Distilling Constraints in Zero-Knowledge Protocols

Elvira Albert[1], Marta Bellés-Muñoz[2], Miguel Isabel[1], Clara Rodríguez-Núñez[1](✉), and Albert Rubio[1]

[1] Complutense University of Madrid, Madrid, Spain
clarrodr@ucm.es
[2] Pompeu Fabra University, Barcelona, Spain

Abstract. The most widely used *Zero-Knowledge* (ZK) protocols require provers to prove they know a solution to a computational problem expressed as a *Rank-1 Constraint System* (R1CS). An R1CS is essentially a system of non-linear arithmetic constraints over a set of signals, whose security level depends on its non-linear part only, as the linear (additive) constraints can be easily solved by an attacker. Distilling the essential constraints from an R1CS by removing the part that does not contribute to its security is important, not only to reduce costs (time and space) of producing the ZK proofs, but also to reveal to cryptographic programmers the real hardness of their proofs. In this paper, we formulate the problem of distilling constraints from an R1CS as the (hard) problem of simplifying constraints in the realm of non-linearity. To the best of our knowledge, it is the first time that constraint-based techniques developed in the context of formal methods are applied to the challenging problem of analysing and optimizing ZK protocols.

1 Introduction

Zero-Knowledge (ZK) protocols [8,15,17,27] enable one party, called prover, to convince another one, called verifier, that a statement is true without revealing any information beyond the veracity of the "statement". In this context, we understand a statement as a relation between an instance, a *public* input known to both prover and verifier, and a *witness*, a *private* input known only to the prover, which belongs to a language $\mathcal{L}$ in the nondeterministic polynomial time (NP) complexity class [5,15]. The most popular, efficient and general-purpose ZK protocols are ZK-SNARKs: ZK Succinct Non-interactive ARguments of Knowledge. While a proof guarantees the existence of a witness in a language $\mathcal{L}$, and *argument of knowledge* proves that, with very high probability, the prover knows a concrete valid witness in $\mathcal{L}$. A ZK-SNARK does not require interaction between the prover and the verifier, and regardless of the size of the statement being proved, the size of the proof is succinct. These appealing properties of ZK-SNARKs have made them become crucial tools in many real-world applications

© The Author(s) 2022
S. Shoham and Y. Vizel (Eds.): CAV 2022, LNCS 13371, pp. 430–443, 2022.
https://doi.org/10.1007/978-3-031-13185-1_21

with strong privacy issues. A prominent such example is Zcash [4]. ZK protocols are also being used in conjunction with smart contracts, in the so-called *ZK-rollups* for enhancing the scalability of distributed ledgers [18].

Like most ZK systems, ZK-SNARKs operate in the model of *arithmetic circuits*, meaning that the NP language $\mathcal{L}$ is that of satisfiable arithmetic circuits. The gates of an arithmetic circuit consist of additions and multiplications modulo p, where p is typically a large prime number of approximately 254 bits [3]. The wires of an arithmetic circuit are called signals, and can carry any value from the prime finite field $\mathbb{F}_p$. In the ZK context, there is usually a set of public inputs known both to the prover and the verifier, and the prover proves that she knows a valid assignment to the rest of signals that satisfies the circuit (i.e., the witness). Most ZK-SNARK protocols draw from a classical algebraic form for encoding circuits and wire assignment called rank-1 constraint system (R1CS). An R1CS encodes a circuit as a set of quadratic constraints over its variables, so that a correct execution of a circuit is equivalent to finding a satisfying variable assignment. This way, a valid witness for an arithmetic circuit translates naturally into a solution of its R1CS representation.

Although ZK protocols guarantee that a malicious verifier cannot extract a witness from a proof, they do not prevent the verifier from attacking the statement directly. Hence, it is important that the prover is aware of the difficulty of the statement being proved. In this regard, it is challenging for cryptographic developers that apply ZK protocols to complex computations to assess the real hardness of the produced computational problem, being hence also difficult to verify and audit the systems. It is partly because a syntactic assessment (e.g. based on counting the number of non-linear constraints) can be inaccurate and misleading. This is the case if the R1CS contains *redundant* constraints, i.e., constraints that can be deduced from others or constraints that follow from linear constraints, since they do not contribute to the hardness of the computational statement. Distilling the relevant constraints is important on one hand for efficiency, to reduce costs (time and space) of producing the ZK proofs, and also because redundancy can mislead developers to believe that the statement is far more complex than it really is. It is clear that when arithmetic circuits are defined over a finite field of small order, the problem can be attacked by brute-force, or if the system consists only of linear constraints, a solution can be found in polynomial time [25]. Moreover, in R1CS-based systems like [17] only multiplication gates add complexity to the statement. Also note that linear constraints induce a way to compute the value of one signal from a linear combination of the others, and hence we can easily extend a witness for the other signals to a witness for all the signals. As a result, the difficulty of finding a solution to a system relies mostly in the number of *non-redundant non-linear constraints*.

