The AZTEC Protocol

Dr Zachary J. Williamson

AZTEC

Version 1.0.1

December 4, 2018

Abstract

The Anonymous Zero-knowledge Transactions with Efficient Communication (AZTEC) protocol describes a set of zero-knowledge proofs that define a confidential transaction protocol, designed for use within blockchain protocols that support Turing-complete general-purpose computation.

The protocol is utilizes a commitment scheme that enables the efficient verification of range proofs. This is combined with a set of zero-knowledge Sigma protocols to enable efficiently verifiable confidential transactions.

The reference implementation of the AZTEC protocol is implemented on the Ethereum public blockchain. It can be used to create confidential representations of existing digital assets. At the time of publication, AZTEC zero-knowledge proofs cost approximately 840,000 'gas' to verify on the Ethereum main-net.

Acknowledgements

I would like to thank Professor Jens Groth, UCL, for his invaluable advice, guidance and contributions over the last few months. Without his assistance this paper would not have been possible.

1 Motivations behind the AZTEC protocol

Blockchain technologies enable transactions to an immutable ledger that can be composed to define a system of digital value transfer. For example, the Bitcoin [1] blockchain defines an ownership registry of the Bitcoin cryptocurrency, where owners of the currency can collectively reach a consensus on how much Bitcoin individuals hold without a centralized entity acting as a source of trust.

This trustless attestation has come at the cost of confidentiality - in order for a transaction to be validated both the inputs, outputs and the transaction validation algorithm must be public. There is no concept of a 'privileged actor' that has access to private information.

A **confidential transaction** describes a transaction on a public blockchain network, where the value of the transaction is hidden. Existing blockchains have implemented confidential transactions, such as Monero [2] or ZCash [3]. However confidential transactions have not yet been implemented on a blockchain platform which also supports general-purpose computation through a Turing-complete virtual machine [4]. This limits existing confidential transactions to transactions where the value being transferred is the native cryptocurrency of the blockchain in question.

The AZTEC protocol enables confidential transactions in a generic form that can be implemented on blockchains that support general-purpose computation, such as the Ethereum blockchain [4]. The AZTEC protocol can be attached to existing digital assets defined on these platforms (for example, digital assets that conform to the ERC20 token standard [5]). The protocol also enables confidential cross-asset trades for digital assets defined on the same blockchain platform via confidential, zero-knowledge decentralized exchanges.

An implementation of the AZTEC protocol has been instantiated on the Ethereum blockchain via a set of op-codes that are defined by the protocol's Turing-complete virtual machine (a 'smart contract'). Gas costs for transactions that verify AZTEC proofs are approximately 840,000 gas for a simple 'join-split' transaction with two inputs and two outputs, although there are efficiency savings when batching multiple proofs together.

1.1 The AZTEC 'note'

An AZTEC note is an encrypted representation of abstract value. How this abstract representation maps to real quantities is a higher-level detail of digital assets that utilize the protocol. The 'note' is an output of the AZTEC commitment function from section 3. It is comprised of a tuple of elliptic curve commitments and three scalars: a **viewing key**, a **spending key** and a **message**. Knowledge of the viewing key allows the note to be decrypted, revealing the message. Knowledge of the viewing key can be used to create valid join-split zero-knowledge proofs. These proofs are then signed by the spending key.

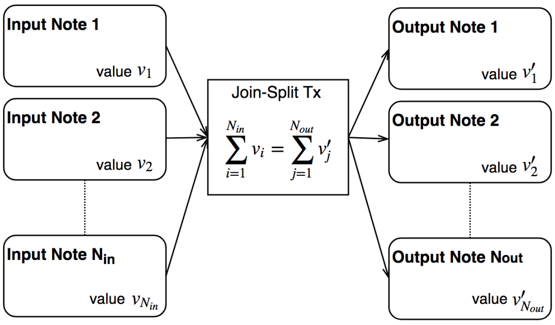

Figure 1: Join-split transactions describe the creation and destruction of notes. By defining different note 'owners' value can be transferred.

1.2 The Join-Split confidential transaction

The join-split transaction type is used by the AZTEC protocol to implement confidential transactions. Traditional digital assets are described by an balance registry - identities are linked to a single balance.

In the AZTEC protocol, this is replaced with a note registry. A note contains an encrypted balance, ownership of the note is defined as possessing knowledge of both the note's viewing key and spending key. In a join-split transaction, at least one note is taken from the note registry (join). These notes are combined and then split into at least one output note (split). The input notes are removed from the note registry and replaced with the output notes. An AZTEC verifier must be able to validate the existence of unspent AZTEC notes (and therefore have access to some form of persistent state), but the manner in which this is achieved is beyond the scope of this paper.

The AZTEC protocol describes how join-split transactions can be constructed and validated in zero-knowledge. The witnesses required to decrypt note balances is not exposed. The witnesses required to prove 'ownership' over a note are not exposed.

The AZTEC protocol validates the legitimacy of a join-split transaction using a combination of homomorphic arithmetic and range proofs. The range proof employed by the AZTEC protocol is efficient to both construct and verify (section 4). The complexity of constructing and verifying a range proof does not scale with the cardinality of the range being proven against. The range proof requires 3 elliptic curve point scalar multiplications and 1 bilinear pairing comparison to verify.

Multiple range proofs can be combined, in which case verifying the set of proofs requires 4 elliptic curve point scalar multiplications per proof and 1 bilinear pairing comparison for the complete set of proofs. Section 8 describes the efficiency of the protocol in the context of implementing a smart-contract verifier described by the Ethereum protocol's Ethereum Virtual Machine [4].

This efficiency comes at the cost of a trusted setup phase that generates a common reference string that contains a set of elliptic curve points. This set scales linearly with the cardinality of the range used by the range proof. This point set in the common reference string is required by a prover to construct proofs, but is not required by the verifier. This makes the protocol ideal for verifiers that are both computationally constrained and storage constrained, such as algorithms embedded into blockchain protocols.

1.3 Who 'owns' an AZTEC note?

When an AZTEC note is issued, a public key, Q, is defined by the transaction issuer. The witness to Q is referred to as the 'spending key'. If this note is used as an input to a future join-split transaction, the prover must provide a signature signed by the spending key. The message of this signature is a hash of the input string of the join-split transaction involving the note.

Ownership of a note is defined as possessing knowledge of the witness to Q.

The spending key is divorced from the commitment function for the AZTEC protocol. The basic protocol enables confidential transactions to be validated; defining ownership of AZTEC notes is a higher-level implementation detail that will vary depending on the capabilities of the blockchain used to implement the AZTEC protocol. This typically will involve validating a signature signed by the witness to Q, however the exact form of the signature will vary depending on the most efficient signature scheme for the platform in question.

Appendix A describes how spending keys are defined and validated for one existing implementation of the AZTEC protocol on the Ethereum blockchain.

1.3.1 How do AZTEC notes interact with traditional public digital assets?

In the context of this paper, a 'traditional' digital asset refers to a program embedded into a blockchain protocol that defines the ownership record and transfer logic for a digital representation of value. One example is an implementation of the ERC20 [5] token standard that builds on top of the Ethereum blockchain [4]. Another example of a digital asset is ethereum itself; the cryptocurrency associated with the consensus mechanism of the protocol. The AZTEC protocol can be used to interact with traditional digital assets that utilize the same underlying blockchain as the AZTEC protocol.

The protocol can be used to define two different types of digital assets: **fully anonymous** assets and **public/private** assets.

Fully anonymous assets are represented exclusively via optimized AZTEC notes and do not have a traditional digital asset representation. The only methods of trading a fully anonymous asset is via join-split transactions or via the AZTEC decentralized exchange protocol.

1.3.2 The AZTEC protocol and anonymity

The AZTEC protocol enables confidential transactions where the messages of individual notes are encrypted. Full anonymity is then provided by combining confidential zero-knowledge transactions with stealth addresses [2]. A description of how stealth addresses are incorporated into the existing Ethereum implementation of the AZTEC protocol is described in appendix A.

2 Setup

2.1 Notation and assumptions about sampling

For a set S we denote by $s \leftarrow S$ the sampling of a uniformly random element from S. When describing protocols, we often assume the abbility to sample uniformly at random from different sets, e.g., $\mathbb{Z}_p$. In reality, randomness sampling may be statistically biased or pseudorandom, in which case security is preserved up to this statistical bias or dependent on the security of the pseudorandomness. Moreover, we assume the sampling process does not reveal intermediate values so the only output is the sampled value s.

