Implementable Witness Encryption from Arithmetic Affine Determinant Programs

Lev Soukhanov Yaroslav Rebenko

lev@allocinit.xyz yar@allocinit.xyz

Muhammad El Gebali Mikhail Komarov

gebali@allocinit.xyz nemo@allocinit.xyz

[[alloc] init]

Abstract.

We propose a Witness Encryption scheme that is practically implementable for an instance that contains verification of a general-purpose SNARK for NP. Our construction is a modification of the Affine Determinant Program framework adapted for a certain class of arithmetic circuits.

1 Introduction

Witness encryption introduced in [GGSW13] is a cryptographic primitive that allows to encrypt a message with a statement of an NP problem and decrypt it with any valid witness.

A scheme from [BIJ⁺20], currently closest to practical, uses the language of ADPs - affine determinant programs.

Theoretically, any WE scheme can be bootstrapped to have a ciphertext size independent of a statement by composing it with a SNARK. Sadly, [BIJ+20] scheme falls short of this goal: it uses VSS (vector subset-sum language) over a field $\mathbb{F}$ and the ciphertext size scales like $\approx n^3|\mathbb{F}|$ where n is the size of the VSS instance. Verification of a SNARK in VSS language has an instance size $\approx 10^9$, which yields 10^{27}-sized ciphertext.

In this note we suggest a modification that we call arithmetic ADPs (AADPs) that allows to use, instead of the VSS (vector subset sum) language a certain class of arithmetic constraint systems. This turns out to be enough to embed a general purpose SNARK in a WE statement, with our preliminary estimates suggesting ciphertext sizes ≈ 338 TB at 100 bits of security.[1]

The paper is organized as follows:

[1] Initially, we wanted to call our paper "Practical Witness Encryption from Arithmetic Affine Determinant Programs". While we are not completely convinced that this range of ciphertext sizes is practical it is definitely something that can be implemented in principle.

– In Section 2 we define AADPs, explain the differences with normal ADPs and the language of projectively safe constraint systems and define AADPs-based witness encryption.
– In Section 3 we discuss some attacks on our scheme and mitigations that we employed.
– In Section 4 we struggle against the extremely restrictive notion of projective safety and suggest a certain pairing-based proof system inspired by Garuda [DMS24] that can be effectively verified using projectively safe constraint systems.

Acknowledgement. Authors thank Handan Kilinc Alper and Patrick Towa for proofreading of the early version of this paper, Elena Tatuzova for help with circuit modeling and Dmitry Khovratovich for comments on algebraic hashes.

2 Arithmetic Affine Determinant Programs

2.1 ADPs

First we give a short review of the ADP construction [BIJ+20]

Definition 1. *An affine determinant program* $\mathsf{ADP} : \{0,1\}^n \to \mathsf{Bool}$ *is specified by square matrices* $M^0, \dots, M^n$ *over a field* $\mathbb{F}$ *and a function* $\mathsf{Eval} : \mathbb{F} \to \mathsf{Bool}$*. It is evaluated on input* $\mathbf{x} \in \{0,1\}^n$ *by computing*

$$\mathsf{Eval}\left(\det\left(M^0 + \sum_{i=1}^{n} x_i M^i\right)\right)$$

We modify this definition to accept inputs from an algebraic closure $\overline{\mathbb{F}}$. Changes are highlighted.

Definition 2. *An arithmetic affine determinant program* $\mathsf{ADP} : \overline{\mathbb{F}}^n \to \mathsf{Bool}$ *is specified by a collection of square matrices* $M^0, \dots, M^n$ *over a field* $\mathbb{F}$*. It is evaluated on a* ***nonzero*** *input* $\mathbf{x} \in \overline{\mathbb{F}}^n$ *by computing*

$$\mathsf{Eval}_0\left(\det\left(M^0 + \sum_{i=1}^{n} x_i M^i\right)\right)$$

Our main objective is to adapt ADPs for arithmetic circuits and, ideally, to avoid decomposition into bits. The changes are motivated as follows:

1. Given that the determinant defines a polynomial defined over $\mathbb{F}$, the inputs are necessarily from $\overline{\mathbb{F}}$ as it is impossible to prevent the evaluation of an ADP on elements of an algebraic closure.
2. Eval mapping is specified to $\mathsf{Eval}_0(v) = \text{if } (v = 0)\ \{1\} \text{ else } \{0\}$ because it is the only case relevant to the witness encryption construction.

A significant effort in the original paper is spent on the "rejection of invalid inputs" property, which enforces $(x_1, \dots x_n) \in \{0,1\}^n$. In our case, we enforce a different quadratic nonlinear relation.

In what follows, we will always assume that $\mathbb{F}$ is cryptographically large. While it is not strictly needed in the original paper, it significantly simplifies our analysis and our main application uses a large field anyway.

In the next section, we will specify the class of constraint systems for which we define our arithmetic ADP programs.

2.2 Projectively safe constraint systems language

For $m \times (n+1)$ matrices A, B, C, D over $\mathbb{F}$ consider the following system of constraints:

$$(i) : (\sum_{j=0}^{n} A_i^j x_j) \cdot (\sum_{j=0}^{n} B_i^j x_j) = (\sum_{j=0}^{n} C_i^j x_j) \cdot (\sum_{j=0}^{n} D_i^j x_j)$$

where $x_0 = 1$ and $(x_1, \dots, x_n)$ are independent variables in $\overline{\mathbb{F}}$.

We will also frequently consider projective solutions - i.e. vectors $(x_0, \dots, x_n) \neq 0$ satisfying this constraint system but without the requirement $x_0 = 1$. Any projective solution can be rescaled to a regular affine solution if $x_0 \neq 0$, solutions with $x_0 = 0$ will be referred as "solutions at infinity".

Definition 3. *We call such a constraint system* ***projectively safe*** *if for each variable x_i there is a constraint (j_i) having the form $x_i^2 = a(x_0, \dots, x_{i-1}) \cdot b(x_0, \dots x_n)$ - i.e. in the matrix notation $A_k^{j_i} = 0$ for $k \geq i$, $C_k^{j_i} = D_k^{j_i} = \delta(i,k)$.*

A constraint (j_i) is called a ***certificate*** *for the variable i.*

The idea is that we want to control the solutions at infinity.

Lemma 1. *Projectively safe constraint system has no solutions at infinity.*

Proof. Indeed, a constraint (j_i) implies that $x_i = 0$ as long as all previous variables are 0. Therefore, for $x_0 = 0$ by induction all $x_i = 0$ (which is a non-solution, as we require $(x_0, \dots, x_n) \neq 0$). $\square$

The reason why we need this notion of projective safety is that all our constructions of ADP programs can not prevent the adversary from evaluating $\sum\limits_{i=0}^{n} x_i M^i$ for $x_0 = 0$. It is interesting that the original construction was able to sidestep this issue completely because the bitchecked constraint systems are automatically projectively safe:

Lemma 2. *Assume that every variable x_i is bitchecked, i.e. is subject to a constraint $x_i^2 = x_0 \cdot x_i$. Then, this system is projectively safe.*

Proof. By definition. $\square$

This lemma means that as long as we operate with variables that are constrained to be bits, the projective safety is not really necessary. It is still more efficient than boolean circuits - while each *variable* needs to be bitchecked, we still can use constraints on linear combinations of bitchecked variables. However, the main optimization techniques that we employ involve finding ways of not bit-checking individual variables while still maintaining projective safety.

2.3 ADP generation procedure

We define an algorithm $\mathsf{AADP.Gen} : (A, B, C, D) \mapsto M$ that takes as an input projectively safe constraint system (A, B, C, D) with n variables and m constraints and outputs an arithmetic affine determinant program M.

We will use the notation $\overset{\$}{\leftarrow}$ to denote an uniform random choice in a set and we will work with linear combinations in variables $x_0, \dots x_n$. In particular, our M will be a linear combination-valued matrix of dimensions $(2m+1) \times (2m+1)$ which corresponds to $M(x) = \sum_{i=0}^{n} x_i M^i$ in standard ADP notation.

