MantaPay Trusted Setup Protocol Specification
v0.0.0

Francisco Hernández Iglesias, Todd Norton [*]

September 28, 2022

Abstract

We describe the protocol for the MantaPay trusted setup ceremony to generate prover and verifier keys for Groth16 ZK-SNARK proofs.

Contents

[*] ordered alphabetically.

1 Introduction

The MantaPay protocol (ref. to the specs) guarantees transaction privacy by using the Groth16 [2] Non-Interactive Zero-Knowledge Proving System (NIZK). In short, Groth16 is defined over a bilinear pairing of elliptic curves $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\tau$ with scalar field $\mathbb{F}_r$. Let $\phi \in \mathbb{F}_r^\ell$ denote the set of *public inputs*, $w \in \mathbb{F}_r^{m-\ell}$ the set of *witnesses*, whose knowledge we want to prove, and let $\tau \in (\mathbb{F}_r^*)^4$ be a set of randomly generated numbers known as the *simulation trapdoor*. Groth16 consists of four parts:

- $(\sigma, \tau) \leftarrow \mathsf{Setup}$: Randomly generates τ, from which it computes σ, which consists of elliptic curve points in $\mathbb{G}_1$ and $\mathbb{G}_2$. σ consists of the proving and verifying keys for Groth16.
- $\pi \leftarrow \mathsf{Prove}(\sigma, \phi, w)$: Computes a proof of knowledge of w, π, for a given setup σ and public input ϕ.
- $0, 1 \leftarrow \mathsf{Verify}(\sigma, \phi, \pi)$: Checks whether the proof π is valid against the setup σ and the public input ϕ.
- $\pi \leftarrow \mathsf{Sim}(\tau, \phi)$: Simulates a proof that will always be valid when verified against the setup σ corresponding to τ and the public input ϕ.

It is important to note that the Sim function is what makes the Groth16 protocol zero-knowledge: you can compute a valid proof π for any given setup σ and public input ϕ without knowledge of the witness w, provided that you have access to the simulation trapdoor τ. But Sim also points to a potential insecurity: if a malicious agent knew τ for a given σ, they could fabricate valid proofs for any witness, regardless of its veracity.

The goal of the *trusted setup* is to compute σ in a secure way, i.e., in such a way that nobody has access to the trapdoor τ that was used to compute it.

2 Context

2.1 Circuit

Throughout this paper, by circuit we mean a *Rank-1 Constraint System (R1CS)*. It is defined as a system of equations over $\mathbb{F}_r$ of the form

$$\sum_{i=0}^{m} a_i u_{i,q} \cdot \sum_{i=0}^{m} a_i v_{i,q} = \sum_{i=0}^{m} a_i w_{i,q}, \qquad q = 1, \dots, n, \tag{1}$$

where $a_0 = 1$. This system of equations, in the context of zero-knowledge proofs, is to be understood as follows:

- The numbers $u_{i,q}, v_{i,q}, w_{i,q}$ are constants in $\mathbb{F}_r$ which represent the operations performed in an arithmetic circuit. Here constant means constant in the circuit, e.g. each MantaPay circuit will have a fixed set of $u_{i,q}, v_{i,q}, w_{i,q}$.
- The numbers $\phi = (a_1, \dots, a_\ell)$ are the public inputs. In MantaPay, these correspond to the TransferPost, excluding the proof.
- The numbers $w = (a_{\ell+1}, \dots, a_m)$ are the private witnesses. In MantaPay, these correspond to the elements of the Transfer which are not part of the TransferPost.
- A R1CS defines the following binary relation

$$R = \Big\{ (\phi, w) \Big|\ \phi = (a_1, \dots, a_\ell),\ w = (a_{\ell+1}, \dots, a_m),\ (1) \text{ is satisfied} \Big\} \subset \mathbb{F}_r^\ell \times \mathbb{F}_r^{m-\ell} \tag{2}$$

- The statements that can be proved in this terminology are of the form, "For a given circuit (1) and public input ϕ, I know a witness w such that $(\phi, w) \in R$."