For an algorithm A, we write $y \leftarrow A(x)$ to refer to the process of running A on an input string x and obtaining output string y. For probabilistic algorithms that consume randomness r, we write $y := A(x, r)$ when we want to explicitly refer to it. If a is deterministic then $y \leftarrow A(x)$ and $y := A(x)$ represent the same process. The notation $y \leftarrow A^B(x)$ refers to running an algorithm A that has access to another algorithm (also called an oracle) B, i.e., A gives B an input and waits until an output from B has been received.

2.2 Common choice of groups with pairing and hash function

We assume that all parties that use the AZTEC protocol share a common choice of $p, \mathbb{G}, \mathbb{G}_2, \mathbb{G}_T, g, g_2, g_T, e, H$, where

- p is a prime and we let $\mathbb{Z}_p$ denote the set of integers modulo p represented by $0, \ldots, p-1$
- $\mathbb{G}, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p, written multiplicatively with neutral element 1, where the group operation and deciding group membership can be computed efficiently
- g, g_2, g_T are the group elements generating $\mathbb{G}, \mathbb{G}_2$ and $\mathbb{G}_T$ respectively
- $e : \mathbb{G} \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is an efficiently computable bilinear map (pairing) such that for all $a, b \in \mathbb{Z} : e(g^a, g_2^b) = g_T^{ab}$
- $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ is an efficienly computable hash function that takes arbitrary length inputs and returns ℓ-bit outputs

The Ethereum blockchain has native support for elliptic curve arithmetic operations of the type required by the AZTEC protocol for one specific curve, the 254-bit pairing friendly Barreto-Naehrig curve [6]. The curve was formerly used by the ZCash blockchain to implement the zerocoin protocol [3]. The curve is colloquially called **bn128** [6] in reference to the nominal security of the curve, or **bn254** in reference to the bit-length of the relevant prime field. The reference implementation of the AZTEC protocol uses this curve to define $p, \mathbb{G}, \mathbb{G}_2, \mathbb{G}_T, e$ as above.

In the reference implementation of the AZTEC protocol, H is the SHA-3 (keccak) hashing algorithm with a 256-bit output [7].

2.3 Common setup over groups with pairing

We also assume all parties share a common choice of $k_{\mathsf{max}}, h, \mu_1, \ldots, \mu_{k_{\mathsf{max}}}, t_2$, where

- k_{max} is a positive integer much smaller than p
- $h, \mu_1, \ldots, \mu_{k_{\mathsf{max}}} \in \mathbb{G}$ and $t_2 \in \mathbb{G}_2$
- $h \neq 1$
- There exists a $y \in \mathbb{Z}_p \setminus \{0, \ldots, k_{\mathsf{max}}\}$ such that $\mu_k = h^{\frac{1}{y-k}}$ for all $k = 0, \ldots, k_{\mathsf{max}}$ and $t_2 = g_2^y$

The AZTEC protocol utilizes range proofs defined over a fixed range $[0; k_{max}]$, where in the reference implementation, $k_{\mathsf{max}} = 2^{25} - 1$. Section 3.1 describes the choices considered when setting k_{max}. In the reference implementation h is chosen uniformly at random from $\mathbb{G} \setminus \{1\}$ and y uniformly at random from $\mathbb{Z}_p \setminus \{0, \ldots, k_{\mathsf{max}}\}$. After computation of $\mu_0, \ldots, \mu_{k_{\mathsf{max}}}$ and t_2 the choice of y and other internal data used in the computation is deleted.

We make the assumption that the group elements have been honestly and securely computed in this way. Specifically, we assume by some trusted procedure h has been sampled uniformly at random. For our choice of $\mathbb{G}$ this can be done using public randomness so no party needs to store any secrets associated with h. We also assume $\mu_1, \ldots, \mu_{k_{\mathsf{max}}}, t_2$ have been sampled so they are valid with respect to y. Here we trust y is generated uniformly at random and it is securely erased together with all intermediate data after the group elements have been computed. We note that the pairing can be used to check the group elements are of the right form, i.e., for $k = 1, \ldots, k_{\mathsf{max}} : e(\mu_k, t_2 g_2^{-k}) = e(h, g_2)$.

It should be noted that the AZTEC protocol's range proof relies on y being destroyed. If an attacker can recover y, they can create a commitment for an integer k that satisfy the AZTEC protocol's verifier algorithm, where $k \notin [0, k_{\mathsf{max}}]$. For any confidential currency using the AZTEC protocol, it follows that the attacker will be able to issue double-spend transactions.

2.4 The q-strong Diffie-Hellman assumption (SDH)

The strong Diffie-Hellman (q-SDH) problem was first defined by Boneh and Boyen [8]. Adapted to our type of groups with pairings it says given $g, g^x, \ldots, g^{x^q}, g_2, g_2^x$ for uniformly random $x \leftarrow \mathbb{Z}_p$ it should be hard to find c, z such that $z = g^{\frac{1}{x+c}}$. More precisely, let $\mathcal{A}$ be an attacker, then we define its advantage against the SDH assumption over our setup to be

$$\mathbf{Adv}_{q,\mathcal{A}}^{\text{SDH}} = \Pr[x \leftarrow \mathbb{Z}_p; (c, z) \leftarrow \mathcal{A}(g, g^x, \ldots, g^{x^q}, g_2, g_2^x) : (c, z) \in \mathbb{Z}_p \times \mathbb{G} \text{ and } z^{x+c} = g].$$

We will assume that all realistic adversaries $\mathcal{A}$ humanity may construct in the immediate future will have miniscule advantage when using our choice of groups and pairing and $q = k_{\mathsf{max}}$.

Please note that the q-SDH assumption implies the hardness of other computational tasks such as the discrete logarithm problem for instance.

2.5 Random oracle model

We will model our choice of hash function as a random oracle [9]. Intuitively this means we believe the hash function mangles the input so much that the output looks random. Our protocols will use the hash function as a subroutine, i.e., algorithms are of the form $A^{\mathcal{O}(\cdot)}$. In our security modelling, we replace the hash function with a random oracle, i.e., instead of using $\mathcal{O}(x) = H(x)$ we define $\mathcal{O}(x)$ to return a random $y \leftarrow \{0,1\}^{\ell}$ when x has not been input before, and otherwise return the same y as was used on the previous oracle call on x. Strengthening the random oracle model, we may also consider the programmable random oracle model, where an additional oracle $\mathcal{O}'$ is provided that on input x ...

2.6 Sigma-protocols

2.7 Non-interactive zero-knowledge proofs

$R = \{(w, x, crs)\} \subseteq \{0,1\}^{\star} \times \{0,1\}^{\star} \times \{0,1\}^{\star}\}$ is a binary relation between a witness string w an input string x and a common reference string crs. An interactive zero knowledge protocol describes a system where a prover, P, must convince a verifier V that, for a given input string x, there exists a witness w such that $(w, x, crs) \in R$. If the zero knowledge protocol is **complete**, V will always validate an input string x, created by P interacting with w, where $(w, x, crs) \in R$. The following uses definitions from [10].

A zero-knowledge protocol is **sound** if there exists a probabilistic polynomial time algorithm E that can extract the witness string w from a cheating prover $P^{\star}$ if given oracle access to $P^{\star}$ (i.e. if $P^{\star}$ can convince V with a common input, E can obtain proofs from $P^{\star}$ for any random challenge). If $P^{\star}$ can satisfy $V(x)$ with probability ϵ, E is able to extract a witness statement w that satisfies R with in $poly(\epsilon)$. i.e. whenever $P^{\star}$ can satisfy $V(x)$, E can extract a valid witness string such that $(w, x, crs) \in R$ with overwhelming probability. Therefore ϵ is negligible for a 'fake' proof where $(w, x, crs) \notin R$.

A proof system is **honest-verifier zero-knowledge** if there exists a probabilistic polynomial-time algorithm S that, for any $(w, x, crs) \in R$, identically recreates the outputs of V.

In the AZTEC protocol, the common reference string is generated through the AZTEC commitment scheme's commitment key generation phase (see section 2.3).