Algorithm 1 Algorithm AADP.Gen

```
for $0 \le i < m$ do
    $a_i(x) \leftarrow \sum_{j=0}^{n} A_i^j x_j$
    ▷ repeat for $b, c, d$ and $B, C, D$ respectively ◁
    $\xi_i(x) \leftarrow \sum_{j=0}^{n} \xi_i^j x_j$ where for $0 \le j \le n : \xi_i^j \overset{\$}{\leftarrow} \mathbb{F}$
    $U_i(x) \leftarrow \begin{pmatrix} a_i(x) & c_i(x) & -\xi_i(x) & 0 \\ d_i(x) & b_i(x) & 0 & \xi_i(x) \\ 0 & 0 & b_i(x) & c_i(x) \\ 0 & 0 & d_i(x) & a_i(x) \end{pmatrix}$
    $L_i \overset{\$}{\leftarrow} [\mathbb{F}]^{(2m+1)\times 4}$
    $R_i \overset{\$}{\leftarrow} [\mathbb{F}]^{4\times(2m+1)}$
    $M_i(x) \leftarrow L_i \cdot U_i(x) \cdot R_i$
end for
return $M(x) = \sum_{i=0}^{m-1} M_i(x)$
```

Lemma 3. *For a given x,* $\operatorname{rank}(U(x)) = \begin{cases} 4 & ab \neq cd \\ 2 & ab = cd \end{cases}$ *except with negligible probability.*

Proof.

- If $ab \neq cd$: $\det(U) = (ab - cd)^2 \neq 0$ hence the matrix has a full rank
- If $ab = cd$: bottom two rows $\begin{bmatrix} 0 & 0 & b & c \\ 0 & 0 & d & a \end{bmatrix}$ are linearly dependent and can be rewritten using Gauss elimination as $\begin{bmatrix} 0 & 0 & v & u \\ 0 & 0 & 0 & 0 \end{bmatrix}$ and the same can be done for the two left columns. Substituting the result back into U we obtain a minor $\widetilde{U} = \begin{bmatrix} u & -\xi & 0 \\ v & 0 & \xi \\ 0 & v & u \end{bmatrix}$ with $\det(\widetilde{U}) = 0$ and $\operatorname{rank}(U) = \operatorname{rank}(\widetilde{U}) \le 2$. Except with

negligible probability we have $\xi \neq 0$ and so rank can not be less than 2 due to a submatrix $\begin{bmatrix} -\xi & 0 \\ 0 & \xi \end{bmatrix}$ having nonzero determinant.

□

Theorem 1. *For a constraint system (A, B, C, D) and a projective input vector $\widetilde{x} = (x_0, \dots, x_n)$*

- *If $\widetilde{x}$ is a valid projective solution of the constraint system,* $\det M(\widetilde{x}) = 0$ *and M is statistically indistinguishable from a random matrix of rank $2m$.*
- *If $\widetilde{x}$ is not a valid projective solution, M is statistically indistinguishable from a random $2m + 1$ (full) rank matrix.*

This property is called "one-time security" in [BIJ+20].

Proof. Consider a matrix $M_i(x) = L_i U_i(x) R_i$ produced on the i-th step of the algorithm AADP.Gen (Algorithm 1). By Lemma 3 if a constraint (i) does not hold, the matrix $U_i(x)$ has a rank 4 and if it does hold, it has rank 2 except with negligible probability.

The distribution of matrices $M_i(x)$ is symmetric under the following changes:

$$L_i \mapsto X L_i$$

$$R_i \mapsto R_i Y$$

and hence

$$M_i(x) \mapsto X M_i(x) Y$$

where X, Y are random invertible $2m + 1 \times 2m + 1$ matrices.

This group of transforms acts transitively on the space of matrices of a fixed rank. This means that $M_i(x)$ is statistically indistinguishable from a random rank 2 matrix if a constraint (i) held, and random rank 4 matrix if it didn't.

Sum of m random equidistributed rank 2 matrices is a matrix of rank $2m$ except with negligible probability. On the contrary, if at least one of the constraints didn't hold, sum is the matrix of the full rank except with negligible probability.

To see that the sum is statistically indistinguishable from a random matrix (of ranks $2m$ and $2m + 1$ respectively), consider once again an action that simultaneously changes each $L_i \mapsto X L_i$ and $R_i \mapsto R_i Y$. This transforms $M(x) \mapsto X M(x) Y$ which is the transitive action. Therefore, $M(x)$ is equidistributed. □

Our construction is analogous to the construction from the original paper [BIJ+20] with slight changes:

- Our construction resembles "formula-based all-accept", but as an individual building block [BIJ+20] uses the following bitcheck matrix $\begin{pmatrix} b & \xi \\ 0 & x_0 - b \end{pmatrix}$

– Our constraints are applied to linear combinations and not individual witness elements. We can also eliminate linear constraints from the original construction in a following way: instead of communicating individual M^i-s we can only include in the ciphertext linear combinations $\sum x_i M^i$ for some basis in the space of the solutions of the linear part of the system.
– We do have the terms ξ_i^0. In the original construction their analogue could be removed via a basis change; it appears that in our case it is impossible.

2.4 Witness encryption from ADPs

Witness encryption reminder
Witness encryption, introduced in [GGSW13] is a generalization of public key encryption for arbitrary NP languages. Specifically, given a message space Msg, an NP-language $\mathcal{L}$ and corresponding witness relation $\mathcal{R}(x, w)$, witness encryption with security parameter 1^λ is a pair of PPT algorithms $\mathsf{WE.Enc}(1^\lambda, x, M)$ and $\mathsf{WE.Dec}(\mathrm{CT}, w)$, such that

– **Correctness**: for all messages $\mathsf{msg} \in \mathsf{Msg}$, $x \in \mathcal{L}$ and w such that $(x, w) \in \mathcal{R}$ holds
$$\Pr[\mathsf{WE.Dec}(\mathsf{WE.Enc}(1^\lambda, x, M), w) = M] = 1 - \mathsf{negl}(\lambda)$$
i.e., x plays the role of a public key and w plays the role of a corresponding private key.
– **Extractable security**: for any PPT adversary $\mathcal{A}$ and poly q there exists a PPT extractor $\mathcal{E}$ and poly p such that for all auxiliary inputs z and for all $x \in \{0,1\}^*$
$$\Pr_{\substack{\mathsf{msg} \leftarrow \mathsf{Msg} \\ \mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, x, M)}}[\mathcal{A}(\mathsf{ct}, x, z) = M] \geq \frac{1}{|\mathsf{Msg}|} + \frac{1}{q(|x|)}$$
$$\implies \Pr[\mathcal{E}(x, z) = w : (x, w) \in \mathcal{R}] \geq \frac{1}{p(|x|)}$$
Or informally: if there is an adversary that can reliably decrypt the message using some knowledge z there is an extractor that can use the same knowledge to solve the instance.

We will omit the 1^λ parameter of $\mathsf{WE.Enc}$.

Witness encryption from ADPs
Now we provide a Witness Encryption construction for AADPs. It is an adaptation of the definition from the section 6.1.3 of the original paper [BIJ+20] to the large field setting.

Definition 4. *Define an algorithm*
$$\mathsf{WE.Enc} : ((A, B, C, D), \mathsf{msg}) \mapsto \widehat{M}$$

which takes as an input a projectively safe constraint system (A, B, C, D) *and a message* $\mathsf{msg} \in \mathbb{F}$ *and outputs an (arithmetic ADP-formatted) ciphertext* $\widehat{M}$ *and a corresponding algorithm*

$$\mathsf{WE.Dec} : \left(x, \widehat{M}\right) \mapsto \mathsf{msg}$$

which takes as an input a solution $x = (x_1, \ldots, x_n)$ *to the constraint system and a ciphertext* $\widehat{M}$ *and outputs the encrypted message* msg*:*

Algorithm 2 Algorithm WE.Enc

```
$M \leftarrow \mathsf{AADP.Gen}((\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}))$
$\widehat{M} \leftarrow M + \mathsf{msg} \cdot x_0 \cdot [\underbrace{0, \ldots, 0}_{2m \text{ times}}, 1]^T \cdot [\underbrace{0, \ldots, 0}_{2m \text{ times}}, 1]$
▷ Here, as before, $x_0$ denotes a formal variable in $M$.  ◁
return $\widehat{M}$
```

Encryption algorithm just produces an arithmetic ADP and adds a message to a single cell in a matrix M^0.

Algorithm 3 Algorithm WE.Dec

```
$E \leftarrow \widehat{M}((1|x))$
Solve in $t$ :   $\det(E - t \cdot x_0 \cdot [\underbrace{0, \ldots, 0}_{2m \text{ times}}, 1]^T \cdot [\underbrace{0, \ldots, 0}_{2m \text{ times}}, 1]) = 0$
return $\mathsf{msg} = t$
```

Decryption algorithm computes a determinant treating msg as an indeterminate variable, and then solves a linear equation in it. Compared to the original method from the section 6.1.3 of [BIJ+20] we do not use any additional columns because our field is cryptographically large which implies that considered linear equation has nonzero coefficient of t except with negligible probability, so the decryption procedure doesn't fail.

