A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK

Sean Bowe, Ariel Gabizon, Matthew Green

June 21, 2017

Abstract

Recent efficient constructions of zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs), require a setup phase in which a common-reference string (CRS) with a certain structure is generated. This CRS is sometimes referred to as the *public parameters of the system*, and is used for constructing and verifying proofs. A drawback of these constructions is that whomever runs the setup phase subsequently possesses trapdoor information enabling them to produce fraudulent pseudoproofs.

Ben-Sasson, Chiesa, Green, Tromer and Virza [BCG$^+$15] presented a generic method for computing this CRS in a multi-party protocol, with the property that only if all players collude together they can reconstruct the trapdoor, or, more generally, deduce any other useful information beyond the resultant CRS. Based on [BCG$^+$15], we devise an arguably simpler method for generating the CRS of the Pinocchio zk-SNARK [PHGR16] with a similar security guarantee: Namely, given that the CRS generated by the protocol is later used to verify proofs; a party controlling all but one of the players will not be able to construct fraudulent proofs except with negligible probability. This method has been used in practice to generate the required CRS for the Zcash cryptocurrency blockchain.

Organization of paper Section 1 introduces some terminology and auxiliary methods that will be used in the protocol. Section 2 describes the protocol in detail. Section 3 describes the security proof of the protocol.

1 Definitions, notation and auxillary methods

Terminology: We always assume we are working with a field $\mathbb{F}_r$ for prime r chosen according to a desired security parameter (more details on this in Section 3). We assume together with $\mathbb{F}_r$ we have generated groups $\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_t$, all cyclic of order r; where we write $\mathbb{G}_1$ and $\mathbb{G}_2$ in additive notation and $\mathbb{G}_t$ in multiplicative notation. Furthermore, we have access to generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, and an efficiently computable pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$, i.e., a non-trivial map such that for any $a, b \in \mathbb{F}_r$

$$e(a \cdot g_1, b \cdot g_2) = g_T^{a \cdot b},$$

for a fixed generator $g_T \in \mathbb{G}_t$. We use the notations $g := (g_1, g_2)$ and $G^* := \mathbb{G}_1 \setminus \{0\} \times \mathbb{G}_2 \setminus \{0\}$.

We think of the field size r as a parameter against which we measure efficiency. In particular, we say a circuit A is *efficient* if its size is polynomial in $\log r$. More precisely, when we refer in the security analysis to an efficient adversary or efficient algorithm, we mean it is a (non-uniform) sequence of circuits indexed by r, of size $\text{poly}\log r$. When we say "with probability p", we mean "with probability at least p".

We assume we have at our disposal a function COMMIT taking as input strings of arbitrary length; that, intutively speaking, behaves like a commitment scheme. That is, it is infeasible to deduce COMMIT's input from seeing its output, and it is infeasible to find two inputs that COMMIT maps to the same output. In our implementation we use the BLAKE-2 hash function as COMMIT. For the actual security proof, we need to assume that COMMIT's outputs are chosen by a random oracle.

Symmetric definitions In the following sections we introduce several methods that receive as parameters elements of both $\mathbb{G}_1$ and $\mathbb{G}_2$. We assume implicitly that whenever such a definition is made, we also have the symmetric definition where the roles are reversed between what parameters come from $\mathbb{G}_1$ and $\mathbb{G}_2$. For example, if we define a method receiving as input a vector of $\mathbb{G}_1$ elements and a pair of $\mathbb{G}_2$ elements. We assume thereafter that we also have the symmetric method receiving as input a vector of $\mathbb{G}_2$ elements and a pair of $\mathbb{G}_1$ elements.

1.1 Comparing ratios of pairs using pairings

Definition 1.1. *Given $s \in \mathbb{F}_r^*$, an* s-pair *is a pair (p, q) such that $p, q \in \mathbb{G}_1 \setminus \{0\}$, or $p, q \in \mathbb{G}_2 \setminus \{0\}$; and $s \cdot p = q$. When not clear from the context whether p, q are in $\mathbb{G}_1$ or $\mathbb{G}_2$, we use the terms* $\mathbb{G}_1$-s-pair *and* $\mathbb{G}_2$-s-pair.

A recurring theme in the protocol will be to check that two pairs of elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, "have the same ratio", i.e., are s-pairs for the same $s \in \mathbb{F}_r^*$.
$\underline{\mathsf{SameRatio}((p, q), (f, H))}$:

1. If one of the elements p, q, f, H is zero; return rej.

2. Return acc if $e(p, H) = e(q, f)$; return rej otherwise.

Claim 1.2. *Given $p, q \in \mathbb{G}_1$ and $f, H \in \mathbb{G}_2$, $\mathsf{SameRatio}((p, q), (f, H)) = \mathsf{acc}$ if and only if there exists $s \in \mathbb{F}_r^*$ such that (p, q) is a $\mathbb{G}_1$-s-pair and (f, H) is a $\mathbb{G}_2$-s-pair.*

Proof. Suppose that $s \cdot p = q$ and $s' \cdot f = H$. Write $p = a \cdot g_1, f = b \cdot g_2$ for some $a, b \in \mathbb{F}_r$. Note that if one of $\{a, b, s, s'\}$ is 0, we return rej in the first step.

Otherwise, we have

$$e(p, H) = (a \cdot g_1, bs' \cdot g_2) = g_T^{abs'},$$

and

$$e(q, f) = (as \cdot g_1, b \cdot g_2) = g_T^{abs},$$

and thus $\mathsf{SameRatio}((p, q), (f, H)) = 1$ if and only if $s = s'$ $(mod\ r)$. $\square$

Let $V = ((p_i, q_i))_{i \in [d]}$, be a vector of pairs in $\mathbb{G}_1$. We say V is an *s-vector* in $\mathbb{G}_1$ if for each $i \in [d]$, (p_i, q_i) is a $\mathbb{G}_1$-s-pair, or is equal to $(0, 0)$. We make the analogous definition for $\mathbb{G}_2$, and similarly to above, sometimes omit the group name when it is clear from the context what group the elements are in, simply using the term s-vector. In our protocol we often want to check if a long vector $((p_i, q_i))_{i \in [d]}$ is an s-vector for some $s \in \mathbb{F}_r^*$. The next claim enables us to do so with just one pairing.

Claim 1.3. *Suppose that $((p_i, q_i))_{i\in[d]}$ is a vector of elements in $\mathbb{G}_1 \setminus \{0\}$ that is not an s-vector. Choose random $c_1, \ldots, c_d \in \mathbb{F}_r$ and define*

$$p \triangleq \sum_{i\in[d]} c_i \cdot p_i, \quad q \triangleq \sum_{i\in[d]} c_i \cdot q_i.$$

Then, with probability at least $1 - 2/r$, both $(p, q) \neq (0, 0)$ and (p, q) is not an s-pair

Proof. Write $p_i = a_i \cdot g_1$ for $a_i \in \mathbb{F}_r$, and $q_i = s_i \cdot p_i$ for some $s_i \in \mathbb{F}_r$. Thus, we have $p = a \cdot g_1$ for $a \triangleq \sum_{i\in[d]} c_i a_i$ and $q = b \cdot g_1$ for $b \triangleq \sum_{i\in[d]} \alpha_i a_i s_i$. Let us assume $a \neq 0$. This happens with probability $1 - 1/r$. Write $[d]$ as a disjoint union $S \cup T$ where S is the set of indices of the s-pairs. That is $S \triangleq \{i \in [d] | s_i = s\}$. We have

$$b/a = \frac{\sum_{i\in[d]} c_i a_i s_i}{\sum_{i\in[d]} c_i a_i} = s + \frac{\sum_{i\in T} c_i \cdot (s - s_i)}{\sum_{i\in[d]} c_i a_i} = s + \frac{\sum_{i\in T} c_i \cdot (s - s_i)}{a}.$$

Thus, $b/a = s$ if and only if the fraction in the right hand side is zero. As the numerator is a random combination of non-zero elements, this happens with probability $1/r$.

We conclude that with probability at least $1 - 2/r$, (p, q) is not an s-pair □

Claim 1.3 implies the correctness of $\mathsf{sameRatio}(V, (f, H))$ that given an s-pair (f, H) in $\mathbb{G}_2$, checks whether V is an s-vector in $\mathbb{G}_1$.

$\underline{\mathsf{sameRatio}(V = ((p_i, q_i))_{i\in[d]}, (f, H)):}$

1. If there exists a pair of the form $(0, a)$ or $(a, 0)$ for some $a \neq 0$ in V; return rej.
2. "Put aside" all elements of the form $(0, 0)$, and from now on assume all pairs in V are in $\mathbb{G}_1 \setminus \{0\}$. (If all pairs are of the form $(0, 0)$ then return acc).
3. Choose random $c_1, \ldots, c_d \in \mathbb{F}_r$.
4. Define $p \triangleq \sum_{i\in[d]} c_i \cdot p_i$, and $q \triangleq \sum_{i\in[d]} c_i \cdot q_i$.
5. If $p = q = 0$, return acc.
6. Otherwise, return $\mathsf{SameRatio}((p, q), (f, H))$.

Corollary 1.4. *Suppose rp_s in a $\mathbb{G}_2$-s-pair, and V is a vector of pairs of $\mathbb{G}_1$ elements. If V is an s-vector, $\mathsf{sameRatio}(V, \mathsf{rp}_s)$ accepts with probability one. If V is not an s-vector, $\mathsf{sameRatio}(V, \mathsf{rp}_s)$ accepts with probability at most $2/r$.*

Let V be a vector of $\mathbb{G}_1$-elements and rp_s be a pair of $\mathbb{G}_2$-elements. We also use a method $\mathsf{sameRatioSeq}(V, \mathsf{rp}_s)$ that given an s-pair rp_s, checks that each two consecutive elements of V are an s-pair. It does so by calling $\mathsf{sameRatio}(V', \mathsf{rp}_s)$ with $V' = ((V_0, V_1), (V_1, V_2), \ldots, (V_{d-1}, V_d))$.

1.2 Schnorr NIZKs for knowledge of discrete log

We review and define notation for using the well-known Schnorr protocol [Sch89]. Given an s-pair $\mathsf{rp}_s = (f, H = s \cdot f)$, and a string h, we define the (randomized) string $\mathrm{NIZK}(\mathsf{rp}_s, h)$ that can be intepreted as a proof that the generator of the string knows s.

