Formalizing Soundness Proofs of SNARKs

Bolton Bailey, Andrew Miller

May 9, 2023

Abstract

Succinct Non-interactive Arguments of Knowledge (SNARKs) have seen interest and development from the cryptographic community over recent years, and there are now constructions with very small proof size designed to work well in practice. A SNARK protocol can only be widely accepted as secure, however, if a rigorous proof of its security properties has been vetted by the community. Even then, it is sometimes the case that these security proofs are flawed, and it is then necessary for further research to identify these flaws and correct the record [35, 51].

To increase the rigor of these proofs, we turn to formal methods. Focusing on the soundness aspect of a widespread class of SNARKs, we formalize proofs for six different constructions, including the well-known Groth '16. Our codebase is written in the Lean 3 theorem proving language, and uses a variety of techniques to simplify and automate these proofs as much as possible.

1 Introduction

Over the past decade, cryptographic research has produced Succinct Non-interactive Arguments of Knowledge (SNARKs), which allow a prover to demonstrate knowledge of a witness corresponding to a statement in some NP relation. The most succinct of these constructions [37, 52, 38] allow a proof consisting of a $O(1)$-sized message. SNARKs promise to have many applications in verifiable computation, blockchains, and identity management [16, 54, 45].

Since practitioners are applying this new technology for tasks where security is a main concern, it is important for them to have confidence in the cryptographic properties of the protocols they implement. To this end, the academic review process ensures that SNARKs published in the literature come with mathematical proofs of security properties. But in practice, errors arise. Research from Parno [51] and Gabizon [35] has identified flaws in the soundness of SNARKs in the `libsnark` library originating from [17], which was a modification of the Pinocchio SNARK [52].

To prevent errors like this from happening in the future, we look to apply formal methods to obtain guarantees on correctness. We focus on the class of "pairing-based" SNARKs to which the above implementations belong and which achieve the fully-succinct $O(1)$ proof-message size previously mentioned. In particular, we deal with SNARKs that work in the structured reference string (SRS) model: All parties are assumed to have access to a collection of elements of a pairing-friendly cryptographic group, generated by some trusted third party.

Our work comprises completed soundness proof compilations for six SNARKs:

- **GGPR** [37]: The first SNARK to encode computations using Quadratic Arithmetic Programs (QAPs), which is the main NP-Complete language to which SNARKs are reduced in this line of work.
- **Pinocchio** [52]: A modification of GGPR which improved efficiency.
- **Groth '16** [38]: A widely-cited SNARK which reduced the proof size to only three group elements.
- **Baghery et al.** [6]: A paper which presented a version of Groth '16 for Type-III pairings.
- **Lipmaa** [42]: Another version of Groth '16, redesigned with an eye to a simulation-extractability property.
- **Baby SNARK** [48]: A simplified version of Groth '16 constructed for educational purposes, designed to have an easy-to-follow soundness proof.

Additionally, we identified and formalized a variety of techniques for manipulating constructions such as these, in order to make our work more extensible.

1.1 The Lean Theorem Prover

Our formalization is carried out in Lean 3 [29]. Lean is a programming language and an interactive theorem prover - it allows a user to construct a formal proof of a mathematical proposition by writing computer code.

The main part of the soundness proofs we complete consists of reasoning about collections of equalities of multivariable polynomials. We base our work on `mathlib` [1], the open-source Lean 3 library that implements structures and lemmas from much of undergraduate-level mathematics, including finite fields and multivariable polynomials.

Lean is built with a metaprogramming facility which allows the user to write "tactic" code which can construct proofs using automation. We take advantage of this in the course of our work - we implement a recursive tactic to resolve subgoals consisting of systems of equations over a integral domain. Additionally, we make use of Lean's simplification attribute feature. This allows the user to tag lemmas under a label so that these can then be passed to the Lean `simp` tactic, which iteratively applies these lemmas to simplify an expression. We construct a variety of Lean simplification attributes for normalizing statements about polynomials and their coefficients. The end product is a system which is capable of compiling an end-to-end proof of the knowledge-soundness theorem for a SNARK.

1.2 Related Work

Cryptography in general and proof systems in particular have been of great interest to the formal verification community. In this section, we will go over some of these contributions and their relevance to the problem at hand.

Lean itself, being relatively new and not focused on cryptography, is a somewhat rare choice for formalization of cryptographic protocols. Nevertheless considered it appropriate for our work here. The comprehensive and well-integrated `mathlib` library and its implementation of numerous lemmas about multivariable polynomials makes Lean ideal in the setting of succinct proofs. The work [4], also uses Lean and `mathlib` and is the only other work of which we are aware that formalizes statements about a general-purpose succinct proof systems. In the case of that paper, there is a full formalization of the Cairo machine, with a Lean proof of *correctness* for its execution, (rather than soundness). It is encouraging to see another aspect of a proof system formalized, and it suggests that a full proof of completeness and soundness of some system could be within grasp. It's worth noting that extensions to Lean could make it a better ground for cryptography in the future: In the time our project has been underway, `mathlib` has added an implementation of elliptic curves [?], as well as a tactic [18] for solving Gröbner basis problems. This tactic, `polyrith`, works by calling a web interface to a Sage Gröbner-basis solver - while the problems that our code generates are too large to be handled by `polyrith`, it's promising that a more general version of the problem is being worked on, and in the future it could make our system more flexible.

Coq [41] is the theorem proving language perhaps most related to Lean - both languages are based on dependent type theory. The Certicrypt Coq library [11] provides tools for developing formalizations of cryptographic protocols. Of the protocols that have been formalized in this framework, more relevant are likely [5], which formalizes the portion of the zero-knowledge stack which compiles relations into the necessary form to be handled by a SNARK, and [12], which uses Certicrypt to formalize Σ-protocols, a class of proof systems involving three rounds of communication. Besides this, there have been several applications of Coq to other (non-proof-system) cryptographic protocols, including the Proof-of-Stake consensus system [58], mix nets [40] and signature schemes [57]. Efforts in Coq to formalize broader classes of cryptographic techniques include SSProve [2], which formalizes a modularization of cryptographic proofs, the formalization [50] of some of the number-theoretic underpinnings of cryptography, and [9], which formalizes the generic model for group-based cryptography and random oracle model . This last is relevant to our own application - the generic group model [55] is related to the algebraic group model that our own work uses, in that both seek to codify an adversary's interaction with a cryptographic group. External to cryptography, Coq also has well developed math libraries, including the [53] tactic for Gröbner bases - the decision to use Lean over Coq for this project comes down to a few matters of convenience, such as Lean's ability to express tactics themselves in Lean.

