Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model

Sean Bowe
sean@z.cash

Ariel Gabizon
ariel@z.cash

Ian Miers
imiers@cs.jhu.edu

October 26, 2017

Abstract

Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) have emerged as a valuable tool for verifiable computation and privacy preserving protocols. Currently practical schemes require a common reference string (CRS) to be constructed in a one-time setup for each statement. Ben-Sasson, Chiesa, Green, Tromer and Virza [5] devised a multi-party protocol to securely compute such a CRS, and an adaptation of this protocol was used to construct the CRS for the Zcash cryptocurrency [7]. The scalability of these protocols is obstructed by the need for a "precommitment round" which forces participants to be defined in advance and requires them to secure their secret randomness throughout the duration of the protocol.

Our primary contribution is a more scalable multi-party computation (MPC) protocol, secure in the *random beacon model*, which omits the precommitment round. We show that security holds even if an adversary has limited influence on the beacon. Next, we apply our main result to obtain a two-round protocol for computing an extended version of the CRS of Groth's SNARK [11]. We show that knowledge soundness is maintained in the generic group model when using this CRS.

We also contribute a more secure pairing-friendly elliptic curve construction and implementation, tuned for use in zk-SNARKs, in light of recent optimizations [13] to the Number Field Sieve algorithm which reduced the security estimates of existing pairing-friendly curves used in zk-SNARK applications.

1 Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs)[12, 15] have seen increased usage both in the literature and the real world, ranging from publicly verifiable computation, to deployed usage for anonymous payment systems such as Zcash [18] and smart contract systems such as Ethereum.[1]

[1] As of the Byzantium hard fork in mid October 2017, Ethereum now supports zkSNARK verification [17]

Despite the power of zk-SNARKs, numerous challenges stand in the way of their widespread use. Software implementations are immature and slow and the concrete security of deployed schemes is lower than originally intended. Most importantly, these schemes are secure in the common reference string (CRS) model, which assumes a trusted setup of parameters used for constructing and verifying proofs. The generation of this CRS is a major challenge, given that corruption or subversion of the parameters can degrade the security guarantees of the system. For example, in Zcash, compromising the CRS generation process allows an attacker to forge transactions and counterfeit millions of dollars. Consequently CRS generation is a major challenge for real world usage of zkSNARKs and similar proof systems: with potentially billions of dollars at stake, "trust me" is not a viable approach.

The current approach for production systems is for the CRS to be generated via a multi-party computation [5, 7]. These protocols guarantee soundness when at least one participant is honest, and guarantee zero-knowledge even if none of the participants are honest.[9] However, these protocols cannot scale beyond a handful of participants, and can even be too expensive to perform for just one or two participants. This is not an engineering and optimization issue, fundamentally it is a cryptographic problem: because of restrictions required to deal with adaptive attackers, participants must commit to their share of the parameters up front and maintain availability and security throughout the entire duration of the protocol even *after* the majority of their individual computation is completed.

These restrictions mean that participants must be selected in advance, and that they must remain online throughout the entire duration of the ceremony. This not only increases the surface area of attacks, but also raises practical problems as participants are required to maintain custody of the hardware during the entire ceremony. Participants also cannot abort,[2] and so attackers can trivially disrupt the ceremony. The net result of these cryptographic limitations is that the participants must be carefully preselected in advance and very limited in number. This moves the setting for SNARKs from "trust me", to "trust us" for a very small group. Particularly for "trustless" blockchains, this is a major limitation.

1.1 Our results

In this paper, we aim to make zk-SNARKs suitable for wide-scale usage by providing a new zk-SNARK scheme and MPC system for CRS generation suitable for real world usage. We offer three contributions:

[2]More precisely, the security analysis in [5, 7] assumes no aborts. It seems plausible it could be extended to deal with at most t aborts where $\binom{N}{t} < 2^\lambda$ where λ is the "security parameter" and N the total number of players, by a simulator that guesses who will abort; however it does not seem straightforward to write such a proof as the adversary may be adaptive in their choice of the aborting subset, according to the message of a simulator emulating the preceding honest player. Moreover, it is of practical interest to allow more than λ participants and arbitrary aborts. Concretely, $\lambda \sim 256$ in our implementation and we would like to have more than 256 participants.

Player-exchangeable MPC Our primary contribution is a new kind of multi-party computation protocol, a *player-exchangeable MPC* (px-MPC) and an efficient and implemented px-MPC protocol for CRS generation.

A px-MPC is described by a sequence of messages players are supposed to send; however, importantly, there is no restriction on the identity of the sender of each message. In particular, although we will discuss multi-round protocols, there is no need to assume the same players participate in different rounds. Since there is no private state between messages, players may be swapped out or removed after every message.

Player exchangeability avoids the issues of pre-selection of participants, the need to select reliable participants who do not abort, and the need for participants to maintain custody of sensitive hardware for extended periods of time. The only requirement is that at least one of the participants in each round honestly follows the protocol and does not collude with other players. As a result, the protocol can scale to a practically unbounded[3] number of participants and do so dynamically during protocol execution.

The key to this new approach is the use of random beacons to support a proof of security which places fewer restrictions on the protocol. We prove security even if an adversary has limited influence on this beacon.

zk-SNARKs with an efficient and amortized px-MPC CRS generation process As a second contribution, we offer a concrete SNARK to use with this approach. To realize this scheme in practice, we must pick a specific SNARK and provide a protocol for generating its CRS. Groth's SNARK [11] is the current state of the art protocol . We prove the security of Groth's SNARK with an extended CRS which allows for a two round px-MPC protocol. More significantly, the first round is agnostic to the statement[4], and so can be performed once for all statements up to some large (but bounded) size. In our implementation, for a circuit size up to 2^{21} multiplication gates, participants in the first round must receive a 1.2GB file, perform a computation that lasts about 13 minutes on a desktop machine, and produce a 600MB file. The second round is statement-specific, but significantly cheaper. This allows the bulk of the cost of setup to amortized over many circuits.

A new efficient curve for zk-SNARKS. In order to implement this protocol we must pick an elliptic curve to use. Existing SNARK implementations, such as those used in Zcash and Ethereum, use a pairing-friendly elliptic curve designed to be efficient for zk-SNARKs [6] which originally targeted the 128-bit security level. However, recent optimizations to the Number Field Sieve algorithm [13] have degraded

[3]Formally, as seen in Theorem 4.1, the number of participants can be any polynomial in the security parameter λ, when assuming efficient attacks on our curve have success probability $\mathrm{negl}(\lambda)$.

[4]Up to statement size.

this security, and so we adopt a new pairing-friendly elliptic curve called *BLS12-381* which targets 128-bit security with minimal performance impact. We provide a stable implementation of this new elliptic curve, written in Rust, with competitive performance, well-defined serialization, and cross-platform support. [16].

1.2 Overview of the proof

We briefly sketch the main idea of Theorem 4.1 and how it differs from [5, 7]. Let's assume for simplicity our CRS consists only of the element $s \cdot g_1$, where g_1 is a generator of some additively written group $\mathbb{G}_1$; and s a uniform element in $\mathbb{F}_p^*$. Suppose we wish to generate such an element by a two party protocol where the first party Alice is honest, and the second, Bob, is malicious. A natural protocol would proceed as follows: Alice chooses a uniform $s_1 \in \mathbb{F}_p^*$, and sends $M = s_1 \cdot g_1$ to Bob. Now, Bob is requested to multiply M by a uniform $s_2 \in \mathbb{F}_p^*$. The protocol output is defined as $s_2 \cdot M = s_1 s_2 g_1$.

The problem is that as Bob is malicious he can adaptively choose a value $s_2 \in \mathbb{F}_p^*$, to manipulate the final output value $s_1 s_2 \cdot g_1$. For this reason, in [5, 7] a precommitment round is added, where both Alice and Bob commit to their values s_1, s_2. In the round after, Alice and Bob will run the natural protocol, but add a proof that they are using the values s_1, s_2 they committed to (the proofs will not expose the values s_1, s_2). This prevents Bob from choosing s_2 adaptively. However, the precommitment round has the following drawbacks.

1. Most obviously, it adds a round to the protocol.
2. The participating players need to be defined in advance.
3. The players need to choose their secret elements in advance and protect them for a while (at least until broadcasting their messages in all subsequent rounds).

The main observation in this paper, is that assuming a public source of randomness that no player has control over, i.e. a *random beacon*, we can omit the precommitment round.

With the random beacon, a simplified version of our protocol, when again, the first party is honest, and second malicious, will proceed as follows.

1. Alice chooses random $s_1 \in \mathbb{F}_p^*$ and broadcasts $M = s_1 \cdot g_1$.
2. Bob chooses (somehow) a value $s_2 \in \mathbb{F}_p^*$ and broadcasts $M' = s_1 s_2 \cdot g_1$.
3. The random beacon is invoked to obtain a uniform $s_3 \in \mathbb{F}_p^*$, and the protocol output is defined as $s_3 \cdot M' = s_1 s_2 s_3 \cdot g_1$.

Note that the protocol output is $s \cdot g_1$ for uniform $s \in \mathbb{F}_p^*$ regardless of Bob's choice of s_2. You may ask, why not skip both players and just output $s \cdot g_1$ with $s \in \mathbb{F}_p^*$ being the beacon's output? The point is that it is important no player, or more generally, no group of colluding players that precludes at least one player, will know s. This means we cannot use the *public* random beacon to select s, only to randomize the choice of s.