NIZK(rp_s, h):

1. Choose random $a \in \mathbb{F}_r^*$ and let $R := a \cdot f$.
2. Let $c := \mathsf{COMMIT}(R \circ h)$ and interpret c as an element of $\mathbb{F}_r$, e.g. by taking it's first $\log r$ bits.
3. Let $u := a + cs$.
4. Define NIZK(rp_s, h) $:= (R, u)$.

Let us denote by π a string that is supposedly of the form NIZK(rp_s, h), for some string h.

VERIFY-NIZK(rp_s, π, h) is a boolean predicate that verifies that π is indeed of this form for the same given h.

VERIFY-NIZK($(f, H), \pi, h$):

1. Let R, u be as in the description above.
2. Compute $c := \mathsf{COMMIT}(R \circ h)$.
3. Return acc when $u \cdot f = R + c \cdot H$; and rej otherwise.

1.3 The random-coefficient subprotocol

A large part of the protocol will consist of invocations of the *random-coefficient subprotocol.* In this subprotocol, we multiply a vector of $\mathbb{G}_1$ elements coordinate-wise by the same scalar $\alpha \in \mathbb{F}_r^*$. α here is a product of secret elements $\{\alpha_i\}_{i \in [n]}$, that we refer to later as *comitted elements*. By this we mean, that before the subprotocol is invoked, for each $i \in [n]$, P_i has broadcasted a $\mathbb{G}_2$-α_i-pair, denoted rp_{α_i}, that is accessible to the protocol verifier. (This will become clearer in the context of Section 2).

RCPC(V, α):

Common Input: vector $V \in \mathbb{G}_1^d$.

Individual inputs: element $\alpha_i \in \mathbb{F}_r^*$ for each $i \in [n]$.

Output: vector $\alpha \cdot V \in \mathbb{G}_1^d$, where $\alpha = \prod_{i=1}^n \alpha_i$.

1. P_1 computes broadcasts $V_1 := \alpha_1 \cdot V$.
2. For $i = 2, \ldots, n$, P_i broadcasts $V_i := \alpha_i \cdot V_{i-1}$.
3. Players ouput V_n (which should equal $\alpha \cdot V$).

Before discussing the transcript verification we define one more useful notation. For vectors $S, T \in \mathbb{G}_1^d$ and a $\mathbb{G}_2$-α-pair rp_α, $\mathsf{sameRatio}((S, T), \mathsf{rp}_\alpha)$ returns $\mathsf{sameRatio}(V, \mathsf{rp}_\alpha)$, where $V_i := (S_i, T_i)$. The transcript verification procedure receives as input $V, V_1, \ldots, V_n$, and for each $i \in [n]$, the $\mathbb{G}_2$-α_i-pair, rp_{α_i}.

$\underline{\mathsf{verifyRCPC}(V,\alpha)\mathbf{:}}$

Input: V, protocol transcript $V_1,\ldots,V_n \in \mathbb{G}_1^d$, for each $i\in[n]$ a $\mathbb{G}_2$-α_i-pair rp_{α_i}.

Output: acc or rej.

1. Run $\mathsf{sameRatio}((V,V_1),\mathsf{rp}_{\alpha_1})$.
2. For $i=2,\ldots,n$, run $\mathsf{sameRatio}((V_{i-1},V_i),\mathsf{rp}_{\alpha_i})$.
3. Return acc if all invocations returned acc; and return rej otherwise.

From the correctness of the $\mathsf{sameRatio}(,)$ method (Corollary 1.4) we have that

Claim 1.5. *If the players follow the protocol correctly, the output is $\alpha\cdot V$, and transcript verification outputs* acc *with probability one. Otherwise, transcript verification outputs* acc *with probability at most $2/r$.*

2 Protocol description

The participants The protocol is conducted by n players, a coordinator, and a protocol verifier. In the implementation the role of the coordinator and protocol verifier can be played by the same server. We find it useful to separate these roles, though, as the actions of the protocol verifier may be executed only after the protocol has terminated, if one wishes to reduce the time the players have to be engaged. Moreover, any party wishing to check the validity of the transcript and generated parameters can do so solely with access to the protocol transcript. On the other hand, this has the disadvantage that non-valid messages will be detected only in hindsight, and the whole process will have to be restarted if one wishes to generate valid SNARK parameters.

Similarly, the role of the coordinator is not strictly necessary if one assumes a blackboard model where each player sees all messages broadcasted. (In our actual implementation the coordinator passes messages between the players). Our security analysis holds when all messages are seen by all players. However, even in such a blackboard model there is an advantage of having of a coordinator role: At the beginning of Round 3 a heavy computation needs to performed (Subsection 2.3) that in theory could be perfomed by the first player before he sends his message for that round. However, as this heavy computation does not require access to any secrets of the players, having the coordinator perform it can save much time, if the coordinator is run on a strong server, and the players have weaker machines.

The protocol consists of four “round-robin” rounds, where for each $i\in[n]$, player P_i can send his message after receiving the message of P_{i-1}. P_1 can send his message after receiving an “initializer message” from the coordinator, which is empty in some of the rounds. An exception of this is the first round, where all players may send their message to the coordinator in parallel. However, security is not harmed if a player sees other players’ messages before sending his in that round. Round 2 is divided into several parts for clarity, however the messages of a player P_i in all parts of that round can be sent in parallel. Similarly, Round 3 and 4 consist of several one round round-robin subprotocols; however, the messages of a player P_i in all these subprotocols can be sent in parallel.

2.1 Round 1: commitments

For each $i \in [n]$, P_i does the following.

1. Generate a set of uniform elements in $\mathbb{F}_r^*$

$$\mathsf{secrets}_i := \{\tau_i, \rho_{A,i}, \rho_{B,i}, \alpha_{A,i}, \alpha_{B,i}, \alpha_{C,i}, \beta_i, \gamma_i\}\,.$$

   Omitting the index i for readability from now on, let

$$\mathsf{elements}_i := \{\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \beta, \gamma, \rho_A\alpha_A, \rho_B\alpha_B,$$
$$\rho_A\rho_B, \rho_A\rho_B\alpha_C, \beta\gamma\}$$

2. Now P_i generates the set of group elements[1]

$$\mathsf{e}_i := (\tau, \rho_A, \rho_A\rho_B, \rho_A\alpha_A, \rho_A\rho_B\alpha_B, \rho_A\rho_B\alpha_C, \gamma, \beta\gamma) \cdot g.$$

3. P_i computes $h_i := \mathsf{COMMIT}(\mathsf{e}_i)$ and broadcasts h_i.

2.2 Round 2

Part 1: Revealing commitments

For each $i \in [n]$

1. P_i broadcasts e_i.
2. The protocol verifier checks that indeed $h_i = \mathsf{COMMIT}(\mathsf{e}_i)$.

Committed elements From the end of Round 2, part 1 of the protocol, we refer to the elements of $\mathsf{elements}_i$ for some $i \in [n]$ as *committed elements*. The reason is that by this stage of the protocol, for each $s \in \mathsf{elements}_i$, P_i has sent an s-pair in both $\mathbb{G}_1$ and $\mathbb{G}_2$, effectively committing him to the value of s. For each such element s, we refer to the s-pair in $\mathbb{G}_1$ by rp_s and the s-pair in $\mathbb{G}_2$ by $\mathsf{rp}^2_{\mathsf{s}}$. We list the corresponding elements and s-pairs, ommitting the i subscript for readability:

- τ: $(\mathsf{rp}^1_\tau, \mathsf{rp}^2_\tau) = (g, \tau \cdot g)$.
- ρ_A: $(\mathsf{rp}^1_{\rho_A}, \mathsf{rp}^2_{\rho_\mathsf{A}}) = (g, \rho_A \cdot g)$.
- ρ_B: $(\mathsf{rp}^1_{\rho_B}, \mathsf{rp}^2_{\rho_\mathsf{B}}) = (g, \rho_B \cdot g)$.
- α_A: $(\mathsf{rp}^1_{\alpha_A}, \mathsf{rp}^2_{\alpha_\mathsf{A}}) = (\rho_A \cdot g, \rho_A\alpha_A \cdot g)$.
- α_B: $(\mathsf{rp}^1_{\alpha_B}, \mathsf{rp}^2_{\alpha_\mathsf{B}}) = (\rho_A\rho_B \cdot g, \rho_A\rho_B\alpha_B \cdot g)$.
- α_C: $(\mathsf{rp}^1_{\alpha_C}, \mathsf{rp}^2_{\alpha_\mathsf{C}}) = (\rho_A\rho_B \cdot g, \rho_A\rho_B\alpha_C \cdot g)$.

[1] In the actual code a more complex set of elements is used that can be efficiently derived from $\mathsf{elements}_i$, as described in Appendix A. The reason we use the more complex set is that it potentially provides more security as it contains less information about $\mathsf{secrets}_i$. However, the proof works as well with this definition of e_i and it provides a significantly simpler presentation. We explain in Appendix A the slight modification for protocol and proof for using the more complex element set.

- β: $(\mathsf{rp}^1_\beta, \mathsf{rp}^2_\beta) = (\gamma \cdot g, \beta\gamma \cdot g)$.
- γ: $(\mathsf{rp}^1_\gamma, \mathsf{rp}^2_\gamma) = (g, \gamma \cdot g)$.
- $\rho_A\alpha_A$: $(\mathsf{rp}^1_{\rho_A\alpha_A}, \mathsf{rp}^2_{\rho_\mathsf{A}\alpha_\mathsf{A}}) = (g, \rho_A\alpha_A \cdot g)$.
- $\rho_B\alpha_B$: $(\mathsf{rp}^1_{\rho_B\alpha_B}, \mathsf{rp}^2_{\rho_\mathsf{B}\alpha_\mathsf{B}}) = (\rho_A \cdot g, \rho_A\rho_B\alpha_B \cdot g)$.
- $\rho_A\rho_B$: $(\mathsf{rp}^1_{\rho_A\rho_B}, \mathsf{rp}^2_{\rho_\mathsf{A}\rho_\mathsf{B}}) = (g, \rho_A\rho_B \cdot g)$.
- $\rho_A\rho_B\alpha_C$: $(\mathsf{rp}^1_{\rho_A\rho_B\alpha_C}, \mathsf{rp}^2_{\rho_\mathsf{A}\rho_\mathsf{B}\alpha_\mathsf{C}}) = (g, \rho_A\rho_B\alpha_C \cdot g)$.
- $\beta\gamma$: $(\mathsf{rp}^1_{\beta\gamma}, \mathsf{rp}^2_{\beta\gamma}) = (g, \beta\gamma \cdot g)$.