Contributions. This case study paper applies techniques developed in the context of formal methods to distill constraints from the R1CS systems used by ZK protocols. The main challenges are related, on the one hand, to reasoning with non-linear information in a finite field and, on the other hand, to dealing with very large constraint systems. Briefly, our main contributions are: (1) we

present a formal framework to reason on circuit reduction which generalizes the application of different existing optimizations and the reduction strategy in which they are applied, (2) we introduce a concrete new optimization technique based on Gaussian elimination that allows deducing linear constraints from the non-linear constraints, (3) we implement our approach within `circom` [21] (a novel domain-specific language and compiler for defining arithmetic circuits) and also develop an interface for using it on the R1CS generated by `ZoKrates` [12], (4) we experimentally evaluate its performance on multiple real-world circuits (including templates from the `circom` library [22] and from [12], on implementations of different SHA-2 hash functions, on elliptic curve operations, etc.).

2 Preliminaries

This section introduces some preliminary notions and notation. We consider $\mathbb{F}_p$ a finite field of prime order p. As usual, $\mathbb{F}_p^n$ is a sequence of n values in $\mathbb{F}_p$. We drop p from $\mathbb{F}$ when it is irrelevant. An arithmetic circuit (over the field $\mathbb{F}$) consists of wires (represented by means of signals $s_i \in \mathbb{F}$) connected to gates (represented by *quadratic constraints*). Signals can be public or private. We now define the concepts of quadratic constraints and R1CS over a set of signals.

Definition 1 (R1CS). *A* quadratic constraint *over a set of signals* $\{s_1, \ldots, s_n\}$ *is an equation of the form* $Q : A \times B - C = 0$*, where* $A, B, C \in \mathbb{F}[s_1, ..., s_n]$ *are linear polynomials over the variables* $s_1, ..., s_n$*, i.e.,* $A = a_0 + a_1 s_1 + \cdots + a_n s_n$*,* $B = b_0 + b_1 s_1 + \cdots + b_n s_n$*, and* $C = c_0 + c_1 s_1 + \cdots + c_n s_n$*, where* $a_i, b_i, c_i \in \mathbb{F}$ *for all* $i \in \{0, \ldots, n\}$*. A* rank-1 constraint system (R1CS) *over a set of signals* T *is a collection of quadratic constraints over* T*.*

We say that a quadratic constraint Q is *linear* when A or B only have the constant term, i.e., $a_i = 0\ \forall i \in \{1, \ldots, n\}$ or $b_i = 0\ \forall i \in \{1, \ldots, n\}$, and is *non-linear* otherwise. As R1CS systems only contain quadratic constraints, in what follows, we simply call them *constraints*, and specify if they are linear or not where needed. We use the standard notation $S \models c$ to indicate that a constraint c is deducible from a set of constraints S and $|S|$ for the number of constraints.

Definition 2 (arithmetic circuit and witness). *An* (arithmetic) circuit *is a tuple* $\mathcal{C} = (U, V, S)$ *where* U *represents the set of public signals,* V *represents the set of private signals, and the R1CS* $S = \{Q_1, \ldots, Q_m\}$ *over the signals* $U \cup V$ *represents the circuit operations. Given an assignment* u *for* U*, a* witness *for* $\mathcal{C}$ *is an assignment* v *for* V *s.t.* u *together with* v *are a solution to the R1CS* S*.*

We use the terms *circuit* and, R1CS or just constraint system, indistinctly when the signals used in the circuit are clear. Given a circuit $\mathcal{C}$ and a public assignment for U, a ZK protocol is a mechanism that allows a prover to prove to a verifier that she knows a private assignment for V that, together with those for U, satisfy the R1CS system describing $\mathcal{C}$. ZK protocols guarantee that the proof will not reveal any information about V.

Example 1. We consider a circuit $\mathcal{C}_1 = (U, V, S_1)$ over a finite field $\mathbb{F}$, with $U = \{v, w\}$, $V = \{x, y, z\}$, and S_1 given by the following constraint system:

$$Q_1 : w \times (y + z) - 4x - 10 = 0, \quad Q_2 : w \times z - w - 3 = 0,$$
$$Q_3 : (x - w + 1) \times v - v + 1 = 0, Q_4 : y - z - 2 = 0.$$

This circuit contains 3 non-linear constraints (Q_1, Q_2, and Q_3) and a linear one (Q_4). Because of its small size, we can easily solve the system (i.e., give the value of each signal in terms of only one of them) and find the set of solutions:

$$W = \{(v, w, x, y, z) \mapsto (1, w, w - 1, 3w^{-1} + 3, 3w^{-1} + 1) \mid w \in \mathbb{F} \setminus \{0\}\}.$$

A cryptographic problem can be modeled by different circuits producing the same solutions. This relation among circuits can be formalized as *circuit equivalence*, which is a natural extension of the constraint system equivalence. We say that two circuits $\mathcal{C} = (U, V, S)$ and $\mathcal{C}' = (U, V, S')$ are *equivalent*, written $\mathcal{C} \simeq \mathcal{C}'$, if S and S' have the same set of solutions. Consequently, if $\mathcal{C}$ and $\mathcal{C}'$ are equivalent, they have the same set of solutions and hence of witnesses.

Example 2. The circuit $\mathcal{C}_2 = (U, V, S_2)$ with the same sets of public and private signals U and V as $\mathcal{C}_1$, and the R1CS S_2 given by the constraints:

$$Q'_1 : w \times y - 3w - 3 = 0, \quad Q'_2 : y - z - 2 = 0, \quad Q'_3 : v - 1 = 0, \quad Q'_4 : x - w + 1 = 0,$$

has the same set of solutions (and thus witnesses) as $\mathcal{C}_1$. Hence, $\mathcal{C}_1 \simeq \mathcal{C}_2$.