We stress that in the proof it is enough to assume the random beacon has low co-entropy; thus the protocol works in the case where the adversary has limited influence on the beacon.

We refer to Theorem 4.1 for precise details.

2 Preliminaries

We will be working over bilinear groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ each of prime order p, together with respective generators g_1, g_2 and g_T. These groups are equipped with a non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, with $e(g_1, g_2) = g_T$. We write $\mathbb{G}_1$ and $\mathbb{G}_2$ additively, and $\mathbb{G}_T$ multiplicatively. For $a \in \mathbb{F}_p$, we denote $[a]_1 := a \cdot g_1, [a]_2 := a \cdot g_2$. We use the notation $\mathbf{G} := \mathbb{G}_1 \times \mathbb{G}_2$ and $\mathbf{g} := (g_1, g_2)$. Given an element $h \in \mathbf{G}$, we denote by $h_1(h_2)$ the $\mathbb{G}_1(\mathbb{G}_2)$ element of h. We denote by $\mathbb{G}_1^*, \mathbb{G}_2^*$ the non-zero elements of $\mathbb{G}_1, \mathbb{G}_2$ and denote $\mathbf{G}^* := \mathbb{G}_1^* \times \mathbb{G}_2^*$.

We assume that we have a generator $\mathcal{G}$ that takes a parameter λ and returns the three groups above having prime order p at least super polynomial in λ, together with uniformly chosen generators $g_1 \in \mathbb{G}_1^*, g_2 \in \mathbb{G}_2^*$. We assume group operations in $\mathbb{G}_1$ and $\mathbb{G}_2$, and the map e can be computed in time $\mathrm{poly}(\lambda)$. When we say an event has probability γ, when mean it has probability γ over the randomness of $\mathcal{G}$, in addition to any other randomness explicitly present in the description of the event.

When we say a party $\mathcal{A}$ is *efficient*, we mean it is a non-uniform sequence of circuits, indexed by λ, of size $\mathrm{poly}(\lambda)$. When we say $\mathcal{A}$ is an efficient oracle circuit we mean it is efficient in the above sense, and during its execution may make $\mathrm{poly}(\lambda)$ queries to an oracle $\mathcal{R}$, taking as input strings of arbitrary length and outputting elements of $\mathbb{G}_2^*$.

We assume such parties $\mathcal{A}$ all have access to the same oracle $\mathcal{R}$ during the protocol, whose outputs are uniform independent elements of $\mathbb{G}_2^*$.

For $a \in \mathbb{F}_p$ and $C \in \mathbf{G}$, we denote by $a \cdot C$ the coordinate-wise scalar multiplication of C by a; that is, $a \cdot C := (a \cdot C_1, a \cdot C_2) \in \mathbf{G}$. We also allow coordinate-wise operations of vectors of the same length. For example, for $a \in \mathbb{F}_p^t$ and $\mathbf{x} \in \mathbb{G}_1^t$, $a \cdot \mathbf{x} := (a_1 \cdot \mathbf{x}_1, \ldots, a_t \cdot \mathbf{x}_t)$.

We think of acc and rej as true and false. Hence when we say "check that $f(x)$" for a function f and input x, we mean check that $f(x) = \mathsf{acc}$.

We use the acronym e.w.p. to mean "except with probability"; i.e., e.w.p. γ means "with probability at least $1 - \gamma$".

We assume a synchronous setting where we have positive integer "slots" of time; we assert that in slot J, parties know what messages were sent (and by whom) in slots $1, \ldots, J - 1$.

2.1 Random beacons

We assume we have at our disposal a "random beacon" RB that outputs elements in $\mathbb{F}_p^*$. We think of RB as a function receiving a time slot J, and positive integer k; and outputting k elements $a_1, \ldots, a_k \in \mathbb{F}_p^*$. It will be convenient to assume RB is defined only for a subset of values J as its first input. We say RB is *resistant to* $\mathcal{A}$, if for any positive integers J and k for which RB is defined: for any random variable X generated by $\mathcal{A}$ before time J - i.e. using calls to $\mathsf{RB}(J', k')$ for $J‘ < J$, and calls to the oracle $\mathcal{R}$ in case $\mathcal{A}$ is an oracle circuit and messages H of honest players following a protocol $\mathcal{A}$ is designed to participate in; the distribution of $\mathsf{RB}(J, k)$ is uniform in $(\mathbb{F}_p^*)^k$ and independent of $(\mathsf{rand}_{\mathcal{A}}, X)$, where $\mathsf{rand}_{\mathcal{A}}$ is $\mathcal{A}$'s randomness.

We now generalize this definition to model adversaries that have limited influence on the value of the beacon. We say RB is *u-co-resistant to $\mathcal{A}$*, if for any positive integers J and k: for any random variable X generated by $\mathcal{A}$ before time J as described above, the distribution of $\mathsf{RB}(J, k)$ conditioned on any fixing of $(\mathsf{rand}_{\mathcal{A}}, X)$ has co-min-entropy at most u (i.e. min-entropy at least $k \cdot \log |\mathbb{F}_p^*| - u$).

Our protocols are always of a round-robin nature, where player P_i sends a single message in each round following player P_{i-1}, and RB is invoked at the end of each round at the time slot after P_N's message. Thus, we implicitly assume the protocol defines that the time slot for P_i to send his round ℓ message is $J = (\ell - 1) \cdot (N + 1) + i$. In this context, it will be convenient to assume $\mathsf{RB}(J, k)$ is defined if and only if J is a multiple of $N + 1$.

2.2 Input domains

We assume implicitly in all method descriptions that if an input is not in the prescribed range the method outputs rej. This means that in an implementation of the protocol a method expecting input in $\mathbb{G}_2^*$ (for example) checks that the received input is indeed in this range and output rej otherwise.

2.3 Player-exchangeable protocols and adaptive adversaries

We assume there are N players $P_1, \ldots, P_N$ in each round of the protocol. Though we use this notation for each round, we do not assume it is the same player P_i in each round, nor that the identity of the player, or equivalently, their behavior in the protocol, was determined before the time slot where they send their message. In particular, it is possible P_i simply aborts adding nothing to the transcript.

When we discuss an adversary $\mathcal{A}$ controlling K players in the protocol, for $1 \leq K \leq N$, we mean that $\mathcal{A}$ can adaptively choose a different subset of K players to control in each round. That is, in time slot $(\ell - 1) \cdot (N + 1) + i$ they can choose whether to control P_i in round ℓ if they have not chosen K players so far in round ℓ.

We denote by $\mathsf{transcript}_{\ell,i}$ the transcript of the protocol up to the point where player i sent his message in round ℓ.

2.4 Preliminary claims

The following claim is not hard to show.

Claim 2.1. *Let A, B be two random variables such that for any fixing a of A, $B|A = a$ has co-min-entropy at most u. Let P be a predicate with range $\{\mathsf{acc}, \mathsf{rej}\}$. Let B' be a random variable independent of A that is uniform on the range of B. Then*

$$\Pr(P(A, B') = \mathsf{acc}) \geq 2^{-u} \cdot \Pr(P(A, B) = \mathsf{acc}).$$

2.5 Auxiliary methods

We define some methods to check whether certain ratios between elements hold, using the pairing function e. The following definition and claim are from [7].

Algorithm 1 Determine if $x \in \mathbb{F}_p^*$ exists such that $B = A \cdot x$, and $D = C \cdot x$.

```
Require: $A, B \in \mathbb{G}_1$ and $C, D \in \mathbb{G}_2$ and none of $A, B, C, D$ are the identity.
function SameRatio($(A, B), (C, D)$)
    if $e(A, D) = e(B, C)$ then
        return acc
    else
        return rej
    end if
end function
```

Claim 2.2. *Given $A, B \in \mathbb{G}_1^*$ and $C, D \in \mathbb{G}_2^*$,* $\mathsf{SameRatio}((A, B), (C, D)) = \mathsf{acc}$ *if and only if there exists $s \in \mathbb{F}_p^*$ such that $B = s \cdot A$ and $D = s \cdot C$.*

Algorithm 2 Check whether the ratio between A and B is the $s \in \mathbb{F}_p^*$ that is encoded in C

```
Require: $A, B \in \mathbb{G}_1^2$ or $A, B \in \mathbf{G}^2$. $C \in \mathbb{G}_2^*$ or $C \in (\mathbb{G}_2^*)^2$.
function Consistent($A, B, C$)
    if $C \in (\mathbb{G}_2^*)^2$ then
        $r \leftarrow$ SameRatio($(A_1, B_1), (C_1, C_2)$)
    else
        $r \leftarrow$ SameRatio($(A_1, B_1), (g_2, C)$)
    end if
    if $A, B \in \mathbb{G}_1$ then
        return $r$
    else
        return $r$ AND SameRatio($(A_1, B_1), (A_2, B_2)$)
    end if
end function
```

We later use the suggestive notation $\mathsf{consistent}(A - B; C)$ for the above function with inputs A, B, C.

2.6 Proofs of Knowledge

We will use a discrete log proof of knowledge scheme based on the Knowledge of Exponent assumption.