Of course, we need to check that P_i has committed to the *same* element $s \in \mathbb{F}_r^*$ by rp_s and rp^2_s. This is done by the protocol verifier in the next stage.

Part 2: Checking commitment consistency between both groups

For each $i \in [n]$, and $s \in \mathsf{elements}_i$, the protocol verifier runs $\mathsf{SameRatio}(\mathsf{rp}_s, \mathsf{rp}^2_\mathsf{s})$, and outputs rej if any invocation returned rej.

Part 3: Proving and verifying knowledge of discrete logs

Let $h := \mathsf{COMMIT}(h_1 \circ \ldots \circ h_n)$ be the hash of the transcript of Round 1. P_1 computes and broadcasts h.

For each $i \in [n]$

1. For $s \in \mathsf{secrets}_i$, let $h_{i,s} := h \circ \mathsf{rp}^1_s$. Note that both P_i and the protocol verifier, seeing the transcript up to this point, can efficiently compute the elements $\{h_{i,s}\}$.
2. For each $s \in \mathsf{secrets}_i$, P_i broadcasts $\pi_{i,s} := \mathrm{NIZK}(\mathsf{rp}^1_s, h_{i,s})$.
3. The protocol verifier checks for each $s \in \mathsf{secrets}_i$ that $\mathrm{VERIFY\text{-}NIZK}(\mathsf{rp}^1_s, \pi_{i,s}, h_{i,s}) = \mathsf{acc}$.

Part 4: The random powers subprotocol:

The purpose of the subprotocol is to output the vector

$$\mathrm{POWERS}_\tau := \left((1, \tau, \tau^2, \ldots, \tau^d) \cdot g_1, (1, \tau, \tau^2, \ldots, \tau^d) \cdot g_2\right),$$

where $\tau := \tau_1 \cdots \tau_n$. Recall that $\tau_1, \ldots, \tau_n$ are committed values from Round 1.

For a vector $V \in \mathbb{G}_1^{d+1}$, and $a \in \mathbb{F}_r$, we use below the notation $\mathsf{powerMult}(V, a) \in \mathbb{G}_1^{d+1}$, defined as

$$\mathsf{powerMult}(V, a)_i \triangleq a^i \cdot V,$$

for $i \in \{0, \ldots, d\}$. We use the analogous notation for a vector $V \in \mathbb{G}_2^{d+1}$.

Phase 1: Computing power vectors

1. P_1 does the following.
   (a) Computes $V_1 = (1, \tau_1, \tau_1^2, \ldots, \tau_1^d) \cdot g_1$ and $V_1' = (1, \tau_1, \tau_1^2, \ldots, \tau_1^d) \cdot g_2$.
   (b) Broadcasts (V_1, V_1').
2. For $i = 2, \ldots, n$, P_i does the following:
   (a) Compute $V_i \triangleq \mathsf{powerMult}(V_{i-1}, \tau_i)$ and $V_i' \triangleq \mathsf{powerMult}(V_{i-1}', \tau_{i-1})$.
   (b) Broadcasts (V_i, V_i').

Phase 2: Checking power vectors are valid The protocol verifier performs the following checks[2] on the broadcasted data from Phase 1:

1. Check that
$$\mathsf{sameRatioSeq}(V_1, \mathsf{rp}^2_{\tau_1}),$$
and
$$\mathsf{sameRatioSeq}(V_1', (V_{1,0}, V_{1,1}))$$
2. For each $i \in [n] \setminus \{1\}$ check that
$$\mathsf{sameRatioSeq}(V_i, (V_{i,0}', V_{i,1}')),$$

$$\mathsf{sameRatioSeq}(V_i', (V_{i,0}, V_{i,1})),$$
and
$$\mathsf{SameRatio}((V_{i-1,1}, V_{i,1}), \mathsf{rp}^2_{\tau_i})$$

The protocol verifier rejects the transcript if one of the checks failed; otherwise, the coordinator defines $(PK_H \triangleq V_n, PK_H' \triangleq V_n')$ is taken as the subprotocol output.

Phase 3: Checking we didn't land in the zeros of Z The zero-knowledge property of the SNARK requires we weren't unlucky and τ landed in the zeroes of $Z(X) := X^d - 1$.

- Protocol verifier and all players check that $Z(\tau) \cdot g_1 = (\tau^d - 1) \cdot g_1 = V_{n,d} - V_{n,0} \neq 1$. If the check fails the protocol is aborted and restarted.

[2] The checks below could be simplified if we had also used $\mathsf{rp}^1_{\tau_i}$. We do not use it as in the actual code, as explained in Appendix A, we do not have a $\mathbb{G}_1$-τ_i-pair.

2.3 Coordinator after Round 2: Computing Lagrange basis using FFT, and preparing the vectors $\vec{A}$,$\vec{B}$ and $\vec{C}$

To avoid a quadratic proving time the polynomials in the QAP must be evaluated in a Lagrange basis. There seems to be no way of directly computing a Lagrange basis at τ in a 1-round MPC in a similar way we did for the standard basis in the Random-Powers subprotocol. Thus we will do 'FFT in the coefficient' to compute the Lagrange basis on the output of the random-powers subprotocol. Details and definitions follow. Let $\omega \in \mathbb{F}_r$ be a primitive root of unity of order $d = 2^\ell$, in code d is typically the first power of two larger or equal to the circuit size.

For $i = 1, \ldots, d$, we define L_i to be the i'th Lagrange polynomial over the points $\{\omega^i\}_{i\in[d]}$. That is, L_i is the unique polynomial of degree smaller than d, such that $L_i(\omega^i) = 1$ and $L_i(\omega^j) = 0$, for $j \in [d] \setminus \{i\}$.

Claim 2.1. *For $i \in [d]$ we have*

$$L_i(X) := c_d \cdot \sum_{j=0}^{d-1} (X/\omega^i)^j,$$

for $c_d := \frac{1}{d}$.

Proof. Substituting $X = \omega^{i'}$ for $i' \neq i$ we have a sum over all roots of unity of order d which is 0. Substituting $X = \omega^i$ we have a sum of d ones divided by d which is one. $\square$

For $\tau \in \mathbb{F}_r^*$, denote by we denote by $\mathrm{LAG}_\tau \in \mathbb{G}_1^d \times \mathbb{G}_2^d$ the vector

$$\mathrm{LAG}_\tau := \left((L_i(\tau) \cdot g_1)_{i\in[d]}, (L_i(\tau) \cdot g_2)_{i\in[d]}\right).$$

The purpose of the FFT-protocol is to compute LAG_τ from POWERS_τ. Let us focus for simplicity how to compute the first half containing the $\mathbb{G}_1$ elements. Computing the second half is completely analogous. We define the polynomial $P(Y)(= P_\tau(Y))$ by

$$P(Y) := \sum_{j=0}^{<d} (\tau \cdot Y)^j.$$

It is easy to check that

Claim 2.2. *For $i \in [d]$*

$$L_i(\tau) = P(\omega^{-i}) = P(\omega^{d-i}),$$

and thus

$$\mathrm{LAG}_\tau = (P(\omega^{-i}))_{i\in[d]} \cdot g$$

Thus our task reduces to computing the vector $(P(\omega^i))_{i\in[d]} \cdot g_1$ (and then reordering accordingly). We describe an algorithm to compute the vector $(P(\omega^i))_{i\in[d]}$ using the vector $(1, \tau, \tau^2, \ldots, \tau^d)$ as input and only linear combination gates. This suffices as these linear combinations can be simulated by scalar multiplication and addition in $\mathbb{G}_1$, when operating on POWERS_τ. We proceed to review standard FFT tricks that will be used.

For a polynomial $P(Y) = \sum_{i=0}^{<d} a_i \cdot Y^i$ of degree smaller than d, where d is even, we define the polynomials

$$P_{\text{EVEN}}(Y) := \sum_{i=0}^{<d/2} a_{2i} \cdot Y^i,$$

and

$$P_{\text{ODD}}(Y) := \sum_{i=0}^{<d/2} a_{2i+1} \cdot Y^i.$$

It is easy to see that

$$P(Y) = P_{\text{EVEN}}(Y^2) + Y \cdot P_{\text{ODD}}(Y^2).$$

In particular, for $i \in [d]$

$$P(\omega^i) = P_{\text{EVEN}}(\omega^{2i}) + \omega^i \cdot P_{\text{ODD}}(\omega^{2i})$$

For $j = 0, \ldots, \ell - 1$ denote $\omega_j \triangleq \omega^{2^j}$. Note further that $\left\{\omega^{2i}\right\}_{i\in[d]}$ is a subgroup if size $d/2$ generated by ω_1. More generally, for $j = 1, \ldots, \ell - 1$ $\left\{\omega_{j-1}^{2i}\right\}_{i\in[d]}$ is a subgroup of size 2^{d-j} generated by ω_j. The above discussion suggests the following (well-known FFT) recursive algorithm.

FFT

input: Polynomial P, given as list of coefficients, element $\omega \in \mathbb{F}_r$ generating a group of size $d = 2^\ell$.

output: The vector $V = (P(\omega^i))_{i\in[d]}$.

1. If $d = 2$ compute V directly.
2. Otherwise,
   (a) Call the method recursively twice; first with P_{EVEN} and ω^2 to obtain output $E := (P_{\text{EVEN}}(\omega^{2i}))_{i\in[d/2]}$, and then with P_{ODD} and ω^2 to obtain the vector $O := (P_{\text{ODD}}(\omega^{2i}))_{i\in[d/2]}$.
   (b) Compute the vector V using E, O and the equality mentioned above. More specifically, each element V_i of V is computed as

$$V_i = P(\omega^i) = P_{\text{EVEN}}(\omega^{2i}) + \omega^i \cdot P_{\text{ODD}}(\omega^{2i}) = E_i + \omega^i \cdot O_i,$$

   (where we subtract $d/2$ from indices of E and O when they are larger than $d/2$).

In summary, we obtain LAG_τ by applying the FFT and the polynomial P described above, with coefficients $1, \tau, \ldots, \tau^{d-1}$ and an ω of order d - which should be the same ω used in the QAP construction. After getting the result from the FFT, we reverse the order of the vector and multiply each element by the scalar $1/d$.