EasyCrypt is another proof assistant. Like Coq, it is written in OCaml, but EasyCrypt is designed specifically with cryptography in mind. EasyCrypt allows one to reason about probabilistic computation, which is convenient for the formalization of game-based cryptographic proofs. Works in EasyCrypt relevant to proof systems include [3], which formalizes the "MPC-in-the-head" paradigm for zero knowledge and [33], which formalizes a variety of protocols including protocols for proof of knowledge of quadratic residues, discrete logarithms and Hamiltonian cycles. The first two of the latter are protocols for specific problems not known to be NP-complete, so they cannot be turned into general-purpose proof systems. The Hamiltonian cycle and MPC-in-the-Head approaches are general purpose, but these proof systems are not succinct - they require the verifier to do work linear or more in the problem. EasyCrypt has also been used for many applications beyond proof systems, including verifications of Multi-party computation itself [31, 39], post-quantum cryptography [8], Pedersen commitments [47], electronic voting [28], and key exchange [10]. The framework also has formalized some more general techniques, including Canetti's Universal Composability framework [22] in [23] and Brzuska et al.'s State separating proofs [19] in [30].

On the other end of the spectrum from strongly-typed languages like Lean and Coq is the Isabelle [49] proof system. The CryptHOL [14] framework is the main mode for cryptography in this language. Butler et al. [21] have done work to formalize Σ-protocols in this framework. CryptHOL additionally has modules for constructive cryptography [43, 13], and oblivious transfer [20]. ACL2, a Lisp-based theorem prover, has been used to verify the Ethereum Recursive Length Prefix encoding scheme [27].

It's also worth mentioning that there is some work done in the space of verification for proof systems which does not prove theorems formally, but nevertheless uses automated processes to check the construction of protocols. These often focus on the circuit compilation component of the SNARK toolchain: Ecne [59] is a Julia project which analyzes Rank-1 Constraint Systems to determine if their outputs are uniquely determined. Picus [24] is a symbolic VM for formal verification of Rank-1 Constraint Systems. [26] describes the Leo language, a DSL for writing SNARK programs with a facility for ACL2 verification.

To summarize, formal modelling of cryptography, like cryptography itself, is diverse both in its scope and in its techniques. For more in-depth surveys, the reader can consult the Systemization of Knowledge papers of [7], [46] or [56]

2 Overview of Pairings and the Algebraic Group Model

As we have mentioned, we focus on *soundness* proofs for pairing-based SNARKs in the structured reference string model. In the field of cryptographic proofs, soundness guarantees that a prover who consistently convinces a verifier of a statement can only do so because they possess knowledge of a witness corresponding to that statement. Typically, this is done by defining an *extractor* that examines the prover and extracts this knowledge. This is challenging for SNARKs, since (unlike in interactive proof systems) the prover could in principle generate the proof deterministically, and so the extractor sees only one valid proof, and this is very little information to work with when trying to recover a witness which may be asymptotically larger. To get around this, we assume that the construction of the proof is intimately tied to a larger piece of random data. This is the structured reference string, which is produced before the proof phase of the protocol begins.

All the SNARKs that our system analyzes assume that the SRS is a collection of elliptic curve group elements. These elements each come from one of three predefined elliptic curves of prime order p, G_1, G_2, G_T, and in each of these groups the discrete logarithm problem is assumed to be hard. The three curves form a *pairing*. That is, they come together with a nontrivial pairing operation $e : G_1 \times G_2 \to G_T$ which satisfies the bilinearity property: $e(g^a, h^b) = e(g, h)^{a \cdot b}$. The SRS elements are always equal to generators of these curves raised to the power of particular multivariable polynomials over $\mathbb{F}_p$. The values of the variables of these polynomials are sometimes called *toxic waste* to reflect the fact that if they become known to the prover, the soundness of the proof system can be broken.

The prover can create a linear combination of the SRS polynomials using curve operations to make a proof-message, and the verifier can then do computations using the pairing to verify the proof-message. In order to formally check the soundness of such a construction, we must make a cryptographic assumption that limits what the prover is capable of doing in assembling the proof-message. In particular, the soundness of these SNARKs can be very neatly formalized with the *Algebraic Group Model* assumption [34], which essentially says that the prover can only output group elements which are indeed linear combinations of SRS elements. The extractor then gets accesss to the coefficients of this linear combination and uses it to reconstruct the witness.

To summarize, formally proving the soundness of a pairing-based SNARK in the AGM model consists of:

- Identifying the toxic waste elements
- Formalizing the SRS elements as polynomials over the toxic waste elements
- Formalizing the proof elements as parameterized linear combinations of SRS elements
- Formalizing the verification equations as equations over the proof elements and SRS elements
- Formalizing the satisfaction condition.
- Formally proving that the verification equations holding implies that the extractor obtains a valid witness.

3 Lean Formalization of the Soundness Proof

In this section, we discuss the specific techniques we used to create a system which is capable of formalizing and verifying, in Lean, the soundness of pairing-based SNARKs. We will also describe some performance considerations, using our Type-III Groth '16 Lean-proof as an example.

3.1 Stage 0: Multivariate Polynomial Formalization

One consideration in our formalization is the data type used to represent the multivariable polynomials in the toxic waste elements which appear throughout the proof. We discuss a few options for representing them, and the benefits and drawbacks.

Stage 0	Proof state consists of polynomial equations in the trapdoor elements and prover coefficients		$(A\alpha+B\beta)(C\alpha+D\beta) = E\alpha\beta$
Stage 1a	Polynomial put in normal form	`simp with polynomial_nf`	$AC\alpha^2+(AD+BC)\alpha\beta+BD\beta^2 = E\alpha\beta$
Stage 1b	Coefficients are isolated	`h := congr_arg (coeff (...)) eqn`	coeff $\alpha\beta$ $(AC\alpha^2+(AD+BC)\alpha\beta+BD\beta^2$ = coeff $\alpha\beta$ $E\alpha\beta$ ...
Stage 1c	Expression broken down into term-by-term coefficient comparisons	`simp only with coeff_simp at h`	if $\alpha\beta = \alpha^2$ then AC else 0 + if $\alpha\beta = \alpha\beta$ then $AD+BC$ else 0 + if $\alpha\beta = \beta^2$ then BD else 0 = if $\alpha\beta = \alpha\beta$ then E else 0 ...
Stage 1d	Coefficient comparisons decided, leaving proof state of polynomial equations in the prover coefficients	`simp only with finsupp_simp at h`	$AC = 0$ $AD + BC = E$ $BD = 0$
Stage 2a	Polynomials are simplified algebraically	`integral_domain_tactic`	$A = 0$ or $C = 0$ $AD + BC = E$ $B = 0$ or $D = 0$
Stage 2b	Proof state consists of simple equations of prover coefficients	`integral_domain_tactic`	$BC = E$ or $AD = E$

Table 1: Describing the stages of a proof. The left row gives an example for a toy (incomplete) SNARK illustrating the type of the hypotheses at each stage.