2.2 Quadratic Arithmetic Programs

Quadratic Arithmetic Programs (QAPs) give an alternative way to describe a circuit, equivalent to R1CS. A QAP is a system of polynomial equations of the form

$$\sum_{i=0}^{m} a_i u_i(X) \cdot \sum_{i=0}^{m} a_i v_i(X) \equiv \sum_{i=0}^{m} a_i w_i(X) \mod t(X), \tag{3}$$

where

- $u_i(X), v_i(X), w_i(X) \in \mathbb{F}_r[X]$ are degree $n-1$ polynomials, and $t(X) \in \mathbb{F}_r[X]$ is a degree 2^k polynomial (see below), all of which are fixed for the protocol.
- The numbers $\phi = (a_1, \ldots, a_\ell)$ are the public inputs.
- The numbers $w = (a_{\ell+1}, \ldots, a_m)$ are the private witnesses.
- A QAP defines the following binary relation

$$R = \Big\{(\phi, w) \Big| \ \phi = (a_1, \ldots, a_\ell),\ w = (a_{\ell+1}, \ldots, a_m),\ (3) \text{ is satisfied}\Big\} \subset \mathbb{F}_r^\ell \times \mathbb{F}_r^{m-\ell} \tag{4}$$

- The statements that can be proved in this terminology are of the form, "For a given circuit (3) and public input ϕ, I know a witness w such that $(\phi, w) \in R$."

We derive the QAP description of a circuit from its R1CS description as follows:

1. Choose k as the minimal integer such that $2^k \geq n$. Let $t(X) = X^{2^k} - 1$. This is the *vanishing polynomial* for the set of 2^k-th roots of unity in $\mathbb{F}_r$.
2. We derive the polynomial $u_i(X)$ from the R1CS vector $(u_{i,q})_{q=1}^n$ via a Lagrange basis $\{L_q(x)\}_{t(q)=0}$ for the set of 2^k-th roots of unity. That is,

$$u_i(X) = \sum_{q=1}^{n} u_{i,q} L_q(X), \tag{5}$$

   where we use the convention $u_{i,q} = 0$ for $q > n$.
3. We repeat the same procedure to define $v_i(X)$ and $w_i(X)$.

One may readily check that (1) is equivalent to (3) with these definitions.

2.3 The Groth16 Setup function

We now recall the Groth16 Setup function. It is defined relative to two choices:

- A pairing curve[1], which consists of a triple of elliptic curves $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ and a non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We fix generators g and h of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. We require that g, h, and $e(g, h)$ all have the same prime order r.
- A circuit, encoded as a QAP $\{u_i(X), v_i(X), w_i(X), t(X)\}$.

All public parameters derive from a simulation trapdoor $\tau = (\alpha, \beta, \delta, x) \leftarrow \mathbb{F}_r^*$ (the famous "toxic waste"). The Groth16 public parameters themselves are elements of $\mathbb{G}_1$ and $\mathbb{G}_2$, specifically

$$\begin{aligned}\sigma_1 = \Bigg[&\alpha, \beta, \delta, \{u_i(x)\}_{i=0}^m, \{v_i(x)\}_{i=0}^m, \{(\beta u_i(x) + \alpha v_i(x) + w_i(x))\}_{i=0}^\ell, \\ &\left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right\}_{i=\ell+1}^m, \left\{\frac{x^i t(x)}{\delta}\right\}_{i=0}^{n-2}\Bigg]_1 \\ \sigma_2 = \Big[&\Big(\beta, \delta, \{v_i(x)\}_{i=0}^m\Big)\Big]_2\end{aligned} \tag{6}$$

where we denote $[y]_1 = y \cdot g$ and $[y]_2 = y \cdot h$ for all $y \in \mathbb{F}_r$. These public parameters make up the Groth 16 proving and verifying keys.

The public output of Setup is $\sigma = (\sigma_1, \sigma_2)$. These are the public parameters from which Groth16 proofs are formed. The trapdoor τ is *not* public; indeed, a malicious prover with knowledge of τ could construct fraudulent proofs. The goal of the trusted setup is to compute σ without revealing τ.