Preparing the vectors $\vec{A}$,$\vec{B}$ and $\vec{C}$ We need to compute the vectors $\vec{A} := (A_i(\tau))_{i\in[0..m+1]} \cdot g_1$, $\vec{B} := (B_i(\tau))_{i\in[0..m+1]} \cdot g_1$, $\vec{B_2} := (B_i(\tau))_{i\in[0..m+1]} \cdot g_2$, and $\vec{C} := (C_i(\tau))_{i\in[0..m+1]} \cdot g_1$. We remark that [BCTV14] use the same notation for vectors of polynomials, while we are looking at the vector of these polynomials evaluated at τ.

Note that[3] $A_{m+1} = B_{m+1} = C_{m+1} := Z[\tau] \cdot g_1 = (\tau^d - 1) \cdot g_1$. After the FFT, we have obtained LAG_τ, so each such element is a linear combination of elements of LAG_τ; except $Z(\tau) \cdot g$, that can be computed using the elements $\tau^d \cdot g$ in POWERS_τ.

2.4 Round 3

After the random-powers subprotocol and the FFT, the MPC consists of a few invocations of the random-coefficient subprotocol. These invocations add a total of two rounds to the MPC, as sometimes and random-coefficient subprotocol will need the output of a previous random-coefficient subprotocol as input.

Part 1: broadcasting result of FFT The coordinator broadcasts the vectors $\vec{A}, \vec{B}, \vec{C}, \vec{B}_2$.

Part 2: Random coefficient subprotocol invocations We apply the random-coefficient subprotocol numerous times to obtain the different key elements. For an element $\alpha_i \in \mathsf{elements}_i$, we abuse notation here and denote $\alpha := \alpha_1 \cdots \alpha_n$ (as opposed to ommitting the index i and writing α for α_i which we did when describing Round 1).

1. $PK_A = \mathrm{RCPC}(\vec{A}, \rho_A)$.
2. $PK_B = \mathrm{RCPC}(\vec{B}_2, \rho_B)$.
3. $PK_C = \mathrm{RCPC}(\vec{C}, \rho_A \rho_B)$.
4. $PK'_A = \mathrm{RCPC}(\vec{A}, \rho_A \alpha_A)$
5. $PK'_B = \mathrm{RCPC}(\vec{B}, \rho_B \alpha_B)$.
6. $PK'_C = \mathrm{RCPC}(\vec{C}, \rho_A \rho_B \alpha_C))$
7. $temp_B = \mathrm{RCPC}(\vec{B}, \rho_B)$
8. $VK_Z = \mathrm{RCPC}(g_2 \cdot Z(\tau), \rho_A \rho_B)$. We use that $g_2 \cdot Z(\tau) = g_2 \cdot (\tau^d - 1)$ can be computed from PK'_H that was computed in Round 2, part 2, as described in Section 2.2.
9. $VK_A = \mathrm{RCPC}(g_2, \alpha_A)$.
10. $VK_B = \mathrm{RCPC}(g_1, \alpha_B)$.
11. $VK_C = \mathrm{RCPC}(g_2, \alpha_C)$.

[3]A technicality is that in the protocol description in [BCTV14] $Z(\tau) \cdot g_2$ is appended with index $m+2$ in $\vec{B}_2$, and $Z(\tau) \cdot g_1$ is appended in index $m+3$ in C. However in the actual libsnark code, they are apppended in index $m+1$, and the prover algorithm is slightly modified to take this into account. But for the security proof we assume later on as in [BCTV14] that $A_{m+1} = C_{m+3} = Z(\tau) \cdot g_1$, $B_{m+2} = Z(\tau) \cdot g_2$, $A_{m+2}, A_{m+3}, B_{m+1}, B_{m+3}, C_{m+1}, C_{m+2} = 0$.

2.5 Round 4: Computing key elements involving β, especially PK_K

Each player (or just the coordinator) computes $V := PK_A + temp_B + PK_C$. The players compute

1. $PK_K = \mathrm{RCPC}(V, \beta)$
2. $VK_\gamma = \mathrm{RCPC}(g_2, \gamma)$
3. $VK^1_{\beta\gamma} = \mathrm{RCPC}(g_1, \beta\gamma)$.
4. $VK^2_{\beta\gamma} = \mathrm{RCPC}(g_2, \beta\gamma)$.

Finally, the protocol verifier will run $\mathsf{verify}\mathrm{RCPC}(,)$ on the input and transcript of each subprotocol executed in Round 3 or 4; and output acc if and only if all invocations of $\mathsf{verify}\mathrm{RCPC}(,)$ returned acc.

3 Security proof

Fix a QAP instance ϕ and input x for ϕ. Let V be the snark verifier of [PHGR16]. We denote by $T \subset [n]$, $|T| = n-1$, the subset of players controled by the adversary B. Our goal is to show that under certain cryptographic assumptions, most notably the Knowledge of Exponent (KEA) assumption, if B can generate a proof that $\mathsf{V}(\phi, x)$ accepts with non-negligible probability, when V is using the parameters generated in the protocol, then there exists an extractor E generating a witness ω satisfying (ϕ, x) with non-negligible probability. Our proofs hold in the random oracle model where a random oracle $\mathcal{R}$ takes the place of COMMIT in the protocol.

Notational conventions To simplify notations we will refer to a fixed pair of groups $(\mathbb{G}_1, \mathbb{G}_2)$. of order r; we implicitly assume that we have a generator $\mathcal{G}$ that when given integer t as parameter, returns a prime r with $r = \mathrm{poly}(t)$ and groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ of order r, with the property that for all cryptographic problems described next (Knowledge of Exponent, Strong Diffie-Helman, Power Diffie-Helman) an efficient adversary has $\mathsf{negl}(\log r)$ success probability; where we remind that by efficient adversary we mean a non-uniform (sequence of) circuit(s) of size $\mathrm{poly}(\log r)$.

3.1 Cryptographic assumptions

We use bilinear versions of the Strong Diffie-Helman, Power Diffie-Helman, and Knowledge of Exponent assumption as in [CFH+15]. It will be convenient to denote $G^* := \mathbb{G}_1 \setminus \{0\} \times \mathbb{G}_2 \setminus \{0\}$, and as before by $g = (g_1, g_2)$ a pair of generators for both groups.

Definition 3.1 (Knowledge of Exponent Assumption (KEA))**.** *For any efficient A there exists an efficient E such that the following holds. Fix positive integer d, and an efficient randomized circuit S with input domain $(G^*)^{2(d+1)}$. Consider the following experiment.*

$\tau, \alpha \in \mathbb{F}_r^*$ and $g \in G^*$ are chosen uniformly. We denote $V := (1, \tau, \ldots, \tau^d, \alpha, \alpha\tau, \ldots, \alpha\tau^d) \cdot g$ Then A is given as input $S(V)$; and outputs a pair of the form (c, d) in $\mathbb{G}_1 \setminus \{0\}$, which he "hopes" is of the form $(c, \alpha c)$. E, given the same input, outputs $a_0, a_1, \ldots, a_d \in \mathbb{F}_r$ such that the probability that both

1. *A "succeeded", i.e., $d = \alpha \cdot c$. But,*

2. *E "failed", i.e., $c \neq (\sum_{i=0}^{d} a_i \tau^i) \cdot g_1$.*

is $\mathsf{negl}(\log r)$. *The same holds when $\mathbb{G}_1 \setminus \{0\}$ is replaced by $\mathbb{G}_2 \setminus \{0\}$ and g_1 is replaced by g_2.*

Remark 3.2. *Bitansky et. al [BCPR14], in fact show that the above assumption is false, assuming the existence of indistinguishability obfuscation, when allowing a general S. If one is troubled by this, one may replace the general S in the above definition, by limitting S to be a circuit that computes one of the functions in Claim 3.5, after choosing some of its inputs (besides the instance ϕ) uniformly. We prefered to stick with the less cumbersome definition above, particularly, as the existence of indistinguishability obfuscation is questionable.*

Definition 3.3 (q-SDH assumption)**.** *Fix positive integer q. Consider the following experiment. $\tau \in \mathbb{F}_r^*$ and $g \in G^*$ are chosen uniformly. Then an efficient A is given as input $(1, \tau, \ldots, \tau^q) \cdot g$. Then the probability that A outputs $e(g_1, g_2)^{\frac{1}{\tau + c}}$, for some $c \in \mathbb{F}_r^*$ is* $\mathsf{negl}(\log r)$.

Definition 3.4 (q-PDH assumption)**.** *Fix positive integer q. Consider the following experiment. $\tau \in \mathbb{F}_r^*$ and $g \in G^*$ are chosen uniformly. Then an efficient A is given as input $(1, \tau, \ldots, \tau^q, \tau^{q+2}, \ldots, \tau^{2q}) \cdot g$. Then the probability that A outputs $\tau^{q+1} \cdot g_1$ is* $\mathsf{negl}(\log r)$.

3.2 The Pinocchio Theorem

We refer to Appendix B of [BCTV14] for a descripition of the Pinocchio protocol [PHGR16] with notation close to what is used here. We also give a semi-formal description of the protocol in Appendix B.

Furthermore, one may refer to [PHGR16] and [BCTV14] for definitions relating to quadratic artihmetic programs (QAPs), that we assume familiarity with here. Given a QAP instance ϕ of degree d, we typically denote by params_ϕ a legitimate set of Pinocchio parameters, i.e., proving key and verification key, for ϕ. We can think of params_ϕ as a deterministic function of the values $\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \gamma, \beta \in \mathbb{F}_r^*$; or alternatively of a vector $\mathsf{secrets} \in (\mathbb{F}_r^*)^8$ consisting of these values. Actually, for our security proof we need to define params_ϕ to include all Pinocchio key elements "given in both groups". Specifically, denoting $g := (g_1, g_2)$,

$$\mathsf{params}_\phi(\mathsf{secrets}) := (P_H, P_\phi, P_{\alpha\phi}, P_K, V),$$

where $P_H := \left\{\tau^i \cdot g\right\}_{i \in [d]}, P_\phi := (P_A, P_B, P_C), P_{\alpha\phi} := (P_{\alpha A}, P_{\alpha B}, P_{\alpha C}), P_K := \beta \cdot (P_A + P_B + P_C)$, where

$$P_A := (A_i \cdot g_A)_{i \in [m]}, P_{\alpha A} := (A_i \alpha_A \cdot g_A)_{i \in [m]},$$

and $P_B, P_{\alpha B}, P_C, P_{\alpha C}$ defined similarly, where $g_A := \rho_A \cdot g, g_B := \rho_B \cdot g$ and $g_C := \rho_A \rho_B \cdot g$, and

$$V := (\rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \rho_A \rho_B Z(\tau), \gamma, \beta\gamma) \cdot g$$

The following claim enables us to use the knowledge of exponent assumption in the proof of Theorem 3.6.

