab88374

Anonymity

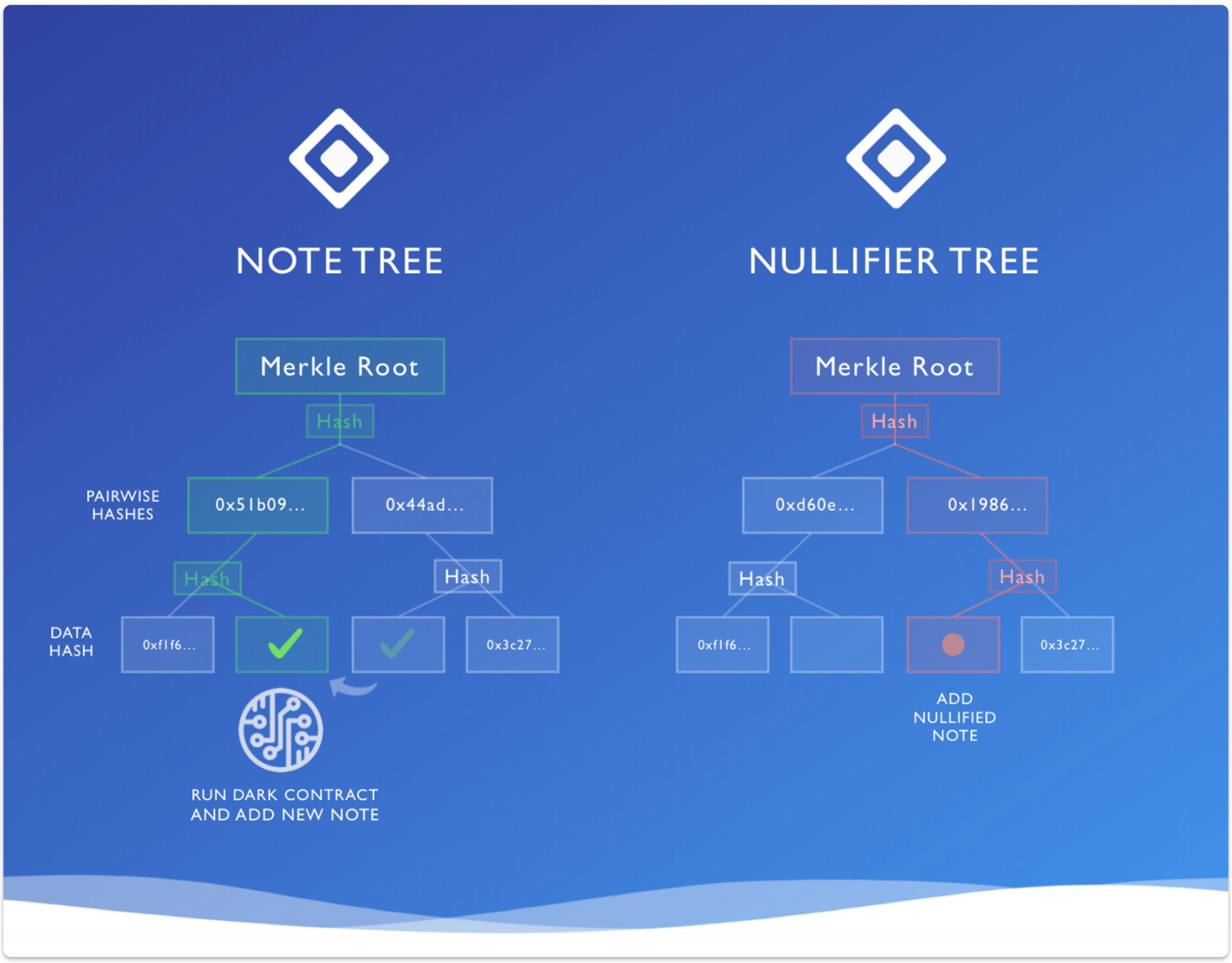
AZTEC notes have 'owners' defined by Ethereum addresses.
On the surface, note ownership is not anonymous (e.g. people can see my ethereum address has a zero-knowledge DAI note); the AZTEC protocol includes a Monero-style stealth-address protocol to derive Ethereum addresses that are single-use and cannot be linked to any other Ethereum address (e.g. if you have an AZTEC wallet, I can 'send' a note to an Ethereum address you control, but nobody but you and me will know this is the case).
The protocol supports both stealth addresses (which require a specific wallet to work; you need