2.4 Multi-Party Computation

A decentralized way to compute σ without revealing τ is to compute σ in such a way that τ becomes a shared secret split among a diverse set of participants. This may be achieved via *secure multi-party computation* (MPC). The MPC we employ is a protocol for computing σ incrementally from private inputs τ_i belonging to participants in the computation.

[1] We use BN254 (see [4] and the references therein)

The key security property of this MPC is that its security is ensured by having at least one honest participant. An honest participant is one who keeps their private input τ_i from all other participants, ideally by permanently clearing it from their system's memory after participation. Put differently, this *1-out-of-N* honest participants guarantee states that to determine the toxic waste τ requires the collusion of *all* participants in the MPC.

By soliciting contributions to the MPC from a diverse set of participants, we increase the difficulty of such collusion. Note that any individual with a stake in the security of MantaPay can guarantee this personally, simply by participating honestly in the Setup MPC.

2.5 Phase Structure

The full Setup MPC splits usefully into two *phases*. The output of *Phase 1* is *universal* in the sense that these parameters may be used by any ZK circuit of small enough size. In *Phase 2* we derive σ from the output of Phase 1. The parameters generated in Phase 2 are *circuit-specific*: they depend on the R1CS description of the circuit (1) and must be computed separately for each ZK circuit. This two-phase splitting of the MPC is formalized in [1].

Phase 1 computes a modified KZG setup [3] consisting of the curve points

$$KZG = (\{[x^i]_1\}_{i=0}^{2n-2}, \{[\alpha x^i]_1\}_{i=0}^{n-1}, \{[\beta x^i]_1\}_{i=0}^{n-1}, [\beta]_2, \{[x^i]_2\}_{i=0}^{n-1}) \tag{7}$$

These Phase 1 parameters are computed via a MPC such that $x, \alpha, \beta \in \mathbb{F}_r$ are shared secrets among the participants. Since Phase 1 parameters don't depend on the specifics of the circuit, a Phase 1 ceremony may be treated as a public good for any project using circuits of appropriate size over the same pairing curve. In our case, we take the Phase 1 parameters computed by the Perpetual Powers of Tau (PPoT) ceremony [5], an ongoing multi-party computation similar to our trusted setup described below, with a 1-out-of-N security assumption. To ensure its soundness, we have personally verified each contribution to PPoT.

3 Groth 16 Phase 2 MPC

Here we define a secure, updateable, verifiable multi-party computation for the Groth16 Setup function. We closely follow the protocol defined in [1].

The MPC proceeds in N Rounds of computation, each of which produces a *state* σ_n, *challenge* c_n, and *proof* π_n. The triple (σ_n, c_n, π_n) is computed recursively from the previous round's triple $(\sigma_{n-1}, c_{n-1}, \pi_{n-1})$ via functions contribute and challenge. A third function verify allows any third party to verify from a transcript $T = \{\sigma_n, c_n, \pi_n\}_{n=1,\ldots,N}$ that each participant computed contribute correctly.

Definition 3.0.1 (State)**.** The *state* of the MPC is a Groth16 prover key (6). We may denote this as

$$\sigma = \left([\delta]_1, [\delta]_2, \left[\frac{\text{cross-term}_i}{\delta}\right]_1, \left[\frac{x^i t(x)}{\delta}\right]_1, \ldots \right) \tag{8}$$

a shorthand for (6) that reminds us which parts of the prover key depend on δ. Only these parts of the prover key are affected by the MPC; all terms summarized by the ellipsis remain invariant.

Definition 3.0.2 (Challenge Hash)**.** *Challenge hashes* are outputs of a collision-resistant hash function H (for example, Blake2 hash with 64-byte digest), see challenge below.

Definition 3.0.3 (Consistent Pair)**.** A *consistent pair* in the bilinear group $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ of size r consists of two pairs $(a_1, b_1) \in \mathbb{G}_1 \times \mathbb{G}_1$, $(a_2, b_2) \in \mathbb{G}_2 \times \mathbb{G}_2$ satisfying

$$e(a_1, b_2) = e(a_2, b_1) \tag{9}$$

Consistent pairs can be used to prove the knowledge of discrete logs:

The main idea is that the pairs satisfy (9) only if there exists a scalar δ such that $b_1 = \delta \cdot a_1$ and $b_2 = \delta \cdot a_2$ (due to the non-degeneracy of e). Thus the following interactive protocol requires the Prover to know the discrete log δ in order to pass:

1. Prover sends (a_1, b_1) to Verifier.
2. Verifier sends challenge point $a_2 \in \mathbb{G}_2$ to Prover.
3. Prover sends matching point $b_2 \in \mathbb{G}_2$ to Verifier.