3 A Formal Framework for R1CS Reduction

R1CS optimizations are applied within state-of-the-art compilers like `circom` [21] or `ZoKrates` [12]. Common to such existing compiler optimizations is the application of rules to simplify and eliminate linear constraints and/or to deduce information from them. As our first contribution, we present a formal framework for R1CS reduction based on a rule-based transformation system which is general enough to be a formal basis for developing specific simplification techniques and reduction strategies. In particular, the simplifications already applied in the above compilers are instantiations of our framework.

The notion of reduction that our framework formalizes is key to define the security level of circuits. When two circuits model the same problem, they provide the same level of security. However, an assessment of their security level based on syntactically counting the number of non-linear constraints in the circuits can lead to a wrong understanding/estimation of their security. For instance, circuits $\mathcal{C}_1$ and $\mathcal{C}_2$ (see Examples 1-2) model the same problem, although $\mathcal{C}_2$ needs a single non-linear constraint to define its set of solutions (instead of three as $\mathcal{C}_1$). This happens because some of the non-linear constraints of $\mathcal{C}_1$ are not essential and can be substituted by linear constraints. Besides, we can observe in $\mathcal{C}_2$ that signals x and z are only involved in linear constraints instead of being on non-linear constraints like in $\mathcal{C}_1$. In other words, *having a circuit with more private signals involved in non-linear constraints (e.g., $\mathcal{C}_1$) does not ensure further security if these private*

signals can be deduced from linear combinations of the others. We build our notion of *circuit reduction* upon this concept.

Definition 3 (circuit-reduction). *Let $\mathcal{C} = (U, V, S)$ be a circuit with $U \cup V = \{s_1, \ldots, s_n\}$, and $\mathcal{C}' = (U, V', S')$ another circuit with $V \subseteq V'$.*

(i) We say that $\mathcal{C}'$ linearly follows *from $\mathcal{C}$, denoted by $\mathcal{C} \models_l \mathcal{C}'$, if $\forall s \in V' \setminus V$, $\exists \lambda_0^s, \lambda_1^s, ..., \lambda_n^s \in \mathbb{F}$, s.t. given an assignment for U, every witness ϕ for $\mathcal{C}$ extended with $s \mapsto \lambda_0^s + \sum_{i=1}^{n} \lambda_i^s * \phi(s_i)$ is a witness for $\mathcal{C}'$.*

(ii) We say that $\mathcal{C}'$ reduces *to $\mathcal{C}$, written $\mathcal{C}' \geq \mathcal{C}$, if $\mathcal{C} \models_l \mathcal{C}'$, $|S'| \geq |S|$ and every witness of $\mathcal{C}'$ restricted to V is a witness for $\mathcal{C}$ for the same assignment of U. We say that $\mathcal{C}'$* strictly reduces *to $\mathcal{C}$, written $\mathcal{C}' > \mathcal{C}$ if $|S'| > |S|$ or $V \subset V'$.*

Intuitively, we have that for every signal defined in V, the values of the two witnesses match, and for the signals defined in $V' \setminus V$, the value of the witness of C' can be obtained from a linear combination of the values from the assignment for U and ϕ.

Example 3. Let $\mathcal{C}_3$ be $(\{v, w\}, \{y\}, S_3)$ with $S_3 = \{Q_1'' : w \times y - 3w - 3 = 0, Q_2'' : v - 1 = 0\}$. Let us show that $\mathcal{C}_1$ (from Example 1) strictly reduces to $\mathcal{C}_3$. From Example 2, we have that every solution of $\mathcal{C}_1$ restricted to $\{v, w, y\}$ is also a solution of $\mathcal{C}_3$ (since $S_3 \subseteq S_2$ and $\mathcal{C}_2 \simeq \mathcal{C}_1$) and that in every witness ϕ' of $\mathcal{C}_2$ we have that $\phi'(x) = \phi'(w) - 1$ and $\phi'(z) = \phi'(y) - 2$. Therefore, taking $\lambda_0^x = -1$, $\lambda_{\text{pos}(w)}^x = 1$, $\lambda_0^z = -2$, $\lambda_{\text{pos}(y)}^z = 1$ (where function $\text{pos}(s_i)$ abstracts the index i of the variable s_i in the set of signals), we have that $\mathcal{C}_3 \models_l \mathcal{C}_1$. Finally, since $\{y\} \subset \{x, y, z\}$ and, given an assignment for $\{v, w\}$, every witness of $\mathcal{C}_1$ restricted to $\{y\}$ is a witness for $\mathcal{C}_3$, and we can conclude.

We now present a set of transformation rules that ensure circuit reducibility. The transformation is based on finding linear consequences of the constraint system to guarantee that the transformed set of constraints linearly follows from the original system. Our transformation rules operate on pairs in $\mathbb{K} \times \mathbb{S}_{\mathbb{L}}$, where $\mathbb{K}$ is the set of arithmetic circuits and $\mathbb{S}_{\mathbb{L}}$ is the set of linear constraint systems. As usual, we use infix notation, writing $(\mathcal{C}, S_L) \Rightarrow (\mathcal{C}', S_L{}')$, and denote respectively by $\Rightarrow^+$ and $\Rightarrow^*$, its transitive and reflexive-transitive closure. Given a circuit $\mathcal{C}$, if $(\mathcal{C}, \emptyset) \Rightarrow^* (\mathcal{C}', S_L)$, then $\mathcal{C}'$ is a reduction for $\mathcal{C}$, and the linear system S_L shows how to prove that $\mathcal{C}' \models_l \mathcal{C}$. In the following, we assume that there exists a total order $<$ among the private signals in V which is used to select a signal among the private signals of a constraint c, denoted by $V(c)$.