two public/private key pairs so a regular Ethereum account won't work) and regular Ethereum addresses (which are not anonymous — if you own a note everybody will be able to see that).

Data Structures and nullifiers

The above is an abstraction of the process, in order to do this in practice 2 merkle trees are used

- A **Note Tree** of all output notes ever created, and
- A **Nullifier Tree** keeping copies of the spent notes
  The idea is — instead of deleting a note from the **Note Tree**, you need to check whether that note also turns up in the Nullifier Tree to work out if it's already spent. If it's not there, it's still spendable.



Hashes required for this :

- **Note Tree:** 30 hashes to add a new output note
- **Nullifier Tree:** 30 hashes to add a note, marking it as spent
- **Total:** 60 Hashes

Each SHA-256 hash in PLONK requires ~27,000 gates for a 64 byte input, so **60 hashes consume ~1.6m gates**.

As with Monero, because we are using finite fields we face the problem of checking the range of the inputs, for this we need range proofs.

For more details of how the proofs are constructed from the details in the trusted setup see this article

zk.money (ZK-ZK-rollups)

Aztec's privacy architecture can enable simple asset transfers, allowing users to privately send $DAI, $ETH, and $renBTC.

Private ZK-rollups provide the scaling benefits of rollups, in addition the transaction inputs/outputs are encrypted.
The zero-knowledge proof that proves the correctness of every transaction also proves that the encrypted data was correctly derived from the non-encrypted 'plaintext' data. But the plaintext is known only to the users that constructed their private transactions.

The rollup cannot simply process a list of transactions like before, it must verify a list of zero-knowledge proofs that each validate a private transaction. (This extra layer of zero-knowledge proof verification is why they're called 'ZK-ZK-rollups').

The result is reduced-cost transactions with full transaction privacy.
Both the identities of senders/recipients are hidden, as well as the values being transferred.