Structured Reference String Components The Groth '16 SNARK uses 5 toxic-waste values in the setup phase:

$$\alpha, \beta, \gamma, \delta, x$$

These values are sampled from $\mathbb{F}^*$. From these, a collection of SRS elements are generated: The first four samples elements appear as the first four SRS elements

$$\alpha, \beta, \gamma, \delta,$$

and the remainder of the SRS elements consist of four indexed collections, the sizes of which depend on the QAP (and ultimately, the size of the circuit on which the SNARK is instantiated).

$$\left\{x^i\right\}_{i=0}^{n-1}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right\}_{i=0}^{l},$$
$$\left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right\}_{i=l+1}^{m}, \left\{\frac{x^i t(x)}{\delta}\right\}_{i=0}^{n-2}$$

As we will later see, while these latter collections consist of multiple elements, it will be the case that whenever one of these elements appears in the proof, *it is always as part of a sum indexed over all members of the collection.* It therefore turns out to be convenient to track this partition of SRS elements into these 8 sets during the course of the proof: We refer to them as SRS components . Table 4 shows how many components there are for each of the SNARKs we considered.

Laurent Polynomials We could almost say that all the SRS elements are multivariable polynomials in the setup values, but this is not quite true: γ and δ appear in the denominator of some of these expressions. Like other SNARK schemes (e.g. Sonic [44]), Groth '16 creates its SRS using the more general notion of *Laurent polynomials.* Laurent polynomials are permitted to have terms with negative exponents, but they have many of the same useful properties as regular polynomials (in particular, a version of the Schwartz-Zippel lemma holds for them).

When trying to formalize the Groth '16 soundness proof while staying as close as possible to the theorem statement given in the paper, we encounter the problem that `mathlib` does not currently have an implementation of multivariable Laurent polynomials. To get around this, we note that it is really not necessary for Groth '16 and these other constructions to actually formalize their results using Laurent polynomials: One can simply multiply all the SRS elements in the construction by the minimum order of every sampled field element to get a SNARK which is functionally equivalent, and which can be formalized in terms of (nonnegative-exponent-term) multivariable polynomials. In the case of Groth '16, this means multiplying all the SRS elements through by $\gamma\delta$. Indeed, it can be shown that the soundness property of the Laurent version follows from the soundness of the non-Laurent version, and we can formalize this fact in Lean.

High-Degree Variables Let us look again at the 8 SRS components, now all multiplied through by $\gamma\delta$.

$$\alpha\gamma\delta, \beta\gamma\delta, \gamma^2\delta, \gamma\delta^2, \{\gamma\delta x^i\}_{i=0}^{n-1}, \{\delta(\beta u_i(x) + \alpha v_i(x) + w_i(x))\}_{i=0}^{l},$$
$$\{\gamma(\beta u_i(x) + \alpha v_i(x) + w_i(x))\}_{i=l+1}^{m}, \{\gamma(x^i t(x))\}_{i=0}^{n-2}$$

Note that in all of these elements, the degrees of α, β, γ, and δ are at most 1, 1, 2 and 2 respectively. Like Pinocchio and others, Groth '16 has the property that there is actually only a single sample element, x, for which the maximum degree of the element depends on the circuit. This leads to an idea that proves crucial in later stages of the automated proof: Instead of formalizing these values as multivariable polynomials in five variables over $\mathbb{F}$, we formalize them as *multivariable polynomials over polynomials.* We use the type `mv_polynomial vars (polynomial F)` where `vars` has four elements corresponding to the bounded degree samples,

```
@[derive decidable_eq]
inductive vars : Type
| α : vars
| β : vars
| γ : vars
| δ : vars
```

We also derive the decidability of equality, to allow us to automatically determine if $\mathbb{N}$-valued functions on `vars` (which correspond to terms of a polynomial over `vars`) are equal.

3.2 Stage 1: Coefficients of the Equations

To prove a SNARK sound in the AGM is to assume that the proof elements have AGM representations, and prove that if these representations satisfy the equations that the protocol specifies they should satisfy, the encoding of the QAP is satisfied. These AGM representations are expressions which include some free variables as coefficients in a linear combination of SRS elements. The type of these expressions is therefore the multivariable polynomial type `mv_polynomial vars (polynomial F)` we defined above. Thus, to formalize this proof in lean, we must take a collection of `mv_polynomial vars (polynomial F)` equality expressions and prove that these equations imply another equation.

A pair of multivariable polynomials are equal if and only if their coefficients are equal. Thus, we can convert our assumptions into a collection of assumptions about the equalities of coefficients of our `mv_polynomial`s. Here is where it becomes important that we have formalized these multivariable polynomials using the type `mv_polynomial vars (polynomial F)`, and ensured that all the orders of variables in the outer polynomial have bounded degree. Because of this, we can extract equalities of coefficients of the finitely many terms for which all the degrees are below this bound.

As we mentioned above in the case of Groth '16, there are 4 bounded toxic waste elements, $\alpha, \beta, \gamma, \delta$ with maximum degrees 1, 1, 2 and 2. Since the single equality tested in the protocol only uses pairings of linear combinations of these elements, the terms in the output can take on:

- One of 3 arities in α, $(1, \alpha, \alpha^2)$
- One of 3 arities in β, $(1, \beta, \beta^2)$
- One of 5 arities in γ, $(1, \gamma, \gamma^2, \gamma^3, \gamma^4)$
- One of 5 arities in δ, $(1, \delta, \delta^2, \delta^3, \delta^4)$

So there are at most $3 \times 3 \times 5 \times 5 = 75$ terms with nonzero coefficient. Inspection of the Lean proof state after extracting all of these into individual equations shows that only 51 of these are actually nonzero, and only a subset of those are actually necessary to complete the natural-language proof. In particular, the presentation in Baghery et al. uses only 14 different coefficients ([6], Theorem 2).