Claim 3.5. *There are efficient (deterministic) functions F_A, F_B, F_C such that the following holds. Fix any vector $\mathsf{secrets} = (\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \gamma, \beta)$ in $(\mathbb{F}_r^*)^8$ and instance ϕ. Denote by $\mathsf{secrets}_A, \mathsf{secrets}_B$ and $\mathsf{secrets}_C$ the vector $\mathsf{secrets}$ when ommitting the element α_A, α_B and α_C respectively. Then*

$$\mathsf{params}_\phi(\mathsf{secrets}) = F_A(P_H, P_H \cdot \alpha_A, \mathsf{secrets}_A, \phi)$$

$$= F_B(P_H, P_H \cdot \alpha_B, \mathsf{secrets}_B, \phi) = F_C(P_H, P_H \cdot \alpha_C, \mathsf{secrets}_C, \phi)$$

Proof. By simple examination of the elements of $\mathsf{params}_\phi(\mathsf{secrets})$. For example, the only elements in $\mathsf{params}_\phi(\mathsf{secrets})$ that are not a function of $\mathsf{secrets}_A$ are $P_{\alpha A}$ and $\alpha_A \cdot g \in V$. Both are a joint efficient function of ρ_A and $P_H \cdot \alpha_A$: We can compute $P_H \cdot \alpha_A \cdot \rho_A = \left\{\tau^i \rho_A \alpha_A \cdot g\right\}_{i\in[d]}$ and then $P_{\alpha A}$ elements are linear combinations of elements of this vector, and $\alpha_A \cdot g$ is simply the first element of $P_H \cdot \alpha_A$.

□

Recall that V denotes the Pinocchio protocol verifier. The inputs of V include the QAP instance ϕ, the instance input x, the CRS or public parameters params_ϕ, and the purported proof π that indeed x satisfies ϕ, and that the prover knows a witness showing this. A slight modification of the security proof of [PHGR16] shows that

Theorem 3.6. *[Pinocchio proof of knowledge] For any efficient A there exists an efficient extractor E such that for any instance ϕ and input x the following holds. The probability over $\mathsf{secrets} \in (\mathbb{F}_r^*)^8$, when A and E are given (ϕ, x) and $\mathsf{params}_\phi(\mathsf{secrets})$ as input that*

1. *A produces π such that $\mathsf{V}(\phi, x, \mathsf{params}_\phi(\mathsf{secrets}), \pi) = \mathsf{acc}$, but*
2. *E does not output a witness ω satisfying (ϕ, x),*

is $\mathsf{negl}(\log r)$.

The proof is almost identical to that of [PHGR16], but we present it here for completeness, at times referring to [PHGR16] for details. An advantage of this proof is that it works for the variant of Pinocchio actually implemented in libsnark, as described in Appendix B of [BCTV14]. To our knowledge, no proof for this variant is written elsewhere. We also recommend looking at the proofs of [GGPR13, CFH+15] for intuition and clarifications.

Proof. We are given A and wish to construct E. We describe how E operates given inputs (ϕ, x) and $\mathsf{params}_\phi(\mathsf{secrets})$. Recall that the purported proof π produced by V has the structure, in the notation of [BCTV14],

$$\pi = (\pi_A, \pi_B, \pi_C, \pi'_A, \pi'_B, \pi'_C, \pi_K, \pi_H).$$

From the description of V, we know that whenever A produces a proof π accepted by V, we have in particular, $\pi'_A = \alpha_A \cdot \pi_A, \pi'_B = \alpha_B \cdot \pi_B$ and $\pi'_C = \alpha_C \cdot \pi_C$, where $\alpha_A, \alpha_B, \alpha_C$ are the corresponding elements of the vector $\mathsf{secrets}$. E works as follows. It gives $\mathsf{params}_\phi(\mathsf{secrets})$ as input to the extractors E_A, E_B, E_C that exist by the KEA assumption together with Claim 3.5. Let $a_0, \ldots, a_d$ be E_A's output. Define $A_{\mathrm{mid}}(X) := \sum_{i=0}^d a_i \cdot X^i$. Let A_{io} be the polynomial of degree at most d defined by the io elements in x; i.e., when $x = (c_1, \ldots, c_n)$, $A_{io}(X) := A_0(X) + \sum_{i=1}^n c_i \cdot A_i(X)$. Define $A(X) := A_{io}(X) + A_{\mathrm{mid}}(X)$. E does an analogous thing with E_B and E_C to obtain polynomials B, C. Now, using linear algebra, E determines whether there exists a set of coefficients $c = (c_0 = 1, c_1, \ldots, c_m)$, such that

$$A(X) = \sum_{i=0}^m c_i \cdot A_i(X), B(X) = \sum_{i=0}^m c_i \cdot B_i(X), C(X) \cdot \sum_{i=0}^m c_i \cdot C_i(X).$$

From the non-degeneracy property[4] [BCTV14], it follows that if there exists such c it coincides with x on $c_1, \ldots, c_n$. If such c exists, output any such c as the proposed QAP witness ω for (ϕ, x).

[4] One needs to make a stronger definition of non-degeneracy than given in [BCTV14]. Specifically, we require that $A_0, \ldots, A_n$ are linearly independent *and* their span is disjoint form the span of $\{A_{n+1}, \ldots, A_m\}$ except for 0.

Otherwise, abort. Let η be the probability that A produced a valid proof but E did not produce a valid assignment for (ϕ, x). Let $q := 4d+4$. We construct an efficient B with the following property. Given a challenge $\mathsf{challenge} = \mathsf{challenge}_s := (1, s, \ldots, s^q, s^{q+2}, \ldots, s^{2q}) \cdot g$, where s is uniform in $\mathbb{F}_r^*$, B outputs with probability $\eta - \mathsf{negl}(\log r)$ either

- $s^{q+1} \cdot g_1$, or
- $e(g_1, g_2)^{\frac{1}{s+t}}$, for some $t \in \mathbb{F}_r^*$.

This implies that $\eta = \mathsf{negl}(\log r)$, as otherwise it would contradict the $2q - SDH$ or $q - PDH$ assumption. Thus, showing $\eta = \mathsf{negl}(\log r)$ suffices to prove the theorem. [5]

Description of B: Given $\mathsf{challenge}$, B begins by constructing a valid set of parameters $\mathsf{params}^{\mathsf{pin}}$, that are a randomized function $\mathsf{params}^{\mathsf{pin}}(s)$ of $s \in \mathbb{F}_r^*$, as follows.

1. $\alpha_A, \alpha_B, \alpha_C, \rho'_A, \rho'_B, \gamma'$ are chosen uniformly in $\mathbb{F}_r^*$
2. We define $\rho_A := \rho'_A \cdot s^{d+1}, \rho_B := \rho'_B \cdot s^{2(d+1)}$
3. For $i \in [0..m]$, define the polynomial $P_i(X) := \rho'_A X^{d+1} \cdot A_i(X) + \rho'_B X^{2(d+1)} \cdot B_i(X) + \rho'_A \rho'_B X^{3(d+1)} \cdot C_i(X)$. Define V to be the $\mathbb{F}_r$-linear space $V := \text{span}\,\{P_i\}_{i \in [0..m]}$ and U to be the $\mathbb{F}_r$-linear space of all polynomials f of degree at most $3d+3$ such that $f \cdot P_i$ has a zero coefficient at X^q for each $i \in [0..m]$.
4. Choose random $f \in U$, and let $\beta := s \cdot f(s)$.
5. Let $\gamma := \gamma' \cdot s^{q+2}$.
6. output $\mathsf{params}^{\mathsf{pin}}(s) := \mathsf{params}_\phi(s, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \gamma, \beta)$.

It can be verified that

1. When s is uniform in $\mathbb{F}_r^*$, $\mathsf{params}^{\mathsf{pin}}(s)$ is distributed as $\mathsf{params}_\phi(\mathsf{secrets})$ for uniform value of $\mathsf{secrets}$.
2. $\mathsf{params}^{\mathsf{pin}}(s)$ can be efficiently computed from $\mathsf{challenge}_s$.

The main point to see the second item is that the power s^{q+1} does not appear in any of the elements of $\mathsf{params}^{\mathsf{pin}}(s)$ (this is immediate for most elements, and requires a calculation for elements containing β).

Now B runs A on $(\phi, x, \mathsf{params}^{\mathsf{pin}}(s))$ to obtain a purported proof $\pi = (\pi_A, \pi_B, \pi_C, \pi'_A, \pi'_B, \pi'_C, \pi_K, \pi_H)$.

B also runs E_A, E_B, E_C on $\mathsf{params}^{\mathsf{pin}}(s)$ to obtain polynomials A, B, C described above, and attempts to find a vector c of coefficients as above. Because of Item 1, together with the KEA and the reasoning above, we know that with probability at least $\eta - \mathsf{negl}(\log r)$ over s,

$$A_{io}(s)\rho_A \cdot g_1 + \pi_A = A(s)\rho_A \cdot g_1; \pi_B = B(s)\rho_B \cdot g_2; \pi_C = C(s)\rho_A\rho_B \cdot g_1,$$

but one of the following happened

[5] For the completely precise argument one must also choose $g \in G^*$ randomly and take success probability over that. To avoid having to "carry the g", we implicitly assume that whenever a statement is made about fixed g it happens with non-negligible probability over a uniform choice of g. e.g. in the theorem statement we are actually assuming A produces a valid proof for $(\phi, x, \mathsf{params}_\phi(\mathsf{secrets}))$ with probability δ, for a non-negligible fraction of $g \in G^*$.

1. There does not exist a vector c as described above.

2. Such c exists but is not a valid QAP witness for (ϕ, x)

Define the polynomial

$$R(X) := \rho'_A X^{d+1} \cdot A(X) + \rho'_B X^{2(d+1)} \cdot B(X) + \rho'_A \rho'_B X^{3(d+1)} \cdot C(X).$$

These two cases are equivalent, respectively, to the following two.

1. $R(X)$ is not in the subspace V.

2. $P(X) := A(X) \cdot B(X) - C(X)$ is not a multiple of $Z(X)$.