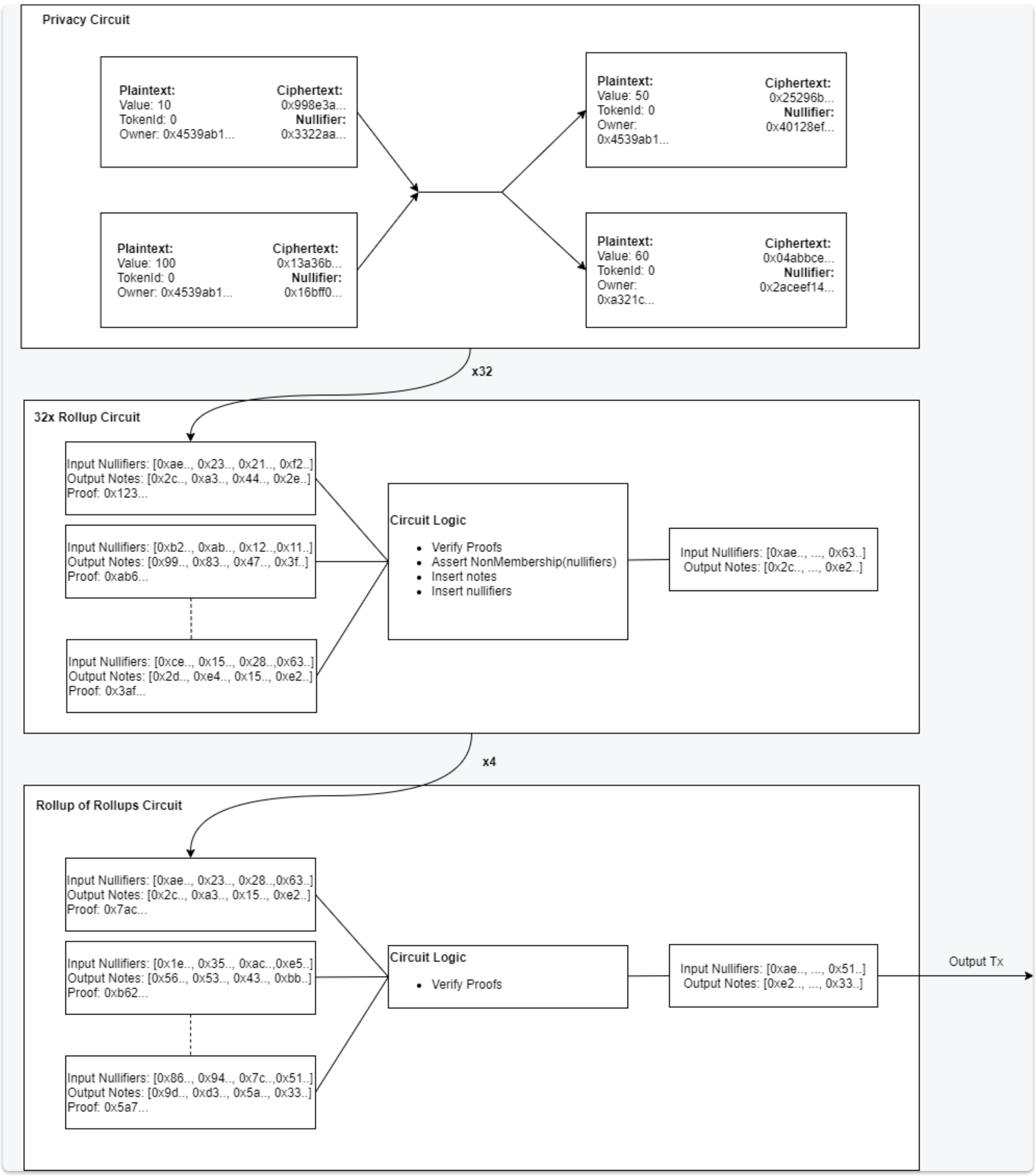
Despite this, users of the protocol can have complete confidence in the correctness of transactions (no double spending etc), because only legitimate transactions can produce a valid zero-knowledge proof of correctness.

The architecture is composed of two programs that are encoded into ZK-SNARK 'circuits': A **privacy** circuit and a **rollup** circuit.

The privacy circuit proves the correctness of a single private transaction. It is constructed by users that want to send private transactions, directly on their hardware to ensure no secrets are leaked.

The rollup circuit validates the correctness of a batch of privacy proofs (currently 128) and updates the rollup's database with the new encrypted transaction data.

The rollup proofs are constructed by a rollup provider, a 3rd party that has access to significant computing resources (for the moment, Aztec are the rollup provider, later Aztec will decentralize the rollup service.
The rollup provider is completely untrusted. They do not have access to any user data and only see the encrypted outputs of privacy proofs. This also makes it impossible to launch selective censorship attacks, because all transactions look like uniform random numbers.