2.8 Sigma Protocols

For a proof relation $R = \{w, x, crs)\} \subseteq \{0,1\}^{\star} \times \{0,1\}^{\star} \times \{0,1\}^{\star}$, a protocol, $\mathcal{P}$ is considered to be a Σ protocol [11] if the following are true:

$\mathcal{P}$ is a three-step protocol between two polynomial-algorithms P, V. Both algorithms have input string x as a common input. The witness to R, w, is a private input to P. $\mathcal{P}$ must conform to the following three-step process:

- P sends V a message a
- V sends P a random challenge $c \overset{\$}{\leftarrow} \mathbb{Z}_p$
- P sends V a response z. V accepts or rejects depending on the variables x, a, c, z

$\mathcal{P}$ must be complete. If P, V follow $\mathcal{P}$ with inputs that satisfy the proof relation, V must always accept.

$\mathcal{P}$ must have the **special soundness** property. If, for any input string x, common reference string crs and a pair of valid protocol parameters $(a, c, z), (a', c', z')$, $\mathcal{P}$ has special soundness if there exists an efficiently-computable algorithm to extract w such that $(w, x, crs) \in R$.

$\mathcal{P}$ is special honest-verifier zero knowledge. There exist a polynomial-time algorithm M that, for input string x and random value $e \leftarrow \mathbb{Z}_p$, outputs valid protocol parameters (a, e, z) with the same probability distribution as honest algorithms P, V acting on input string x.

2.9 The Fiat-Shamir Transform

The protocols in this paper utilize the Fiat-Shamir transform [12], which makes an interactive protocol non-interactive through the use of a hash function H. Instead of c being provided by the verifier, c is the output of a hash function, whose input is the set of the Σ protocol's commitments. The challenge, c, is treated as a pseudo-random number whose entropy source is the input string of proof. The Fiat-Shamir transform is secure in the random oracle model [9].

3 The AZTEC Commitment Scheme

The AZTEC system specifies a non-interactive commitment scheme. This enables a sender to commit to a secret value $k \in \{0, \ldots, k_{\mathsf{max}}\}$ and send the commitment to a receiver. Later, the sender may open the commitment by revealing the value and randomness used in creating the commitment, which the receiver can then check. The commitment scheme is *hiding* such that the commitment itself reveals nothing about the committed value, and *binding* such that once a commitment has been made the prover cannot open it to different values. The commitment scheme utilizes a set membership proof similar to that described by Camenisch et al [13] and Arfaoui et al [14].

The AZTEC commitment function $\mathsf{com} : [1; k_{\mathsf{max}}] \times \mathbb{Z}_p^* \rightarrow \mathbb{G} \times \mathbb{G}$ is defined from the setup as

$$\mathsf{com}(k; a) := (\mu_k^a, \mu_k^{ka} h^a).$$

When a sender desires to commit to a value $k \in [1; k_{\mathsf{max}}]$ she picks at random $a \leftarrow \mathbb{Z}_p$ and computes a commitment $(\gamma, \sigma) = \mathsf{com}(k; a)$.

Lemma 1 *The commitment scheme* com *is perfectly hiding.*

Proof. Consider a commitment to a value $k \in [1; k_{\mathsf{max}}]$. It is of the form

$$(\gamma, \sigma) = (\mu_k^a, \mu_k^{ka} h^a) = (h^{\frac{a}{y-k}}, h^{\frac{ka}{y-k}} h^a) = (h^{\frac{a}{y-k}}, h^{\frac{ka+ya-ka}{y-k}}) = (h^{\frac{a}{y-k}}, (h^{\frac{a}{y-k}})^y).$$

Since a is chosen uniformly at random from $\mathbb{Z}_p^*$ we have $h^{\frac{a}{y-k}}$ is uniformly random in $\mathbb{G} \setminus \{1\}$ and hence reveals nothing about k. □

Lemma 2 *The commitment scheme* com *is binding if the k_{max}-SDH problem is hard.*

Proof. Suppose an attacker finds two distinct openings to the same commitment, i.e., $(\gamma, \sigma) = (\mu_k^a, \mu_k^{ka} h^a) = (\mu_{k'}^{a'}, \mu_{k'}^{k'a'} h^{a'})$ with $k, k' \in [1; k_{\mathsf{max}}]$ and $a, a' \in \mathbb{Z}_p^*$. Since $k \neq k'$ and $a, a' \neq 0$ we see from γ that $a \neq a'$. Looking closer at the first component of the commitment we then have

$$\mu_k^a = \mu_{k'}^{a'} \Rightarrow h^{\frac{a}{y-k}} = h^{\frac{a'}{y-k'}} \Rightarrow \frac{a}{y-k} = \frac{a'}{y-k'} \Rightarrow y = \frac{ak' - a'k}{a - a'} \mod p.$$

The ability to compute y means such an attacker can break the q-SDH assumption with $q = k_{\mathsf{max}}$. □

The AZTEC commitment scheme is efficient to integrate into our protocols because it has an integrated range proof with constant verification time. To explain this, we first observe that for a correctly computed commitment

$$e(\gamma, t_2) = e(\sigma, g_2).$$

This follows from

$$e(\mu_k^a, t_2) = e(h^{\frac{a}{y-k}}, g_2^y) = e(h^{\frac{ay}{y-k}}, g_2) = e(h^{\frac{ak}{y-k}} h^a, g_2) = e(\mu_k^{ka} h^a, g_2).$$

Second, we observe that in a valid commitment

$$\sigma = \gamma^k h^a.$$

From these two properties we get a guarantee that $k \in [0; k_{\mathsf{max}}]$. Namely, suppose an attacker produces $k \in \mathbb{Z}_p$ and $a \in \mathbb{Z}_p$ such that $\gamma \neq 1$ and $\sigma = \gamma^k h^a$ then we are guaranteed the restriction $k \in [1; k_{\mathsf{max}}]$.

Lemma 3 *An attacker that can produce $\gamma \in \mathbb{G} \setminus 1$ and $a, k \in \mathbb{Z}_p$ such that $e(\gamma, t_2) = e(\gamma^k h^a, g_2)$ with $k \notin [0; k_{\mathsf{max}}]$ could be used to break the k_{max}-SDH assumption.*

Proof. The pairing equation $e(\gamma, t_2) = e(\gamma^k h^a, g_2)$ shows us $e(\gamma^{y-k} h^a, g_2) = 1$ and therefore $\gamma = h^{\frac{a}{y-k}}$. Since $\gamma \neq 1$ we know $a \neq 1$ so this allows the computation of $\gamma^{\frac{1}{a}} = h^{\frac{1}{y-k}}$. If $k \in [0; k_{\mathsf{max}}]$ this just means $\gamma^{\frac{1}{y-k}} = \mu_k$, however, if $k \notin [0; k_{\mathsf{max}}]$ we now have a weak Boneh-Boyen signature on $-k$ not previously seen. The weak Boneh-Boyen signature scheme is existentially unforgeable under selective chosen message attack assuming the k_{max}-SDH assumption holds though, so such an attacker is unlikely to succeed. □

3.1 Choice of k_{max} for the AZTEC commitment function

Given (γ, σ, a) it should be possible to recover the message k of a commitment. Three factors influence the choice of k_{max}

- The time taken to recover g^k through a brute-force algorithm
- The security of the $k_{max} - sdh$ problem
- The size of commitment key ck

Qualitative tests indicate that a value of $k_{max} \approx 2^{32}$ is an approximate upper threshold given these constraints. The reference implementation of the AZTEC protocol defines $k_{max} = 2^{25} - 1$ to enable the fast recovery of k given g^k.

4 The AZTEC protocol zero-knowledge proofs

In the AZTEC protocol, users will have commitments to input values and commitments to output values that need to balance against each other. Since the commitments are perfectly hiding this cannot be directly guaranteed, however, instead users may produce proofs of knowledge of openings of the commitments that balance out. The AZTEC protocol therefore includes non-interactive zero-knowledge (NIZK) proofs that can be used to demonstrate two sets of commitments have known openings that sum to the same total. We describe these proofs after giving a quick background on zero-knowledge proofs.

4.1 Sigma-protocols and the Fiat-Shamir heuristic

Proof systems allow two parties, called the prover and the verifier, to interact in a way that convinces the verifier a particular statement is true. Let R be a binary relation with an efficient algorithm that can determine whether $(x, w) \in R$. We will call x an instance and when $(x, w) \in R$ we call w a witness to x. In our proof system, statements will be defined by the relation R, which is defined by our setup, and instances x, and be the claim that the prover knows a witness w such that $(x, w) \in R$.