Suppose that $R(X)$ is not in V. We show that in this case B can efficiently compute $s^{q+1} \cdot g_1$: It follows from Lemma 10 of [GGPR13] that except with probability $1/r$ (over s and the inner randomness of $\mathsf{params}^{\mathsf{pin}}$), $R'(X) := X \cdot f(X) \cdot R(X)$ has a non-zero coefficient at X^{q+1}. But note that as π was valid,

$$\pi_K = \beta \cdot (\rho_A \cdot A(s) + \rho_B \cdot B(s) + \rho_A \rho_B \cdot C(s)) \cdot g_1 = R'(s) \cdot g_1.$$

B can thus use $\mathsf{challenge}$ to erase all elements of $R'(s) \cdot g_1$ of powers different than $q+1$, obtaining $s^{q+1} \cdot g_1$.

Now suppose that $P(X)$ is not a multiple of $Z(X)$. The same argument as in [PHGR16] can now be used to obtain $e(g_1, g_2)^{\frac{1}{s+t}}$, for some $t \in \mathbb{F}_r^*$. □

We say an adversary A defeats the protocol, if it is able to produce a set of parameters for which it can construct a valid proof, when COMMIT is replaced by a random oracle. More formally,

Definition 3.7. *Let B be an adversary controlling $n-1$ out of n players in our protocol. We assume B is deterministic (as we can fix its randomness to maximize its success probability as defined next). Let $P \in \{P_1, \ldots, P_n\}$ be the player not controlled by B. We say B δ-*defeats *the protocol on (ϕ, x) if with probability at least δ over the randomness of P in the protocol, and over the choices of a random oracle choosing the output values of COMMIT in the protocol description, the protocol verifier accepts the transcript, and B afterwards produces a proof π such that $\mathsf{V}(\phi, x, \pi, \mathsf{params}_\phi^B) = \mathsf{acc}$, where params_ϕ^B are the Pinocchio parameters for instance ϕ that were generated by the protocol.*

The correctness of the protocol lies in the following theorem.

Theorem 3.8. *For any efficient B controlling some subset of $n-1$ out of n players, there is an efficient A such that the following holds. Fix any instance ϕ and input x. Suppose that B δ-defeats the protocol on (ϕ, x). Then A $\frac{\delta}{\mathrm{poly}(\log r)}$-defeats Pinocchio on (ϕ, x). In particular, if B's success probability is non-negligible, so is A's success probability.*

Before proving the theorem note that an immediate corollary of Theorems 3.6 and 3.8 is

Corollary 3.9. *[Knowledge Soundness of Protocol] For any efficient B controlling some subset of $n-1$ out of n players, there is an efficient extractor E such that the following holds. Fix any instance ϕ and input x. Suppose that B δ-defeats the protocol on (ϕ, x). Then E produces a witness ω satisfying (ϕ, x) with probability $\frac{\delta}{\mathrm{poly}(\log r)}$ over $\mathsf{secrets} \in (\mathbb{F}_r^*)^8$, when given $(\phi, x, \mathsf{params}_\phi(\mathsf{secrets}))$ as input.*

We proceed to prove Theorem 3.8.

Proof of Theorem 3.8

For ease of notation, we assume that B controls $P_2, \ldots, P_n$. Also, we describe a non-uniform algorithm making choices depending on (ϕ, x). However, inspection shows that making all these choices uniformly (and independently of (ϕ,x)) succeeds with probability $\delta/\text{poly}\log r$. We denote by $\mathcal{R}$ the random oracle that chooses the values of COMMIT in the protocol. We assume the range (i.e. domain of replies) of $\mathcal{R}$ is of size $\mathsf{M} = \text{poly}(r)$ which is exponential in our prespective. We will denote by π the purported proof and params_ϕ^B the Pinocchio parameters generated in the protocol when B is participating. (These are randomized functions of various elements as will be discussed below). We will say params_ϕ^B and π are *accepted by* V, if $\mathsf{V}(\phi, x, \mathsf{params}_\phi^B, \pi) = \mathsf{acc}$. It will be convenient to view the protocol as divided into two main phases.

1. The *commit and prove phase* which consists of Round 1 and Parts $1-3$ of Round 2, i.e., all parts of Round 2 except the random powers subprotocol.

2. the *compute parameters phase* which consists of the random powers subprotocol in Round 2, together with Rounds 3 and 4.

Let $\mathsf{e} = \{\mathsf{e}_i\}_{i\in[n]}$ be the set of elements broadcast in Round 2 Part 1 during some execution of the protocol. Inspection of the protocol shows that

1. If all players follow the protocol, the transcript of the compute parameters phase is a deterministic function $\mathsf{comp\text{-}transcript}(\mathsf{e})$ of e.

2. Using the correctness of the $\mathsf{sameRatio}(,)$ method, given the value of e in the commit and prove phase, if one of the players writes a message that does not coincide with $\mathsf{comp\text{-}transcript}(\mathsf{e})$ in the compute parameters phase, the transcript will be accepted by the protocol verifier with probability at most $2/r$. Thus, we assume that after the commit and prove phase, B follows the protocol correctly in the compute parameters phase; as otherwise we may replace him by another adversary B' that does so, and $\delta - \mathsf{negl}(\log r)$-defeats the protocol.

3. In Round 2 Part 3, whenever B broadcasts an element R in one of the nizks, if he has not queried $\mathcal{R}(R \circ h)$ where h will be the appropriate element $h_{s,j}$, the transcript will be accepted with probability at most $1/\mathsf{M} = \mathsf{negl}(\log r)$. Thus we can assume B always makes these queries, as otherwise he may be replaced with an B' that does and $\delta - \mathsf{negl}(\log r)$ defeats the protocol.

4. Similarly, if B has not queried $\mathcal{R}(\mathsf{e}_i)$ before broadcasting h_i, and has not indeed broadcasted $\mathcal{R}(\mathsf{e}_i)$ as h_i, where e_i is what he will broadcast in Round 2 Part 1 his success probability is negligible, and we can assume this is not the case.

Note that P_1, acting honestly makes 10 calls to $\mathcal{R}$- one in the first round to compute his message h_1, and 9 to compute the elements $h, \{h_{1,s}\}$ in Round 2 Part 3. Let us assume B makes exactly Q' queries to $\mathcal{R}$ during Round one and two, and let $Q := Q' + 10$ be the total number of queries made to $\mathcal{R}$. Note that $Q = \text{poly}(\log r)$ since the size of B is $\text{poly}(\log r)$. Denote the answers of $\mathcal{R}$ by $C := \{c_1, \ldots, c_Q\}$. Denote the queries by $q_1, \ldots, q_Q$. Assume $q_1, \ldots, q_{Q/2}$ are exactly the ones made before the broadcast of e_1 in Round 2. Denote by M the set of messages of P_1 in the Round 2 part of the commit and prove phase, that are not outputs of $\mathcal{R}$- so M consists of the values e_i and the outputs $\{\pi_{1,s}\}$ of the nizks from Round 2 Part 3.

Under this assumption, together with items 2 and 3 above, the whole protocol transcript, and in particular the Pinocchio parameters params_ϕ^B and the purported proof π generated by B, are a deterministic function F of (C, M); i.e., we can denote $(\mathsf{params}_\phi^B, \pi) = F(C, M)$. From the fact that B δ-defeats the protocol, we know that there is a set of density δ of sequences (C, M) that cause B to produce a valid proof π when V uses params_ϕ^B. We can thus, using an averaging argument, fix a set of values S for (C, M) such that

1. S has probability mass $\delta' = \delta - \mathsf{negl}(\log r)$ in the space of all possible values for (C, M), when C is random, and M is distributed according to an honest player P_1's messages.

2. C contains all distinct elements.

3. For any $(C, M) \in S$, $F(C, M)$ is accepted by V.

Denote $E := \cup_{i \in [2..n]} \mathsf{secrets}_i$, and enumerate the elements of E somehow as $E_1, \ldots, E_\ell$, where $\ell := 8 \cdot (n-1)$. Note that the elements of E are determined (although not efficiently) by the protocol transcript and thus by (C, M). Note that in Round 2, B has to present Schnorr nizk proofs for all elements of E, each requiring a query to $\mathcal{R}$ on the corresponding element $R \circ h$. By another averaging argument similar to the "forking lemma", there exists a permutation σ on $[\ell]$ and indices $1 \leq i_1 < \ldots < i_\ell \leq Q$ such that there is a subset of $T \subset S$ of density (meaning probability mass) $\delta'' := \delta'/Q^\ell$ (in the space of all values for $(C$,$M)$) such that for each $(C, M) \in T$

1. $F(C, M)$ is accepted by V.

2. When executing the protocol with (C, M), for each $j \in [\ell]$, B uses the value c_{i_j} for the challenge c in the nizk of $E_{\sigma(j)}$.

We assume from now on that σ is the identity for simplicity of notation. Yet another averaging argument, using the non-negligibility of δ'', allows us to construct a string C^* such that

1. $(C^*, M) \in T$ with probability δ'' over M.

2. For any $j \in [\ell]$, there are strings $(C', M'), (C'', M') \in T$, i.e., that agree on the M' part, such that C' and C'' agree with C^* on first $i_j - 1$ indices, but disagree with each other on the i_j'th coordinate.

We now note an important point:

Fix some $j \in [\ell]$. Then any string $(C', M') \in T$ in which C' agrees with C^* on indices $1, \ldots, i_j - 1$, must lead to the same value of E_j. This is because conditioned on being in T, the query q_{i_j} to $\mathcal{R}$ will contain $\mathsf{rp}^1_{E_j}$- which uniquely determines E_j, and the value of q_{i_j} is a deterministic function of $c_1, \ldots, c_{i_j - 1}$, when $i_j \leq Q/2$, and otherwise using item 4 E_j is determined by $c_1, \ldots, c_{Q/2}$. The second property above about the elements $(C', M'), (C'', M') \in T$ implies we have access to two valid Schnorr nizks for E_j with the same R but different challenges c, which implies using the well-known Schnorr extractability property that A can extract the value E_j used by B given any $(C^*, M) \in T$.