To isolate these equations automatically, we create three Lean simplification attributes:

- `polynomial_nf`: This puts the polynomial equations into a sum-of-products normal form, with the monomials grouped together.
- `coeff_simp`: This takes an expression consisting of the `mv_polynomial.coeff` function (which takes an argument specifying a monomial and returns the coefficient of that monomial in the given polynomial) evaluated on some normal-form polynomial. The attribute reduces this expression to a form involving compositions of if-then-else statements depending on the equality of various monomials.
- `finsupp_eq`: Finally, this decides the equality of the monomial equivalences to reduce the if-then-else expressions to simple expressions of equality between (sums over) `polynomial F` values.

All told, for each of the 14 terms, there are 10 LOC to construct the proof of the equality of the coefficients. For the Type-III Groth '16, and this takes approximately 40 seconds on a 2.9 GHz Quad-Core Intel Core i7. See Table 1 for a visualisation of the effect of each of these steps.

3.3 Stage 2: Mutual Simplification

At this point we have our collection of equations over `polynomial F`. Our goal is also an equation of values having type `polynomial F`. All that remains is to prove our hypotheses imply this goal (the reader can see a snapshot of the proof state showing what these equations look like in Appendix A). We must now simplify these equations, hopefully in an automated a way as possible.

One convenient fact is that many of the indexed sums which occur over u, v, w and multiplications thereof with other polynomials occur multiple times throughout the hypotheses, and since the validity of the SNARKs depend on u, v, w being general, the proofs of validity do not require any structure on these sums. We can therefore treat these summations as atoms, and deal with equations which are simple additions and multiplications of these atoms.

A further convenience is that, because the SNARK equations arise through pairings, our hypotheses are all quadratic in these atoms. In fact, many of the equations are of the form `A * B = 0` for atoms `A` and `B`. This is by design, as it is necessary for the proofs to leverage the fact that the product of two values equating to zero implies at least one of the multiplicands is zero. This leads us to formulate the following approach to simplifying the goal: We use the fact (inferred by Lean's typeclass system) that a polynomial ring over a field is an integral domain, and we simplify all equations of the form `A * B = 0` to `A = 0 or B = 0`. We can then split these hypotheses into two cases and prove the goal for each case, simplifying our hypotheses by rewriting `A` or `B` to 0, and carry on this process until we are left with a collection of goals that cannot be simplified through these rules. To facilitate this, we wrote a tactic `integral_domain_tactic`, which carries out the above simplifications and calls itself recursively until it reaches a point where it can make no more progress:

```
meta def integral_domain_tactic : tactic unit := do
  trace "Call to integral_domain_tactic",
  -- Factor statements of the form a * b = 0 into a = 0  b = 0
  -- and mutually simplify the resulting hypotheses.
  `[simp only [*] with integral_domain_simp
    at * {fail_if_unchanged := ff}],
  -- Eliminate true and false hypotheses, halt if done
  try `[cases_type* true false],
  _::_ ← get_goals | skip,
  -- Identify disjunctions
  try `[clear found_zero],
  cases_success <- try_core `[cases <_  _> with found_zero found_zero],
  -- Do case work on disjunctions
  match cases_success with
  | some _ := all_goals' `[done <|> id { integral_domain_tactic }]
  | none := skip
  end
```

We can then either solve these goals by hand, either one at a time, or by dispatching multiple subgoals at once using built-in Lean tactics, until none are left.

In the Type-III Groth 16, `integral_domain_tactic` deals with 37 different branches of cases, after which only an additional 20 LOC of tactics are needed to close remaining goals generated. This takes approximately 90 seconds on our 2.9 GHz Quad-Core Intel Core i7 processor.

4 Evaluation

Table 4 shows data about the sizes of various parameters of the various SNARKs and the time it takes Lean to verify the proofs. In each case, the proof time is dominated by the mutual simplification phase, as one might expect, given that this this phase requires case analyses that potentially blow up exponentially. The most expensive SNARK to verify is the generic Groth '16, which takes almost two orders of magnitude longer than the second longest, which is the Type III variant. This is consistent with the fact that there are more symmetries in the Groth '16 SNARK, which are dealt with in the paper via a without-loss-of-generality argument, but which `integral_domain_tactic` handles by brute force.

Name	# Toxic Samples	# Proof elements	# SRS Components	# Checks	Compile Time
GGPR [37]	5	7 (6*)	19	5 (4*)	140.61 s
Pinocchio [52]	8	8	21	5	342.89s
Groth '16 [38]	5	3	8	1	13741.86s
Baghery et al. [6]	5	3	7 and 4	1	552.67s
BabySNARK [48]	3	3	4	2	74.98s
Lipmaa [42]	2	3	7 and 4 and 1	1	81.82s

Table 2: Data on different SNARK variants. *GGPR includes in the paper a proof element and a check which are not strictly necessary for the soundness proof.

5 Comments on Proof Exposition

In this section, we discuss some of the proofs of the SNARKs we have covered, not as we formalized them, but as they appear in the original references on which our formalizations were based. For each of these references we managed to produce a formalization and proof that was fully checked by the Lean kernel, so it would not be appropriate to call any of these constructions broken. However, the process of creating these formalizations was not always smooth, in part because there are a few places in which the arguments presented in the papers are misleading, or even incorrect. In the interest of a better understanding of these proofs by the community as a whole, we will take the time to explain why we found these proofs confusing and the impact it had on our proof efforts.

5.1 Pinocchio

First, we discuss Pinocchio. Specifically, we refer to the Protocol 2 of that paper (that is, the SNARK designed to work for regular QAPs rather than strong QAPs), and the subsequent theorem 1, which provides the "Security Intuition". To recap the design of this SNARK, three of the proof elements in it are meant to be derived directly from the witness, namely $g^{V_{mid}}, g^{W_{mid}}$, and $g^{Y_{mid}}$. Specifically, these are meant to be constructed as a linear combination of, respectively, the three sets of SRS elements $\{g_v^{v_k(s)}\}_{k \in I_{mid}}$, $\{g_w^{w_k(s)}\}_{k \in I_{mid}}$, and $\{g_y^{y_k(s)}\}_{k \in I_{mid}}$. Three of the other proof elements ($g^{V'_{mid}}$, $g^{W'_{mid}}$, and $g^{Y'_{mid}}$) are likewise derived from $\{g_v^{\alpha_v v_k(s)}\}_{k \in I_{mid}}$, $\{g_w^{\alpha_w w_k(s)}\}_{k \in I_{mid}}$, and $\{g_y^{\alpha_y y_k(s)}\}_{k \in I_{mid}}$. Three of the checks the verifier then carries out use these proof elements, $e(g_v^{V'_{mid}}, g) = (g_v^{V_{mid}}, g^{\alpha_v})$, $e(g_w^{W'_{mid}}, g) = (g_w^{W_{mid}}, g^{\alpha_w})$, and $e(g_y^{Y'_{mid}}, g) = (g_y^{Y_{mid}}, g^{\alpha_y})$.