(REMOVE) $((U, V, S \cup \{c\}), S_L) \Rightarrow ((U, V, S), S_L)$, **if** $S \models c$.

(DEDUCE) $((U, V, S), S_L) \Rightarrow ((U, V, S), S_L \cup \{c\})$, **if** c linear, $S \models c$, $c \notin S_L$.

(SIMPLIFY) $((U, V, S), S_L \cup \{s = l\}) \Rightarrow ((U, V \setminus \{s\}, S[s \mapsto l]), S_L[s \mapsto l] \cup \{s = l\})$, **if** $s \in V$ and $\forall x \in V(l)$, $x < s$.

Fig. 1. Circuit transformation rules.

The REMOVE rule allows us to remove redundant constraints. The DEDUCE rule is needed to extract from S linear relations among the signals. Finally, the SIMPLIFY rule allows us to safely remove a signal s from V by replacing it by an equivalent linear combination of public and (strictly) smaller private signals in S. The fact that we replace a private signal by strictly smaller ones prevents this rule from being applied infinitely many times. When no constraint or private signal can be removed from a circuit (e.g., from $\mathcal{C}_3$) after applying a sequence of reduction rule steps, the circuit is considered *irreducible* and we call it a *normal form*. Note that the linear constraints in S_L with signals not belonging to $U \cup V$ are the ones that track how to obtain the missing signals from the remaining ones.

The three rules from Fig. 1 are terminating and they are contained in the circuit reducibility relation (Definition 3) when projected to the first component (the circuit). Regarding confluence, we have that if $(\mathcal{C}, S_L) \Rightarrow^* (\mathcal{C}_1, S_{L1})$ and $(\mathcal{C}, S_L) \Rightarrow^* (\mathcal{C}_2, S_{L2})$, then we have that $(\mathcal{C}_1, S_{L1}) \Rightarrow^* (\mathcal{C}'_1, S_{L1}{}')$ and $(\mathcal{C}_2, S_{L2}) \Rightarrow^* (\mathcal{C}'_2, S_{L2}{}')$ such that $\mathcal{C}'_1$ and $\mathcal{C}'_2$ are equivalent (see Appendix).

Example 4. Let us apply our reduction system to find a normal form of $(\mathcal{C}_1, \emptyset)$ which corresponds to its reduction. At each step we label the arrow with the applied rule and show only the component that is modified from the previous step (we use $_$ to indicate the value of the component as in the previous step):

$$\begin{aligned}((U, V, S_1), \emptyset) &\overset{\text{DEDUCE}}{\Rightarrow} ((_, _, _), \{L_1 : z = y - 2\}) \overset{\text{SIMPLIFY}}{\Rightarrow} (_, _ \setminus \{z\}, _[z \mapsto y - 2]), _) \\ &\overset{\text{REMOVE}}{\Rightarrow} ((_, _, _ \setminus \{0 = 0\}), _) \overset{\text{DEDUCE}}{\Rightarrow} ((_, _, _), _ \cup \{L_2 : x = w - 1\}) \\ &\overset{\text{SIMPLIFY}}{\Rightarrow} ((_, _ \setminus \{x\}, _[x \mapsto w - 1]), _) \\ &\overset{\text{REMOVE}}{\Rightarrow} ((_, _, _ \setminus \{Q : w \times (2y - 2) - 4w - 6 = 0\}), _)\end{aligned}$$

Here $(\mathcal{C}_3, \{L_1, L_2\})$ is a normal form of $(\mathcal{C}_1, \emptyset)$ and, as we have already seen in Example 3, $\mathcal{C}_3$ is a reduction for $\mathcal{C}_1$. Note that $\{L_1, L_2\}$ shows how to obtain the values of the removed signals as a linear combination.

4 Circuit Reduction Using Constraint Simplification

In this section, we introduce different strategies to apply the transformation rules described in Fig. 1, and also to approximate the deduction relation $S \models c$ in rules REMOVE and DEDUCE. Note that the classical representation of our problem is undecidable, but since we work in a finite field, it becomes decidable. However, as the order of $\mathbb{F}$ is large, it is still impractical and approximation is required.

As an example, let us show how the simplification techniques applied in `ZoKrates` and `circom` fit in our framework. In both languages, besides the removal of tautologies, all simplification steps are made using linear constraints that are part of the set of constraints. In particular, in a first step both languages handle the so-called *redefinitions* (i.e., constraints of the form $x = y$), and in a second step all the remaining linear constraints are eliminated applying the necessary substitutions. In our framework, these simplification steps can be

described as a sequence of DEDUCE to obtain the linear constraints that will be applied as substitutions, followed by a sequence of SIMPLIFY, and a sequence of REMOVE to delete the tautologies obtained after the substitutions. The whole sequence can be repeated until no linear constraints are left in the circuit. The specific strategy followed to perform the sequence of DEDUCE steps to obtain the substitutions used to simplify the circuit from its linear constraints has a big impact in the efficiency of the process. For instance, `circom` considers all maximal clusters of linear constraints (sharing signals) in the system and then infers in one go all the substitutions to be applied for every cluster, using a lazy version of Gauss-Jordan elimination. This process can be very expensive when the number of constraints in the R1CS is very large (e.g. hundreds of millions in ZK-Rollups like Hermez [20]).