Definition 2.3 (Knowledge of Exponent Assumption (KEA)). *For any efficient $\mathcal{A}$ there exists an efficient deterministic χ such that the following holds. Consider the following experiment. $\mathcal{A}$ is given an arbitrary "auxiliary information string" z, together with a uniformly chosen $r \in \mathbb{G}_2^*$, that is independent of z. He then generates $x \in \mathbb{G}_1^*$ and $y \in \mathbb{G}_2^*$. χ, given the same inputs r and z and the internal randomness of $\mathcal{A}$, outputs $\alpha \in \mathbb{F}_p^*$. The probability that both*

1. *$\mathcal{A}$ "succeeded", i.e.,* $\mathsf{SameRatio}((g_1, x), (r, y))$,
2. *χ "failed", i.e., $x \neq [\alpha]_1$,*

is $\mathrm{negl}(\lambda)$.

Remark 2.4. *Let's see that the assumption is the standard KEA assumption, besides the partition of the elements to $\mathbb{G}_1$ and $\mathbb{G}_2$: Suppose that $r = [\gamma]_2$ and $x = [\alpha]_1$. Then* $\mathsf{SameRatio}((g_1, x), (r, y))$ *implies $y = [\alpha \cdot \gamma]_2$. Thus (x, y) is a pair of 'ratio' γ, generated from the given pair (g_1, r) also of ratio γ; and the KEA states to create such a pair we must know the ratio with the original pair, namely α.*

Note that KEA is usually phrased for groups written in multiplicative notation, thus a better name here might have been "Knowledge of Coefficient Assumption".

Algorithm 3 Construct a proof of knowledge of α

```
Require: $\alpha \in \mathbb{F}_p^*$
1: function POK($\alpha$, string $v$)
2:     $r \leftarrow \mathcal{R}([\alpha]_1, v) \in \mathbb{G}_2^*$
3:     return $([\alpha]_1, \alpha \cdot r)$
4: end function
```

Algorithm 4 Verify a proof of knowledge of α

```
Require: $a \in \mathbb{G}_1^*$, $b \in \mathbb{G}_2^*$
1: function CheckPOK($a$, string $v$,$b$)
2:     $r \leftarrow \mathcal{R}(a, v) \in \mathbb{G}_2^*$
3:     return SameRatio$((g_1, a), (r, b))$
4: end function
```

Claim 2.5. *Under the KEA assumption, for any efficient oracle circuit $\mathcal{A}$, there exists an efficient χ such that the following holds. Fix any string z that was generated without queries to $\mathcal{R}$. Given z and random oracle replies $r_1, \ldots, r_\ell$, $\mathcal{A}$ produces $a \in \mathbb{G}_1, y \in \mathbb{G}_2$ and a string v; and χ, given the same inputs together with the internal randomess used by $\mathcal{A}$, produces $\alpha \in \mathbb{F}_p^*$. The probability that both*

1. *$\mathcal{A}$ "succeeds", i.e., $\mathsf{CheckPOK}(a, v, y) = \mathsf{acc}$,*

2. *χ "failed", i.e., $a \neq [\alpha]_1$,*

is $\mathrm{negl}(\lambda)$.

Proof. Fix $\mathcal{A}$ and z such that given z and oracle access to $\mathcal{R}$, $\mathcal{A}$ produces a pair $a \in \mathbb{G}_1, y \in \mathbb{G}_2$ and string v. Let $\ell = \mathrm{poly}(\lambda)$ be the number of oracle calls $\mathcal{A}$ makes to $\mathcal{R}$. We can think of $\mathcal{A}$ as a deterministic function of z, the sequence $\mathbf{r} = r_1, \ldots, r_\ell$ of replies from $\mathcal{R}$, and its internal randomness $\mathsf{rand}_{\mathcal{A}}$. For $i \in [\ell]$, we construct $\mathcal{A}_i$, that given z and $r \in \mathbb{G}_2$ does the following. It invokes $\mathcal{A}$ on $(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}})$, where r_j is chosen uniformly for $j \neq i$, and $r_i = r$; and $\mathsf{rand}_{\mathcal{A}}$ is chosen uniformly. Let $(a, v, y) := \mathcal{A}(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}})$ and let $q_1, \ldots, q_\ell$ be its sequence of queries to $\mathcal{R}$. Let D_i be the set of $(\mathbf{r}, \mathsf{rand}_{\mathcal{A}})$ such that $q_i = (a, v)$ and i is the first such index. If $(\mathbf{r}, \mathsf{rand}_{\mathcal{A}}) \notin D_i$, $\mathcal{A}_i$ aborts. Otherwise, $\mathcal{A}_i$ outputs (a, y). By the KEA, there exists an efficient χ_i such that the probability over uniform $\mathbf{r}, \mathsf{rand}_{\mathcal{A}}$ that both

1. $\mathsf{SameRatio}((g_1, a), (r, y))$,

2. χ_i given $z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}}$ didn't output α such that $a = [\alpha]_1$,

is $\mathrm{negl}(\lambda)$. We can think of $\mathcal{A}_i$ as a deterministic function $\mathcal{A}_i(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}})$, that takes r_i as its input r and $r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_\ell$ as its randomness for answering the calls to $\mathcal{R}$ for $j \neq i$. We can think of χ_i as a function $\chi_i(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}})$ in the same way.

Now we construct an efficient χ as follows. χ determines the sequence $q_1, \ldots, q_\ell$ of queries to $\mathcal{R}$ made by $\mathcal{A}(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}})$ and its output (a, v, y). Suppose that $(\mathbf{r}, \mathsf{rand}_{\mathcal{A}}) \in D_i$ for some $i \in [\ell]$, then χ returns $\alpha := \chi_i(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}})$; otherwise χ aborts. Now suppose that $(\mathbf{r}, \mathsf{rand}_{\mathcal{A}}) \in D_i$ and "$\mathcal{A}$ beats χ". That is,

1. $\mathsf{CheckPOK}(a, v, y) = \mathsf{acc}$.

2. $\chi(z, v, \mathbf{r}, \mathsf{rand}_{\mathcal{A}}) = \alpha$ where $a \neq [\alpha]_1$.

We have $\mathcal{R}(a, z) = r_i$, and $\chi_i(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}}) = \chi(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}})$. Hence,

1. $\mathsf{SameRatio}((g_1, r_i), (a, y))$.

2. $\chi_i(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}}) = \alpha$ where $a \neq [\alpha]_1$.

But this can only happen for a $\mathrm{negl}(\lambda)$ fraction of $(\mathbf{r}, \mathsf{rand}_{\mathcal{A}})$. Also, if $(\mathbf{r}, \mathsf{rand}_{\mathcal{A}}) \notin D_i$ for any $i \in [\ell]$, the value of $\mathcal{R}(a, v)$ is yet unknown and uniformly distributed and thus the probability that $\mathsf{CheckPOK}(a, v, y)$ is $\mathrm{negl}(\lambda)$.

A union bound over $i \in [\ell]$ now gives the claim. $\square$

3 Multi-party Computation

3.1 The circuit structure

We assume we have an artithmetic circuit $\mathbf{C}$ over $\mathbb{F}_p$ with the following structure, which may seem a bit adhocish; however it is satisfied for a circuit computing the extended CRS of [11] described in Section 5 and allows us to simplify the protocol design of [5].

The circuit consists of alternate multiply/divide layers $C_1, \ldots, C_d$, and linear combination layers $L_1, \ldots, L_d$. We call d the *depth* of the circuit.[5] (A layer can have depth larger than one in the regular sense.) The circuit inputs $\mathbf{x}$ are partitioned into disjoint sets $\mathbf{x}^1, \ldots, \mathbf{x}^d$ corresponding to the layers. Specifically, we think of $\mathbf{x}^\ell$ as the inputs of the multiply/divide layer C_ℓ, and at times use the notation $x \in C_\ell$ to mean $x \in \mathbf{x}^\ell$. A multiply/divide layer C satisfies the following:

1. All gate outputs in C are outputs of the circuit.
2. $C = C_\ell$ has an input *gate* for each of its inputs $x \in \mathbf{x}^\ell$. When another gate wishes to use one of these inputs, it uses a wire from the corresponding input gate (i.e. there are no "direct" input wires). In particular, every input is part of the circuit output.
3. All gates in C, besides the input gates, are division and mutiplication gates of fan-in two. The left input is a gate from C or previous layers; and the right input is an input gate belonging to C.[6] In case of a division gate, the right input is always the denominator.

A linear combination layer L consists of linear combination gates of unbounded fan in, whose inputs are gates from L or previous layers.

3.2 The protocol coordinator

In addition to messages of the players, the protocol description includes messages that are to be sent by the *protocol coordinator*. These messages are a deterministic function of the protocol description and the transcript up to that point. In practice, it can be helpful to have a computationally strong party fill this role. However, there is no need to trust this party, and anyone can later verify that the protocol coordinator's messages in the protocol transcript are correct. In particular, the role of the protocol verifier will include, in addition to the steps explicitly described, to compute the protocol coordinator's messages independently and check they are correct.

[5]This notion is similar to S-depth in [5], though we have not determined the precise relation.

[6]In fact, we can allow the right input to be any gate that is 'purely' from C; meaning that the directed tree of gates leading to the right input only contains gates from C. But for the Groth circuit [11] which is our main usecase, we can assume the right input is an actual input from the same layer.