The reason given for these checks, as stated in the paper, is to "Check that the linear combinations of $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{Y}$ are in their appropriate spans". These spans are identified as "the $v_k(x)$'s, $w_k(x)$'s, and $y_k(x)$'s, respectively". Somewhat confusing is whether this is meant to be just the k indices in the witness or in the statement too - A hiccup is that if the witness and statement polynomials are not linearly independent, then an AGM adversary can add these linear combinations to their proof coefficients, leading to a verifying set of coefficients for which the coefficients of the witness polynomials in $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{Y}$ do not match the witness itself. This does not actually change the proof elements, but it makes them trickier to reason about formally in the model.

A more glaring issue is that, in fact, these checks do not even guarantee that the proof elements will be in the span of the statement and witness polynomials combined. While it is indeed true that the checks are intuitively intended to guarantee this, it is technically possible for a prover to construct proof elements with nonzero coefficients for verifier key elements, but for which these checks pass. Specifically, after constructing a key honestly, the prover can multiply $g_v^{V_{mid}}$ and $g_v^{V'_{mid}}$ by g^1 and g_v^{α} respectively, and similarly with W_{mid} and Y_{mid}. We noticed this by way of our attempt to construct a proof that the only nonzero coefficients for $\mathcal{V}$ are in this set, finding that even after simplifying using the equalities generated by this check, there were still terms that we could not eliminate.

It is only through the next check that this attack is caught: If one includes the fifth check, which is intended to guarantee "that the same coefficients were used in each of the linear combinations over $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{Y}$", it becomes impossible to construct proof terms which fall outside the span. This caused us to make a slight change to the plan for the proof, to get around the fact that proving $\mathcal{V}$ was exactly equal to $g^{V_{mid}}$ was not possible to do simply. Instead, we left the g^1 terms (as well as the statement polynomial terms) in our simplifications of $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{Y}$, then proved that when one plugs these into the fifth check, these extra terms can be ignored, and the coefficients of the $g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}$ terms in Z can be seen to be equal to the supposed coefficients of $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{Y}$, so that *these* coefficients can still act as the extracted witness.

5.2 Groth '16

The written proof of soundness of Groth16 makes a similar overassumption about what certain equalities of terms in their check polynomial guarantees about relationships between coefficients of the SRS elements. The soundness proof given by Theorem 1 of [38] proceeds by first analyzing the $\alpha^2, \alpha\beta$, and β^2 coefficients of the polynomial, which allows one to simplify this polynomial, zeroing out a few of the terms without loss of generality. This logic is fine. We are then shown that the terms involving $1/\delta^2$ give us, without loss of generality.

$$\sum_{i=l+1}^{m} A_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + t(x)A_h(x) = 0$$

(Actually there is a typo in the Groth 16 paper here, as he writes A_t when he should write A_h). But then comes there is the more worrying claim that "The terms in $\alpha \frac{\sum_{i=l+1}^{m} B_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + B_h(t)}{\delta} = 0$ now show us that also:

$$\sum_{i=l+1}^{m} B_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + t(x)B_h(x) = 0 \tag{1}$$

What are "the terms in" this expression? It seems Groth is saying that the coefficients corresponding to the monomials α^2/δ, $\alpha\beta/\delta$ and α/δ, the only monomials that appear in this expression, imply the equation. α^2/δ gives us $\sum_{i=l+1}^{m} B_i v_i(x) = 0$, and $\alpha\beta/\delta$ gives us $\sum_{i=l+1}^{m} B_i u_i(x) = 0$, and this reduces the term to

$$\sum_{i=l+1}^{m} B_i w_i(x) + t(x)B_h(x)$$

But notice that the coefficient of α/δ actually includes a term of the form $A(x)\sum_{i=l+1}^{m} B_i \frac{\alpha v_i(x)}{\delta}$, so summing these three coefficients does not give us that this term is zero, it gives us that the term is equal to $-A(x)\sum_{i=l+1}^{m} B_i \frac{\alpha v_i(x)}{\delta}$. It is not immediately clear that this is even true! However, one eventually finds that the terms with coefficient $1/\delta$ give $\left(\sum_{i=l+1}^{m} B_i w_i(x) + t(x)B_h(x)\right) A(x) = 0$, and after doing casework on whether or not $A(x) = 0$, one sees that either way, the full Equation 1 can indeed be reduced to 0. Similar reasoning appears immediately after this with "The terms in $\alpha\frac{\sum_{i=l+1}^{m} B_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))+B_h(t)}{\gamma} = 0$", but this can be fixed in the same way.

Helpful in resolving this confusion was Baghery et al. [6] Theorem 2, which presents a correct proof of soundness by carefully listing the equations from each coefficient, and explicitly clarifying that the proof uses cases on $A(x) = 0$. They do this for a version of the Groth '16 SNARK intended for use with type-III pairing friendly curves, but the proof carries over to the non-type-III case.

5.3 Lipmaa's Simulation Extractable SNARK

Lipmaa [42] presents a SNARK with a simulation-extractability property. In the process of this construction, the paper constructs the SNARK S_{qap}, which is heavily based on a version of Groth '16 - In particular, it is very similar to the type-III SNARK presented in Baghery et al. Instead of 4 degree-bounded field samples, Lipmaa uses one sample giving a group element Y, and uses different exponents of this group element to represent the samples of Baghery et al. (for reference $\alpha, \beta, \gamma, \delta$ from Baghery et al. become $\gamma - \beta, \delta - \beta, \eta - \beta, \alpha - \beta$ and then an additional β is added to each sum of these exponents in the SRS, so for example, $\beta u_i(x)/\delta$ becomes $u_i y^{\beta-\alpha+\delta}$). Some small additive factors to the proofs are shifted around as well; the a (and b) proof terms includes an α (β) term in Baghery et al. but in Lipmaa these terms are added in by the verifier. None of these differences is substantive from the point of view of the AGM, and in fact these "transformations" can be generalized, as we will elaborate in Section 6. Thus, it is indeed the case that the Baghery et al. proof can be adapted to show the Lipmaa SNARK is sound.