Lemma 4. $\mathsf{WE.Dec}$ *is correct: i.e. it successfully decrypts the output of* $\mathsf{WE.Enc}$*.*

Proof. Fix any valid witness x and let $E \leftarrow \widehat{M}((1 \mid x))$ be the evaluation of the ciphertext ADP at $(x_0, x) = (1, x)$.

Topleft $2m \times 2m$ minor of the matrix $\widehat{M}(x)$ has nonzero determinant except with negligible probability. This is clear because the matrix $M(x)$ has rank $2m$ and (for a fixed x and varying setup parameters) is equidistributed in the space of all matrices of rank $2m$ by Theorem 1.

Hence, the linear equation in Algorithm 3 has a nonzero coefficient before t and therefore a unique solution.

Now, it is remains to see that original encrypted msg is indeed a solution. This just means that $\det M(1|x) = 0$ which is true by Theorem 1. □

3 Cryptanalysis

Neither AADPs nor original ADPs have a reduction to a well-established security assumption, so we need to come up with and analyze specific attacks on our scheme.

3.1 Correlated span attack and projective safety

Our starting point is the correlated span attack (see Section 7.1.1 of [BIJ+20] for the original description). It is a randomness recovery attack that requires the adversary to know multiple solutions of the original statement. However, it also would work if the adversary was able to sample a lot of projective solutions at infinity: which is a main motivation behind our notion of projective safety.

We give here a generalized version of the correlated span attack adapted to our setting. We assume the attacker has access to multiple projective solutions to the constraint system. In the description below, we will assume that these solutions are generic in a sense that they have some nonzero values on each gate and various linear systems formed from them have an expected dimensions of solution spaces. In the cases where an attacker can only sample solutions from some linear subspace it might be harder to mount the attack.

Consider a solution w to a constraint system that corresponds to AADP program M. As $\det(M(w)) = 0$, there is a kernel vector $M(w) \cdot \alpha_w = 0$. If it is unique up to rescaling (which happens except with negligible probability), then $\forall i : U_i(w)R_i\alpha_w = 0$. This happens because rank of each of these systems is 2 and there are m conditions in total, so they do have a common solution vector. This common vector also necessarily lies in a kernel of $M(w)$, so it must be α_w.

Consider the i-th gate $a_ib_i = c_id_i$. Denote the rows of the matrix R as R_i^1, R_i^2, R_i^3, R_i^4. Then, for $(b_i(w), c_i(w)) \neq (0, 0)$ the third row of the matrix U_i gives us the following ξ-independent equation:

$$b_i(w)R_i^3\alpha_w + c_i(w)R_i^4\alpha_w = 0$$

for each i.

Treating R_i^3, R_i^4 as variables and sampling $2m$ independent pairs (w, α_w), we obtain enough linear equations to find all the coefficients of R_i^3 and R_i^4.

Analogously, starting from a dual kernel $\beta_w M(w) = 0$ one can recover the first two columns L_i^1, L_i^2 of the L-matrices.

Lemma 5. *Let w be a valid projective solution, v any vector satisfying*

$$(b_i(w)R_i^3 + c_i(w)R_i^4)v = 0$$

and u any covector satisfying

$$u(c_i(w)L_i^1 + b_i(w)L_i^2) = 0$$

Then, a term $uL_iU_i(w)R_iv = 0$

Proof. Explicitly write down $uL_iU_i(w)R_iv$ and simplify it using $a_i(w)b_i(w) = c_i(w)d_i(w)$. We will avoid writing evaluation in w to reduce the visual clutter:

$$\left(uL_i^1,\, uL_i^2,\, uL_j^3,\, uL_j^4\right) \cdot \begin{pmatrix} a_i & c_i & -\xi_i & 0 \\ d_i & b_i & 0 & \xi_i \\ 0 & 0 & b_i & c_i \\ 0 & 0 & d_i & a_i \end{pmatrix} \cdot \begin{pmatrix} R_i^1v \\ R_i^2v \\ R_i^3v \\ R_i^4v \end{pmatrix} =$$

$$\left(uL_i^1,\, uL_i^2,\, uL_j^3,\, uL_j^4\right) \cdot \begin{pmatrix} a_iR_i^1v + c_iR_i^2v - \xi_iR_i^3v \\ d_iR_i^1v + b_iR_i^2v + \xi_iR_i^4v \\ 0 \\ 0 \end{pmatrix} =$$

$$a_i(uL_i^1R_i^1v) + d_i(uL_i^2R_i^1v) + c_i(uL_i^1R_i^2) + b_i(uL_i^2R_i^2v) + \xi(uL_i^2R_i^4v - uL_i^1R_i^3v)$$

Because of our assumption $b_i(R_i^3v) + c_i(R_i^4v) = 0$ we can express $R_i^3v = c_is$, $R_i^4v = -b_is$ for some s and similarly $uL_i^1 = b_it$, $uL_i^2 = -c_it$. Substituting this yields:

$$\ldots = (a_ib_it - d_ic_it)R_i^1v + (c_ib_i - b_ic_i)R_i^2v + \xi ts(c_ib_i - b_ic_i) = 0$$

□

This structure can be exploited to determine the remaining parts of the matrices, i.e. columns L_i^3, L_i^4 and rows R_i^1, R_i^2. In order to do that, pick a vector v and a covector u satisfying the following conditions:

$$\forall j \neq i : (b_j(w)R_j^3 + c_j(w)R_j^4)v = 0$$

$$\forall j \neq i : u(c_j(w)L_j^1 + b_j(w)L_j^2) = 0$$

$$R_i^3v = 0 \qquad R_i^4v = 0$$

Now, compute the value $uM(w)v = \sum_j uL_jU_j(w)R_jv$. By Lemma 5 only the i-th term survives. Writing it down explicitly gives:

$$uM(w)v =$$

$$uL_iU_i(w)R_iv = \left(uL_i^1,\, uL_i^2,\, uL_j^3,\, uL_j^4\right) \cdot \begin{pmatrix} a_iR_i^1v + d_iR_i^2v \\ c_iR_i^1v + b_iR_i^2v \\ 0 \\ 0 \end{pmatrix} =$$

$$(a_iuL_i^1 + c_iuL_i^2)R_i^1v + (d_iuL_i^1 + b_iuL_i^2)R_i^2v$$

By this point, all the values L_i^1, L_i^2 are already found, so this represents an inhomogeneous linear equation on the values of the matrices R_i^1, R_i^2. Sampling enough equations of this form, we fully recover values of R. Recovery of L-values is done analogously.

Now, it remains to only recover ξ-values. Notice under conditions that values of L_i and R_i are already known, any expression of the form $uM(w)v$ is a linear equation on values of ξ. To make this process more efficient, we can use only such pairs u, v that satisfy the system

$$\forall j \neq i : (b_j(w)R_j^3 + c_j(w)R_j^4)v = 0$$

$$\forall j \neq i : u(c_j(w)L_j^1 + b_j(w)L_j^2) = 0$$

which ensures that only coefficients of a single ξ_i participate in the equation.

Asymptotic complexity:

Each phase of the attack requires sampling $O(m)$ solutions and $O(m^4)$ field operations. A single step where it is not directly obvious is sampling ($O(m)$ per each of m gates) vectors u and v for each witness sample w in the second phase of the attack. To see that, observe that for each witness sample all the vectors v are taken from the common subspace of a linear system $\forall j : (b_j(w)R_j^3 + c_j(w)R_j^4)v = 0$, which can be found in $O(m^3)$.

This is somewhat worse than the original correlated span attack described in [BIJ$^+$20] which runs in $O(m^3)$ time, but our version has the significant advantage of being mountable in any case where many (generic enough) solutions are known. It is also likely that in case an additional structure is known on the solution space this attack can be significantly sped up.

3.2 Necessity of the projective safety condition

Due to the existence of the correlated span attack, projectively unsafe circuits are typically non-secure. An attacker can ignore the existence of the matrix M^0 and run the correlated span attack for the solutions at infinity to learn all the values of L_i, R_i and ξ_i^j-s for $j > 0$. It can then compute ξ_i^0 and msg by directly running the $\mathsf{WE.Enc}$ algorithm in indeterminates and solving the resulting system of linear equations.