3.3 The MPC

The goal of the protocol is to compute $\mathbf{C}(\mathbf{x})\cdot\mathbf{g}$ for uniformly chosen $\mathbf{x} \in (\mathbb{F}_p^*)^t$, where t is the number of $\mathbf{C}$'s inputs. More specifically, we will have $\mathbf{x} = \mathbf{x}_1 \cdots \mathbf{x}_N \cdot \mathbf{x'}$ (recall this product is defined coordinate-wise), where $\mathbf{x}_i \in (\mathbb{F}_p^*)^t$ is the input of P_i, and $\mathbf{x}$' is a random beacon output.

Denote the layers of $\mathbf{C}$ by $C_1, L_1, \ldots, C_d, L_d$. The protocol consists of d rounds corresponding to the layers.

3.4 The round structure

We fix a layer $\ell \in [1..d]$ and denote $C = C_\ell, L = L_\ell$. We assume that for all gates g in previous layers - $C_1, L_1, \ldots, C_{\ell-1}, L_{\ell-1}$, we have already computed an output value $[\mathsf{g}] \in \mathbf{G}$.

Note that the output of every gate $\mathsf{g} \in C$, is a Laurent monomial (i.e. ratio of two monomials) in C's inputs, possibly multiplied by an output of some gate g' from a previous layer. Denote this monomial M_{g}, and the output from the previous layer by $\mathsf{g}_{\mathsf{src}}$; if no such output exists let $\mathsf{g}_{\mathsf{src}} := \mathbf{g}$.

1. For $j \in [N]$, Player j does the following.
   (a) For each input x used in C, output $[x_j]_1$, and $y_{x,j} := \mathsf{POK}(x_j, v)$, where $v = \mathsf{transcript}_{\ell,j-1}$ is the protcol transcript before the current player.
   (b) For each gate $\mathsf{g} \in C$:
      - If $j = 1$, output $[\mathsf{g}]^{\mathbf{1}} := M_{\mathsf{g}}(\mathbf{x}_1^\ell) \cdot \mathsf{g}_{\mathsf{src}}$.
      - Otherwise, when $j > 1$, output $[\mathsf{g}]^{\mathbf{j}} := M_{\mathsf{g}}(\mathbf{x}_j^\ell) \cdot [\mathsf{g}]^{\mathbf{j-1}}$.
2. Let $J - 1$ be the time slot on which P_N was supposed to broadcast in this round. The protocol coordinator computes and outputs $\mathbf{x'}^\ell := \mathsf{RB}(J, t_\ell)$, and $[\mathsf{g}] := M_{\mathsf{g}}(\mathbf{x'}^\ell) \cdot [\mathsf{g}]^{\mathbf{N}}$ for each $\mathsf{g} \in C$.
3. Finally, the protocol coordinator computes and outputs, in the same time slot, the values $[\mathsf{g}]$ for all gates g in the linear combination layer $L = L_\ell$.

Verification:
For each $j \in N$, the protocol verifier does the following.

1. For each input $x \in C$, let $r_{x,j} = \mathcal{R}([x_j]_1, \mathsf{transcript}_{\ell,j-1})$ check that $\mathsf{CheckPOK}([x_j]_1, \mathsf{transcript}_{\ell,j-1}, y_{x,j})$; and $\mathsf{consistent}([x]^{\mathbf{j-1}} - [x]^{\mathbf{j}}; (r_{x,j}, y_{x,j}))$.
2. Let g_{L} and g_{R} be the inputs of g.
3. If $\mathsf{g}_{\mathsf{L}} \in C$ then

- If g is a multiplication gate check that $\mathsf{consistent}([\mathsf{g_L}]^{\mathbf{j}} - [\mathsf{g}]^{\mathbf{j}}; [\mathsf{g_R}]^{\mathbf{j}})$
- If g is a division gate check that $\mathsf{consistent}([\mathsf{g}]^{\mathbf{j}} - [\mathsf{g_L}]^{\mathbf{j}}; [\mathsf{g_R}]^{\mathbf{j}})$

4. If $\mathsf{g_L}$ is from a previous layer, then
   - If g is a multiplication gate check that $\mathsf{consistent}([\mathsf{g_L}] - [\mathsf{g}]^{\mathbf{j}}; [\mathsf{g_R}]^{\mathbf{j}})$
   - If g is a division gate check that $\mathsf{consistent}([\mathsf{g}]^{\mathbf{j}} - [\mathsf{g_L}]; [\mathsf{g_R}]^{\mathbf{j}})$

4 Security Proof

We denote by $\mathbf{C}_S$ a random variable equal to the encoded output of the circuit $\mathbf{C}$ with uniformly chosen input. That is, $\mathbf{C}_S := [\mathbf{C}(s)]$ for uniform $s \in (\mathbb{F}_p^*)^t$.

Let $\mathcal{A}$ be an adversary that controls a subset of $N-1$ players in each round as described in Section 2.3. We denote by $\mathbf{C}_{\mathcal{A}}$ the circuit output generated by $\mathcal{A}$ participating in the protocol together with an honest player in each round. We think of $\mathcal{A}$ as outputting a string z after the end of the protocol. $\mathbf{C}_{\mathcal{A}}$ and z are random variables that are a function of $\mathcal{A}$'s randomness $\mathsf{rand}_{\mathcal{A}}$, the honest player's inputs - which consist of uniformly distributed independent elements of $\mathbb{F}_p^*$, the random oracle $\mathcal{R}$'s outputs - which are uniformly distributed elements of $\mathbb{G}_2$; and the random beacon's outputs $\mathsf{rand_{beacon}}$ (which are elements of $\mathbb{F}_p^*$, over which $\mathcal{A}$ may have some limited influence).

For a predicate P with range $\{\mathsf{acc}, \mathsf{rej}\}$, we define

$$\mathrm{adv}_{\mathcal{A},P} := \Pr(P(\mathbf{C}_{\mathcal{A}}, \mathsf{z}) = \mathsf{acc}).$$

Note that $\mathrm{adv}_{\mathcal{A},P}$ depends on RB and the amount of influence $\mathcal{A}$ has on RB. We think of RB as fixed and thus don't use it as an extra parameter.

Theorem 4.1. *Fix any efficient oracle circuit $\mathcal{A}$ and $u > 0$. Fix a number of players N with $N(\lambda) = \mathrm{poly}(\lambda)$. There exists an efficent $\mathcal{B}$ such that if RB is u-co-resistant to $\mathcal{A}$, then for every predicate P*

$$\Pr(P(\mathbf{C}_S, \mathcal{B}(\mathbf{C}_S)) = \mathsf{acc}) \geq 2^{-ud} \cdot \mathrm{adv}_{\mathcal{A},P} - \mathrm{negl}(\lambda).$$

Suppose P is a predicate that runs a SNARK verifier with some fixed public input, using its first input as the SNARK parameters, and the second as the proof; take a constant d and $u = O(\log \lambda)$. The theorem implies that if $\mathcal{A}$ cannot construct a correct proof with non-negligible probability for independently generated parameters, it cannot do so for parameters generated in the protocol in which it participated.

Proof. Denote by H the set of inputs of the honest player in each round. Denote by $\mathsf{rand_{beacon}}$ the replies of the random beacon to the protocol coordinator at the end of each round. Denote by $\mathsf{rand_{oracle}}$ the replies of the random oracle to the honest player (when computing $\mathsf{POK}(x,z)$ for $x \in H$) and to $\mathcal{A}$'s queries. The circuit output $\mathbf{C}_{\mathcal{A}}$ and the string z $\mathcal{A}$ outputs after the protocol can be viewed as a function of $\mathsf{x} = (\mathsf{rand}_{\mathcal{A}}, H, \mathsf{rand}_{\mathsf{oracle}}, \mathsf{rand}_{\mathsf{beacon}})$. Call this function F; i.e. $F(\mathsf{x}) = (\mathbf{C}_{\mathcal{A}}(\mathsf{x}), \mathsf{z}(\mathsf{x}))$. Let $\mathcal{X}$ be the set of such x's. We have d calls to RB- one at the end of each round corresponding to the string $\mathsf{rand}_{\mathsf{beacon}} = \mathsf{rand}_{\mathsf{beacon}1}, \ldots, \mathsf{rand}_{\mathsf{beacon}d}$. As RB is u-co-resistant to $\mathcal{A}$, we know that during the protocol $\mathsf{rand}_{\mathsf{beacon}\ell}$ has co-min-entropy at most u conditioned on any fixing of $\mathsf{rand}_{\mathcal{A}}, H, \mathsf{rand}_{\mathsf{oracle}}, \mathsf{rand}_{\mathsf{beacon}1}, \ldots, \mathsf{rand}_{\mathsf{beacon}\ell-1}$. In particular,

$$\mathrm{adv}_{\mathcal{A},P} = \Pr(P(\mathbf{C}_{\mathcal{A}}(A,B), \mathsf{z}(A,B)) = \mathsf{acc}).$$

for a uniformly distributed A on the possible values of $(\mathsf{rand}_{\mathcal{A}}, H, \mathsf{rand}_{\mathsf{oracle}})$, and a random variable B having co-min-entropy at most ud conditioned on any fixing of A, describing the value of $\mathsf{rand}_{\mathsf{beacon}}$. It now follows from Claim 2.1 that

$$\Pr_{\mathsf{x} \leftarrow \mathcal{X}}(P(\mathbf{C}_{\mathcal{A}}(\mathsf{x}), \mathsf{z}(\mathsf{x})) = \mathsf{acc}) \geq 2^{-ud} \cdot \mathrm{adv}_{\mathcal{A},P}.$$

(where $\mathsf{x} \leftarrow \mathcal{X}$ refers to a uniform choice of x.)