But unfortunately the proof presented in [42] hews more closely to the Groth proof than the Baghery et al. proof. That proof uses 14 monomial equations, but Lipmaa makes the suspect claim that actually only 6 are needed to prove the soundness of the SNARK. The flaw in Lipmaa's reasoning turns out to be exactly analogous to that made by Groth discussed in the previous section! The $Y^{\beta+\gamma}$ terms are claimed (in Figure 2, for example) to show that $(a_\gamma + 1)v_b(X) - v(X) = 0$, but the output of the Lean computation shows that there should actually be a term for $b_\alpha \sum_{i \in wit} C_i v_i(X)$ which comes from the component of the witness SRS elements. Seeing this kind of mistake carrying over from one paper to another justifies the importance of catching these mistakes quickly.

6 SNARK Transformations

We have noted previously that many of the constructions covered in this paper are conceptually similar to others in their construction, but differ in details that make their soundness proofs different at a low level, while still being analogous at a high level. This illuminates the following principle: Many SNARKs can be described by first describing another SNARK, and then manipulating that SNARK according to transformations that preserve the soundness. Often these transformations can be applied to any SNARK construction, or to any SNARK construction satisfying certain criteria, and affect the aspects of the performance or explainability. A few examples of this include:

- Translating the degree of polynomials: That is to say, given a SNARK, it is possible to multiply through each SRS element by an existing (or new) toxic waste sample without affecting the soundness of the SNARK. This follows from the fact that all the checks carried out by the verifier take the form of comparisons of degree-2 polynomials in the proof elements: a toxic waste element τ that is multiplied through the SRS elements will yield a τ^2 in the polynomial check, which can be factored out. This is a fact we have implicitly used in our formalization to avoid dealing with Laurent polynomials with negative exponents.
- If a toxic waste σ element appears to maximum degree $< n$ in a check polynomial, and τ is another toxic waste element, then it is possible to replace τ with σ^n is the maximum degree, thus reducing the number of toxic waste samples needed. This does not affect the soundness, as any equivalence between coefficients of terms of the form $\sigma^i\tau^j$ in the first SNARK will be implied by coefficient of the term σ^{i+nj} . One can view this transformation as one of the conceptual differences between the Lipmaa SNARK [42] relative to [?].
- If the coefficient of a particular SRS element in a particular proof element is the same for any satisfying statement-witness pair, then we can treat that component as a constant and remove the dependence of the proof element linear combination on that SRS element. This could potentially be useful in optimizing SNARK circuits after the trusted setup. For example: Suppose we have a system where a particular prover always provides input specific to them (a private key, say) which causes part of the circuit computing the relation to consistently evaluate the same. Then we can collapse all the

witness elements associated with that part of the circuit into one, alleviating the requirement on the prover to store this data.

To demonstrate the extensibility of our development, we implement the above transformations, along with proofs that they preserve soundness. To do this, we create a data structure, `AGM_proof_system`, defined in `proof_system_fin.lean`, to represent sound SNARKs This structure includes:

- The relation being proved
- The SRS elements, as multivariable polynomials over a finite type of sample elements,
- the verifier checks, as a list of mulitvariable polynomials over a finite type of indices into proof elements.
- An extractor, which takes AGM coefficients (in the form of field elements in a mtrix, one coefficient for each SRS element component in each proof element) to field element values for each witness variable.
- A soundness proposition, which indicates that if the verifier checks pass for a particular statement and AGM coefficient matrix, then the relation applied to that statement and the extractor on that matrix is satisfied.

The transformations are then implemented as functions that take in a term of this type (along with potentially some other data) and return a new term of the same type. The biggest challenge in writing code for these transformation is the construction of the preservation of the soundness proposition. While this is slightly different for each of the transformations above, the process can broadly be described like this:

1. We are given the soundness statement for a input SNARK, and we need to prove the statement for the transformed SNARK.
2. Since the soundness proposition for the transformed SNARK is universally quantified over a statement and an AGM coefficient matrix for which the polynomial checks holding implies extractability, we always introduce the statement, matrix and proof of the transformed polynomial checks holding into the proof state.
3. Since the resulting goal is now simply that the relation holds for a particular statement and witness, and this is the output conclusion of the soundness proposition for the input SNARK, we apply this conclusion via the `apply` tactic.
4. This leaves in the goal the statement that the input SNARK polynomial checks hold. We must then find a way to prove this from the hypothesis that polynomial checks hold for the transformed SNARK.

This proof pattern highlights the reductive nature of the transformations. A statement of the rough form "If the proof checks don't pass for the input SNARK, then they shouldn't pass for the output SNARK" is contrapositively transformed into "If the checks pass for the output SNARK, then they pass for the input SNARK as well".

7 Future Work

There are a few things that could be future directions:

- One could optimize the Lean code to speed up the compilation. In particular one might think of using mathlib's `wlog` tactic to take advantage of symmetries, in particular in the Groth '16 SNARK.
- Going in the opposite direction, one could generalize the code to apply in broader settings. One option we considered was rewriting the `integral_domain_tactic` to use Gröbner basis methods so that it would require less by-hand tweaking to complete the proofs. We ultimately decided, given the poor computational complexity of these methods, that this would be too much effort to lose too much performance. Nevertheless, such a design could prove useful for other formalization efforts.
- One could add more aspects of the soundness proof which are ancillary to the main goal, such as formalizing the Schwartz-Zippel lemma application as seen in e.g., Lemma 1 of [48], or formalizing some basic complexity theory to prove the (trivial) extraction operation is indeed polynomial-time.

Possibly the most generic idea for future work in the direction of this project would be to study broader classes of SNARKs. In particular, one could imagine expanding the scope of the project to move beyond the class of SRS SNARKs shown here to other constructions, such as Sonic, PlonK, Marlin, or others [44, 36, 25]. A challenge we forsee in this direction is formally managing the Fiat-Shamir paradigm [32], which has become common in the design of SNARKs. This paradigm involves the random oracle model as a way of designing a proof system [15], and so it would likely require machinery to deal with probability distributions over hash functions.

8 Conclusion

We have presented our efforts to formalize the Groth '16 SNARK and similar constructions in Lean. Our work includes a variety of programs that help accomplish this task – we have described a variety of pitfalls associated with this challenge and our strategies for overcoming them. It is our hope going forward to continue formalizing, and eventually provide a library with coverage of several major SNARKs, and with tools which security researchers can use for assistance in in analyzing SNARKs in the wild.

Acknowledgements

The authors would like to thank Bryan Parno for discussions on early drafts of this paper.

This material is based upon work supported by the National Science Foundation under the Graduate Research Fellowship Program with Grant No. DGE – 1746047.

References

[1] The lean mathematical library. *CoRR*, abs/1910.09336, 2019.