3.3 Parametric rank drop search

It is not hard to come up with some string x such that $\det M(x) = 0$ in a non-blackbox way. For example, an adversary could pick a 1-dimensional family of inputs $x(t)$, compute $\chi_x(t) = \det M(x(t))$ as a polynomial in t and solve it, finding a root.

This can be promoted to a potential attack idea. What if the adversary finds such a projective family of assignments $\mathcal{X}$ that:

- It is fully at infinity.
- $\ell + 1$ constraints are broken.

– $\dim(\mathcal{X}) > \ell + 1$.

In this case, an adversary could inspect the situations where $\det M(x) = 0, x \in \mathcal{X}$ and at least some of those would correspond to the case where simultaneously $> \ell + 1$ fully vanishing constraints will also have the masking value $\xi = 0$. These would reveal a lot of useful information about the kernels of the individual constraint matrices.

Remark. While this is only an idea of the attack (as it is not clear how to detect these solutions in the space of all solutions to $\det M(x) = 0; x \in \mathcal{X}$), a slightly tweaked version of our protocol is suspectible to a concrete attack of this form using a 1-dimensional family. This happens if one tries to mix our multiplication matrices and bitcheck matrices from [BIJ+20].

The rank of the bitcheck constraint from [BIJ+20] is 1 when it is correct, 2 when broken and 0 when degenerate. Consider a family of solutions at infinity parametrized by t that breaks two bitcheck constraints: given a "projectively safe" system which involves a pair of bitchecked values b_1, b_2 set $b_1 = t, b_2 = (1 - t)$ and everything else to 0. This family passes through multiple values of t corresponding to situations where a value ξ of some (totally vanishing) multiplication constraint is 0. These situations will be revealed as roots of $\det M(x(t))$ as the rank of $M(x(t))$ jumps by 2 due to the break of two bitcheck constraints, but falls by 2 due to a single multiplication constraint being degenerate.

We prove the following:

Lemma 6. *Let $\mathcal{X} \hookrightarrow \mathbb{P}^{n-1} = \mathbb{P}(x_1 : \ldots : x_n)$ be a k-dimensional algebraic variety. A generic $x \in \mathcal{X}$ breaks at least $k + 1$ constraints.*

Proof. Let a generic element of this family pass all but $\ell+1$ certifying constraints corresponding to the variables $x_{i_0}, \ldots x_{i_\ell}$ (and also break any number of non-certifying ones). Consider the space of solutions $\mathcal{Y}$ to our system without certifying constraints corresponding to these variables (i.e., constraints $j_{i_s} : x_{i_s}^2 = \ldots$ are allowed to be broken). Obviously, $\mathcal{X} \subset \mathcal{Y}$.

On the other hand, $\mathcal{Y}$ is embedded into a sequence of finite coverings of the coordinate plane $\mathbb{P}(x_{i_0} : \ldots : x_{i_\ell})$, corresponding to certifying constraints $(j) : x_j^2 = \ldots$ for all $j \notin \{i_0, \ldots, i_\ell\}$. This means that $\mathcal{Y}$ and hence $\mathcal{X}$ is not more than ℓ-dimensional, implying $k \leq \ell$. □

This justifies that this attack is likely impossible in our version of the protocol.

3.4 Linearization attack

We make a best-effort attempt at launching a linearization attack analog to the one descriped in [BIJ+20, Section 7.1.2]. Similar to the original paper, we do not actually manage to mount it, but we both show that if it exists, then

– For practically relevant instances it has infeasible runtime.

– From theoretical standpoint, there is a mitigation.

The attack in the original paper was possible when the subset-sum instance was sparse and as such one could isolate some B_i matrices in the clear.

The analog for AADP is the existence of inputs x such that they are nonzero only on a small amount of quadratic constraints. Define

$$g(x) = 2 \cdot |\text{invalid constraints on x}| + |\text{nonzero but valid constraints on x}|$$

and denote $g_i = g(\delta_i)$ where δ_i is an input assignment that has $x_j = \delta(i,j)$.

Trivially, we know that $g(x) \geq 1$ for all x. In fact, for a projectively safe system we can assume wlog that $g(x) \geq 3$. Indeed:

– Any δ_i input has at least the certifying constraint (j_i) and if it is the only constraint it is involved in we can safely drop the variable i without changing the constraint system functionality.
– Any input which is not of the form δ_i is involved in at least 2 certifying constraints.

Therefore, for each x we have at least two constraints and at least one of them must not hold because of the projective safety condition.

Let $L_i^{(p,q)}$ consist of columns p and q of L_i and similarly $R_i^{(p,q)}$ consist of rows p and q of R_i.

For any input x denote $D(x) \in \mathbb{F}^{2m \times 2m}$ the diagonal matrix

$$D(x) = \mathsf{diag}([-\xi_0(x), \xi_0(x), \ldots, -\xi_{m-1}(x), \xi_{m-1}(x)])$$

$P \in \mathbb{F}^{2m+1 \times 2m}$ the horizontal concatenation of $L_j^{(3,4)}$ for $0 \leq j < m$ and Q the vertical concatenation of $R_j^{(1,2)}$.

Now, we claim that

$$M(x) = PD(x)Q + \mathsf{LowRank}_{\leq 2g(x)}$$

Indeed, the product $PD(x)Q$ contains all the contributions of the top right 2×2 blocks of $U_i(x)$ from the Algorithm 1. It remains to add the contributions from the top left and bottom right blocks. If a constraint is nonzero but does hold, the contribution, which is equal to

$$(L_i^1, L_i^2)\begin{pmatrix} a_i(x) & c_i(x) \\ d_i(x) & b_i(x) \end{pmatrix}\begin{pmatrix} R_i^1 \\ R_i^2 \end{pmatrix} + (L_i^3, L_i^4)\begin{pmatrix} b_i(x) & c_i(x) \\ d_i(x) & a_i(x) \end{pmatrix}\begin{pmatrix} R_i^3 \\ R_i^4 \end{pmatrix}$$

has a rank 2 and if it does not hold it has a rank 4. Summing them together, we get a matrix with rank bounded by $2g(x)$.

Now, for a pair of inputs x, x' compute $\widetilde{M} = M(x')^{-1}M(x)$ and denote $\widetilde{D} = D(x')^{-1}D(x)$. Then, we claim that a matrix N defined via

$$Q\widetilde{M} = \widetilde{D}Q + N$$

has a low rank. Indeed, multiply both sides by $PD(x')$ on the left (which does not change the rank because it is a multiplication by a full rank $2m+1 \times 2m$ matrix). We get:

$$PD(x')QM' = PD(x')\widetilde{D}Q + N'$$

where $N' = PD(x')N$ has the same rank as N. This yields

$$(M(x') + \mathsf{LowRank}_{\leq 2g(x')})M(x')^{-1}M(x) = M(x) + \mathsf{LowRank}_{\leq 2g(x)} + N'$$

$$\mathsf{LowRank}_{\leq 2g(x')} + \mathsf{LowRank}_{\leq 2g(x)} = N'$$

Typically, the rank of N will be $2g(x) + 2g(x')$, though there is a possibility it might be smaller if some additional relations hold on x and x' - but no smaller than $2\min(g(x), g(x'))$. Denote this rank bound $b-1$. As $g(x) \geq 3$, we can at least estimate $b \geq 7$.

As in [BIJ+20], the adversary can now set up a system of algebraic equations for the condition

$$\mathsf{rank}(Q\widetilde{M} - \widetilde{Q}) < b$$

, treating Q and $\widetilde{Q} = \widetilde{D}Q$ as formal variables. Denote $k = 2m$. In the following calculation we will use the symbol $\approx$ to denote the leading term of a polynomial.

We obtain $\approx (\frac{k^b}{b!})^2$ degree b equations stating that determinant of each $b \times b$ minor of the matrix $Q\widetilde{M} - \widetilde{Q}$ vanishes. Additionally, there are equations stating that $\widetilde{Q}$ is a product of Q with some diagonal matrix (i.e. it's columns are rescaled columns of Q). This is another $\approx \frac{k^3}{2}$ quadratic equations.

By itself, this amount of equations is not enough to launch a successful linearization attack - total amount of degree b monomials is $2k^2$ variables is $\approx \frac{2^b}{(2b)!}k^{2b}$.