Privacy Circuit
Plaintext: Value: 10 TokenId: 0 Owner: 0x4539ab1...
Ciphertext: 0x998e3a... Nullifier: 0x3322aa...
Plaintext: Value: 100 TokenId: 0 Owner: 0x4539ab1...
Ciphertext: 0x13a36b... Nullifier: 0x16bff0...
Plaintext: Value: 50 TokenId: 0 Owner: 0x4539ab1...
Ciphertext: 0x25296b... Nullifier: 0x40128ef...
Plaintext: Value: 60 TokenId: 0 Owner: 0xa321c...
Ciphertext: 0x04abbce... Nullifier: 0x2aceef14...
x32
32x Rollup Circuit
Input Nullifiers: [0xae.., 0x23.., 0x21.., 0xf2..] Output Notes: [0x2c.., 0xa3.., 0x44.., 0x2e..] Proof: 0x123...
Input Nullifiers: [0xb2.., 0xab.., 0x12..,0x11..] Output Notes: [0x99.., 0x83.., 0x47.., 0x3f..] Proof: 0xab6...
Input Nullifiers: [0xce.., 0x15.., 0x28..,0x63..] Output Notes: [0x2d.., 0xe4.., 0x15.., 0xe2..] Proof: 0x3af...
Circuit Logic
Verify Proofs
Assert NonMembership(nullifiers)
Insert notes
Insert nullifiers
Input Nullifiers: [0xae.., ..., 0x63..] Output Notes: [0x2c.., ..., 0xe2..]
x4
Rollup of Rollups Circuit
Input Nullifiers: [0xae.., 0x23.., 0x28..,0x63..] Output Notes: [0x2c.., 0xa3.., 0x15.., 0xe2..] Proof: 0x7ac...
Input Nullifiers: [0x1e.., 0x35.., 0xac..,0xe5..] Output Notes: [0x56.., 0x53.., 0x43.., 0xbb..] Proof: 0xb62...
Input Nullifiers: [0x86.., 0x94.., 0x7c..,0x51..] Output Notes: [0x9d.., 0xd3.., 0x5a.., 0x33..] Proof: 0x5a7...
Circuit Logic
Verify Proofs
Input Nullifiers: [0xae.., ..., 0x51..] Output Notes: [0xe2.., ..., 0x33..]
Output Tx

Aztec Connect

From article

Aztec Connect allows users to bridge private assets to mainnet for a DeFi interaction and return to Aztec in the same transaction.

Aztec Connect serves as a bridge to Ethereum, allowing users to bring privacy-shielded zk-assets on Aztec to public DeFi protocols on Ethereum.

As a result, users save 80–90% on gas fees with privacy thrown in for free.

Users deposit funds into Aztec’s Layer 1 rollup contract, and privacy-shielded notes are minted by the Aztec system, identified by their zk- prefixes (e.g. zkETH and zkDAI).

Previously, functionality and usability of zk-assets was limited to internal-to-Aztec private sends and private withdrawals to Ethereum L1 addresses.

Now, users can do *any* DeFi transaction supported by an Aztec Connect Bridge Contract

- a 50 to 100-line interface allowing Aztec's roll-up to interact with a given Layer 1 smart contract.

Looking at a Uniswap swap

A basic Uniswap swap, which costs ~130,000 gas

Because the Aztec rollup supports large batch sizes, up to 896 transactions at launch, courtesy of Flashbots, the cost of validating Aztec zero-knowledge proofs is amortized across many users.
At current proof construction costs, this is just 1,875 gas per transaction.

Say 100 users want to execute the same swap on Uniswap
Splitting the cost of the Uniswap transaction and cost of posting data on Ethereum, they each pay 15,762 gas.

In total, the cost of a Uniswap transaction becomes just 17,637 gas,
an 86% savings over L1.
Swaps become 7.4x cheaper, with iron-clad privacy as a bonus.

Aztec Connect *vastly* expands Aztec Network's capabilities at launch, adding whitelisted DeFi functionality with select partners.
Any developer looking to integrate Aztec to an existing DeFi application can write an Aztec Connect Bridge Contract.

To contribute see repo

Noir Language

See Noir
Noir is a domain specific language for creating and verifying proofs. Design choices are influenced heavily by Rust.

Noir is much simple and flexible in design as it does not compile immediately to a fixed NP-complete language. Instead Noir compiles to an intermediate language which itself can be compiled to an arithmetic circuit or a rank-1 constraint system.

An example function

```
fn main(message : [62]u8, index : Field, hashpath : [40]Field, root : Field) {
        priv leaf = std::hash::hash_to_field(message);
        priv is_member = std::merkle::check_membership(root, leaf, index, 
hashpath);
        constrain is_member == 1;
}
```