A Sigma-protocol is a particular type of proof system where the prover sends an initial commitment message to the verifier, gets back a random challenge from the verifier, and then sends a final response. Afterwards the verifier checks the instance x and the transcript of commitment, challenge and response and decides whether to accept or reject. We are going to use a special type of Sigma-protocols that can be described by four efficient algorithms $\mathcal{P}, \mathcal{U}, \mathcal{E}, \mathcal{S}$ and where the verifier chooses challenges from $\{0,1\}^\ell$.

$\mathcal{P}(x, w)$: The prover algorithm is stateful and works in two phases. Assuming the input satisfies $(x, w) \in R$ it first generates an initial commitment message B. Then after receiving a challenge $c \in \{0,1\}^\ell$ it computes a response z.

$\mathcal{U}(x, c, z) \to B$: The deterministic unique commitment reconstruction algorithm on instance x, challenge c and answer z, returns an initial commitment B. If the input is malformed, it may instead return an error symbol $\perp$.

The algorithm implicitly defines a verification procedure of a transcript (B, c, z) that accepts if and only if $B = \mathcal{U}(x, c, z) \neq \perp$. We require (perfect) completeness, which means for all $(x, w) \in R$ and all $c \in \{0,1\}^\ell$ the outputs B and z of $\mathcal{P}(x, w)$ will satisfy $B = \mathcal{U}(x, c, z) \neq \perp$.

$\mathcal{E}(x, c, z, c', z') \to w$: The deterministic extraction algorithm returns a string w. We require (perfect) special soundness, which means whenever $\mathcal{U}(x, c, z) = \mathcal{U}(z, c', z') \neq \perp$ then the output w will be a witness such that $(x, w) \in R$.

$\mathcal{S}(x, c) \to z$: The simulation algorithm returns a simulated answer z. We require (perfect) special honest verifier zero-knowledge, which means for all $(x, w) \in R$ and $c \in \{0,1\}^\ell$ the output has the same probability distribution as a response z obtained from running $\mathcal{P}(x, w)$ on challenge c.

In order to evaluate homomorphic sums of input and output commitment messages, it is necessary to prove the existence of witnesses that open each commitment. Specifically, for each commitment tuple $(\gamma_{in_i}, \sigma_{in_i}) \forall i \in [1, n_{in}]$ there must exist witnesses $k_i, a_i \in \mathbb{Z}_p$ such that $\gamma_{in_i}^{k_i} h^{a_i} = \sigma_{in_i}$.

Similarly, for each commitment tuple $(\gamma_{out_j}, \sigma_{out_j}) \forall j \in [1, n_{out}]$ there must exist witnesses $k_j, a_j \in \mathbb{Z}_p$ such that $\gamma_{out_j}^{k_j} h^{a_j} = \sigma_{out_j}$.

Once the existence of these witnesses has been proven, the following relationship must hold: $\sum_{i=1}^{n_{in}} k_i - \sum_{j=1}^{n_{out}} k_j = 0$.

Finally, it is necessary to validate that the witness k for every **output** commitments is within the range $[1, k_{max}]$. As long as $k_{max} << p$ this prevents the creation of commitments where $k > \frac{p}{2}$. This is necessary to prevent a malicious prover from using modular arithmetic to construct commitments of arbitrary value.

It is only necessary to prove this for output commitments; it is assumed that input commitments must themselves have been output commitments of a previous joinsplit transaction. We can inductively assume that every input note is within the range $[1, k_{max}]$. Specifically, the following relationship must be valid:

$$\forall j \in [1, n_{out}] e(\gamma_j, t_2) = e(\sigma_j, g_2)$$

5 The Joinsplit Protocol

We assume that the trusted setup algorithm $\mathcal{T}$ has been executed correctly, producing a commitment key ck defined over groups $\mathbb{G}, \mathbb{G}_2$ with pairing function e and hash function $H : \{0,1\}^* \to \{0,1\}^\ell$, where $\ell < |p|$. The commitment key ck contains group elements $g, h, \mu_1, \ldots, \mu_{k_max} \in \mathbb{G}$ and $g_2, t_2 \in \mathbb{G}_2$ that satisfy the relationship $k = 1, \ldots, k_{\max} : e(\mu_k, t_2) = e(\mu_k^k h, g_2)$.

We define the relationship $\mathcal{R}_{\mathsf{balance}}$ where

$$\mathcal{R}_{\mathsf{balance}} = \left\{ \begin{array}{c} (x, w) = (((\gamma_i, \sigma_i)_{i=1}^n, m, k_{public}), (k_i, a_i)_{i=1}^n) \mid \\ \text{For all } i \in \{1, \ldots, n\} : \gamma_i, \sigma_i \in \mathbb{G}, k_i, a_i \in \mathbb{Z}_p, \sigma_i = \gamma_i^{k_i} h^{a_i} \\ \text{and } 0 \leq m \leq n \text{ and } n > 0 \text{ and } \sum_{i=1}^m k_i = \sum_{i=m+1}^n k_i + k_{public} \pmod{p} \end{array} \right\}$$

The scalar $k_{public} \in \mathbb{Z}_p$ represents a public representation of 'value', for situations where a public value is converted into a confidential AZTEC commitment. We describe a proof system for $\mathcal{R}_{\mathsf{balance}}$ through a Sigma-protocol defined by four algorithms $(\mathcal{P}, \mathcal{U}, \mathcal{E}, \mathcal{S})$ for respectively proofs, reconstruction of unique initial message, extraction of witness and simulation [1]. We use these algorithms to prove that the protocol has both special soundness and special honest-verifier zero knowledge.

5.0.1 $\mathcal{P}_{\mathsf{balance}}$ and $\mathcal{V}_{\mathsf{balance}}$

Figure 2 describes the protocols used to construct and verify proofs for $\mathcal{R}_{\mathsf{balance}}$.

[1] See Matilda Backendal, Mihir Bellare, Jessica Sorrell, Jiahao Sun: The Fiat-Shamir Zoo: Relating the Security of Different Signature Variants. IACR Cryptology ePrint Archive 2018: 775 (2018) for related ideas.

```
$\mathcal{P}_{\mathsf{balance}}(((\gamma_i,\sigma_i)_{i=1}^n, m, k_{public}), (k_i, a_i)_{i=1}^n)$:
    Validate $(((\gamma_i,\sigma_i)_{i=1}^n, m), (k_i, a_i)_{i=1}^n) \in \mathcal{R}_{\mathsf{balance}}$
    Pick $b_{a_1}, b_{k_2}, b_{a_2}, \ldots, b_{k_n}, b_{a_n} \leftarrow \mathbb{Z}_p$
    Set $b_{k_1} = \sum_{i=m+1}^n b_{k_i} - \sum_{i=2}^m b_{k_i}$
    First output of $\mathcal{P}$ is
        $B_1 = \gamma_1^{b_{k_1}} h^{b_{a_1}}, \ldots, B_n = \gamma_n^{b_{k_n}} h^{b_{a_n}}$
    Compute the challenge $c = H(((\gamma_i,\sigma_i)_{i=1}^n, m), (B_i)_{i=1}^n)$
    Second output of $\mathcal{P}$ is
        $\bar{k}_1 = ck_1 + b_{k_1}, \bar{a}_1 = ca_1 + b_{a_1}, \ldots, \bar{k}_n = ck_n + b_{k_n}, \bar{a}_n = ca_n + b_{a_n} \pmod{p}$
    Return $\pi = (c, \bar{a}_1, \bar{a}_2, \bar{k}_2, \ldots, \bar{a}_n, \bar{k}_n)$

$\mathcal{V}_{\mathsf{balance}}(((\gamma_i,\sigma_i)_{i=1}^n, m, k_{public}), \pi)$:
    Validate $0 \le m \le n$ and for $i \in \{1, \ldots, n\} : \gamma_i, B_i \in \mathbb{G}$
    Parse $\pi = (\bar{a}_1, \bar{a}_2, \bar{k}_2, \ldots, \bar{a}_n, \bar{k}_n) \in \mathbb{Z}_p^{2n}$
    If $m = 0$ Set $\bar{k}_1 = -\sum_{i=2}^n \bar{k}_i - k_{public}c$
    If $m > 0$ Set $\bar{k}_1 = \sum_{i=m+1}^n \bar{k}_i - \sum_{i=2}^m \bar{k}_i + k_{public}c$
    If all checks pass the output of $\mathcal{U}$ is
        $B_1 = \gamma_1^{\bar{k}_1} h^{\bar{a}_1} \sigma_1^{-c}, \ldots, B_n = \gamma_1^{\bar{k}_n} h^{\bar{a}_n} \sigma_n^{-c}$
    Return 1 if $c = H(((\gamma_i,\sigma_i)_{i=1}^n, m), (B_i)_{i=1}^n)$ else return 0
```