Similar techniques based on analyzing the linear constraints are applied in other circuit-design languages. However, up to our knowledge, no language uses the non-linear part of the circuit to infer new linear constraints, or to remove redundant constraints, and this constitutes the second main contribution of this work. In the remaining of this section, we present a new approach inspired by techniques used in program analysis and SMT-solving like [9,11], where the non-linear reasoning is reduced to linear-reasoning. We can assume that we have applied first the aforementioned strategies to obtain an R1CS containing only non-linear constraints (or linear constrains with only public signals). Then, in our framework, the problem of inferring new linear constraints from non-linear R1CS can be formalized as a synthesis problem as follows: "*given a circuit* (U, V, S), *where* $U \cup V = \{s_1, \ldots, s_n\}$, *our goal is to find a linear expression* $l = c_0 + c_1 s_1 + \ldots + c_n s_n$ *with* $c_0, c_1, \ldots, c_n \in \mathbb{F}$ *such that* $S \models l = 0$." In order to solve this problem, we follow an efficient approach in which we restrict ourselves to the case where $l = 0$ can be expressed as a linear combination of constraints in S, i.e., of the form $\sum \lambda_k * Q_k$ with $Q_k \in S$ and $\lambda_k \in \mathbb{F}$. It is clear that any constraint $l = 0$ obtained using this approach satisfies $S \models l = 0$, but we are only interested in the ones that are linear. In the following two stages, we describe how to obtain linear expressions l, and hence, infer the constraints.

Stage 1. First, for each constraint $Q_k : A_k \times B_k - C_k = 0$, $k \in \{1, \ldots, m\}$, we expand the multiplication $A_k \times B_k$, obtaining the expression $\sum_{1 \leq i \leq j \leq n} Q_k[i, j] * s_i s_j + L_k$, where $Q_k[i, j]$ for $1 \leq i \leq j \leq n$ denotes the coefficient of the monomial $s_i s_j$ in the constraint Q_k, and L_k is the linear part of $A_k \times B_k$.

Example 5. Let us consider the circuit from Example 4 after applying the first three transformation rules, i.e. after removing the linear constraints. We denote the resulting circuit $\mathcal{C}_4 = (U, V_4, S_4)$, where $U \cup V_4 = \{v, w, x, y\}$ and S_4 is given by:

$$Q_1 : w \times (2y - 2) - 4x - 10 = 0,\ Q_2 : w \times (y - 2) - w - 3 = 0,$$
$$Q_3 : (x - w + 1) \times v - v + 1 = 0.$$

Here, we have for Q_1 that $A_1 = w$, $B_1 = 2y - 2$ and $C_1 = 4x + 10$ (recall that we consider $A_1 \times B_1 - C_1 = 0$). Then, we expand the multiplication $A_1 \times B_1 = 2wy - 2w$, so that $L_1 = -2w$ and $Q_1[2, 4] = 2$ (for wy), where the later is the only non-zero coefficient of a quadratic monomial. Similarly, for Q_2 we have

$C_2 = w + 3$, $Q_2[2,4] = 1$ (also for wy) and $L_2 = -2w$. Finally, for Q_3 we have $C_3 = v - 1$, and $Q_3[1,3] = 1$ (for vx) and $Q_3[1,2] = -1$ (for vw) and $L_3 = v$.

Stage 2. Now, we can model a sufficient condition of linearity using the previous ingredients: if there exist $\lambda_1, \ldots, \lambda_m \in \mathbb{F}$ such that, for every i, j with $1 \leq i \leq j \leq n$, we have that $\sum_{k=1}^{m} \lambda_k * Q_k[i,j] = 0$, then $l = \sum_{k=1}^{m} \lambda_k * (L_k - C_k)$ is linear and $S \models l = 0$. Moreover, assuming that S is consistent, we have that either $l = 0$ is a tautology $0 = 0$ or it is a non-trivial linear constraint. In the first case, any of the constraints Q_k with $\lambda_k \neq 0$ follows from the rest of the constraints and we can apply the REMOVE rule. In the second case, we can apply DEDUCE and later SIMPLIFY if l has at least one private signal. Note that, after applying SIMPLIFY one of the constraints Q_k with $\lambda_k \neq 0$ will follow from the rest, and we will be able to finally apply REMOVE.

Example 6 (continued). Following the example, we need to find $\lambda_1, \lambda_2, \lambda_3$ such that (considering only the non-zero coefficients $Q[i,j]$) $2\lambda_1 + \lambda_2 = 0$ (for $Q[2,4]$), $2\lambda_3 = 0$ (for $Q[1,3]$), and $-\lambda_3 = 0$ (for $Q[1,2]$). Since the monomials vx and vw occur only once, the only solution for λ_3 is 0. Now solving $2\lambda_1 + \lambda_2 = 0$, we get that $\lambda_2 = -2\lambda_1$. Hence, we take the solution $\lambda_1 = 1$ and $\lambda_2 = -2$. With this solution, $l = 1 * (-2w - (4x + 10)) + (-2) * (-2w - (w + 3)) + 0 * (v - (v - 1))$. Hence, we obtain $4w - 4x - 4 = 0$, which is equivalent to $x - w + 1 = 0$ that is the deduced linear constraint used in Example 4 to reduce the original circuit.