Amount of linear equations that we obtain is $\approx (\frac{1}{(b!)^2} + \frac{1}{2(2b-3)!})k^{2b}$. For $b \geq 7$ this value is significantly smaller than the amount of monomials. So, a naive attempt of launching this attack fails.

Authors of [BIJ+20] suggest that Kipnis-Shamir [KS99] relinearization technique could, potentially, allow to launch the attack. While we were unable to do it, out of the abundance of caution we provide a trivial estimate that suggests that it's runtime would be far beyond security level that we aim for. Indeed, for instance sizes, even without any additional relinearization the size of the linear system is at least $O(k^{2b})$. As $b \geq 7$ and the runtime for solving a linear system is $\geq O(N^2)$, we have $\geq O(k^{28})$ runtime. Our concrete instance has the size $k > 128$, which makes the attack practically impossible.

As a theoretical mitigation, we suggest enforcing that a constraint system has superconstant minimal value of $g(x) > m^{\varepsilon}$. This can be achieved by repeating each gate m^{ε} times.

Overall, we consider evidence for the linearization attack inconclusive and believe that further research is required in this direction.

3.5 Security claims

Assumption 1. ***(Theoretical security claim)*** *for a projectively safe constraint system with $n > O(\lambda)$, $g(x) > O(m^{\varepsilon})$ the pair Algorithm 2, Algorithm 3 are extractable witness encryption scheme.*

Remark. Extractability is, of course, non-falsifiable. We prefer it to the weaker notion of soundness because the constraint systems that we compose our WE scheme with are SNARK verifiers that always have a valid (even if computationally infeasible to find) solutions.

Assumption 2. ***(Concrete security claim)*** *the circuits in the next chapter, combined with the instantiation of our WE scheme provide at least* 100 *bits of security.*

Remark. They could provide higher security level if not for the curve choice - BN254, for which we instantiate our circuits is known to have around 100 bits of security for the composition of our scheme and a SNARK.

4 Projectively safe gadgets

4.1 Notation

In order to build a set of gadgets we need to keep track of projective safety of the resulting constraint system.

Definition 5. *Recall that in the terminology of Definition 3 for (not necessarily projectively safe) constraint system we call a constraint (j_i) of form*

$$(j_i) : x_i^2 = a(x_0, \dots, x_{i-1}) \cdot b(x_0, \dots x_n)$$

*a **certificate** for variable x_i.*

*If a variable x_i has a **certificate** in the constraint system we say that this variable is **certified**.*

Note 1. A constraint system is projectively safe iff all of its variables are certified.

To meaningfully apply a gadget one needs to know an order of the variables used in it and which of these variables must be certified beforehand, which will be certified by the gadget and which will be left uncertified. To keep track of this information we will annotate each gadget with a list of all variables. Each variable in this list will be marked as

$\underline{\underline{x}}$: *required to be certified to correctly apply the gadget*
$\underline{x}$: *will be certified by the gadget*
x : *will not be certified by the gadget*

In our formulas for gadgets we will use the pseudocode notation resembling Circom language: we will use the symbol "=" to denote constraint, the symbol "$\leftarrow$" to denote the hint to the constraint system solver and the symbol $\Leftarrow$ to denote both at the same time. We will denote $\mathbb{1} = x_0$ to improve the readability of our formulas.

4.2 First examples

In this section, we will give some examples of projectively safe circuits - mainly, related to exponentiation. We hope this helps the reader to familiarize with our toolbox and so motivates the choice of our proof system later in the exposition.

The main downside of the projectively safe circuits is that it seems impossible to directly multiply elements of the field without additional logarithmic amount of certification checks. It is, however, possible to multiply them and compute a square root:

Gadget 1: Multiply and square root
Functionality:
Given variables a, b, output c that is (some) square root of ab.
Variable ordering:

$$\underline{\underline{a}}, \underline{\underline{b}} \prec \underline{c}$$

Builder:

$c^2 = a \cdot b$
$c \leftarrow \sqrt{ab}$

We will denote application of this gadget $c \Leftarrow \sqrt{ab}$.

Using square roots, we can compute the exponent in the odd-order subgroup of some extension field $\mathcal{K}/\mathbb{F}$ up to a cofactor twist:

Gadget 2: Odd-order subgroup exponent with fixed power
Functionality:
Given a variable $x \in \mathcal{K}$ and a constant power N:
Denote $\mathcal{K}^* = G_{\mathsf{odd}} \times \mu_{2^k}$ the decomposition of the multiplicative group of K into the odd-order subgroup and roots of unity of order 2^k.
Denote $\ell = \lfloor \log(N) \rfloor + 1$.
Denote $M = N/2^\ell \mod |G_{\mathsf{odd}}|$.
This gadget outputs a new variable y with the following properties:

> **Soundness:** y is constrained to be an exponent with power M up to the power-of-two root of unity multiplier: $y/x^M \in \mu_{2^\infty} \subset \overline{\mathbb{F}}$
> **Completeness:** y can be set to x^M if the original $x \in G_{\mathsf{odd}}$.

There is no guarantee that this constraint system if solvable if $x \notin G_{\mathsf{odd}}$.
Variable ordering:

$$\underline{\underline{x}} \prec \underline{y[1]} \prec \underline{y[2]} ... \prec \underline{y[\ell]} = \underline{y}$$

Builder:
Let $N = \sum\limits_{i=0}^{\ell-1} 2^i \cdot \mathsf{bits}[i]$ be a bit decomposition of N.

```
$y[0] \Leftarrow \mathbb{1}$
for $0 \le i \le \ell - 1$ do
    if $\mathsf{bits}[i] = 0$ then
        $y[i+1] \Leftarrow \sqrt{y[i] \cdot \mathbb{1}}$
    else
        $y[i+1] \Leftarrow \sqrt{x \cdot y[i]}$
    end if
end for
return $y \Leftarrow y[\ell]$
```

This constraint system indeed constrains a certain power of x up to the μ_{2^∞} multiplier. Explicitly computing the power yields

$$\sum_{i=0}^{\ell-1} (1/2)^{i+1} \mathsf{bits}[i] = (\sum_{i=0}^{\ell-1} 2^{\ell-i-1} \mathsf{bits}[i])/2^\ell = N/2^\ell = M \pmod{|G_{\mathsf{odd}}|}$$

Completeness for $x \in G_{\mathsf{odd}}$ follows from the fact that the computation of the square root in the odd-order group is always possible.

Gadget 3: Odd-order subgroup exponent with variable power
Functionality:
Notation and functionality is almost the same as functionality of the previous Gadget 2, with changes being:
N is now not a constant and decomposed into the sum $N = \sum_{i=0}^{\ell-1} 2^i \mathsf{bits}[i]$ of variables.
Final power M is shifted by another factor of two: $M = N/2^{\ell+1} \mod |G|_{\mathsf{odd}}$.
Variable ordering:

$$\underline{\underline{x}}, \underline{\underline{\mathsf{bits}[]}} \prec \underline{s[0]} \prec \underline{y[1]} \prec \underline{s[1]} \prec \underline{y[2]} ... \prec \underline{s[\ell-1]} \prec \underline{y[\ell]} = \underline{y}$$

Builder:

```
$y[0] \Leftarrow \mathbb{1}$
for $0 \le i \le \ell - 1$ do
    $s[i]^2 = \mathsf{bits}[i] \cdot x$
    $s[i] \leftarrow$ if $\mathsf{bits}[i]$ $\{\sqrt{x}\}$ else $\{0\}$
    $y[i+1]^2 = (\mathbb{1} - \mathsf{bits}[i] + s[i]) \cdot y[i]$
    $y[i+1] \leftarrow$ if $\mathsf{bits}[i]$ $\{(\sqrt{x} \cdot y[i])^{1/2}\}$ else $\{y[i]^{1/2}\}$
end for
return $y \Leftarrow y[\ell]$
```

4.3 Proof system

Design goals. Considering the restricted functionality of projectively safe circuits, the proof system ideally should optimize for the following properties:

– Pairing-based. It is allowed to compute few hashes, but the main work must be arithmetic over a large field.
– Pairing only with some fixed elements of $\mathcal{G}_2$. As we will see later, the computation of the Miller loop is possible in this case.
– Minimal amount of arithmetic outside of Miller loops and exponents. All other arithmetic seems to be extremely inefficient - to the point where instead of computing a scalar multiple of $\mathcal{G}_1$-point it is easier to map it into $\mathcal{G}_t$ first using pairing.