Figure 2: Algorithms $\mathcal{P}_{\mathsf{balance}}$, $\mathcal{V}_{\mathsf{balance}}$

We define $\mathcal{E}(((\gamma_i,\sigma_i)_{i=1}^n, m, k_{public}), c, (\bar{k}_i, \bar{a}_i)_{i=1}^n, c', (\bar{k}'_i, \bar{a}'_i)_{i=1}^n) = (k_i, a_i)_{i=1}^n$, where

$$k_i = \frac{\bar{k}'_i - \bar{k}_i}{c' - c} \text{ and } a_i = \frac{\bar{a}'_i - \bar{a}_i}{c' - c} \pmod{p}.$$

We define $\mathcal{S}(((\gamma_i,\sigma_i)_{i=1}^n, m, k_{public}), c) = (\bar{k}_i, \bar{a}_i)_{i=1}^n$ using uniformly random $\bar{k}_i, \bar{a}_i \leftarrow \mathbb{Z}_p$ under the condition $\sum_{i=1}^m \bar{k}_i = \sum_{i=m+1}^n \bar{k}_i + k_{public}$.

Lemma 4 *The core protocols ($\mathcal{P}$,$\mathcal{U}$,$\mathcal{E}$,$\mathcal{S}$) describe the core of a proof system for $\mathcal{R}_{\mathsf{balance}}$ with perfect completeness, perfect special soundness and perfect honest verifier zero-knowledge.*

5.0.2 Proof of perfect completeness

By definition, if the proof system has perfect completeness then for an input $(x, w) = (((\gamma_i,\sigma_i)_{i=1}^n, m), (k_i, a_i)_{i=1}^n) \in \mathcal{R}_{\mathsf{balance}}$ and $c \in \{0,1\}^\ell$ the protocol $\mathcal{P}_{\mathsf{balance}}$ will return 1.

Algorithm $\mathcal{P}$ always produces initial message $(B_1, \ldots, B_n)$, challenge c and challenge response $\bar{a}_1, \bar{a}_2, \bar{k}_2, \ldots, \bar{a}_n, \bar{k}_n$, such that $(B_1, \ldots, B_n) = \mathcal{U}(((\gamma_i, \sigma_i)_{i=1}^n, m), c, (\bar{a}_1, \bar{a}_2, \bar{k}_2, \ldots, \bar{a}_n, \bar{k}_n))$. This follows from the fact for a valid statement and challenge, all checks by $\mathcal{P}$ and $\mathcal{U}$ pass. From this, for all i we have

$$B_i = \gamma_i^{\bar{k}_i} h^{\bar{a}_i} \sigma_i^{-c} = \gamma_i^{ck_i + b_{k_i}} h^{ca_i + b_{a_i}} (\gamma_i^{k_i} h^{a_i})^{-c} = \gamma_i^{b_{k_i}} h^{b_{a_i}}.$$

5.0.3 Proof of special soundness

By definition, perfect special soundness means that for two proof transcripts $(\bar{a}_1, \bar{a}_2, \bar{k}_2, \ldots, \bar{a}_n, \bar{k}_n)$ and $(\bar{a}'_1, \bar{a}'_2, \bar{k}'_2, \ldots, \bar{a}'_n, \bar{k}'_n)$ with distinct challenges $c \neq c' \in \{0,1\}^\ell$ where $\mathcal{U}$ returns the same initial message $(B_i)_{i=1}^n$ it is possible to extract a valid witness. From the verification equations we have for each i that

$$B_i = \gamma_i^{\bar{k}_i} h^{\bar{a}_i} \sigma_i^{-c} = \gamma_i^{\bar{k}'_i} h^{\bar{a}'_i} \sigma_i^{-c'}.$$

Where $\bar{k}_1 = \sum_{i=m+1}^n \bar{k}_i - \sum_{i=2}^m \bar{k}_i$ and $\bar{k}'_1 = \sum_{i=m+1}^n \bar{k}'_i - \sum_{i=2}^m \bar{k}'_i$. This implies that

$$\sigma_i^{c'-c} = \gamma_i^{\bar{k}'_i - \bar{k}_i} h_i^{\bar{a}'_i - \bar{a}_i} \implies \sigma_i = \gamma_i^{\frac{\bar{k}'_i - \bar{k}_i}{c'-c}} h_i^{\frac{\bar{a}'_i - \bar{a}_i}{c'-c}}$$

From this, we have $\sigma_i = \gamma_i^{\frac{\bar{k}'_i - \bar{k}_i}{c'-c}} h_i^{\frac{\bar{a}'_i - \bar{a}_i}{c'-c}} = \gamma_i^{k_i} h^{a_i}$ where k_i, a_i match the output of extractor $\mathcal{E}$.

Finally, from $\bar{k}_1 = \sum_{i=m+1}^n \bar{k}_i - \sum_{j=1}^m \bar{k}_j + k_{public} c$ and $\bar{k}'_1 = \sum_{i=m+1}^n \bar{k}'_i - \sum_{j=1}^m \bar{k}'_j + k_{public} c'$ we obtain the following relationship

$$\frac{\bar{k}'_1 - \bar{k}_1}{c' - c} = \sum_{i=m+1}^{n} \frac{\bar{k}'_i - \bar{k}_i}{c' - c} - \sum_{i=2}^{m} \frac{\bar{k}'_i - \bar{k}_i}{c' - c} + k_{public} = \sum_{i=m+1}^{n} k_i - \sum_{i=2}^{m} k_i + k_{public} = k_1$$

5.0.4 Proof of special honest verifier zero-knowledge

By definition, perfect honest verifier zero-knowledge means that for a valid statement $(x, w) \in \mathcal{R}_{\mathsf{balance}}$ and challenge $c \in \{0,1\}^\ell$ a valid proof transcript can be perfectly simulated. This can be achieved by sampling $(\bar{a}_i, \bar{k}_i)_{i=1}^n$ from $\mathbb{Z}_p$ at random, conditional on $\sum_{i=1}^m \bar{k}_i = \sum_{i=m+1}^n \bar{k}_i$. A valid proof transcript can be constructed by sampling $(b_{k_i}, b_{a_i})_{i=1}^n$ from $\mathbb{Z}_p$ at random, conditional on $\sum_{i=1}^m b_{k_i} = \sum_{i=m+1}^n b_{k_i}$, and calculating values $(k_i, a_i)_{i=1}^n$ where

$$k_i = \frac{\bar{k}_i - b_{k_i}}{c} \quad a_i = \frac{\bar{a}_i - b_{a_i}}{c}.$$

Input string $(\gamma_i, \sigma_i)_{i=1}^n$ is then constructed, where $\gamma_i = \mu_{k_i}^{a_i} \quad \sigma_i = (\mu_{k_i} h)^{a_i}$. Finally, a valid proof transcript can be constructed by calculating initial message $(B_i)_{i=1}^n$, where $B_i = \gamma_i^{\bar{k}_i} h^{\bar{a}_i} \sigma_i^{-c}$.

6 The AZTEC joinsplit transactions

The motivation for the AZTEC proofs is to enable a system of 'join-split' confidential transactions, involving AZTEC 'notes'. A note is defined by the following:

- An AZTEC commitment tuple γ, σ, where witness k defines the 'value' of note and witness a defines the note's 'viewing key'
- A public key that defines the note 'owner'. The corresponding private key is defined as the note's 'spending key'

In the reference implementation of the AZTEC protocol, a note 'owner' is defined via an Ethereum address. For the remainder of this section, the reference implementation of the AZTEC protocol is used to describe 'join-split' transactions, however it should be noted that some of these methods are implementation specific, for example the specific signature scheme used to define note 'ownership'.

'Join-split' transactions require a **note registry** - persistent state that records the existence of all un-spent AZTEC notes. Different digital assets utilizing the AZTEC protocol each posess a unique note registry.