4. Verifier checks (9).

In Step 3, a prover with knowledge of δ merely computes $b_2 = \delta \cdot a_2$. A prover without knowledge of δ must first solve a discrete log problem to find b_2, so this proof is sound for r large enough.

In practice, step 2 above is replaced by a Fiat-Shamir transform. When this is the case, we define

Definition 3.0.4 (Ratio Proof)**.** A *ratio proof* is a pair $(a_1, b_1) \in \mathbb{G}_1 \times \mathbb{G}_1$ and a *matching point* $b_2 \in \mathbb{G}_2$, together with a prescription for computing a *challenge point* $a_2 \in \mathbb{G}_2$ from a_1, b_1 and a challenge hash, such that (9) is satisfied. We will denote ratio proofs by π.

In addition to making the above protocol non-interactive, the Fiat-Shamir step forms a *contribution chain* by computing the challenge for Round n from the data of Round $n-1$. Verifying the MPC means checking that state σ_n was computed from state σ_{n-1} using some δ attested to by a ratio proof π_n whose a_2 challenge point is computed from (a commitment to) $\sigma_{n-1}, \sigma_n, \pi_{n-1}$, and the Round $n-1$ challenge point.

We now define the functions needed for the MPC.

fn initialize

Inputs: Phase 1 output (7) and R1CS circuit description.

Outputs: Groth 16 proving key σ_0 (6) with $\delta = 1$, challenge hash $c_0 = [0, 0, \ldots 0]$.

Definition:

1. Compute commitments to the Lagrange basis polynomials for the set of 2^k-th roots of unity (where $2^k \geq n$ is the least power of two with this property).

   Specifically, these are commitments of the form $[L_q(x)]_1$, $[L_q(x)]_2$ that we compute from the KZG parameters $[x^i]_j$ via a Fourier transform. Similarly, we compute the commitments of the form $[\alpha \cdot L_q(x)]_1$, $[\beta \cdot L_q(x)]_1$, $[\beta \cdot L_q(x)]_2$ from the KZG parameters $[\alpha \cdot x^i]_1, [\beta \cdot x^i]_j$.

2. For each circuit input i (public or private), compute a commitment to the Lagrange interpolation polynomial $w_i(X)$ defined by (5) from the commitments to the Lagrange polynomials. Similarly, compute these commitments for $\beta \cdot u_i$, $\alpha \cdot v_i$.

3. Compute commitments $[x^i t(x)]_1$ from the KZG parameters.

4. The commitments $[\alpha]_1, [\beta]_1, [\beta]_2$ are already part of the KZG parameters.

5. Assemble the above pieces to form the commitments found in (6) with $\delta = 1$.

fn contribute

Inputs: σ_{n-1}, c_{n-1}, the state and challenge hash from Round $n-1$.

Outputs: σ_n, π_n, the state and proof of Round n.

Definition:

1. Sample $\delta_n \in \mathbb{F}_r$

2. From σ_{n-1} of the form (8), compute

$$\sigma_n = \left(\delta_n \cdot [\delta]_1, \delta_n \cdot [\delta]_2, \delta_n^{-1} \cdot \left[\frac{\text{cross-term}_i}{\delta}\right]_1, \delta_n^{-1} \cdot \left[\frac{x^i t(x)}{\delta}\right]_1, \ldots \right)$$

3. Form ratio proof π_n: Sample $a_1 \in \mathbb{G}_1$, compute $b_1 = \delta_n \cdot a_1$. Compute challenge point a_2 from c_{n-1}, a_1, b_1. Compute $b_2 = \delta_n \cdot a_2$ and let $\pi_n = (a_1, b_1, b_2)$.