Given $\mathcal{A}$ we construct $\mathcal{B}$ with the following property. $\mathcal{B}$ receives $[\mathbf{C}(s)]$ which is an output value of the random variable $\mathbf{C}_S$. Given $[\mathbf{C}(s)]$ it produces an output $\mathsf{z}(\mathsf{x})$, for x such that

1. x is uniform in $\mathcal{X}$ (over the randomness of $s \in (\mathbb{F}_p^*)^t$ and the randomness of $\mathcal{B}$).
2. The values x for which $\mathcal{B}$ does not produce an output $\mathsf{z}(\mathsf{x})$ with $\mathbf{C}_{\mathcal{A}}(\mathsf{x}) = [\mathbf{C}(s)]$ have density $\mathrm{negl}(\lambda)$.

It follows that

$$\Pr(P(\mathbf{C}_S, \mathcal{B}(\mathbf{C}_S)) = \mathsf{acc}) \geq \Pr_{\mathsf{x} \leftarrow \mathcal{X}}(P(\mathbf{C}_{\mathcal{A}}(\mathsf{x}), z(\mathsf{x})) = \mathsf{acc}) - \mathrm{negl}(\lambda)$$

$$\geq 2^{-ud} \cdot \mathrm{adv}_{\mathcal{A},P} - \mathrm{negl}(\lambda).$$

We proceed to describe $\mathcal{B}$ and show that its output is as claimed.

We have $[\mathbf{C}(s)] = \{[\mathsf{g}(s)]\}_{\mathsf{g} \in \mathrm{M}_{\mathbf{C}}}$, where $\mathrm{M}_{\mathbf{C}}$ is the set of all gates in all multiply/divide layers of $\mathbf{C}$. $\mathcal{B}$ runs the protocol with $\mathcal{A}$ as follows. We think of $\mathcal{B}$ as running an internal oracle circuit $\mathcal{B}^*$ that makes queries to $\mathcal{R}$. When $\mathcal{B}^*$ makes a new query to $\mathcal{R}$, $\mathcal{B}$ answers uniformly in $\mathbb{G}_2^*$, and otherwise it answers consistently with the previous answer. If $\mathcal{B}^*$ aborts in the description below, $\mathcal{B}$ outputs $\mathsf{z}(\mathsf{x}')$ for some fixed arbitrary string x'.

$\mathcal{B}^*$ in turn runs $\mathcal{A}$ as follows.

1. $\mathcal{B}^*$ intializes an empty table T of "exceptions" to responses of $\mathcal{R}$.
2. Whenever $\mathcal{A}$ makes a query q to $\mathcal{R}$, $\mathcal{B}^*$ checks if the reply $\mathcal{R}(q)$ is present in T; if so it answers according to that, otherwise according to $\mathcal{R}$. It answers queries to RB as specified below.

3. For each $\ell \in [1..d]$, it emulates the ℓ'th round as follows.

   (a) Let j be the index of the honest player in round ℓ.[7] Let $C := C_\ell$. Recall that $\mathbf{x}^\ell$ denotes the inputs belonging to C. $\mathcal{B}^*$ begins by executing the round up to player P_{j-1} by invoking $\mathcal{A}$ on the transcript from previous rounds.

   For each $1 \le j' < j$ such that P'_j aborted or wrote an invalid message that the protocol verifier rejected, $\mathcal{B}$ sets $\mathbf{x}^\ell_{j'} = (1,\ldots,1) \in (\mathbb{F}^*_p)^{t_\ell}$. Otherwise, for each $x \in \mathbf{x}^\ell_{j'}$, $P_{j'}$ has output $[x]_1$ and $y \in \mathbb{G}_2$ with $\mathsf{CheckPOK}([x]_1, \mathsf{transcript}_{\ell,j'-1}, y)$. Let χ be the extractor obtained from Claim 2.5 when taking there $\mathcal{A}$ to be a variant of $\mathcal{B}^*$ that uses the same random string and runs identically to $\mathcal{B}^*$ but stops when reaching this point and outputs $[x]_1, \mathsf{transcript}_{\ell,j-1}, y$; and taking $z = [\mathbf{C}(s)]$. $\mathcal{B}^*$ computes $x^* = \chi(z, \mathbf{r}, \mathsf{rand}_{\mathcal{B}^*})$ where $\mathbf{r}$ is the sequence of replies to $\mathcal{B}^*$ from $\mathcal{R}$ up to the point of outputting $[x]_1, y$. If χ's output x^* is not equal to x, $\mathcal{B}^*$ aborts. (This can be checked by checking if $[x^*]_1 = [x]_1$.)

   (b) If $\mathcal{B}^*$ has not aborted it has obtained $\mathbf{x}^\ell_1, \ldots, \mathbf{x}^\ell_{j-1}$. $\mathcal{B}^*$ now chooses uniform $b \in (\mathbb{F}^*_p)^{t_\ell}$, and defines

   $$\mathbf{x}^\ell_j := \frac{bs^\ell}{\mathbf{x}^\ell_1 \cdots \mathbf{x}^\ell_{j-1}}.$$

   Note that as $\mathcal{B}^*$ doesn't know s it can't compute $\mathbf{x}^\ell_j$. However, it has $\left[s^\ell\right]$ as part of $[\mathbf{C}(s)]$, where s^ℓ is the restriction of s to the inputs $\mathbf{x}^\ell$ of C. Thus it can compute

   $$\left[\mathbf{x}^\ell_j\right] = \frac{b \cdot \left[s^\ell\right]}{\mathbf{x}^\ell_1 \cdots \mathbf{x}^\ell_{j-1}}$$

   Note that $\mathbf{x}^\ell_1 \cdots \mathbf{x}^\ell_j = bs^\ell$. So, for each $\mathsf{g} \in C$, $\mathcal{B}^*$ can compute and broadcast

   $$[\mathsf{g}]^{\mathbf{j}} = M_{\mathsf{g}}(\mathbf{x}^\ell_1 \cdots \mathbf{x}^\ell_j) \cdot \mathsf{g}_{\mathsf{src}} = M_{\mathsf{g}}(b) M_{\mathsf{g}}(s^\ell) \cdot \mathsf{g}_{\mathsf{src}} = M_{\mathsf{g}}(b) \cdot [\mathsf{g}(s)].$$

   ($[\mathsf{g}(s)]$ is given as part of $[\mathbf{C}(s)]$.) Thus, $\mathcal{B}^*$ can correctly play the role of P_j with this value of $\mathbf{x}^\ell_j$ in Step 1b of Section 3.4 and produces a valid message.

   (c) What is left is generating $\mathsf{POK}([x]_1, \mathsf{transcript}_{\ell,j-1})$ for $x \in \mathbf{x}^\ell_j$ as in step 1 of Section 3.4. If $\mathcal{R}([x]_1, \mathsf{transcript}_{\ell,j-1})$ has been queried by $\mathcal{A}$ it aborts. Otherwise, $\mathcal{B}^*$ chooses random $r \in \mathbb{F}^*_p$ and adds the query

[7]Note that j may only be determined by $\mathcal{A}$ after the message of P_{j-1}, but the description of $\mathcal{B}^*$ in this step doesn't require knowing j before, and $\mathcal{B}^*$ can just execute $\mathcal{A}$ until reaching a player j that $\mathcal{A}$ doesn't choose to control.

$(([x], \mathsf{transcript}_{\ell,j-1}), [r]_2)$ to the exceptions table T. It outputs $y := r \cdot [x]_2$. Note that if we had $\mathcal{R}([x]_1, \mathsf{transcript}_{\ell,j-1}) = [r]_2)$ then we would have $\mathsf{CheckPOK}([x]_1, \mathsf{transcript}_{\ell,j-1}, y)$; so from $\mathcal{A}$'s point of view this is a correct message given H and $\mathsf{rand}_{\mathsf{oracle}}$.

(d) Now $\mathcal{B}^*$ uses $\mathcal{A}$ to run the parts of $P_{j+1}, \ldots, P_N$ in round ℓ. Again, for any $j+1 \le j' \le N$ such that $P_{j'}$ did not output a valid message, $\mathbf{x}_{j'}^{\ell}$ is set to the vector $(1, \ldots, 1)$.

(e) Similary to before, for any $j+1 \le j' \le N$ such that $P_{j'}$ did broadcast a valid message, for each $x \in \mathbf{x}_{j'}^{\ell}$, $P_{j'}$ has output $[x]_1$ and $y \in \mathbb{G}_2$ with $\mathsf{CheckPOK}([x]_1, \mathsf{transcript}_{\ell,j'-1}, y)$. Let χ be the extractor obtained from Claim 2.5 when taking there $\mathcal{A}$ to be a variant of $\mathcal{B}^*$ that runs up to this point and outputs $[x]_1, y$; and taking $z = [\mathbf{C}(s)]$. $\mathcal{B}^*$ computes $x^* = \chi(z, \mathbf{r}, \mathsf{rand}_{\mathcal{A}}) = x$ where $\mathbf{r}$ is the sequence of replies to $\mathcal{B}^*$ from $\mathcal{R}$ up to the point of outputting $[x]_1, y$. If χ's output is not equal to x, $\mathcal{B}$ aborts.