A 'join-split' transaction maps to taking an input $((\gamma_i, \sigma_i, P_i)_{i=1}^{n}, m, k_{public}, (Sig_i)_{i=1}^{m}, (P_i)_{i=1}^{n})$, where $(\gamma, \sigma)_{i=1}^{m}$ describe a vector of **input commitments** and $(\gamma, \sigma)_{i=m+1}^{n}$ describe a vector of **output commitments**. $(P_i)_{i=1}^{n}$ describes the set of 'owner' public keys that map to each commtiment. $(Sig_i)_{i=1}^{m+1}$ describes the set of signatures that map to each input commitment.

The reference implementation of the AZTEC protocol uses the Ethereum protocol's native signature scheme; ECDSA signatures [15] defined over the *secp256k1* curve [16]. This enables normal Ethereum private keys to function as AZTEC note spending keys. The message being signed by each note owner contains the following:

- The note's commitment γ, σ
- The challenge variable c generated by $\mathcal{P}_{\mathsf{balance}}$ for this join-split transaction

The challenge variable is unique to a single zero-knowledge proof, and a signature against this challenge is considered an explicit approval by a note owner of the outcome of the 'join-split' transaction.

A valid 'join-split' transaction proves the following properties about these commitments:

- $((\gamma, \sigma)_{i=1}^{n}, m, k_{public})$ is a valid input string to the proof system $\mathcal{R}_{\mathsf{balance}}$
- $(\gamma, \sigma)_{i=m+1}^{n}$ are valid outputs of the AZTEC commitment function
- Commitments $(\gamma, \sigma)_{i=1}^{m}$ exist as entries in the **note registry**, whose owners are described by $(P_i)_{i=1}^{m}$
- Commitments $(\gamma, \sigma)_{i=m+1}^{n}$ do *not* exist as entries in the **note registry**
- Every $(Sig_i)_{i=1}^{m}$ is a valid ECDSA signature signed by public key $(P_i)_{i=1}^{m}$
- Every $(P_i)_{i=1}^{m}$ maps to a valid ethereum address

- The value that maps to k_{public} is authorized as a legitimate input to the join-split transaction

Defining the nature of k_{public}

The variable k_{public} in $\mathcal{R}_{\mathsf{balance}}$ enables a public 'value' to be either converted into or out of AZTEC note form. This is to support value either being added or removed from the confidential AZTEC note form and converted from/to a public analogue. The reference implementation of the AZTEC protocol enables AZTEC note registries to be attached to existing ERC-20 tokens. Here, k_{public} maps to token balances controlled by the ethereum account that issues the 'join-split transaction'.

The AZTEC protocol can be used to define a digital asset that has no public representation. For a fully confidential asset, the first 'join-split' transaction is the *origination* transaction, which in turn defines the initial note registry. In this situation, k_{public} is equal to the total size of the note registry. For all future 'join-split' transactions for a fully confidential asset, $k_{public} = 0$.

6.1 Join-split transactions

Figure 3 describes the full protocol to construct and validate join-split transactions, using the proof system for $\mathcal{R}_{\mathsf{balance}}$. The final addition is the validate the commitments $(\gamma_i, \sigma_i)_{i=m+1}^{n}$ are valid outputs of the AZTEC commitment function. It is not neccessary to validate commitments $(\gamma_i, \sigma_i)_{i=1}^{m}$ are valid outputs of the AZTEC commitment function as it is assumed that all inputs to join-split transactions were outputs of a previous join-split transaction.

One other modification is that the ethereum address of the transaction sender is included in the input string of $\mathcal{P}_{\mathsf{balance}}$, and is added as a input when generating the challenge variable c, for $\mathcal{R}_{\mathsf{balance}}, \mathcal{P}_{\mathsf{balance}}$. This is done to prevent 'front-running', where an attacker takes a valid proof from a yet-to-be-mined transaction, and integrates it into a transaction issued by the attacker. This addition makes zero-knowledge proofs specific to a given transaction sender.

The validation logic for a 'join-split' transaction is represented by a deterministic algorithm embedded into a blockchain protocol. At the conclusion of a valid 'join-split' transaction, all input notes are removed from the note registry. Similarly, all output notes are added into the note registry.

```
$\mathcal{P}_{\mathsf{joinsplit}}\ (((\gamma_i, \sigma_i, k_i, a_i)_{i=1}^{m}, m, k_{public}), (k_i)_{i=m+1}^{n})$:
    Validate for $i = 1$ to $m$ that $\gamma_i \in \mathbb{G}, k_i, a_i \in \mathbb{Z}_p$ and $\sigma_i = \gamma_i^{k_i} h^{a_i}$ and $k_{public} \in \mathbb{Z}_p$
    Validate for $i = m + 1$ to $n$ that $k_i \in \{1, \ldots, k_{\max}\}$
    For $i = m{+}1$ to $n$ pick $a_i \leftarrow \mathbb{Z}_p$ and compute commitments $(\gamma_i, \sigma_i) = (\mu_{k_i}^{a_i}, (\mu_{k_i}^{k_i} h)^{a_i})$
    Compute $\pi \leftarrow \mathcal{P}_{\mathsf{balance}}(((\gamma_i, \sigma_i)_{i=1}^{n}, m, k_{public}), (k_i, a_i)_{i=1}^{n})$
    Return $(\gamma_{m+1}, \sigma_{m+1}, \ldots, \gamma_n, \sigma_n, \pi)$
$\mathcal{V}_{\mathsf{joinsplit}}\ ((\gamma_i, \sigma_i)_{i=1}^{n}, m, k_{public}, \pi)$:
    Validate $1 \leq m < n$ and for $i = 1$ to $n$ that $\gamma_i, \sigma_i \in \mathbb{G}$
    For $i = m + 1$ to $n$ verify $e(\gamma_i, t_2) = e(\sigma_i, g_2)$ and reject if either fails
    Accept if $\mathcal{V}_{\mathsf{balance}}(((\gamma_i, \sigma_i)_{i=1}^{n}, m), \pi) = 1$ and else reject
```

Figure 3: Join-split protocol

7 Optimizing batched AZTEC commitments

The AZTEC protocol enables transactions that involve multiple AZTEC commitments. In the naive protocol, a bilinear pairing comparison must be performed for each output commitment. It is possible to validate multiple AZTEC commitments using a single bilinear pairing comparison with a small modification to the verification equations, at the expense of one additional scalar multiplication per output note. This modification is introduced as a result of the way elliptic curve arithmetic is implemented in the Ethereum protocol [4]. The protocol introduces fixed costs for scalar multiplications and bilinear pairings, which do not take into account efficiencies in calculating multi-exponentiations via Shamir's trick [17]. Bilinear pairing comparisons are also significantly more expensive than scalar multiplications. A more detailed analysis of the Ethereum protocol's capabilities and costs is described in section 8.

We want to be able to validate that, for every tuple (γ, σ), $\gamma^y = \sigma$. Instead of validating $e(\gamma_i, t_2) \overset{?}{=} e(\sigma_i, g_2) \forall i \in [1, n_{out}]$, we can condense the validation into a single bilinear pairing comparison of the form $e(\gamma_1^{x_1} \cdot ... \cdot gamma_{n_{out}}^{x_n}, t_2) \overset{?}{=} e(\sigma_1^{x_1} \cdot ... \cdot \sigma_{n_{out}}^{x_n}, g_2)$, where $x_1, ..., x_n$ are random variables.

i.e. if every term were exponentiated by an independent random variable, it would be possible to sum together these exponentiated terms and perform a single bilinear pairing comparison whilst treating every term as an isolated pairing comparison. We can further reduce the amount of work required by setting $x_1 = 1$ and reducing the number of scalar multiplications required.

We introduce an intermediate challenge x, where $x = h(\boldsymbol{\gamma}_{in}, \boldsymbol{\sigma}_{in}, \boldsymbol{\gamma}_{out}, \boldsymbol{\sigma}_{out})$.

The intermediate challenge is used to create a linear combination of γ and σ terms. The modified protocols $\mathcal{P}_{\mathsf{balance}'}, \mathcal{V}_{\mathsf{balance}'}, \mathcal{P}_{joinsplit'}, \mathcal{V}_{joinsplit'}$ are described in figures 4 and 5 respectively.

To summarise, the pairing optimization removes a pairing calculation per output commitment (excluding the first output commitment), at the expense of one extra scalar multiplication.