Now let T' be the subset of T consisting of elements beginning with C^*

Now let $\mathsf{params}_\phi = \mathsf{params}_\phi(\mathsf{secrets})$ be the parameters given as a challenge to A (for which he should construct an accepting proof). Let $\mathsf{secrets}_1 := \mathsf{secrets}/(\mathsf{secrets}_2 \cdots \mathsf{secrets}_n)$ be the coordinate wise division of the corresponding secrets vectors. Recall $E = \mathsf{secrets}_2 \cup \ldots \cup \mathsf{secrets}_n$ is fixed conditioned on $(C, M) \in T'$, and A has extracted the values E. Inspection of the protocol shows that A can efficiently play the role of P_1 when he chooses this value of $\mathsf{secrets}_1$, just from knowing E and $\mathsf{params}_\phi(\mathsf{secrets})$, *with one potential exception:* Constructing valid nizks of the elements of $\mathsf{secrets}_1$ *which he does not know*. However, he can do this also using his ability to program $\mathcal{R}$. He will choose random $\mathbb{F}_r$ elements as the answers u in the nizks and then compute $R := c \cdot H - u \cdot f$. and set $\mathcal{R}(R) = c$, unless $\mathcal{R}(R)$ has been queried by B in which case he aborts. He then conducts the protocol with B using the values (C^*, M), and outputs the proof π generated by B. Note that when following this strategy $\mathsf{secrets}_1$ and all the corresponding R's in the nizks are uniformly distributed. Thus the probability that $(C^*, M) \in T'$ when M is derived from the R's and $\mathsf{secrets}_1$ is at least δ''. Thus the probability that $F(C^*, M)$ will satisfy V when using this strategy is at least δ'' which is non-negligible when δ is non-negligible. $\square$

3.3 Zero-Knowledge

Campanelli, Gennaro, Goldfeder and Nizzardo [CGGN17] have recently noted that a mallicious choice of params_ϕ can potentially break the zero-knowledge guarantee of the Pinocchio protocol. We prove that statistical zero-knowledge holds when using the parameters generated by our protocol, *even in the case that all n players are mallicous and colluding*, provided the prover verifies the protocol transcript before sending her proof. Interestingly, this means the protocol is useful also when[6] *run by one player*; as the transcript will provide proof to the prover that sending her proof will not leak additional information.

Denote by P the (honest) prover of the Pinocchio protocol. We can think of P as a randomized function $\mathrm{P}(\phi, x, \mathsf{params}_\phi, \omega)$ generating a proof according to the instance, input, witness and public parameters. (See Appendix B of [BCTV14] for a full description of P.) We slightly alter P and think of it as an algorithm $\mathrm{P}(\phi, x, \mathsf{transcript}, \omega)$ that also receives a protocol transcript $\mathsf{transcript}$, and first applies the protocol verifier on $\mathsf{transcript}$. If the protocol verifier rejects, P outputs rej and nothing else. If the protocol verifier accepts, P derives the parameters params_ϕ from the transcript, and outputs $\mathrm{P}(\phi, x, \mathsf{params}_\phi, \omega)$ for the original prover P.

Theorem 3.10. *[Statistical Zero-Knowledge] For any polynomial P, there is an efficient sim such that the following holds. Fix any efficient B controlling all n players in the protocol, instance ϕ, input x and witness ω. Let $\mathcal{D}$ be the distribution obtained by outputting the protocol transcript $\mathsf{transcript}(B)$ concatenated with the proof $\mathrm{P}(\phi, x, \mathsf{transcript}(B), \omega)$ (over the randomness of B in the protocol, the randomness of the oracle $\mathcal{R}$, and that of P in generating the proof). Then the distribution $\mathcal{D}_{\mathrm{sim}}$ of the output of sim^B has distance at most $1/P(\log r)$ to $\mathcal{D}$.*

Proof. Note that if we can simulate $\mathcal{D}$ with error $1/P(\log r)$ conditioned on every fixing of the randomness of B, we can simulate the fully randomized B with the same error. So the simulator begins by choosing the randomness of B uniformly, and we can assume we are working with a deterministic B. Assume B always makes exactly Q queries to the random oracle $\mathcal{R}$ emulating COMMIT (if this is just an upper bound we can think of B not reading the last answers sometimes). As before, we denote by M the range size of $\mathcal{R}$ and assume $r \le \mathsf{M} \le \mathrm{poly}(r)$. The transcript output $\mathsf{transcript}(B)$ is a deterministic function of a sequence r of answers of $\mathcal{R}$ to queries of B. We denote this transcript $\mathsf{transcript}(\mathsf{r})$. The corresponding set of queries $\{q_1, \ldots, q_Q\}$ of B to $\mathcal{R}$ is also a deterministic function of r. We denote by $\mathcal{D}_\mathsf{r}$ the distribution of $\mathrm{P}(\phi, x, \mathsf{transcript}(\mathsf{r}), \omega)$.

[6]Thank to Eran Tromer for pointing this out.

We will show that for a $(1 - 1/P(\log r))$-fraction of the sequences r, sim can simulate $\mathcal{D}_{\mathsf{r}}$ with error $\mathsf{negl}(\log r)$ given r. Clearly this suffices - as sim can choose a random r, and then try to simulate $\mathcal{D}_{\mathsf{r}}$ using this method.

sim begins by sampling a random sequence r and characterizing it as either good, negligible or bad. This characterization will have the property that a bad sequence r will have $\mathcal{D}_{\mathsf{r}}$ be rej with probability $1 - \mathsf{negl}(\log r)$, and the negligible r together will have probability $\mathsf{negl}(\log r)$.

Characeterizing r: If r has repetitions it is labeled negligible - because since $Q = \mathrm{poly}\log r$, such sequences are a $\mathsf{negl}(\log r)$ fraction. Otherwise sim executes $B(\mathsf{r})$. If B aborts before producing a full transcript, r is labeled bad. In particular, sim has checked if the values $\{\mathsf{e}_i\}_{i\in[n]}$ and nizk proofs $\{\pi_{i,s}\}_{i\in[n],s\in\mathsf{secrets}_i}$ are present in the Round 2 part of the transcript and otherwise labeled r bad.

It then checks if the queries $\{\mathcal{R}(\mathsf{e}_i)\}_{i\in[n]}$,$h := \mathcal{R}(\mathcal{R}(\mathsf{e}_1)\circ\ldots\circ\mathcal{R}(\mathsf{e}_n))$,$\{\mathcal{R}(R_S \circ h \circ \mathsf{rp}_s^1)\}_{i\in[n],s\in\mathsf{secrets}_i}$ were made (recall that B's sequence of queries is a determinstic function of r that can be efficiently derived given blackbox access to B); here R_s is the first part of $\pi_{i,s}$. If not all queries were made, r is labeled as bad - as the transcript is correct only if

- the values $\{h_i\}$ in the transcript are equal to $\{\mathcal{R}(\mathsf{e}_i)\}$,
- the nizk proofs $\pi_{i,s}$ are verified with challenges $\{c_{i,s}(\mathsf{r}) := \mathcal{R}(R_S \circ h \circ \mathsf{rp}_s^1)\}$, which in turn determine a unique uniformly distributed correct $\pi_{i,s}$,

and this requires guessing in advance at least one output of $\mathcal{R}$, that can succeed only with probability $1/\mathsf{M}$.

If all queries were made, sim derives the sequence of indices $I = I(\mathsf{r})$ of the queries $\mathcal{R}(\mathsf{e}_1), \ldots, \mathcal{R}(\mathsf{e}_n)$, and a set of indices $J = J(\mathsf{r})$ where the queries $\{c_{i,s}\}$ were made. Note that in this case the transcript validity is a deterministic function of $\mathsf{transcript}(\mathsf{r})$ and r; as r contains all queries to $\mathcal{R}$ of the protocol verifier. sim checks if the protocol verifier accepts and if not labels r bad. sim checks if the query to h was made after all queries in I, if not r is labeled negligible as the transcript will only be valid if for some $i \in [n]$ h_i corresponds to the later determined $\mathcal{R}(\mathsf{e}_i)$. sim then checks if $I < J$ in the sense that all elements of I are smaller than all elements of J. If this is not the case, r is labeled negligible - as in this case the transcript is valid only if for some $i \in [n]$ the value h in the queries $\mathcal{R}(h \circ \mathsf{rp}_s^1)$ equals the later determined $\mathcal{R}(\mathsf{e}_1 \circ \ldots \circ \mathsf{e}_n)$. If r has been labeled so far sim outputs rej. If sim has not been labeled so far (as bad or neutral), it is labeled good. In this case we can define the vector $\mathsf{secrets}(\mathsf{r})$, such that the parameters derived from $\mathsf{transcript}(\mathsf{r})$ are $\mathsf{params}_\phi(\mathsf{secrets})$, which is the vector of secrets uniquely determined by the product of secrets of each player which in turn are uniquely determined by the elements $\mathsf{e}_1, \ldots, \mathsf{e}_n$ in $\mathsf{transcript}(\mathsf{r})$.

The proof now follows from two claims. The first tells us that for a good r, $\mathcal{D}_{\mathsf{r}}$ can be simulated given $\mathsf{secrets}(\mathsf{r})$. The second tells us that for a $(1 - 1/P(\log r))$-fraction of good r, we can obtain $\mathsf{secrets}(\mathsf{r})$ with probability $1 - 1/\mathsf{M}$.

Claim 3.11. *For any polynomial P, there is an algorithm* alg *that for a $(1 - 1/P(\log r))$-fraction of good* r*, obtains* $\mathsf{secrets}(\mathsf{r})$ *given* r *with probability* $1 - 1/\mathsf{M}$ *in time* $\mathrm{poly}\log r$.

Proof. For $I \subset [Q], |I| = n, J \subset [Q], |J| = 8 \cdot n, x \in [\mathsf{M}]^{Q-8n}$, denote by $W_{x,I,J}$ the set of $\mathsf{r} \in [\mathsf{M}]^Q$ that are good with $I(\mathsf{r}) = I, J(\mathsf{r}) = J$, and $\mathsf{r}|_{[Q]\setminus J} = x$. For $J \subset [Q], |J| = Q - 8n, x \in [\mathsf{M}]^{Q-8n}$ denote by $W|_{\bar{J}=x}$ the set of $\mathsf{r} \in [\mathsf{M}]^Q$ with $\mathsf{r}|_{[Q]\setminus J} = x$.

Observe that for good $\mathsf{r}, \mathsf{r}' \in W_{x,I,J}$ we have $\mathsf{secrets}(\mathsf{r}) = \mathsf{secrets}(\mathsf{r}')$ - as the values up to the indices of J are identical in r and r', and thus the queries q_i for indicies $i \in I$ are identical in r and r' - and these queries determine the elements $\mathsf{e}_1, \ldots, \mathsf{e}_n$ appearing in the transcript which in turn determine $\mathsf{secrets}$.