4. Clear δ_n from memory. (This step is not related to the correct computation of σ_n, π_n, but rather to the security of the MPC.)

The first and last steps of this protocol have famously led to participants sampling δ using randomness from Chernobyl and later smashing their computers to clear it from memory. An ambitious participant can easily modify our implementation to accommodate exotic sources of randomness.

fn challenge

Inputs: $c_{n-1}, \sigma_{n-1}, \sigma_n, \pi_n$, the state and challenge hash from Round $n-1$ and the state and ratio proof from Round n.

Outputs: c_n, the challenge hash of Round n.

Definition: for some CRH H, let $c_n = H(c_{n-1}|\sigma_{n-1}|\sigma_n|\pi_n)$, the hash of the concatenated byte representations of each piece of data.

fn verify

Inputs: $c_{n-1}, \sigma_{n-1}, \sigma_n, \pi_n$ as above.

Outputs: boolean, indicating whether σ_n was formed from σ_{n-1} according to protocol.

Definition:

1. Check that the Phase 2 invariants of σ_{n-1} and σ_n match.
2. Compute challenge point a_2 from c_{n-1} and a_1, b_1 of π_n.
3. Check that a_2, and $\pi_n = (a_1, b_1, b_2)$ form a consistent pair (9).
4. Check that a_1, b_1 of π_n form a consistent pair with the $[\delta]_2$ points of σ_{n-1} and σ_n.
5. Check that the $[\delta]_1$ points of σ_{n-1}, σ_n form a consistent pair with their $[\delta]_2$ points.
6. Check that the terms $\left[\frac{\text{cross-term}_i}{\delta}\right]_1, \left[\frac{x^i t(x)}{\delta}\right]_1$ of σ_{n-1}, σ_n form a consistent pair with the $[\delta]_2$ points. Here we reverse the order of the pair of $[\delta]_2$ points to account for the fact that $\left[\frac{\text{cross-term}_i}{\delta}\right]_1, \left[\frac{x^i t(x)}{\delta}\right]_1$ is proportional to δ^{-1}.

In practice, Step 6 is performed quickly by computing random linear combinations of the points $\left[\frac{\text{cross-term}_i}{\delta}\right]_1, \left[\frac{x^i t(x)}{\delta}\right]_1$ and checking this pair's consistency (see [1]).

4 Requirements

Goals

- Anonymous: Registration only requires a Twitter handle and an email address, none of which needs to be linked to a participant's identity. Participants are encouraged to publicly announce their contribution, but this is not mandatory.
- Low barrier to entry: participation does not require sophisticated understanding of cryptography. Participants can contribute using a free, open source, independently audited client.
- Low downtime: a ceremony coordinator will minimize the waiting time between contributions.
- Updatable: More contributions can be added to the ceremony at a later stage if desired.
- Verifiable: The ceremony and all its contributions can be verified by independent auditors. We ensure that is the case by:
  - Publishing the transcript of the ceremony, including the validity proof and hash of each contribution.
  - Asking participants to publish their hashes in independent locations, e.g. Twitter.
  - Distributing an open-source verification library.

Non-Goals

- Permisionless: We require participants to pre-register to contribute to the ceremony. However, registration is open to anyone with a Twitter account and an email address.
- Support for independently computed contributions: We make no special effort to support contribution clients other than our own. However, the server code is open-source and a knowledgeable user could write their own contribution client, provided it adheres to the messaging protocol 5.2.

5 Design

5.1 Ceremony Protocol

Here we describe a MPC protocol for N contributors $C_1, \dots C_N$ to participate in the computation of the public parameters σ. Because the participants must contribute in serial, we use a central Coordinator to order the participants and check their contributions. We emphasize that the Coordinator exists solely to help organize the ceremony and has no effect on the security properties of the MPC.

The protocol is initialized by the Coordinator and proceeds in repeated rounds, in which the Coordinator and a Contributor exchange messages. The messaging protocol is described in Section 5.2.