[2] Carmine Abate, Philipp G Haselwarter, Exequiel Rivas, Antoine Van Muylder, Théo Winterhalter, Cătălin Hriţcu, Kenji Maillard, and Bas Spitters. Ssprove: A foundational framework for modular cryptographic proofs in coq. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, pages 1–15. IEEE, 2021.

[3] José Bacelar Almeida, Manuel Barbosa, Manuel L Correia, Karim Eldefrawy, Stéphane Graham-Lengrand, Hugo Pacheco, and Vitor Pereira. Machine-checked zkp for np relations: Formally verified security proofs and implementations of mpc-in-the-head. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2587–2600, 2021.

[4] Jeremy Avigad, Lior Goldberg, David Levit, Yoav Seginer, and Alon Titelman. A verified algebraic representation of cairo program execution, 2021.

[5] José Bacelar Almeida, Manuel Barbosa, Endre Bangerter, Gilles Barthe, Stephan Krenn, and Santiago Zanella Béguelin. Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 488–500, 2012.

[6] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-snark. Cryptology ePrint Archive, Report 2020/811, 2020. `https://ia.cr/2020/811`.

[7] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. Sok: Computer-aided cryptography. In *2021 IEEE symposium on security and privacy (SP)*, pages 777–795. IEEE, 2021.

[8] Manuel Barbosa, Gilles Barthe, Xiong Fan, Benjamin Grégoire, Shih-Han Hung, Jonathan Katz, Pierre-Yves Strub, Xiaodi Wu, and Li Zhou. Easypqc: Verifying post-quantum cryptography. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2564–2586, 2021.

[9] Gilles Barthe, Jan Cederquist, and Sabrina Tarento. A machine-checked formalization of the generic model and the random oracle model. In *Automated Reasoning: Second International Joint Conference, IJCAR 2004, Cork, Ireland, July 4-8, 2004. Proceedings 2*, pages 385–399. Springer, 2004.

[10] Gilles Barthe, Juan Manuel Crespo, Yassine Lakhnech, and Benedikt Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 689–718. Springer, 2015.

[11] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 90–101, 2009.

[12] Gilles Barthe, Daniel Hedin, Santiago Zanella Béguelin, Benjamin Grégoire, and Sylvain Heraud. A machine-checked formalization of sigma-protocols. In *2010 23rd IEEE Computer Security Foundations Symposium*, pages 246–260. IEEE, 2010.

[13] David Basin, Andreas Lochbihler, Ueli Maurer, and S Reza Sefidgar. Abstract modeling of system communication in constructive cryptography using crypthol. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, pages 1–16. IEEE, 2021.

[14] David A Basin, Andreas Lochbihler, and S Reza Sefidgar. Crypthol: Game-based proofs in higher-order logic. *Journal of Cryptology*, 33:494–566, 2020.

[15] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[16] 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, 2013. `https://ia.cr/2013/507`.

[17] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879, 2013. `https://ia.cr/2013/879`.

[18] Dhruv Bhatia. A tactic using sage to solve polynomial equalities with hypotheses, 2022. `https://github.com/leanprover-community/mathlib/pull/14878`.

[19] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. State separation for code-based game-playing proofs. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*, pages 222–249. Springer, 2018.

[20] David Butler, David Aspinall, and Adrià Gascón. Formalising oblivious transfer in the semi-honest and malicious model in crypthol. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 229–243, 2020.

[21] David Butler, Andreas Lochbihler, David Aspinall, and Adrià Gascón. Formalising σ-protocols and commitment schemes using crypthol. *Journal of Automated Reasoning*, 65(4):521–567, 2021.

[22] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[23] Ran Canetti, Alley Stoughton, and Mayank Varia. Easyuc: Using easycrypt to mechanize proofs of universally composable security. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 167–16716. IEEE, 2019.

[24] Yanju Chen, Clara Rodriguez, Yu Feng, and Bryan Tan. Picus, 2022.

[25] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. Cryptology ePrint Archive, Report 2019/1047, 2019. `https://ia.cr/2019/1047`.

[26] Collin Chin, Howard Wu, Raymond Chu, Alessandro Coglio, Eric McCarthy, and Eric Smith. Leo: A programming language for formally verified, zero-knowledge applications. *Cryptology ePrint Archive*, 2021.

[27] Alessandro Coglio. Ethereum's recursive length prefix in acl2. *arXiv preprint arXiv:2009.13769*, 2020.

[28] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. Machine-checked proofs for electronic voting: privacy and verifiability for belenios. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 298–312. IEEE, 2018.

[29] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The lean theorem prover. 2015.

[30] François Dupressoir, Konrad Kohbrok, and Sabine Oechsner. Bringing state-separating proofs to easycrypt a security proof for cryptobox. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*, pages 227–242. IEEE, 2022.

[31] Karim Eldefrawy and Vitor Pereira. A high-assurance evaluator for machine-checked secure multiparty computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 851–868, 2019.

[32] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1987.

[33] Denis Firsov and Dominique Unruh. Zero-knowledge in easycrypt. *Cryptology ePrint Archive*, 2022.

[34] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. Cryptology ePrint Archive, Report 2017/620, 2017. `https://ia.cr/2017/620`.

[35] Ariel Gabizon. On the security of the bctv pinocchio zk-snark variant. Cryptology ePrint Archive, Report 2019/119, 2019. `https://ia.cr/2019/119`.

[36] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://ia.cr/2019/953`.

[37] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. Cryptology ePrint Archive, Report 2012/215, 2012. `https://ia.cr/2012/215`.

[38] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Report 2016/260, 2016. `https://ia.cr/2016/260`.

[39] Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. Computer-aided proofs for multiparty computation with active security. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 119–131. IEEE, 2018.

[40] Thomas Haines, Rajeev Goré, and Bhavesh Sharma. Did you mix me? formally verifying verifiable mix nets in electronic voting. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1748–1765. IEEE, 2021.

[41] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. The coq proof assistant a tutorial. *Rapport Technique*, 178, 1997.

[42] Helger Lipmaa. Simulation-extractable snarks revisited. Cryptology ePrint Archive, Report 2019/612, 2019. `https://ia.cr/2019/612`.

[43] Andreas Lochbihler, S Reza Sefidgar, David Basin, and Ueli Maurer. Formalizing constructive cryptography using crypthol. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 152–15214. IEEE, 2019.

[44] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings. Cryptology ePrint Archive, Report 2019/099, 2019. `https://ia.cr/2019/099`.

[45] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. Cryptology ePrint Archive, Report 2020/934, 2020. `https://ia.cr/2020/934`.

[46] Catherine A Meadows and Catherine A Meadows. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology—ASIACRYPT'94: 4th International Conferences on the Theory and Applications of Cryptology Wollongong, Australia, November 28–December 1, 1994 Proceedings 4*, pages 133–150. Springer, 1995.