7.0.1 Security of the pairing optimization

Every term in $\gamma_{m+1}\Pi_{i=m+2}^{n}\gamma_i^{cx^i}$ can be treated as an independent variable. If the pairing comparison is satisfied then it must also be satisfied for each individual term.

Consider the point $\gamma_{m+1}\Pi_{i=m+2}^{n}\gamma_i^{cx^i}$. Because every element of $\mathbb{G}$ is a generator of the group, each term can be represented as a linear combination of the remaining terms and a set of transformation scalars $\{q_i \forall i \in [1, n]\}$:

$$\gamma_i^{cx^{i-1}} = \sum_{j=1|j \neq i}^{n} \gamma_j^{cx^{j-1}q_j} \tag{1}$$

This can be rearranged to:

```
$\mathcal{P}_{\mathsf{balance}}(((\gamma_i, \sigma_i)_{i=1}^n, m, k_{public}), (k_i, a_i)_{i=1}^n)$:
    Validate $(((\gamma_i, \sigma_i)_{i=1}^n, m), (k_i, a_i)_{i=1}^n) \in \mathcal{R}_{\mathsf{balance}}$
    Set $x = H(\gamma_i, \sigma_i)_{i=m+1}^n$ Pick $b_{a_1}, b_{k_2}, b_{a_2}, \ldots, b_{k_n}, b_{a_n} \leftarrow \mathbb{Z}_p$
    Set $b_{k_1} = \sum_{i=m+1}^n b_{k_i} - \sum_{i=2}^m b_{k_i}$
    Set $(b_{k'_i}, b_{a'_i})_{i=1}^n = (x^i b_{k_i}, x^i b_{a_i})_{i=m+1}^n$
    First output of $\mathcal{P}$ is
        $B_1 = \gamma_1^{b_{k'_1}} h^{b_{a'_1}}, \ldots, B_n = \gamma_n^{b_{k'_n}} h^{b_{a'_n}}$
    Compute the challenge $c = H(((\gamma_i, \sigma_i)_{i=1}^n, m), (B_i)_{i=1}^n)$
    Second output of $\mathcal{P}$ is
        $\bar{k}_1 = ck_1 + b_{k_1}, \bar{a}_1 = ca_1 + b_{a_1}, \ldots, \bar{k}_n = ck_n + b_{k_n}, \bar{a}_n = ca_n + b_{a_n} \pmod{p}$
    Return $\pi = (c, \bar{a}_1, \bar{a}_2, \bar{k}_2 \ldots, \bar{a}_n, \bar{k}_n)$

$\mathcal{V}_{\mathsf{balance}}(((\gamma_i, \sigma_i)_{i=1}^n, m), \pi)$:
    Validate $0 \leq m \leq n$ and for $i \in \{1, \ldots, n\} : \gamma_i, B_i \in \mathbb{G}$
    Set $x = H(\gamma_i, \sigma_i)_{i=m+1}^n$ Parse $\pi = (\bar{a}_1, \bar{a}_2, \bar{k}_2, \ldots, \bar{a}_n, \bar{k}_n) \in \mathbb{Z}_p^{2n}$
    If $m = 0$ Set $\bar{k}_1 = -\sum_{i=2}^n \bar{k}_i - k_{public}c$
    If $m > 0$ Set $\bar{k}_1 = \sum_{i=m+1}^n \bar{k}_i - \sum_{i=2}^m \bar{k}_i + k_{public}c$
    If all checks pass the output of $\mathcal{U}$ is
        $B_1 = \gamma_1^{\bar{k}_1 x^i} h^{\bar{a}_1 x^i} \sigma_1^{-cx^i}, \ldots, B_n = \gamma_1^{\bar{k}_n} h^{\bar{a}_n} \sigma_n^{-c}$
    Return 1 if $c = H(((\gamma_i, \sigma_i)_{i=1}^n, m), (B_i)_{i=1}^n)$ else return 0
```

Figure 4: Algorithms $\mathcal{P}_{\mathsf{balance}'}$, $\mathcal{V}_{\mathsf{balance}'}$ with pairing optimization

$$\gamma_i = \left(\sum_{j=1}^{n-1} \gamma_j^{x^{j-1} q_j} \right)^{\frac{1}{x^{i-1}}} \tag{2}$$

Assume a prover, P, can construct the input string to valid zero-knowledge proof by using equation 2. If this is the case, then the terms in equation 2 are not independent and the proof statement for Π_{commit} is not proven. However the rhs of equation 2 contains a factor of x for every element in the sum. As x is a hash of the commitment set, it is not possible to construct the commitment set by using equation 2 and every term can be treated as an independent variable.

```
$\mathcal{P}_{\mathsf{joinsplit'}}\ (((\gamma_i, \sigma_i, k_i, a_i)_{i=1}^m, m, k_{public}), (k_i)_{i=m+1}^n)$:
    Validate for $i = 1$ to $m$ that $\gamma_i \in \mathbb{G}, k_i, a_i \in \mathbb{Z}_p$ and $\sigma_i = \gamma_i^{k_i} h^{a_i}$ and $k_{public} \in \mathbb{Z}_p$
    Validate for $i = m + 1$ to $n$ that $k_i \in \{1, \ldots, k_{\max}\}$
    For $i = m+1$ to $n$ pick $a_i \leftarrow \mathbb{Z}_p$ and compute commitments $(\gamma_i, \sigma_i) = (\mu_{k_i}^{a_i}, (\mu_{k_i}^{k_i} h)^{a_i})$
    Compute $\pi \leftarrow \mathcal{P}_{\mathsf{balance}}(((\gamma_i, \sigma_i)_{i=1}^n, m, k_{public}), (k_i, a_i)_{i=1}^n)$
    Return $(\gamma_{m+1}, \sigma_{m+1}, \ldots, \gamma_n, \sigma_n, \pi)$
$\mathcal{V}_{\mathsf{joinsplit'}}\ ((\gamma_i, \sigma_i)_{i=1}^n, m, k_{public}, \pi)$:
    Validate $1 \leq m < n$ and for $i = 1$ to $n$ that $\gamma_i, \sigma_i \in \mathbb{G}$
    Set $x = H(\gamma_i, \sigma_i)_{i=1}^n$
    Verify $e(\gamma_{m+1} \Pi_{i=m+2}^n \gamma_i^{cx^i}, t_2) = e(\sigma_{m+1} \Pi_{i=m+2}^n \sigma_i^{cx^i}, g_2)$ and reject if either fails
    Accept if $\mathcal{V}_{\mathsf{balance}}(((\gamma_i, \sigma_i)_{i=1}^n, m, k_{public}), \pi) = 1$ and else reject
```

Figure 5: Join-split protocol with pairing optimization

8 Efficiency analysis of the AZTEC protocol

The efficiency of the AZTEC protocol is examined in the context of minimizing *gas costs* required to validate confidential transactions via a smart contract described by the Ethereum protocol.

8.1 Ethereum and Smart Contracts

The Ethereum protocol provides for the ability to embed hexadecimal byte-code into a transaction. The byte-code represents operations performed on a Turing-complete virtual machine, whose parameters are strictly defined [4]. This enables deterministic algorithms to be executed with code, inputs and outputs defined in transactions embedded into blockchain blocks. The outputs of such transactions can be verified and validated by any observer.

Blocks of byte-code, termed **smart contracts**, can be embedded into one of Etherem's Patricia-Merkle trees. Future Ethereum transactions can then execute the byte-code associated with the smart contract, with input parameters defined by the transaction sender. Smart contracts also contain persistent state; data embedded into a separate Patricia-Merkle tree that can be deterministically modified by the smart contract.

The Ethereum Virtual Machine (EVM) provides for limited set of elliptic curve operations and only for the pairing-friendly Barreto-Naehrig curve **bn128**. The implicit cost of an Ethereum transaction (denoted in **gas**) is a function of the computational complexity of the transaction.

Each EVM opcode has an associated gas cost that reflects the amount of computation required to execute the opcode. These are described in the Ethereum Yellow Paper's gas schedule [4]. Because of the computational overheads of simulating a virtual machine when executing smart contracts, the gas cost of elliptic curve computations through EVM opcodes is unfeasibly high. To compensate for this, the Ethereum protocol specifies a set of *precompile* smart contracts. These precompiles are not described by EVM opcodes, but instead will execute a specific algorithm that is implemented in native machine-code. This machine-code implementation is defined by individual clients that implement the Ethereum protocol. The gas cost of these 'precompile' smart contracts is set to reflect the computational cost of executing the native algorithm.