Initialization

The Coordinator computes an initial state σ_0 and challenge hash c_0 as

$$(\sigma_0, c_0) = \mathsf{initialize}(\text{circuits, KZG parameters})$$

The inputs to initialize are the QAP descriptions (3) of some ZK circuits and modified KZG parameters (7) (the output of a prior Phase 1 ceremony). The function initialize computes σ_0, a Groth16 proving key with $\delta = 1$ (see (6)), and an initial challenge hash c_0.

Note that initialize is deterministic and the circuit descriptions and KZG parameters are public. Therefore any observer may verify that Coordinator completed this step correctly.

Contribution Round

The n^{th} round of contribution begins with the state σ_{n-1} and challenge c_{n-1} of the previous round. Contributor C_n will interact with Coordinator to produce the next state σ_n and a proof π_n that it was computed according to the protocol. The round consists of these steps: (refer to Section 3)

1. Coordinator sends (σ_{n-1}, c_{n-1}) to Contributor C_n.
2. Contributor computes $(\sigma_n, \pi_n) = \mathsf{contribute}(\sigma_{n-1}, c_{n-1})$.
3. Contributor sends (σ_n, π_n) to Coordinator (as a signed message)
4. Coordinator computes $\mathsf{verify}(\sigma_{n-1}, c_{n-1}, \sigma_n, \pi_n)$. This checks that the Contributor has formed σ_n from σ_{n-1} according to the protocol.
   (a) If the check fails, Coordinator rejects this contribution. Nothing is added to the transcript and the Coordinator proceeds to next round with state and challenge unchanged ($\sigma_n = \sigma_{n-1}$ and $c_n = c_{n-1}$).
   (b) Otherwise, Coordinator computes challenge $c_n = \mathsf{challenge}(\sigma_{n-1}, c_{n-1}, \sigma_n, \pi_n)$ and records σ_n, π_n, c_n to the Transcript.
5. Coordinator proceeds to next round with (σ_n, c_n).

This process repeats until all Contributors have made their contribution. At the end (assuming all contributions were valid) we have a Groth16 prover key σ with $\delta = \delta_1 \cdot \delta_2 \cdot \ldots \cdot \delta_N$ and a Transcript $T = \{(\sigma_i, \pi_i, c_i)\}_{i=1}^N$ recording all contributions to the ceremony and allowing a third-party to verify that σ was computed according to protocol.

5.2 Messaging Protocol

In each round, the Coordinator and Contributor communicate via (unencrypted?) messages. Messages from the Contributor to the Coordinator are signed with an Ed25519 signature. The Coordinator accepts only those messages with a valid signature whose public verifying key belongs to a Registry of participants. The Registry and signature checks prevent a DDoS attack on the ceremony in which malicious participants fill up the contribution queue and intentionally time-out without contributing.

The Contributor sends one of two messages to the Coordinator: a QueryRequest or an UpdateRequest. Each message follows the format

$$(\text{Participant ID} \mid \text{Domain Tag} \mid \text{Nonce} \mid \text{Payload}).$$

In a QueryRequest the Payload is empty, whereas in an UpdateRequest the Payload contains a new state and proof, (σ_n, π_n). In both cases the message is serialized to bytes and signed with an Ed25519 signature.

The Coordinator responds to these messages according to the current state of the protocol:

- QueryRequest: the Coordinator responds with a QueryResponse containing:
  - Current state σ_{n-1} and challenge c_{n-1}, if Contributor is at front of Queue (see below).
  - Contributor's current position in Queue, if Contributor is not at front of queue.
  - Error messsage, if Contributor has already participated in ceremony.
- UpdateRequest: the Coordinator responds with an UpdateResponse informing the Contributor of whether their contribution was successfully verified.
- If the message from Contributor fails signature verification, the Coordinator responds with the expected nonce for the Contributor's Participant ID. (If the Participant ID is invalid the Coordinator ignores the message.)

5.3 Server State Machine

The Coordinator role is performed by a central server state machine. The Coordinator's duties are to enforce the MPC protocol and organize the contributions.

To enforce the MPC protocol, the Coordinator performs:

- Parameter Initialization: a reproducible initial state σ_0 and initial challenge c_0 are computed from public data.
- Contribution Verification: each contribution to the ceremony is checked to conform to the MPC protocol.
- Contribution Archival: each successful contribution to the ceremony is recorded, together with the proof that it conforms to the MPC protocol.