[47] Roberto Metere and Changyu Dong. Automated cryptographic analysis of the pedersen commitment scheme. In *Computer Network Security: 7th International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2017, Warsaw, Poland, August 28-30, 2017, Proceedings 7*, pages 275–287. Springer, 2017.

[48] Andrew Miller, Ye Zhang, and Sanket Kanjalkar. Baby snark (do do dodo dodo. 2020.

[49] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

[50] David Nowak. On formal verification of arithmetic-based cryptographic primitives. In *Information Security and Cryptology–ICISC 2008: 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers 11*, pages 368–382. Springer, 2009.

[51] Bryan Parno. A note on the unsoundness of vntinyram's snark. Cryptology ePrint Archive, Report 2015/437, 2015. `https://ia.cr/2015/437`.

[52] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. Cryptology ePrint Archive, Report 2013/279, 2013. `https://ia.cr/2013/279`.

[53] Loïc Pottier. Nsatz: a solver for equalities in integral domains, 2021.

[54] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[55] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*, pages 256–266. Springer, 1997.

[56] Nikolaj Sidorenco, Sabine Oechsner, and Bas Spitters. Formal security analysis of mpc-in-the-head zero-knowledge protocols. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, pages 1–14. IEEE, 2021.

[57] Sabrina Tarento. Machine-checked security proofs of cryptographic signature schemes. In *ESORICS*, volume 5, pages 140–158. Springer, 2005.

[58] Søren Eller Thomsen and Bas Spitters. Formalizing nakamoto-style proof of stake. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, pages 1–15. IEEE, 2021.

[59] Franklyn Wang. Ecne: Automated verification of zk circuits, 2022.

A Proof State after Simplifcation

Below, we show the proof state for the Baghery et al. SNARK just before mutual simplification.

The variables starting `A_`, `B_`, and `C_` correspond to components of the linear combination making up the A, B and C proof elements given by the AGM. The goal of the mutual simplification is to prove that as many of these values are 0 as possible.

```
state:
F : Type u,
_inst_1 : field F,
n_stmt n_wit n_var : ,
u_stmt : fin n_stmt → polynomial F,
u_wit : fin n_wit → polynomial F,
v_stmt : fin n_stmt → polynomial F,
v_wit : fin n_wit → polynomial F,
w_stmt : fin n_stmt → polynomial F,
w_wit : fin n_wit → polynomial F,
r : fin n_wit → F,
A_α A_β A_δ B_β B_γ B_δ C_α C_β C_δ : F,
A_x B_x C_x : fin n_var → F,
A_l C_l : fin n_stmt → F,
A_m C_m : fin n_wit → F,
A_h C_h : fin (n_var - 1) → F,
a_stmt : fin n_stmt → F,
h1122 : C A_α * C B_β = 1,
h1121 : ( (x : fin n_wit), v_wit x * C (A_m x)) * C B_β = 0,
h1112 : ( (x : fin n_stmt), v_stmt x * C (A_l x)) * C B_β = 0,
h1022 :
  C A_α *  (i : fin n_var), C (B_x i) * X ^ ↑i +
        ( (x : fin n_stmt), v_stmt x * C (A_l x)) * C B_γ +
      ( (x : fin n_wit), v_wit x * C (A_m x)) * C B_δ =
     (x : fin n_stmt), C (a_stmt x) * v_stmt x +  (x : fin n_wit), v_wit x * C (C_m x),
h0222 : C A_β * C B_β = 0,
h0221 : ( (x : fin n_wit), u_wit x * C (A_m x)) * C B_β = 0,
h0212 : ( (x : fin n_stmt), u_stmt x * C (A_l x)) * C B_β = 0,
h0122 :
  C A_β *  (i : fin n_var), C (B_x i) * X ^ ↑i +
          ( (i : fin n_var), C (A_x i) * X ^ ↑i) * C B_β +
        ( (x : fin n_stmt), u_stmt x * C (A_l x)) * C B_γ +
      ( (x : fin n_wit), u_wit x * C (A_m x)) * C B_δ =
     (x : fin n_stmt), C (a_stmt x) * u_stmt x +  (x : fin n_wit), u_wit x * C (C_m x),
h0121 :
  ( (x : fin n_wit), u_wit x * C (A_m x)) *  (i : fin n_var), C (B_x i) * X ^ ↑i +
        ( (x : fin n_wit), w_wit x * C (A_m x)) * C B_β +
      ( (x : fin (n_var - 1)), X ^ ↑x * t * C (A_h x)) * C B_β =
    0,
h0112 :
  ( (x : fin n_stmt), u_stmt x * C (A_l x)) *  (i : fin n_var), C (B_x i) * X ^ ↑i +
      ( (x : fin n_stmt), w_stmt x * C (A_l x)) * C B_β =
    0,
h0022 :
  (  (i : fin n_var), C (A_x i) * X ^ ↑i) *   (i : fin n_var), C (B_x i) * X ^ ↑i +
          (  (x : fin n_stmt), w_stmt x * C (A_l x)) * C B_γ +
        (  (x : fin n_wit), w_wit x * C (A_m x)) * C B_δ +
      (  (x : fin (n_var - 1)), X ^ ↑x * t * C (A_h x)) * C B_δ =
     (x : fin n_stmt), C (a_stmt x) * w_stmt x +
      (  (x : fin n_wit), w_wit x * C (C_m x) +   (x : fin (n_var - 1)), X ^ ↑x * t * C (C_h x)),
h0021 :
  (  (x : fin n_wit), w_wit x * C (A_m x)) *   (i : fin n_var), C (B_x i) * X ^ ↑i +
      (  (x : fin (n_var - 1)), X ^ ↑x * t * C (A_h x)) *   (i : fin n_var), C (B_x i) * X ^ ↑i =
    0,
h0012 : (  (x : fin n_stmt), w_stmt x * C (A_l x)) *   (i : fin n_var), C (B_x i) * X ^ ↑i = 0
 (  (i : fin n_stmt), u_stmt i * C (a_stmt i) +   (i : fin n_wit), u_wit i * C (C_m i)) *
      (  (i : fin n_stmt), v_stmt i * C (a_stmt i) +   (i : fin n_wit), v_wit i * C (C_m i)) =
     (i : fin n_stmt), w_stmt i * C (a_stmt i) +   (i : fin n_wit), w_wit i * C (C_m i) +
       (x : fin (n_var - 1)), X ^ ↑x * t * C (C_h x)
```