To conclude, finding $\lambda_1, \ldots, \lambda_m \in \mathbb{F}$ such that for every i, j with $1 \leq i \leq j \leq n$, then $\sum_{k=1}^{m} \lambda_k * Q_k[i,j] = 0$, is a linear problem that can be solved using Gaussian elimination or similar techniques. Note that we are only interested in solutions with at least one $\lambda_k \neq 0$. Therefore, we can efficiently synthesize new linear constraints or show that some constraint follows from the others using this approach.

Regarding the practical application of our technique, since sometimes we are handling very large sets of non-linear constraints, additional engineering is needed to make it work. For instance, we need to remove those constraints that have a quadratic monomial that appears in no other constraint, and after that, compute maximal clusters sharing the same quadratic monomials. We have observed in our experimental evaluation that, in general, even for large circuits, each cluster remains small. Thanks to this, we obtain rather small independent sets of constraints that can be solved in parallel using Gaussian elimination.

5 Experimental Results

This section describes our experimental evaluation on two settings: On one hand (Sect. 5.1), we have implemented them within `circom` [21], a novel domain-specific language and compiler for defining arithmetic circuits, fully written in Rust. The `circom` compiler generates executable code (`WebAssembly` or `C++`) to compute the witness, together with the R1CS, since both are later needed by ZK tools to produce ZK proofs. The implementation is available in a public fork of

the compiler [1]; On the other hand (Sect. 5.2), we have decoupled the constraint optimization module from the circom compiler in a new project, which is accessible online [2], in order to be able to use it after other cryptographic-language compilers that produce R1CS, in our case with ZoKrates [12]. ZoKrates is a high-level language that allows the programmer to abstract the technicalities of building arithmetic circuits. The input to our optimizer is the R1CS in the smtlib2 format generated by ZoKrates. The goal of our experiments is two fold: (1) assess the scalability of the approach when applied to real-world circuits used in industry and (2) evaluate its impact on code already highly optimized –such as circom's libraries developed on a low-level language by experienced programmers– and on code automatically compiled from a high-level language such as ZoKrates. In both cases, the optimizations of linear constraints that the compilers include (see Sect. 4) are enabled so that the reduction gains are due only to our optimization. Experimental results have been obtained using an AMD Ryzen Threadripper PRO 3995WX 64-Cores Processor with 512 GB of RAM (Linux Kernel Debian 5.10.70-1).

5.1 Results on circom Circomlib

circom is a modular language that allows the definition of parameterizable small circuits called "templates" and has its own library called circomlib [22]. This library is widely used for cryptographic purposes and contains hundreds of templates such as comparators, hash functions, digital signatures, binary and decimal converters, and many more. Our experiments have been performed on the available test cases from circomlib. Many of them have been carefully programmed by experienced cryptographers to avoid unnecessary non-linear constraints and our optimization cannot deduce new linear constraints. Still, we are able to reduce 26% of the total tests (12 out of 46).

Table 1 shows the results for the five circuits that we optimize the most. For each of them, we show: (**#C**) the number of generated constraints, (**#R**) the number of removed constraints, (**G%**) the gains expressed as **#R/#C x 100**, and (**T(s)**) the compilation time. The largest gain is for pointbits_loopback, where circom generates 2.333 constraints and we remove 381 of them, our gain is 16.33% and the compilation time is 13.4s. As explained in Sect. 4, for each linear constraint deduced by our technique, we are always able to remove a non-linear constraint and, in general, also a signal. Note that we sometimes produce new linear constraints in which all the involved signals are public and thus, none of them can be removed. Importantly, in spite of the manual simplifications already made in most of the circuits in circomlib, our techniques detect further redundant constraints in a short time. Such small reductions in templates of circomlib

Table 1. Results on circomlib.

Circuit	#C	#R	G%	T(s)
sha256_2_test	30134	32	0.11%	15.6s
eddsamimc_test	5712	46	0.81%	1.9s
eddsaposeidon_test	4217	46	1.09%	1.7s
eddsa_test	7554	556	7.36%	4.8s
pointbits_loopback	2333	381	16.33%	13.4s

can produce larger gains, since they are repeatedly used as subcomponents in industrial circuits.

5.2 Results on ZoKrates Stdlib

We have used two kind of circuits from the `ZoKrates stdlib` for our experimental evaluation: (1) The first four circuits `shaXbit` are implementations of different SHA-2 hash functions [19], where `X` indicates the size of the output. SHA-2 hashes are constructed from the repeated use of simple computation units that heavily use bit operations. Bit operations are very inefficient inside arithmetic circuits [13] and, as a result, the number of constraints describing these circuits is very large, see in Table 2. The number of constraints deduced is quite low for this kind of circuits since specialized optimization for bitwise operation is required (other compilers like `xJsnark` [23] are specialized on this). This also happens in the `circom` implementation of SHA-256-2 (row 1 of Table 1). However, Poseidon [16] is a recent hash function that was designed taking into account the nature of arithmetic circuits in a prime field $\mathbb{F}$, and as a result, the function can be described with many less constraints. Our approach is able to optimize the current implementation of Poseidon by more than 20%, which represents a very significant reduction. (2) The second kind are the last four circuits: they correspond to the ground for implementing elliptic curve cryptography inside circuits. Our optimizer detects, in a negligible time, that more than 23% of constraints are redundant and can be removed. Verifying if a pair of public/private keys matches (`ProofOfOwnership`) is fundamental in almost every security situation, hence the optimization of this circuit becomes particularly relevant for saving blockchain space. For this reason, we have parameterized `ProofOfOwnership` to the number of pairs public/private keys to be verified and we have measured the performance impact (time and memory consumption) of `snarkjs` setup step of these circuits without simplification (Table 3) and after simplification (Table 4). The results show the effect of our reduction when the constraints are later used by `snarkjs` to produce ZK proofs.