To organize the contributions, the Coordinator performs:

- Registry Maintenance: a registry of authorized participants is kept.
- Signature Verification: all messages from a Contributor to the Coordinator will be signed.
- Queue Management: during the ceremony, participants are ordered in a queue.

State

The above Coordinator tasks are accomplished by a machine whose state consists of:

- Registry: For each participant, a record of their public signing key, current signature nonce, and whether they have already contributed to the ceremony.
- MpcState: The current pair (σ_n, c_n) (see Section 3).
- Transcript: The history of MPC states and proofs.
- Queue: an ordering of the participants waiting to contribute. This may be a priority queue, if desired.[2]
- TimedLock: a lock is given to the participant at the front of the queue while they compute their contribution. This lock times out after a specified duration and drops the participant from the queue.

State Changes

The following function calls change the state of the machine:

- initialize: set MpcState to (σ_0, c_0) (see above).
- enqueue(ParticipantId): check that participant is registered and has not already contributed. If so, add to end of Queue.
- register: add new participants to the Registry.
- update: Compute verify$(\sigma_{n-1}, c_{n-1}, \sigma_n, \pi_n)$ to check that the latest contribution conforms to MPC protocol. If so,
  - compute challenge c_n, update MpcState to (σ_n, c_n)

[2]A priority queue can help defend against a DDoS attack in which malicious participants intentionally time out without contributing. If a participant times out too many times they can be de-prioritized, allowing more reliable participants to skip them in the queue.

- Add (σ_n, π_n, c_n) to Transcript
- Set participant's contribution status in Registry.
- Update Queue and TimedLock.

If contribution is invalid, update only Queue and TimedLock (and downgrade participant's Queue priority, if applicable).

Operation

The Coordinator state machine initializes its state and then listens for messages and processes these according to the messaging protocol (see above). Two types of messages are recognized, QueryRequest and UpdateRequest. The state machine responds in the following way to each request:

- QueryRequest
  1. Parse ParticipantId from request.
  2. Consult Registry, confirm participant is registered and has not contributed.
  3. Consult Queue; if participant is not in Queue, call enqueue.
  4. Consult Queue; if participant is at front of Queue, send a message containing MpcState to participant and set TimedLock. Otherwise, send participant a message containing their current position in Queue.
- UpdateRequest
  1. Parse ParticipantId from request. Check that the TimedLock refers to this participant.
  2. Parse state, proof (σ_n, π_n) from request.
  3. Call $\mathsf{update}(\sigma_{n-1}, c_{n-1}, \sigma_n, \pi_n)$.

If any of the above checks do not pass, the Coordinator responds with an appropriate error message.

The operation of this machine is summarized in Figure 1.

5.4 Client State Machine

The duty of a Contributor is to contribute according to the MPC protocol. To this end, a client state machine performs:

- Ed25519 Keypair Generation: each Contributor generates their own signature credentials.
- Message Signing: messages to Coordinator are signed.
- Contribution: formed according to MPC protocol.

State

The state of this machine is an Ed25519 private key and a message nonce (unsigned integer). The only state changes during operation are updates to the message nonce. The machine also needs access to some random oracle to perform keypair generation and contribution.

This state is initialized by generate-keypair, which generates an Ed25519 private/public keypair. Note that the same state can later be recovered by prompting users to enter their private key or seed phrase (the correct nonce can be recovered by queries to the server).

Operation

1. Initialize state.
2. Send signed QueryRequest to Coordinator. Await response containing MpcState and challenge, (σ_{n-1}, c_{n-1}).
3. Compute $(\sigma_n, \pi_n) = \mathsf{contribute}(\sigma_{n-1}, c_{n-1})$.
   (a) Sample random δ
   (b) Transform σ_{n-1} by δ to get σ_n
   (c) Produce ratio proof π_n
4. Send signed UpdateRequest to Coordinator. Await response confirming submission.

5. Clear δ from memory.

6 References

References

[1] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive, Paper 2017/1050, 2017. https://eprint.iacr.org/2017/1050.