(f) If $\mathcal{B}^*$ has not aborted it has obtained $\mathbf{x}_{j+1}^{\ell}, \ldots, \mathbf{x}_N^{\ell}$. It defines $\mathbf{x}'^{\ell} := \frac{1}{b \cdot \mathbf{x}_{j+1}^{\ell} \cdots \mathbf{x}_N^{\ell}}$. and outputs $\mathbf{x}'^{\ell}$ as the beacon output $\mathsf{RB}(J, t_\ell)$. Note that if we have reached this point without aborting we have $\mathbf{x}_1^{\ell} \cdots \mathbf{x}_N^{\ell} \cdot \mathbf{x}'^{\ell} = s^{\ell}$.

4. Finally $\mathcal{B}^*$ outputs $\mathcal{A}$'s output z at the end of the protocol.

We proceed to prove the first property - we need to show that the elements ($\mathsf{rand}_{\mathcal{A}}$,$H$,$\mathsf{rand}_{\mathsf{beacon}}$,$\mathsf{rand}_{\mathsf{oracle}}$) used in the protocol are uniform and independent of each other.

- $\mathsf{rand}_{\mathcal{A}}$- $\mathcal{B}^*$ runs $\mathcal{A}$ with a uniform choice of its random coins, so $\mathsf{rand}_{\mathcal{A}}$ is uniformly distributed.

- $\mathsf{rand}_{\mathsf{oracle}}$- $\mathcal{B}$ choses the outputs of $\mathcal{R}$ uniformly and independent of any other event. The other elements of $\mathsf{rand}_{\mathsf{oracle}}$ are the elements $[r]_2$ chosen in step 3c which are uniform in $\mathbb{G}_2^*$ and independent of any other variable here.

- H- the honest input $\mathbf{x}_j^{\ell}$ of each layer C_ℓ is chosen as $\frac{b \cdot s^{\ell}}{a}$ where a is the product of inputs in the same layer by the players controlled by $\mathcal{A}$ participating before the honest player. b and s^{ℓ} are both uniform in $(\mathbb{F}_p^*)^{t_\ell}$; and independent from each other, a and the same variables from other layers. Hence H is uniform and independent from previous variables.

- $\mathsf{rand}_{\mathsf{beacon}}$- the part of $\mathsf{rand}_{\mathsf{beacon}}$ from layer $C = C_\ell$ is of the form $\frac{1}{a \cdot b}$, where a contains inputs of the player controlled by $\mathcal{A}$ following the honest player. The only other place b appears in is in $\mathbf{x}_j^{\ell}$. But even fixing $\mathbf{x}_j^{\ell}$ leaves b, and hence the part of $\mathsf{rand}_{\mathsf{beacon}}$ from round ℓ, uniform.

To prove the second property we note we note that the values x for which the protocol output as described will not be $[\mathbf{C}(s)]$ are those that cause an abort in steps 3a,3e or 3c. An abort in steps 3a,3e happens for a $\mathrm{negl}(\lambda)$ fraction of $\mathsf{x} \in \mathcal{X}$ according to Claim 2.5; aborting in step 3c happens only when $\mathcal{A}$ chose in advance to query $\mathcal{R}$ in a later uniformly chosen input in a domain of size at least $|\mathbb{G}_2^*|$, and thus happens only for a $\mathrm{negl}(\lambda)$ fraction of $\mathsf{x} \in \mathcal{X}$.

□

5 Reducing the Depth of Groth's CRS

In this section we assume familiarity with Quadratic Artihmetic Programs [10] and the work of Groth [11]. As in [11] we first describe the Non-Interactive Linear Proof (NILP) from which the zk-SNARK is built.

The extended Groth CRS: Let $\{u_i, v_i, w_i\}_{i\in[0..m]}$ be the polynomials of the QAP of degree less than n, and let t be the target polynomial of degree n. Suppose that $1, \ldots, \ell < m$ are the indices of the public input.

For $\alpha, \beta, \delta, x \in \mathbb{F}_p^*$. *Groth*$(\alpha, \beta, \delta, x)$ is defined as the set of elements:

$$\beta, \delta, \left\{x^i\right\}_{i\in[0..2n-2]}, \left\{\alpha x^i\right\}_{i\in[0..n-1]}, \left\{\beta x^i\right\}_{i\in[1..n-1]}, \left\{x^i \cdot t(x)/\delta\right\}_{i\in[0..n-2]},$$
$$\left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right\}_{i\in[\ell+1..m]}.$$

The additional elements, compared to [11] are $\left\{x^i\right\}_{i\in[n..2n-2]}, \left\{\alpha x^i\right\}_{i\in[1..2n-1]}, \left\{\beta x^i\right\}_{i\in[1..2n-1]}$. On the other hand the elements

$$\left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right\}_{i\in[0..\ell]}, \gamma$$

that appear in the CRS of [11] and have disappeared here; they were needed there to enable the verifer to compute

$$\sum_{i=0}^{\ell} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x));$$

which can be computed as a linear combination of above CRS with our added elements.

We claim that *Groth* can be computed by a depth two circuit according to the definition of depth in Section 3.1:

- C_1**:** The layer inputs are $\mathbf{x}^1 = (x, \alpha, \beta)$. The layer computes α, β, $\left\{x^i\right\}_{i\in[0..2n-2]}$, $\left\{\alpha x^i\right\}_{i\in[0..2n-1]}$, $\left\{\beta x^i\right\}_{i\in[0..2n-1]}$, which are all products of inputs in $\mathbf{x}^1$.
- L_1**:** We compute $\left\{x^i \cdot t(x)\right\}_{i\in[0..n-2]}$ that are linear combinations of $\left\{x^i\right\}_{i\in[0..2n-2]}$ since t has degree n. We also compute $\{\beta u_i(x) + \alpha v_i(x) + w_i(x)\}_{i\in[0..m]}$, which are linear combinations of elements from the first layer.

- C_2**:** The layer input $\mathbf{x}^2 = \delta$. Compute δ, $\left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}\right\}_{i\in[\ell+1..m]}$, $\left\{x^i t(x)/\delta\right\}_{i\in[0..n-2]}$.

Groth prover and verifier: Fix public input $a_1,\ldots,a_\ell$. The prover chooses random $r,s \in \mathbb{F}_p$ and computes from the CRS and her witness $a_{\ell+1},\ldots,a_m$; the elements

$$A = \alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta, B = \beta + \sum_{i=0}^{m} b_i v_i(x) + s\delta$$

$$C = \frac{\sum_{i=\ell+1}^{m} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta} + As + Br - rs\delta.$$

The verifier, given A,B,C, checks that:

$$A\cdot B = \alpha\cdot\beta + \sum_{i=0}^{\ell} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + C\cdot\delta.$$

Proving knowledge soundness From [11] it is enough to prove that we can extract a witness for the QAP given A,B,C that are linear combinations of the CRS elements such that the verification equation holds as a polynomial identity. That is, we assume we are given

$$A = A_\alpha(x)\alpha + A_\beta(x)\beta + A_\delta\delta + A(x)$$
$$+ \sum_{i=\ell+1}^{m} \frac{A_i\cdot(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\delta} + A_h(x)\frac{t(x)}{\delta}$$

where A_α, A_β are known polynomials of degree at most $n-1$, A is a polynomial of degree at most $2n-2$, A_h is of degree at most $n-2$ and $A_i, \{A_i\}_{i\in[\ell+1..m]}, A_\delta$ are known field elements. B and C are defined similarly. And we assume for these given polynomials and constants that

$$A\cdot B \equiv \alpha\cdot\beta + \sum_{i=0}^{\ell} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + C\cdot\delta$$

as rational functions in x,α,β,δ. Let us denote by C^* the right hand of the equation for a given C; i.e,

$$C^* := \alpha\cdot\beta + \sum_{i=0}^{\ell} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + C\cdot\delta$$

and denote the "part without C in C^* by C_0; i.e

$$C_0 := \alpha\cdot\beta + \sum_{i=0}^{\ell} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))$$

When we discuss monomials from now on we mean the quotient of two monomials in α, β, δ, x that have no common factors; e.g. $\frac{\alpha}{\delta}$. For a monomial M let us use the notation $M \in A$ to mean M has a non-zero coefficient in A; i.e., when writing A as (the unique) linear combination of monomials in α, β, δ, x, M appears with non-zero coefficient. Use the same notation for $B, C, A \cdot B, C_0, C^*$.

When we say a monomial is *in the CRS*, we mean it is present with non-zero coefficient in one of the elements of the CRS $groth(\alpha, \beta, \delta, x)$ when writing that element as a combination of monomials.

Our focus is to show the new monomials we have added to the CRS - $\{x^i\}_{i \in [n..2n-2]}$, $\{\alpha x^i\}_{i \in [1..n-1]}$, $\{\beta x^i\}_{i \in [1..n-1]}$ are not used in A, B, C; this will imply correctness using [11], as there it is proven that given A, B, C that are linear combinations of the original CRS elements for which verification holds, a witness can be extracted.