Fix good r, and let $I = I(\mathsf{r}), J = J(\mathsf{r}), x = \mathsf{r}|_{[Q]\setminus J}$. Denote $P' := PQ^{9n}$. alg samples $T := 2P'(\log r) \cdot \log r$ sequences $\mathsf{r}' \in W|_{\bar{J}=x}$. For each such r', she checks if $\mathsf{r}' \in W_{x,I,J}$, and if so, whether $c_{i,s}(\mathsf{r}) \neq c_{i,s}(\mathsf{r}')$ for each $i \in [n], s \in \mathsf{secrets}$. If this is the case, alg can extract $\mathsf{secrets}_i$ for each $i \in [n]$; as Now the question is what is the probability of not finding an appropriate r' among the samples. We claim this probability is at most $1/r$, when the density of $W_{x,I,J}$ in $W|_{\bar{J}=x}$ is at least $1/P'$: The density of elements of $W_{x,I,J}$ such that $c_{i,s}(\mathsf{r}') \neq c_{i,s}$ for each $i \in [n], s \in \mathsf{secrets}$ is at least $1/P' - Q/\mathsf{M} \geq 1/2P'(\log r)$ for large enough r. Thus, the probability that no such element r' is found is at most $(1 - /2P'(\log r))^{2P'(\log r)\cdot \log r} \leq 1/r$.

As for given x, there are at most Q^{9n} sets $W_{x,I,J}$ (going over possibilites for I and J), it follows that the set of good $\mathsf{r} \in W|_{\bar{J}=x}$ belonging to a set $W_{x,I,J}$ of density smaller than $1/P'(\log r)$ is at most $1/P(\log r)$. Thus, for a $(1 - 1/P(\log r))$-fraction of good r sim simulates $\mathcal{D}_{\mathsf{r}}$ with error $1/r$. □

Claim 3.12. *Suppose that* r *is good. Then given* $\mathsf{secrets}(\mathsf{r})$*,* $\mathcal{D}_{\mathsf{r}}$ *can be efficiently simulated with no error*

Proof. The proof is similar to that of Theorem 13 in [GGPR13]. Note that when r is good the elements $A_{m+1} = B_{m+2} = C_{m+3}$ in $\mathsf{params}_\phi(\mathsf{secrets}(\mathsf{r}))$ are non-zero, as this is required for the transcript to be accepted.

We abuse notation and denote the discrete logs of the proof elements by themselves. As $A_{m+1}, B_{m+2}, C_{m+3}$ are non-zero, π_A, π_B, π_C are uniformly distributed in $\mathbb{F}_r$ since $c_{m+1}, c_{m+2}, c_{m+3}$ are chosen uniformly by P. Now, in a valid proof the other proof elements $\pi'_A, \pi'_B, \pi'_C, \pi_K, \pi_H$ are deterministic functions of π_A, π_B, π_C, derived from the verification constraints (see Appendix B). Moreover, given $\mathsf{secrets}$ and the discrete logs of π_A, π_B, π_C, they can be efficiently derived: Defining A_{io} as in the proof of Theorem 3.6, we have

1. $\pi'_A = \alpha_A \cdot \pi_A$
2. $\pi'_B = \alpha_B \cdot \pi_B$
3. $\pi'_C = \alpha_C \cdot \pi_C$
4. $\pi_H = ((A_{io}(s) + \pi_A) \cdot \pi_B - \pi_C)/Z(s)$
5. $\pi_K = \beta \cdot ((A_{io}(s) + \pi_A) + \pi_B + \pi_C)$

Thus, given the value of $\mathsf{secrets}(\mathsf{r})$, sim can efficiently simulate the distribution $\mathcal{D}_{\mathsf{r}}$. □

□

Acknowledgements

We thank Eli Ben-Sasson, Alessandro Chiesa, Jens Groth, Daira Hopwood, Hovav Shacham, Eran Tromer, Madars Virza, Nathan Wilcox and Zooko Wilcox for helpful discussions. We thank Daira Hopwood for pointing out some technical inaccuracies. We thank Eran Tromer for bringing to our attention the work of [CGGN17], and the relevance of our protocol to that work.

References

[BCG+15] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 287–304, 2015.

[BCPR14] N. Bitansky, R. Canetti, O. Paneth, and A. Rosen. On the existence of extractable one-way functions. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 505–514, 2014.

[BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 781–796, 2014.

[CFH+15] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 253–270, 2015.

[CGGN17] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. *Commun. ACM*, 2017.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 626–645, 2013.

[PHGR16] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: nearly practical verifiable computation. *Commun. ACM*, 59(2):103–112, 2016.

[Sch89] C. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 239–252, 1989.

A Actual, more complicated version of e_i in code

For each $i \in [n]$, P_i first constructs $\mathsf{elements}_i$ as in Subsection 2.1. That is

1. As described in Generate a set of uniform elements in $\mathbb{F}_r^*$

$$\mathsf{secrets}_i := \{\tau_i, \rho_{A,i}, \rho_{B,i}, \alpha_{A,i}, \alpha_{B,i}, \alpha_{C,i}, \beta_i, \gamma_i\}$$

, and let

$$\mathsf{elements}_i := \{\tau_i, \rho_{A,i}, \rho_{B,i}, \alpha_{A,i}, \alpha_{B,i}, \alpha_{C,i}, \beta_i, \gamma_i, \rho_{A,i}\alpha_{A,i}, \rho_{B,i}\alpha_{B,i},$$
$$\rho_{A,i}\rho_{B,i}, \rho_{A,i}\rho_{B,i}\alpha_{C,i}, \beta_i\gamma_i\}$$

2. Now each player generates a somewhat complex set of group elements from its set of secrets. We omit the index i for clarity of notation, but it should be appended to all elements below:

(a) P_i chooses random elements $f_1, f_2, f_3 \in \mathbb{G}_2 \setminus \{0\}$ and random elements $f_4, f_5, f_6, f_7, f_8 \in \mathbb{G}_1 \setminus \{0\}$.

(b) P_i stores the sets of $\mathbb{G}_2$ elements

$$\mathsf{e}_i^2 := \{f_1, f_1 \cdot \rho_A, f_1 \cdot \rho_A\alpha_A, f_1 \cdot \rho_A\rho_B\alpha_C, f_1 \cdot \rho_A\rho_B, f_1 \cdot \rho_A\rho_B\alpha_B,$$

$$f_2, f_2 \cdot \beta, f_2 \cdot \beta\gamma, f_3, f_3 \cdot \tau\},$$

and the set of $\mathbb{G}_1$ elements

$$\mathsf{e}_i^1 := \{f_4, f_4 \cdot \alpha_A, f_5, f_5 \cdot \alpha_C, f_6, f_6 \cdot \rho_B, f_7, f_7 \cdot \rho_A, f_8, f_8 \cdot \gamma\}.$$

3. Finally, define $\mathsf{e}_i := \mathsf{e}_i^1 \circ \mathsf{e}_i^2$.

Validity of more complicated definition The only requirement for e_i to run the protocol is that we always have the needed s-pair for each $s \in \mathsf{elements}_i$. Inspection shows this is the case with the definiton here. For example, P_i has broadcasted (ommitting index i) $f_1, f_1\rho_A$ and $f_1\rho_A\rho_B$ as part of e_i^2, out of which we can construct the ρ_A-pair $(f_1, f_1\rho_A)$, the ρ_B-pair $(f_1\rho_A, f_1\rho_A\rho_B)$, and the $\rho_A\rho_B$-pair $(f_1, f_1\rho_A\rho_B)$. Since here we don't have a $\mathbb{G}_1$-s-pair and $\mathbb{G}_2$-s-pair for each $s \in \mathsf{elements}_i$, less consistency checks are performed in Part 2 of Round 2. (Note that to run the protocol we don't need an s-pair for each $s \in \mathsf{elements}_i$ in each group. For example, we only need a $\mathbb{G}_2$-ρ_A-pair which is used in Round 3 to compute PK_A, but we never use a $\mathbb{G}_1$-ρ_A-pair).

B Implementation details

In our implementation we use the libsnark `alt_bn128` curve implementation where

1. $\mathbb{G}_1 = E/\mathbb{F}_q$ is a BN curve over $\mathbb{F}_q$. That is the set of solutions in $\mathbb{F}_q^2$ of an equation of the form $y^2 = x^3 + b$.
2. $\mathbb{G}_2 = E'/\mathbb{F}_{q^2}$ is a subgroup of order r of a sextic twist of $\mathbb{G}_1$. Where a sextic twist of $\mathbb{G}_1$ means the set of solutions in $\mathbb{F}_{q^2}$ of $y^2 = x^3 + b/\xi$, where $\xi \in \mathbb{F}_{q^2} \setminus \mathbb{F}_q$ is an element such that the polynomial $W^6 - \xi$ is irreducible over $\mathbb{F}_{q^2}$.
3. $\mathbb{G}_t$ is a subgroup of order r in $\mathbb{F}_{q^{12}}$.

C Pinocchio reminder

We give a brief reminder of how the proof and verification procedure of [PHGR16] look like using the notation of [BCG+15]. The prover has in his hand a QAP solution $(c_0 = 1, c_1, \ldots, c_m)$ that coincides with the public input $x = (c_1, \ldots, c_n)$ and satisfies the following. If we define $A := \sum_{i=0}^{m} c_i \cdot A_i, B := \sum_{i=0}^{m} c_i \cdot B_i$, and $C := \sum_{i=0}^{m} c_i \cdot C_i$; then the polynomial $P := A \cdot B - C$ will be divisble by the target polynomial Z. Given $\mathsf{params}_\phi(\mathsf{secrets})$, V will compute

1. $\pi_A := \rho_A A(s) \cdot g_1$, $\pi'_A := \alpha_A \rho_A A(s) \cdot g_2$.
2. $\pi_B := \rho_B B(s) \cdot g_2$, $\pi'_B := \alpha_B \rho_B B(s) \cdot g_1$.

3. $\pi_C := \rho_A\rho_B C(s)\cdot g_1$, $\pi'_C := \alpha_C\rho_A\rho_B C(s)\cdot g_2$.
4. $\pi_K := \beta(\rho_A A(s) + \rho_B B(s) + \rho_A\rho_B C(s))\cdot g_1$.
5. $\pi_H := (P(s)/Z(s))\cdot g_1$.

Now, abuse notation and denote the discrete log of each element above in base g_1 or g_2 by the element itself. The verifier, using pairings and the verification key V, will check the following.

1. $\pi'_A = \alpha_A\pi_A$.
2. $\pi'_B = \alpha_B\pi_B$.
3. $\pi'_C = \alpha_C\pi_C$.
4. $\pi_K = \beta(\pi_A + \pi_B + \pi_C)$.
5. $\pi_A\cdot\pi_B - \pi_C = \pi_H\cdot Z(s)\rho_A\rho_B$.