This set of objectives is very different from normal optimization targets, so the system we come up with is slightly custom. It is heavily inspired by Garuda [DMS24] in a sense that it combines KZG-commitments and circuit-specific trusted setup.

We will use the following notation:

– $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_t$ is a pairing-friendly curve of Type 3. In our estimates, we will assume it to be the curve BN254.
– Pairing is denoted $\langle \cdot, \cdot \rangle : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_t$
– Basis elements are denoted $[1]_1, [1]_2, [1]_t = \langle [1]_1, [1]_2 \rangle$ respectively and $[x]_\star$ means $x \cdot [1]_\star$.
– Base field of $\mathcal{G}_1$ is our WE field $\mathbb{F}$.
– Embedding extension field is $\mathcal{K} = \mathbb{F}^d$.
– Group has a prime order r, scalar field is $\mathbb{F}_r$.

Quadratic Arithmetic Programs. We will use slightly modified versions of Quadratic Arithmetic Programs (QAPs). We quickly recall this notion.

Let's fix the following parameters:

n - solution size
$\mathsf{pub} < n$ number of public parameters
m - number of constraints
$\mathcal{Z} \subset \mathbb{F}_\mathtt{r}$ - constraint set, subject to $|\mathcal{Z}| = m$

We will denote elements of this set $\{z_0, \dots z_{m-1}\} = \mathcal{Z}$ and set $Z(x) = \prod_{i=0}^{m-1} (x - z_i)$

Definition 6. *QAP instance with these parameters is given by a set of triples of polynomials $(A^i(x), B^i(x), C^i(x))$ for $0 \le i < n$ of degree $< m$ and a public input vector $x = (x_0, \dots x_{\mathsf{pub}-1})$.*

QAP witness is given by a pair of vectors (w, h) where $w = (w_0, \dots, w_{n-1})$, $h = (h_0, \dots h_{m-2})$ such that $w_i = x_i$ for $i < \mathsf{pub}$ and

$$A(x)B(x) - C(x) = Z(x)H(x)$$

where

$$A(x) = \sum_{i=0}^{n-1} w_i A^i(x) \qquad B(x) = \sum_{i=0}^{n-1} w_i B^i(x)$$

$$C(x) = \sum_{i=0}^{n-1} w_i C^i(x) \qquad H(x) = \sum_{i=0}^{m-2} h_i x^i$$

Consider some rank one constraint system

$$(j) : \left(\sum_{i=0}^{n} a_j^i w_i\right) \cdot \left(\sum_{i=0}^{n} b_j^i w_i\right) = \left(\sum_{i=0}^{n} c_j^i w_i\right)$$

with public input condition $w_i = x_i$ for $i < \mathsf{pub}$.

It is a standard fact that this system of constraints is equivalent to the QAP defined via interpolation:

$$A^i(z_j) = a_j^i \qquad B^i(z_j) = b_j^i \qquad C^i(z_j) = c_j^i$$

Indeed, for a vector w a polynomial $A(x)B(x) - C(x)$ is divisible by $Z(x)$ if and only if the original R1CS constraint system was satisfied.

We will modify the standard QAP notion in few aspects. First of all, we will set $A = B$. This version of QAPs is sometimes called "squaring programs" as it is only capable of verifying R1CS constraints of the form $a^2 = c$. As multiplication can be expressed through addition and squaring, this does not really harm expressiveness of the underlying SNARK that much, and decreases the verification costs.

Another change is the way that we inject public inputs - instead of requiring that some subset of w is bound to the public input, we will directly inject them to the right-hand side in a coefficient form. In what follows, we restrict ourselves to a single public input denoted u.

Definition 7. *Modified squaring QAP is the same as the normal QAP with the following changes: $B^i(x) = A^i(x)$, there is no direct condition on public inputs and instead the final equation is*

$$A^2(x) - C(x) = Z(x)H(x) + u$$

This naturally corresponds to the following modified squaring-R1CS notion:

Definition 8. *Modified squaring R1CS is a system of equations of the form*

$$(j) : (\sum_{i=0}^{n-1} a_j^i w_i)^2 - (\sum_{i=0}^{n-1} c_j^i w_i) = u$$

Modified squaring R1CS is equivalent to modified squaring QAP in exactly the same way regular R1CS is equivalent to a regular QAP. In order to simulate regular squaring R1CS (with coefficients a_j^i, b_j^i, c_j^i) via modified squaring R1CS, we can do:

$$(0)' : w_0 = u$$

$$(j+1)' : (\sum_{i=0}^{n-1} a_j^i w_i)^2 - (\sum_{i=0}^{n-1} c_j^i w_i) + w_0 = u$$

It is also possible to instead use a far more aggressive option and to restrict R1CS as $a^i_j = b^i_j = c^i_j$. This does not seem to give any verification-efficiency gains due to how we embed our public inputs.

Remark 1. Normally we desire to have at least **two** public inputs - an actual public input and the value 1. However, as long as we verify $u = 1 \pmod 2$ outside of the circuit, the value 1 can be recovered in the circuit by bit-splitting u:

$$u = \sum_i 2^i b_i;\ \text{now } b_0 \text{ can be used as } 1$$

While this is definitely non-standard, our verification budget is so tight that this optimization is justified.

Informal overview
Given a collection of polynomials $P_1, \ldots, P_k$ we can encode them in the following vector-KZG commitment:

$$\mathcal{P} = \mathsf{VecKZGCommit}(P_1, \ldots, P_k) = [\alpha_1 P_1(\tau) + \ldots + \alpha_k P_k(\tau)]$$

The vector opening argument $P_i(z) = v_i$ is similar to a standard version:

$$\langle \mathcal{P}, [1]_2 \rangle = \langle \mathcal{Q}, [\tau]_2 - z \cdot [1]_2 \rangle + \sum v_i [\alpha_i]_t$$

The polynomials that we plan to package in this way are A, C, H. In the same way as in Garuda, CRS elements of $\mathcal{G}_1$ will only contain elements of the form $\mathcal{W}^i = [\gamma(\alpha_A A^i(\tau) + \alpha_C C^i(\tau))]_1, \mathcal{H}^i = [\gamma \alpha_H \tau^i]_1$ twisted by an additional factor γ. This will enforce that the prover can only commit to a correct QAP witness. The pairing will take the form $\langle P, [\gamma^{-1}]_2 \rangle$ (to compensate for the twist), and the quotient commitment $\mathcal{Q}$ will be expressed in terms of elements of the form $[\alpha_U \tau^i]_1$.

In the QAP verification equation we will also need to compute $Z(z)$. This is a significant cost in terms of non-native arithmetic: even for $Z(x)$ of the standard form $x^{2^k} - 1$ it is still a logarithmic amount of non-native multiplications. Instead, opening of Z will also be added into the verification equation (in the quotient term).

The final pairing check that we obtain will have the following form:

$$\langle \mathcal{P}, [\gamma^{-1}]_2 \rangle + [\alpha_Z Z(\tau)]_t = \langle \mathcal{Q}, [\tau]_2 - z[1]_2 \rangle + \sum v_U [\alpha_U]_t$$

where $U \in \{A, C, H, Z\}$.

Considering that we are unable to perform neither $\mathcal{G}_1$ nor $\mathcal{G}_2$ operations, we also open the brackets and get:

$$\langle \mathcal{P}, [\gamma^{-1}]_2 \rangle + [\alpha_Z Z(\tau)]_t = \langle \mathcal{Q}, [\tau]_2 \rangle - z \cdot \langle \mathcal{Q}, [1]_2 \rangle + \sum v_U [\alpha_U]_t$$

Formal description

The setup algorithm takes modified squaring QAP and outputs a CRS.