Table 2. Results on `stdlib`.

Circuit	#C	#R	G%	T(s)
`sha256bit`	25730	288	1.1%	35.0s
`sha512bit`	26838	544	2.0%	37.8s
`sha1024bit`	54284	1312	2.4%	82.4s
`sha1536bit`	81730	2080	2.6%	128s
`Poseidon`	3912	851	21.8%	0.3s
`EdwardsAdd`	17	4	23.6%	0.07s
`EdwardsOrderCheck`	56	15	26.8%	0.07s
`EdwardsScalarMult`	9989	2304	23.1%	0.2s
`ProofOfOwnership`	9984	2306	23.0%	0.5s

Table 3. Results on different instantiations of `ProofOfOwnership` from `stdlib` without nonlinear simplification. The generated ERROR in last row is an out-of-memory-error.

Circuit	Generation			`snarkjs`	
	T(s)	#C	Size	T(s)	Memory
`ProofOfOwnership-400`	1m58.1s	3,902,378	582MB	7m26.8s	14.4GB
`ProofOfOwnership-1000`	4m54.7s	9,740,978	1.5GB	37m50.0s	33.1GB
`ProofOfOwnership-1200`	6m09.6s	11,687,178	1.7GB	47m15.7s	36.2GB
`ProofOfOwnership-1400`	6m50.1s	13,633,378	2.0GB	ERROR	ERROR

Table 4. Results on different instantiations of `ProofOfOwnership` from `stdlib` with nonlinear simplification.

Circuit	Generation			`snarkjs`	
	T(s)	#C	Size	T(s)	Memory
`ProofOfOwnership-400`	3m11.0s	2,970,072	451MB	5m00.1s	12.7GB
`ProofOfOwnership-1000`	8m05.1s	7,413,672	1.1GB	23m40.8s	24.6GB
`ProofOfOwnership-1200`	9m43.8s	8,894,872	1.4GB	31m46.8s	30.7GB
`ProofOfOwnership-1400`	11m06.4s	10,376,072	1.6GB	38m31.0s	32.7GB

The impact of our simplification on the setup step of `snarkjs` is relevant and goes beyond the increase in the compilation time. However, this step is applied only once. We have also measured the impact in performance when generating a ZK-proof for a given witness using `snarkjs` after the setup step. This action that is the one repeated many times when used in a real context. Our experiments show that, e.g., with `ProofOfOwnership-400` we improve from 41 s to 35 s and with `ProofOfOwnership-1000` we improve from 1 m 53 s to 1 m 12 s.

In conclusion, our experiments show that the higher the level of abstraction is, the more redundant constraints the compiler introduces in the R1CS. Our proposed techniques are an efficient and effective solution to enhance the performance in this setting. On the other hand, circuits written in a low-level language by security experts (usually optimized by hand), or circuits using bitwise operations, leave small room for optimization by applying our techniques.

6 Related Work and Conclusions

We have proposed the application of (non-linear) constraint reasoning techniques to the new application domain of ZK protocols. Our approach has wide applicability as, in the last few years, much effort has been put in developing new programming languages that enable the generation and verification of ZK proofs and that also focus on the design of arithmetic circuits and the constraint encoding. Among the different solutions, we can distinguish: libraries (`bellman` [7], `libsnark` [29], `snarky` [28]), programming-focused languages (`ZoKrates` [12],

`xJsnark` [23], `zinc` [24], `Leo` [10]), and hardware-description languages (`circom`). As opposed to the initial library approach, both programming and hardware-description languages put focus on the design of arithmetic circuits and the constraint encoding. In this regard, `ZoKrates`, `xJsnark`, and the `circom` compiler implement one simple but powerful R1CS-specific optimization called *linearity reduction*: it consists in substituting the linear constraints to generate a new circuit whose system only consists of non-linear constraints. However, they do not deduce new constraints to detect further redundancies in the system. Linear reduction is a particular case of our reduction rules in which the only linear constraints that can be deduced and added to the linear system are those that follow from linear constraints present in the constraint system. On the other side, the constraint system generated by `Leo` is only optimized at the level of its intermediate representation not at R1CS-level, as our method works.

Finally, there has been a joint effort towards standardizing and allowing the interoperability between different programs, like `CirC` [26], an infrastructure for building compilers to logical constraint representation. Currently, `CirC` only applies the linearity reduction explained above. Recently, an interface called `zkInterface` [6] has been built to improve the interoperability among several frontends, like `ZoKrates` and `snarky`. It provides means to express statements in a high-level language and compile them into an R1CS representation; and several backends that implement ZK protocols like Groth16 [17] and Pinocchio [27] that use the R1CS representation to produce ZK proofs. `zkInterface` could benefit from our optimization to apply our reduction to every circuit generated by any of the accepted frontends. `zkInterface` is also written in Rust, then our optimizer could be easily integrated as a new gadget for the tool in the future. Finally, we believe that the techniques presented in this paper can lead us to new reduction schemes to be applied over PlonK [14] constraint systems.