Several precompile contracts exist to perform elliptic curve cryptographic operations on the *bn128* curve. Figure 6 shows the relative gas costs of elliptic curve operations at the time of publication, relative to other EVM operations and the native cost of a transaction. All figures are taken from the Ethereum Yellow Paper [4]. Recent developments have lead to the clients that implement the Ethereum protocol to upgrade the implementations of their precompile smart contracts. This has significantly reduced the computational cost of elliptic curve cryptography executed by these clients and has triggered a planned reduction of the associated gas costs in the near future. These reduced gas costs are described by bracketed terms in figure 6. The updated gas schedule is taken from EIP-1108 [18]

- Addition of two elliptic curve points in $\mathbb{G}$: 500 (100) gas
- Multiplication of an elliptic curve point by a scalar: 40,000 (6,000) gas
- Validating the bilinear pairing $\Pi_{i=0}^{n} e(p_i, q_i) \overset{?}{=} 1$, where $p_i \in \mathbb{G}, q_i \in \mathbb{G}_2$: $100,000(35,430) + 80,000n(28,300n)$ gas
- Calculating the sum of two integers: 3 gas
- Base cost of an Ethereum transaction: $21,000$ gas
- Writing 32 bytes of data to persistent storage: $20,000$ gas
- Adding 32 bytes of non-zero data to a transaction as input data: $2,176$ gas
- Validating an ECDSA signature using the *secp256k1* [16] curve: $3,000$ gas

Figure 6: Gas cost of elliptic curve operations compared with some other Ethereum opcodes [4]

The current implementation does not provide an interface to use Shamir's trick [17] to perform multiple scalar multiplications. Multiple scalar multiplications must be performed individually and then summed together.

The total number of scalar multiplications required in $\mathcal{V}_{\mathsf{joinsplit}}$ is $\mathbf{3n + m - 1}$, with $2n + 2m - 2$ additions, with 1 bilinear pairing comparison.

With the current gas schedule, this translates to approximately $121,000n + 41,000m + 219,000$ gas to verify a proof. With the updated gas schedule, the gas costs drop to $12,200n + 6,200m + 85,930$ gas.

For an example join-split transaction with two input notes and two output notes, the gas cost of the elliptic curve operations required to verify the transaction is equal to $795,000$ gas, dropping to $147,130$ gas with the updated gas schedule.

Group elements in $\mathbb{G}$ on the *bn128* curve are represented in 64 bytes if using uncompressed coordinates, or 32 bytes using compressed coordinates. Scalars in $\mathbb{Z}_p$ are represented in 32 bytes. The size of a proof transcript $((\gamma_1, \sigma_1, \bar{a}_1, \gamma_2, \sigma_2, \bar{a}_2, \bar{k}_2, \ldots, \gamma_n, \sigma_n, \bar{a}_n, \bar{k}_n), c)$ is $128n$ bytes using compressed coordinates, or $192n$ bytes using compressed coordinates. For a 2-input-2-output join-split transaction this translates to a proof transcript size of 512 bytes compressed, 768 bytes uncompressed. The reference implementation utilises uncompressed coordinates, as the gas costs for elliptic curve point decompression are greater than the gas saved by reducing the proof size.

The protocol $\mathcal{P}_{\mathsf{joinsplit}}$ requires $3n$ scalar multiplications. This is well within the capabilities of the embedded processors utilized by hardware wallets, however values from the Boneh-Boyen signature set from the commitment key ck, M, will need to be supplied externally due to the large size of M.

A AZTEC notes and spending keys

The reference implementation of the AZTEC protocol is tailored to minimize the gas costs required to validate an AZTEC join-split transaction through a smart contract defined by the Ethereum protocol's Ethereum Virtual Machine.

Because of this, the note's spending key Q is defined over the *secp256k1* [16] curve, as opposed to the *bn128* curve that is used for the AZTEC zero-knowledge protocols. The gas cost to validate an ecdsa signature on the *secp256k1* curve is 3,000 gas. This contrasts with the 40,000 gas cost of performing a scalar multiplication on the *bn128* curve, leading to an 80,000 gas cost to validate an equivalent ecdsa signature. [2]

Using the *secp256k1* curve enables the protocol to define 'ownership' using the same elliptic curve that the base Ethereum protocol uses, allowing for the potential re-use of Ethereum addresses. We use g' to describe the generator of this curve.

In the reference implementation of the AZTEC protocol, Q is a stealth address that is generated via a modified form of the dual-key stealth address protocol. This protocol is used by several existing blockchains and a summary of the protocol is presented below.

A stealth address 'wallet' contains two public/private key pairs, a **scan** key $(V, v), V = g'^v$, and an **issue** key $(S, s), s = g'^s$.

When constructing an AZTEC note, the transaction sender generates an ephemeral key pair $(B, b), B = g'^b$. The sender then constructs a Diffie-Hellman shared secret between B and V, $x = H(g'^{vb}) = H(V^b)$.

The note's spending key, Q, is defined as $Q = S \cdot g'^x$. In addition, for every note the ephemeral key B is also published.

The advantage of the dksap protocol is twofold: Q is not directly linkable to S or V. It also enables a third party, such as an auditor, to be given enough information to view transactions from an address without revealing information required to issue transactions. By revealing a wallet's **scan** key, an entity can view all AZTEC notes that were issued to the wallet, as well as notes that were used by the wallet as inputs to join-split transactions.

The **viewing key** a, required to decrypt an AZTEC note, is generated by taking x mod p, where p is the order of the *bn128* group g.

[2] the algorithm used to implement secp256k1 elliptic curve operations is significantly more optimized than the bn128 curve. If the costs of using the bn128 curve approaches parity with secp256k1 then the signature scheme will be adapted to use the bn128 curve

References

[1] S. Nakamoto. Bitcoin, a Peer-to-Peer Electronic Cash System. `"https://bitcoin.org/bitcoin.pdf"`, 2008.

[2] A. Mackenzie *et al* S. Noether. Ring Confidential Transactions. *Cryptology ePrint Archive Report*, 2017:1098, 2015.

[3] E. Ben-Sasson *et al.* Zerocash: Decentralized Anonymous Payments from Bitcoin. *Proceedings of the IEEE Symposium on Security & Privacy Oakland*, pages 459–474, 2014.

[4] G. Wood. A Secure Decentralised Generalised Transaction Ledger. `"https://github.com/ethereum/yellowpaper"`, 2015.

[5] ERC-20 Token Standard. `"https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md"`, 2015.

[6] M. Naehrig P. Barreto. Pairing-Friendly Elliptic Curves of Prime Order. *Selected Areas in Cryptography*, pages 319–331, 2006.

[7] G. Bertoni *et al.* KECCAK. `"https://keccak.team/keccak.html"`, 2017.

[8] X. Boyen D. Boneh. Short Signatures Without Random Oracles. *Advances in Cryptology, EUROCRYPT*, pages 56–73, 2004.

[9] Philip Rogaway M. Bellare. Random Oracles are pratical: A paradigm for designing efficient protocols. *ACM conference on Computer and Communications Security*, pages 62–73, 1993.

[10] O. Goldreich M. Bellare. On Defining Proofs of Knowledge. *Advances in Cryptology — CRYPTO '92*, pages 390–420, 1992.

[11] I. Damgård. On the Existence of Bit Commitment Schemes and Zero-Knowledge Proofs. *Proc. of Crypto[5]*, 1989.

[12] A. Shamir A. F. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. *Conference on the Theory and Application of Cryptographic Techniques*, pages 189–194, 1986.

[13] A. Shelat J. Camenisch, R. Chaabouni. Efficient Protocols for Set Membership and Range Proofs. *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security*, pages 234–252, 2008.

[14] G. Arfaoui *et al.* A Practical Set-Membership Proof for Privacy-Preserving NFC Mobile Ticketing. *Proceedings on Privacy Enhancing Technologies*, 2:25–45, 2015.

[15] D. Johnson *et al.* The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[16] secp256k1 curve parameters. `"https://en.bitcoin.it/wiki/Secp256k1"`, 2015.

[17] A. Menezes D. Hankerson. Guide to Elliptic Curve Cryptography. *Springer*, 2004.

[18] EIP 1108 Reduce altbn128 precompile gas costs. `"https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1108.md"`, 2018.