As $\alpha\beta \in A \cdot B$ we must have $\alpha \in A, \beta \in B$ - or $\beta \in A, \alpha \in B$. Assume the first option w.l.g. Now take the maximal i such that the monomial $x^i \in A$. It follows that $\beta x^i \in A \cdot B$. So we must have $\beta x^i \in C^*$; which means either $\beta x^i/\delta \in C$ -but such monomials are possibly in the CRS only for $i \le n-1$ (in the terms $\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}$); or either $\beta x^i \in C_0$, which again can only happen for $i \le n-1$. A similar argument shows that $x^i \notin B$ for $i \ge n$. If $x^i \in C$ it implies $x^i\delta \in A \cdot B$, which means $x^i \in A$ or $x^i \in B$, and thus $i < n$. Therefore, the new terms $\{x^i\}_{i \in [n..2n-1]}$ are not used in the proof.

Now suppose $\alpha x^i \in A$ for $i > 0$; then $\alpha\beta x^i \in A \cdot B$ - which means either $\alpha\beta x^i/\delta \in C$ - but no such monomial exists in the CRS, or $\alpha\beta x^i \in C_0$, but such monomial exists in C_0 only for $i = 0$. A symmetrical argument shows that the other three options $\beta x^i \in A, \alpha x^i \in B, \beta x^i \in B$, can only hold for $i = 0$ - as otherwise we would have a monomial $Mx^i/\delta \in C$ or $Mx^i \in C_0$, for a monomial M of degree 2 in α, β - and such exists only in C_0 for $i = 0$.

Now assume $\alpha x^i \in C$ - then $\alpha x^i \delta \in A \cdot B$ which means αx^i is in A or B; and we have seen this is possible only for $i = 0$. Same holds when $\beta x^i \in C$. In summary, we have shown the new terms $\{\alpha x^i, \beta x^i\}_{i \in [1..n-1]}$ do not appear in the proof.

6 Multi-party Computation for Groth's SNARK

We know instantiate the protocol of Section 3 to get a protocol for computing the CRS of the zk-SNARK corresponding to that of the NILP described in Section 5.

The output will have the form

$$\left\{[x^i]\right\}_{i \in [0..n-1]}, \left\{[x^i]_1\right\}_{i \in [n..2n-2]}, \left\{[\alpha x^i]_1\right\}_{i \in [0..n-1]}, [\beta], \left\{[\beta x^i]_1\right\}_{i \in [1..n-1]},$$
$$\left\{[x^i \cdot t(x)/\delta]_1\right\}_{i \in [0..n-2]}, \left\{\left[\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right]_1\right\}_{i \in [\ell+1..m]}$$

Notice that some outputs are given only in $\mathbb{G}_1$, whereas the protocol description in Section 3 gave all outputs in both groups. It's straightforward to see that if this is the case only for outputs later used as inputs only for other outputs given only in $\mathbb{G}_1$ as well, the security proof goes through the same way.

In the protocol below , if M is an output in $\mathbb{G}_1, \mathbb{G}_2$ or $\mathbf{G}$ that we want to compute, and $j \in [N]$, we will denote by $[M]^{\mathbf{j}}$, the "partial M" after players $P_1, \ldots, P_j$ have contributed their shares. $[M]^{\mathbf{0}}$ will be set to some initial value as part of the protocol description. We assume $\mathbf{g}$ is publicly known.

6.1 Round 1: 'Powers of τ'

We need to compute

$$\mathrm{M}_1 = \left\{ \left\{ \left[x^i\right] \right\}_{i \in [0..n-1]}, \left\{ \left[x^i\right]_1 \right\}_{i \in [n..2n-2]}, \left\{ \left[\alpha x^i\right]_1 \right\}_{i \in [0..n-1]}, [\beta], \left\{ \left[\beta x^i\right]_1 \right\}_{i \in [1..n-1]} \right\}.$$

Initialization: We initialize the values

1. $[x^i]^{\mathbf{0}} := \mathbf{g}$, $i \in [1..n-1]$.
2. $[x^i]^{\mathbf{0}} := g_1$, $i \in [n..2n-2]$.
3. $[\alpha x^i]^{\mathbf{0}} := g_1$, $i \in [0..n-1]$.
4. $[\beta]^{\mathbf{0}} := \mathbf{g}$.
5. $[\beta x^i]^{\mathbf{0}} := g_1$, $i \in [1..n-1]$.

Computation: For $j \in [N]$, P_j outputs:

1. $[\alpha_j]_1$,$[\beta_j]_1$,$[x_j]_1$
2. $y_{\alpha,j} := \mathsf{POK}(\alpha_j, \mathsf{transcript}_{1,j-1})$
3. $y_{\beta,j} := \mathsf{POK}(\beta_j, \mathsf{transcript}_{1,j-1})$
4. $y_{x,j} := \mathsf{POK}(x_j, \mathsf{transcript}_{1,j-1})$
5. For each $i \in [1..2n-2]$, $[x^i]^{\mathbf{j}} := x_j^i \cdot [x^i]^{\mathbf{j-1}}$
6. For each $i \in [0..n-1]$, $[\alpha x^i]^{\mathbf{j}} := \alpha_j x_j^i \cdot [x^i]^{\mathbf{j-1}}$
7. For each $i \in [0..n-1]$, $[\beta x^i]^{\mathbf{j}} := \beta_j x_j^i \cdot [x^i]^{\mathbf{j-1}}$

Let $J-1$ be the time-slot where P_N sends their message. Let $(x', \alpha', \beta') := \mathsf{RB}(J, 3)$. We define

1. $\left[x^i\right] := x'^i \cdot [x^i]^{\mathbf{N}}$, $i \in [1..2n-2]$.
2. $\left[\alpha x^i\right] := \alpha' x'^i \cdot [\alpha x^i]^{\mathbf{N}}$, $i \in [0..n-1]$.
3. $\left[\beta x^i\right] := \beta' x'^i \cdot [\beta x^i]^{\mathbf{N}}$, $i \in [0..n-1]$.

Verification: The protocol verifier computes for each $j \in [N]$

$r_{\alpha,j} := \mathcal{R}([\alpha_j]_1, \mathsf{transcript}_{1,j-1}), r_{\beta,j} := \mathcal{R}([\beta_j]_1, \mathsf{transcript}_{1,j-1}), r_{x,j} := \mathcal{R}([x_j]_1, \mathsf{transcript}_{1,j-1})$,

and checks for each $j \in [N]$ that

1. $\mathsf{CheckPOK}([\alpha_j]_1, \mathsf{transcript}_{1,j-1}, y_{\alpha,j})$
2. $\mathsf{CheckPOK}([\beta_j]_1, \mathsf{transcript}_{1,j-1}, y_{\beta,j})$
3. $\mathsf{CheckPOK}([x_j]_1, \mathsf{transcript}_{1,j-1}, y_{x,j})$
4. $\mathsf{consistent}([\alpha]^{\mathbf{j-1}} - [\alpha]^{\mathbf{j}}; (r_{\alpha,j}, y_{\alpha,j}))$,
5. $\mathsf{consistent}([\beta]^{\mathbf{j-1}} - [\beta]^{\mathbf{j}}; (r_{\beta,j}, y_{\beta,j}))$,
6. $\mathsf{consistent}([x]^{\mathbf{j-1}} - [x]^{\mathbf{j}}; (r_{x,j}, y_{x,j}))$,
7. For each $i \in [1..2n-2]$, $\mathsf{consistent}([x^{i-1}]^{\mathbf{j}} - [x^i]^{\mathbf{j}}; [x]^{\mathbf{j}})$.
8. For each $i \in [1..n-1]$, $\mathsf{consistent}([x^i]^{\mathbf{j}}_1 - [\alpha x^i]^{\mathbf{j}}; [\alpha]^{\mathbf{j}})$.
9. For each $i \in [1..n-1]$, $\mathsf{consistent}([x^i]^{\mathbf{j}}_1 - [\beta x^i]^{\mathbf{j}}; [\beta]^{\mathbf{j}})$.

6.2 Linear combinations between rounds

For $i \in [0..n-2]$, we compute as linear combinations of $\left\{ [x^i]_1 \right\}_{i\in[0..2n-2]}$ the element

$$H'_i := \left[t(x)x^i\right]_1.$$

Let $\omega \in \mathbb{F}_p$ be a primitive root of unity of order $n = 2^t$, in code n is typically the first power of two larger or equal to the circuit size.

For $i \in [1..n]$, we define L_i to be the i'th Lagrange polynomial over the points $\{\omega^i\}_{i\in[n]}$. That is, L_i is the unique polynomial of degree smaller than n, such that $L_i(\omega^i) = 1$ and $L_i(\omega^j) = 0$, for $j \in [n] \setminus \{i\}$. For $x \in \mathbb{F}_p^*$, we denote by $\mathrm{LAG}_x \in \mathbf{G}^n$ the vector

$$\mathrm{LAG}_x := ([L_i(x)])_{i\in[n]}.$$

LAG_x can be computed in an FFT using $O(n \log n)$ group operations from $\left\{ [x^i] \right\}_{i\in[0..n-1]}$, as decribed in Section 3.3 of [7]. Similarly, since the FFT is linear, using exactly the same operations, but only on the $\mathbb{G}_1$ coordinate and starting from $\left\{ [\alpha x^i]_1 \right\}_{i\in[0..n-1]}$ and $\left\{ [\beta x^i]_1 \right\}_{i\in[0..n-1]}$, we obtain $(\alpha\cdot\mathrm{LAG}_x)_1$ and $(\beta\cdot\mathrm{LAG}_x)_1$.