Algorithm 4 Algorithm $\mathsf{SNARK.Setup}$

```
$\tau \xleftarrow{\$} \mathbb{F}_r$
$\gamma \xleftarrow{\$} \mathbb{F}_r$
$\alpha_U \xleftarrow{\$} \mathbb{F}_r$ for $U \in \{A, C, H, Z\}$
$\mathsf{CRS} \leftarrow ()$
for $0 \leq i < n$ do
    $\mathcal{W}^i \leftarrow [\gamma(\alpha_A A^i(\tau) + \alpha_C C^i(\tau))]_1$
    $\mathsf{CRS} \leftarrow \mathsf{CRS}||(\mathcal{W}^i)$
end for
for $0 \leq i < m - 1$ do
    $\mathcal{H}^i \leftarrow [\gamma\alpha_H\tau^i]_1$
    $\mathsf{CRS} \leftarrow \mathsf{CRS}||(\mathcal{H}^i)$
end for
for $U \in \{A, C, H, Z\}$ do
    if $U \in \{A, C\}$ do $k \leftarrow m - 1$
    if $U \in \{H\}$ do $k \leftarrow m - 2$
    if $U \in \{Z\}$ do $k \leftarrow m$
    for $0 \leq i < k$ do
        $\mathcal{T}_U^i \leftarrow [\alpha_U\tau^i]_1$
        $\mathsf{CRS} \leftarrow \mathsf{CRS}||(\mathcal{T}_U^i)$
    end for
end for
$\mathsf{CRS} \leftarrow \mathsf{CRS}||([1]_2, [\tau]_2, [\gamma^{-1}]_2)$
$\mathsf{CRS} \leftarrow \mathsf{CRS}||([\alpha_Z Z(\tau)]_t, [\alpha_U]_t$ for $U \in \{A, C, H, Z\})$
return $\mathsf{CRS}$
```

Algorithm 5 Algorithm $\mathsf{SNARK.Prove}$

```
Input: Solution $(w, h)$ to a modified squaring QAP instance with a public input $u$.
$\mathcal{P} \leftarrow \sum w_i\mathcal{W}^i + \sum h_i\mathcal{H}^i$
$z \leftarrow \mathsf{Hash}(\mathsf{pp}, \mathcal{P}, u) \in \mathbb{F}_r$
$v_U \leftarrow U(z)$ for $U \in \{A, C, H, Z\}$
$Q_U(x) \leftarrow (U(x) - U(z))/(x - z)$ for $U \in \{A, C, H, Z\}$
$\mathcal{Q} \leftarrow \sum(Q_U)_{(i)}\mathcal{T}_U^i$
return $\mathsf{Proof} = (u, \mathcal{P}, \mathcal{Q}, v_U$ for $U \in \{A, C, H, Z\})$
```

Algorithm 6 Algorithm SNARK.Verify

```
**Input:** Proof $= (u, \mathcal{P}, \mathcal{Q}, v_U$ for $U \in \{A, C, H, Z\})$
**Require:** $v_A^2 - v_C = v_H \cdot v_Z + u$
**Require** $\langle \mathcal{P}, [\gamma^{-1}]_2 \rangle + [\alpha_Z Z(\tau)]_t = \langle \mathcal{Q}, [\tau]_2 \rangle - z \cdot \langle \mathcal{Q}, [1]_2 \rangle + \sum_U v_U [\alpha_U]_t$
```

Lemma 7. *This SNARK is sound in the generic group model (and ROM for Fiat-Shamir transform)*

Proof. $\mathcal{P}$ only contains the terms from CRS (defined in Algorithm 4) divisible by γ as there are no other product terms potentially containing γ^{-1}. So wlog $\mathcal{P} = \sum w_i \mathcal{W}^i + \sum h_i \mathcal{H}^i$. Similarly, $\mathcal{Q}$ can only contain terms from $\mathcal{G}_1$ that correspond to monomials not divisible by γ in the setup, so it has the form $\mathcal{Q} = \sum (q_U)_i \mathcal{T}_U^i$. Splitting $\mathcal{Q}$ into separate terms by U, we get four independent KZG opening arguments in separate independent bases, so they must all hold independently. □

Embedding strategy The rest of this chapter is devoted to the embedding of this proof system (specifically the verifier Algorithm 6) into our AADP-based witness encryption scheme. Roughly, we will need the following gadgets:

- Non-native arithmetic in the field $\mathbb{F}_r$.
- An arithmetic hash (we will use the most common and well-established option, namely, Poseidon).
- Extension field constructor. Our circuits will have coefficients over $\mathbb{F}$ and we will construct the basis of the extension $\mathcal{K} = \mathbb{F}^d$ inside of the circuit.
- Miller loop with a fixed $\mathcal{G}_2$ point.
- Final exponent.

All of this needs to be done on an extremely tight constraint budget, as our main witness encryption protocol scales as $O(N^3)$.

Final exponent is made possible by the method in the Section 4.2. We proceed with the rest of the construction.

4.4 Finding the extension basis

We have a projectively safe constraint system over $\mathbb{F}$, and want to use the elements of the extension $\mathcal{K} = \mathbb{F}^d$ in it (recall that solutions to the system are allowed to have values in $\overline{\mathbb{F}}$).

Let $E(x)$ be a degree d monic polynomial defining the extension: $\mathcal{K} \simeq \mathbb{F}[x]/(E(x))$.

A straightforward idea is to construct a (projectively safe) constraint system that validates, for a certain element s that $E(s) = 0$ and then to use the powers of s as the basis of $\mathcal{K}$.

Gadget 4: FindExtensionBasis
Functionality:
Constrain and output variables $s_1, ...s_{d-1}$ satisfying $\forall i : s_i = s^i$, $E(s) = 0$.
Variable ordering:
No ordering dependency. All variables $\underline{s_1}, \ldots, \underline{s_{d-1}}$ are certified.
Builder:

```
$s_0 \Leftarrow \mathbb{1}$
$s_1 \leftarrow$ any root of $E(x)$ in $\overline{\mathbb{F}}$
for $2 \leq i < d$ do
    $s_i \leftarrow s_1 \cdot s_{i-1}$
    $\mathbb{1} \cdot s_i = s_1 \cdot s_{i-1}$
    $\mathbb{1} \cdot (-\sum_{i=0}^{d-1} E_{(i)} s_i) = s_1 \cdot s_{d-1}$
end for
▷ By this point, all $s_i$-s are constrained correctly, but nothing is certified ◁
for $0 \leq i < d$ do
    $R(x) \leftarrow x^{2i} \mod E(x)$
    $s_i^2 = \mathbb{1} \cdot \sum_{i=0}^{d-1} R_{(i)} s_i$
    ▷ These constrains hold automatically given the previous ones and certify
      individual $s_i$-s ◁
end for
return $(s_1, \ldots s_{d-1})$
```

4.5 Certification toolbox

Now we will discuss some tricks that allow for variable certification.

Gadget 5: Certification by binary decomposition
Notes:
Let $\ell = \lfloor \log q \rfloor$
Functionality:
Certify variable a
Variable ordering:
There is no ordering on variables $\underline{b_0}, \ldots, \underline{b_\ell}$
Builder:

```
$b[0 \ldots \ell] \leftarrow \mathsf{bits}(a)$
for $0 \leq i \leq \ell$ do
    $b_i^2 = \mathbb{1} \cdot b_i$
end for
```

Notes:
One can see that variable a does not appear in the constraints and variable

ordering. Instead, we alias variable a to linear combination $\sum_{i=0}^{\ell} 2^i b_i$. So, formally, there is no such variable as a. When any gadget's ordering would include variable a we will replace it with all variables b_i.

An alternative way to certify uses the exponentiation. This version of exponentiation is different from Gadget 2: it doesn't need to work in the odd subgroup, but it's certification properties are reversed: it takes uncertified input, certifies it, but leaves the output uncertified. Another difference is that this gadget is defunct for the zero input and is allowed to output anything.

Gadget 6: Alternative exponent with fixed power
Functionality:
Given variable x and a constant power $t > 2$ output x^t if $x \neq 0$. For zero input it is allowed to output anything.
Variable ordering:

$$\underline{x} \prec \underline{y[0]} \prec \ldots \prec \underline{y[N-1]} \prec y[N]$$

Builder:
First we construct an array bits$[0\ldots(N-1)]$ such that $t = 1 + 2^N + \sum_{i=0}^{N-1} \text{bits}[i] 2^i$

```
$x^2 = 1 \cdot y[0]$
for $0 \le i < N-1$ do
    if bits$[N-1-i] = 1$ then
        $y[i]^2 = \mathbb{1} \cdot y[i+1]$
        $y[i+1] \leftarrow y[i]^2$
    else
        $y[i]^2 = x \cdot y[i+1]$
        $y[i+1] \leftarrow \frac{y[i]^2}{x}$
    end if
end for
```

Notes:
This resembles normal double-and-add method, but instead of multiplying by x we divide by it. This also leads to defunct behavior in the case $x = 0$.