[2] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[3] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT (10)*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[4] Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings, 2010. https://cryptojedi.org/papers/dclxvi-20100714.pdf.

[5] Ethereum Foundation Privacy and Scaling Exploration Team. Perpetual powers of tau ceremony.

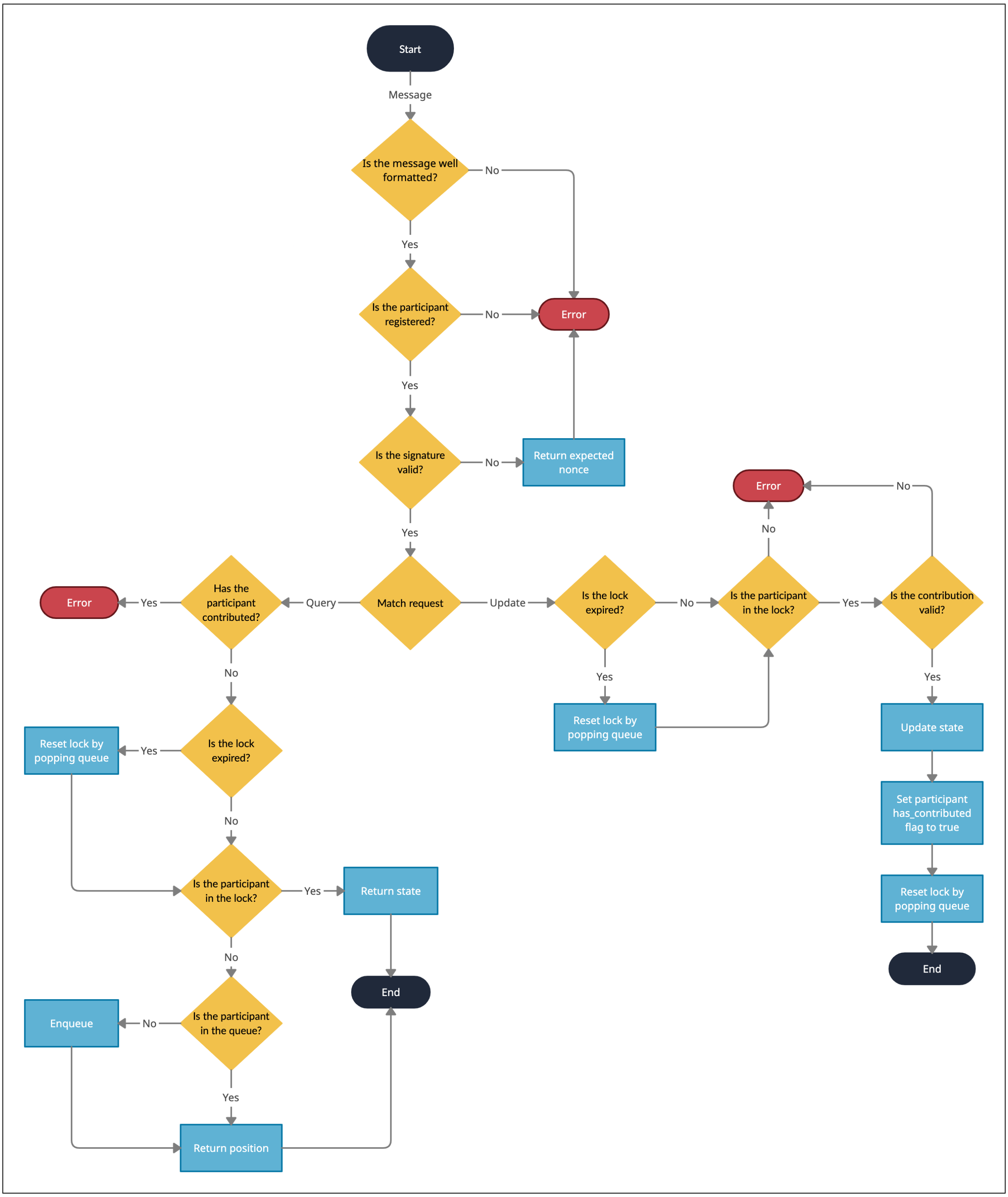

Figure 1: Operation of the server state machine.