Now, as the QAP polynomials $\{u_i, v_i, w_i\}_{i\in[0..m]}$ are typically[8] each a linear combination of at most three different L_i, we can now compute using $O(m)$ group operations the elements $\{[\beta u_i(x)]_1\}_{i\in[0..m]}$, $\{[\alpha v_i(x)]_1\}_{i\in[0..m]}$ and $\{[w_i(x)]\}_{i\in[0..m]}$.

[8]This is the case in the reduction of arithmetic circuits to QAPs; in general the cost of this step is $O(a)$ operations where a is the total number of non-zero coefficients in one of the QAP polynomials.

Finally, we compute as linear combinations, for $i \in [\ell + 1..m]$, the element

$$K'_i := [\beta u_i(x) + \alpha v_i(x) + w_i(x)]_1 \,.$$

We also output, as linear combinations of LAG_x the elements $\{[u_i(x)]_1\}_{i\in[0..m]}$ and $\{[v_i(x)]_2\}_{i\in[0..m]}$ (To allow faster prover computation).

6.3 Round two

For $i \in [\ell + 1..m]$, denote

$$K_i := \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}.$$

For $i \in [0..n-2]$, denote

$$H_i := \frac{t(x)x^i}{\delta}.$$

We need to compute

$$\mathrm{M}_2 = \left\{ [\delta]\,, \{[K_i]_1\}_{i\in[\ell+1..m]}\,, \{[H_i]_1\}_{i\in[0..n-2]} \right\}.$$

Initialization: We initialize

1. $[K_i]^{\mathbf{0}} := K'_i$, $i \in [\ell + 1..m]$,
2. $[H_i]^{\mathbf{0}} := H'_i$, $i \in [\ell + 1..m]$.
3. $[\delta]^{\mathbf{0}} := \mathbf{g}$.

Computation: For $j \in [N]$, P_j outputs

1. $[\delta_j]_1$.
2. $y_{\delta,j} := \mathsf{POK}(\delta_j, \mathsf{transcript}_{2,j-1})$
3. $[\delta]^{\mathbf{j}} := [\delta]^{\mathbf{j-1}}/\delta_j$.
4. For each $i \in [\ell + 1..m]$, $[K_i]^{\mathbf{j}} := ([K_i]^{\mathbf{j-1}})/\delta_j$.
5. For each $i \in [0..n-2]$, $[H_i]^{\mathbf{j}} := ([H_i]^{\mathbf{j-1}})/\delta_j$.

In the end, we define Let $J - 1$ be the time-slot where P_N sends their message. Let $\delta' := \mathsf{RB}(J, 1)$. We define

1. $[\delta] := [\delta]^{\mathbf{N}}/\delta'$
2. $[K_i]_1 := [K_i]^{\mathbf{N}}/\delta'$
3. $[H_i]_1 := [H_i]^{\mathbf{N}}/\delta'$

Verification: The protocol verifier computes for each $j \in [N]$

$$r_{\delta,j} := \mathcal{R}([\delta_j]_1, \mathsf{transcript}_{2,j-1}),$$

and for each $j \in [N]$ checks that

1. $\mathsf{CheckPOK}([\delta_j]_1, \mathsf{transcript}_{2,j-1}, (r_{\delta,j}, y_{\delta,j}))$.
2. For $j \in [N]$, $\mathsf{consistent}([\delta]^{\mathbf{j-1}} - [\delta]^{\mathbf{j}}; [\delta_j])$.
3. For each $i \in [\ell+1..m]$, $j \in [N]$, $\mathsf{consistent}([K_i]^{\mathbf{j}} - [K_i]^{\mathbf{j-1}}; [\delta_j])$.
4. For each $i \in [0..n-2]$, $j \in [N]$, $\mathsf{consistent}([H_i]^{\mathbf{j}} - [H_i]^{\mathbf{j-1}}; [\delta_j])$.

7 BLS12-381

The most common pairing-friendly elliptic curve construction used in zk-SNARK software is a Barreto-Naehrig [4] (BN) construction with a 254-bit base field and group order, as designed in [6]. That construction equipts $\mathbb{F}_p$ with a large 2^n root of unity for efficient polynomial evaluation. Although the construction originally targeted the 128-bit security level, recent optimizations to the Number Field Sieve algorithm [13] have reduced its concrete security.

Subsequent analysis [14] recommended that BN curves and Barreto-Lynn-Scott (BLS) curves [3] with embedding degree $k = 12$ have approximately 384-bit base fields in order to target 128-bit security. BN curves are thus not ideal for our purposes, as these larger base fields are accompanied by similarly larger group orders, which substantially increases the cost of multi-exponentiation and fast-fourier transforms and harms the usability of protocols that use $\mathbb{F}_p$ to encode keying material. BLS12 curves with 384-bit base fields, in contrast, give rise to 256-bit group orders, making them ideal for use with zk-SNARKs. In more conservative contexts, the larger constructions proposed in [2] are recommended.

BLS curves with $k = 12$ are parameterized by an integer x. The existing BN curve has $2^{28}|p-1$ to ensure a 2^{28} root of unity is available. We target the same by ensuring that $2^{14}|x$. We target prime p of less than 2^{255} in order to accomodate efficient approximation algorithms and reductions. We desire efficient extension field towers and twisting isomorphisms, following recommendations from [1]. In addition, we desire x of small Hamming weight for optimal pairing efficiency.

The largest construction with smallest Hamming weight that meets our requirements is $x = -2^{63} - 2^{62} - 2^{60} - 2^{57} - 2^{48} - 2^{16}$, which we name **BLS12-381**. This curve exists within a subfamily of curves, as in [8], which have immediately determined curve parameters. We provide an implementation of this curve in Rust. [16]

Acknowledgements

We think Paulo Barreto for helpful feedback about the BLS12-381 elliptic curve. We thank Daira Hopwood for helpful comments.

References

[1] Diego F. Aranha, Laura Fuentes-Castaneda, Edward Knapp, Alfred Menezes, and Francisco Rodriguez-Henriquez. *Implementing Pairings at the 192-bit Security Level*. Cryptology ePrint Archive, Report 2012/232. `http://eprint.iacr.org/2012/232`. 2012.

[2] Razvan Barbulescu and Sylvain Duquesne. *Updating key size estimations for pairings*. Cryptology ePrint Archive, Report 2017/334. `http://eprint.iacr.org/2017/334`. 2017.

[3] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. *Constructing Elliptic Curves with Prescribed Embedding Degrees*. Cryptology ePrint Archive, Report 2002/088. `http://eprint.iacr.org/2002/088`. 2002.

[4] Paulo S. L. M. Barreto and Michael Naehrig. *Pairing-Friendly Elliptic Curves of Prime Order*. Cryptology ePrint Archive, Report 2005/133. `http://eprint.iacr.org/2005/133`. 2005.

[5] 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*. 2015, pp. 287–304.

[6] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. *SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge*. Cryptology ePrint Archive, Report 2013/507. `http://eprint.iacr.org/2013/507`. 2013.

[7] S. Bowe, A. Gabizon, and M. D. Green. "A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK". In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 602. URL: `http://eprint.iacr.org/2017/602`.

[8] Craig Costello, Kristin Lauter, and Michael Naehrig. *Attractive Subfamilies of BLS Curves for Implementing High-Security Pairings*. Cryptology ePrint Archive, Report 2011/465. `http://eprint.iacr.org/2011/465`. 2011.

[9] Georg Fuchsbauer. *Subversion-zero-knowledge SNARKs*. Cryptology ePrint Archive, Report 2017/587. `http://eprint.iacr.org/2017/587`. 2017.

[10] 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.* 2013, pp. 626–645.

[11] J. Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II.* 2016, pp. 305–326.

[12] Joe Kilian. "A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)". In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada.* 1992, pp. 723–732. DOI: `10.1145/129712.129782`. URL: `http://doi.acm.org/10.1145/129712.129782`.

[13] Taechan Kim and Razvan Barbulescu. *Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case.* Cryptology ePrint Archive, Report 2015/1027. `http://eprint.iacr.org/2015/1027`. 2015.

[14] Alfred Menezes, Palash Sarkar, and Shashank Singh. *Challenges with Assessing the Impact of NFS Advances on the Security of Pairing-based Cryptography.* Cryptology ePrint Archive, Report 2016/1102. `http://eprint.iacr.org/2016/1102`. 2016.

[15] Silvio Micali. "Computationally Sound Proofs". In: *SIAM J. Comput.* 30.4 (2000), pp. 1253–1298. DOI: `10.1137/S0097539795284959`. URL: `https://doi.org/10.1137/S0097539795284959`.

[16] *pairing.* URL: `https://github.com/ebfull/pairing` (visited on 2017-10-14).

[17] Ethereum Team. *Byzantium HF Announcement.* `https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/`. October 2017.

[18] *Zcash.* URL: `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf` (visited on 2017-10-14).