Acknowledgements. This research was funded by the Spanish MCIN-AEI-10.13039/501100011033-FEDER "Una manera de hacer Europa" projects RTI2018-094403-B-C31 and RTI2018-094403-B-C33, by the CM project S2018/TCS-4314 co-funded by EIE Funds of the European Union, and by the project RTI2018-102112-B-100 (AEI/FEDER, UE).

References

1. Albert, E., Bellés-Muñoz, M., Isabel, M., Rodríguez-Núñez, C., Rubio, A.: circom fork including non-linear simplification. GitHub (2022). github.com/clararod9/circom. Accessed 21 Jan 2022
2. Albert, E., Bellés-Muñoz, M., Isabel, M., Rodríguez-Núñez, C., Rubio, A.: An optimizer for non-linear constraints. GitHub (2022). github.com/miguelis/nonlinearoptimizer. Accessed 21 Jan 2022
3. Bellés-Muñoz, M., Whitehat, B., Baylina, J., Daza, V., Tapia, J.L.M.: Twisted edwards elliptic curves for zero-knowledge circuits. Mathematics, 9(23), 2021
4. Sasson, E.B., et al.: Zerocash: decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 459–474 (2014)

5. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_6
6. Benarroch, D., Gurkan, K., Kahat, R., Nicolas, A., Tromer, E.: zkInterface, a standard tool for zero-knowledge interoperability, June 2019. github.com/QED-it/zkinterface/blob/master/zkInterface.pdf. Accessed 15 Jan 2022
7. Bowe, S.: Bellman: zk-snark library. github.com/ebfull/bellman. Accessed 15 Jan 2022
8. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 315–334 (2018)
9. Champion, A., Gurfinkel, A., Kahsai, T., Tinelli, C.: CoCoSpec: a mode-aware contract language for reactive systems. In: De Nicola, R., Kühn, E. (eds.) SEFM 2016. LNCS, vol. 9763, pp. 347–366. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41591-8_24
10. Chin, C., Wu, H., Chu, R., Coglio, A., McCarthy, E., Smith, E.: Leo: a programming language for formally verified, zero-knowledge applications. IACR Cryptology ePrint Archive, Report 2021/651 (2021). ia.cr/2021/651. Accessed 15 Jan 2022
11. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. ACM Trans. Comput. Log., **19**(3), 19:1–19:52 (2018)
12. Eberhardt, J., Tai, S.: ZoKrates - scalable privacy-preserving off-chain computations. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 1084–1091 (2018)
13. Gabizon, A., Williamson, Z.J.: Plookup: a simplified polynomial protocol for lookup. IACR Cryptology ePrint Archive, Report 2020/315 (2020). ia.cr/2020/315. Accessed 15 Dec 2021
14. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953 (2019). eprint.iacr.org/2019/953
15. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, STOC '85, pp. 291–304, New York, NY, USA (1985). Association for Computing Machinery
16. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: a new hash function for zero-knowledge proof systems. IACR Cryptology ePrint Archive, Report 2019/458, 2019. ia.cr/2019/458. Accessed 15 Dec 2021
17. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
18. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: layer-two blockchain protocols. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 201–226. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_12
19. Handschuh, H.: SHA Family (Secure Hash Algorithm), pp. 565–567. Springer, US, Boston, MA (2005)
20. Hermez Network. Hermez whitepaper, October 2020. hermez.io/hermez-whitepaper.pdf. Accessed 15 Dec 2021

21. Iden3. Circom: Circuit compiler for zero-knowledge proofs. GitHub (2020). ithub.com/iden3/circom. Accessed 21 Jan 2022
22. Iden3. Circomlib: Library of circom templates. GitHub (2020). github.com/iden3/circomlib. Accessed 15 Dec 2021
23. Kosba, A., Papamanthou, C., Shi, E.: xJsnark: a framework for efficient verifiable computation. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 944–961 (2018)
24. Matter Labs. Zinc v0.2.3. Cryptology ePrint Archive, Report 2019/953 (2019). ia.cr/2020/352. Accessed 15 Dec 2021
25. Nakos, G.C., Turner, P.R., Williams, R.M.: Fraction-free algorithms for linear and polynomial equations. SIGSAM Bull. **31**(3), 11–19 (1997)
26. Ozdemir, A., Brown, F., Wahby, R.S.: Unifying compilers for snarks, smt, and more. IACR Cryptology ePrint Archive, Report 2020/1586, 2020. ia.cr/2020/1586. Accessed 15 Jan 2022
27. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 238–252. IEEE, May 2013
28. Protocol, M., Snarky. GitHub (2020). minaprotocol.com/blog/snarky-a-high-level-language-for-verifiable-computation. Accessed 21 Jan 2022
29. Succinct Computational Integrity and Privacy Research (SCIPR) Lab. libsnark: a c++ library for zk-snark proofs. GitHub, First release, June 2014. github.com/scipr-lab/libsnark. Accessed 15 Jan 2022

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.