Gadget 7: Certification by exponent
Functionality:
Given variable x prove that it is in the base field $\mathbb{F}$ (and certify it).
Variable ordering:
$\underline{x} \prec \underline{x_1}$
Builder:

```
Use Gadget 6 to constrain $x_1 \Leftarrow x^{\frac{q-1}{2}}$
$x_1^2 = \mathbb{1} \cdot \mathbb{1}$
```

Notes:
Interestingly enough, for $x = 0$ this constraint system is still solvable because the gadget Gadget 6 is defunct: we can set all $y[i]$-s to 0 up to the last division and to $\mathbb{1}$ afterwards and the constraints will all hold.

4.6 Pairing computation

We proceed with the pairing computation. We will use the optimal Ate pairing algorithm on BN curves. BN curves family is parametrized by a parameter u satisfying $r = 36u^4 + 36u^3 + 18u^2 + 6u + 1$. The extension degree is $d = 12$ and our base field $\mathbb{F} = \mathbb{F}_q$. In the algorithm below, the notation $\ell_{A,B}$ means the equation of the line passing through the points A, B (normalized as $y + \star x + \star$).

Algorithm 7 Miller loop on BN curves[AKL+10]

```
Input: $P \in \mathcal{G}_1, Q \in \mathcal{G}_2, m = |6u+2| = \sum_{i=0}^{\lfloor \log(m) \rfloor} m_i 2^i$
Output: $\langle P, Q \rangle \in \mathcal{G}_t$
▷ Miller loop:                                                    ◁
$T \leftarrow Q, f \leftarrow 1$
for $0 \le j < \lfloor \log(m) \rfloor$ do
    $i \leftarrow \lfloor \log(m) \rfloor - j - 1$
    $f \leftarrow f^2 \cdot \ell_{T,T}(P), T \leftarrow 2T$
    if $m_i = 1$ then
        $f \leftarrow f \cdot \ell_{T,Q}(P), T \leftarrow T + Q$
    end if
end for
$Q_1 \leftarrow \mathsf{Frob}(Q)$
$Q_2 \leftarrow \mathsf{Frob}^2(Q)$
if $u < 0$ then
    $T \leftarrow -T$
    $f \leftarrow f^{-1}$
end if
$f \leftarrow f \cdot \ell_{T,Q_1}(X), T \leftarrow T + Q_1$
$f \leftarrow f \cdot \ell_{T,-Q_2}(X), T \leftarrow T - Q_2$
▷ Final exponent:                                                 ◁
$f \leftarrow f^{(q^{12}-1)/r}$
```

For simplicity, in what follows we will assume $u > 0$ (this holds for our target curve BN254).

For a fixed point Q, it is possible to precompute all the values of T for all the iterations of the loop. Moreover, the resulting value of f is a product of 2^i-powers of some collection of linear inhomogeneous polynomials in variables $(x, y) = P$ with coefficients in $\mathbb{F}_q^{12}$.

We will precompute and certify all the values of the form xs_i, ys_i for $i < 12$ and then will use a method similar to Gadget 2 to compute the Miller loop.

Gadget 8: Multiplication terms of the Miller loop
Functionality:
Given a point $P = (x, y)$, checks that it is on a curve $y^2 = x^3 + b$ and outputs and certifies $u_i = xs_i, v_i = ys_i$ for all $i < 12$.

Variable ordering:
Ordering is not important, but all outputs $\underline{u_i}, \underline{v_i}$ are certified.

Builder:

```
Apply Gadget 7 to $x$ and $y$. Denote $x_2$, $y_2$ squares of $x, y$ and that these gadgets generate.
▷ *This will validate that $x$ and $y$ are in the base field and also will give us some powers of $x, y$ that we will reuse later* ◁
$\mathbb{1} \cdot (y_2 - b) = x \cdot x_2$
▷ *Validate that the point $(x, y)$ is on a curve. It might be possible to optimize this slightly by using this equation to certify $y$ instead, but this would fail to check that it is in the base field.* ◁
**for** $1 \leq i < 12$ **do**
    $u_i \leftarrow xs_i$
    $v_i \leftarrow ys_i$
    $u_i \cdot \mathbb{1} = x \cdots s_i$
    $v_i \cdot \mathbb{1} = y \cdots s_i$
    ▷ *These constraints are not certifying yet* ◁
    $R(x) \leftarrow x^{2i} \mod E(x)$
    $u_i^2 = x_2 \cdot \sum_{i=0}^{11} R_{(i)} s_i$
    $v_i^2 = y_2 \cdot \sum_{i=0}^{11} R_{(i)} s_i$
    ▷ *Certify using the same trick as for the extension elements themselves* ◁
**end for**
**return** $v_i, u_i$ for $i < 12$
```

In the Miller loop gadget, we will use the following notation: for a precomputed line function $\ell(x, y)$ we denote $\ell_{i,j} \in \mathbb{F}_p$ its coefficients

$$\ell(x, y) = \sum \ell_{0,j} u_j + \sum \ell_{1,j} v_j$$

Algorithm 8 Precomputation for the reverse Miller loop gadget

```
**Input:** Point $Q \in \mathcal{G}_2$
**Output:** Coefficients of all $\ell_i, \ell'_i, \ell_A, \ell_B$.
$T \leftarrow Q, f \leftarrow 1$
for $0 \leq j < \lfloor \log(m) \rfloor$ do
    $i \leftarrow \lfloor \log(m) \rfloor - j - 1$
    $\ell_i \leftarrow \ell_{T,T}, T \leftarrow 2T$
    if $m_i = 1$ then
        $\ell'_i \leftarrow \ell_{T,Q}, T \leftarrow T + Q$
    end if
end for
$Q_1 \leftarrow \mathsf{Frob}(Q)$
$Q_2 \leftarrow \mathsf{Frob}^2(Q)$
$\ell_L \leftarrow \ell_{T,Q_1}, T \leftarrow T + Q_1$
$\ell_R \leftarrow \ell_{T,-Q_2}, T \leftarrow T - Q_2$
```

We also need a gadget that outputs the twists from μ_{2^k}-cofactor to move everything into the order subgroup. In our target curve BN254, $k = 5$.

Gadget 9: Twist check
Functionality:
Checks that $x \in \mu_{2^k}$.
Variable ordering:
Not important, and everything is certified.
Builder:

```
$x_1 \Leftarrow x$
for $1 \leq i < k - 1$ do
    $x_{i+1} \leftarrow x_i^2$
    $x_{i+1} \cdot \mathbb{1} = x_i^2$
end for
$x_{k-1}^2 = \mathbb{1} \cdot \mathbb{1}$
```

We will denote finding a cofactor twist and applying this gadget $\Leftarrow \mu_{2^k}$. The suggested solver value of the twist is inferred from the context (it is instantly multiplied by a value in order to put the result in an odd-order subgroup).

Gadget 10: $\mathsf{MillerLoop}$ **- reverse Miller loop**
Functionality:
Given the output data of Gadget 8 corresponding to an in input point $P \in \mathcal{G}_1$, computes the value of the Miller loop up to the μ_{2^k}-cofactor and the static shift $2^{-(\lfloor \log m \rfloor + 2)}$ in the exponent.
Variable ordering:
This gadget only uses output-certifying gadgets as the components, so it has the same behavior: inputs must be certified, outputs are certified automatically.
Builder:

```
$a_L \Leftarrow \sum_{i=0}^{11}((\ell_L)_{0,i}u_i + (\ell_L)_{1,i}v_i)$
$t_L \Leftarrow \mu_{2^k}$
$b_L \Leftarrow \sqrt{a_L t_L}$
$a_R \Leftarrow \sum_{i=0}^{11}((\ell_R)_{0,i}u_i + (\ell_R)_{1,i}v_i)$
$t_R \Leftarrow \mu_{2^k}$
$b_R \Leftarrow \sqrt{a_R t_R}$
$f[0] \Leftarrow \sqrt{b_L b_R}$
▷ This value contains the contributions from two last exceptional multiplications taken to the power $2^{-2}$. ◁
for $0 \leq j < \lfloor \log(m) \rfloor$ do
    $a_1[j] \Leftarrow \sum_{i=0}^{11}((\ell_i)_{0,i}u_i + (\ell_i)_{1,i}v_i)$
    $t_1[j] \Leftarrow \mu_{2^k}$
    $b_1[j] \Leftarrow \sqrt{a_1[j]t_1[j]}$
    if $m_i = 1$ then
        $a_2[j] \Leftarrow \sum_{i=0}^{11}((\ell'_i)_{0,i}u_i + (\ell'_i)_{1,i}v_i)$
        $t_2[j] \Leftarrow \mu_{2^k}$
        $b_2[j] \Leftarrow \sqrt{a_2[j]t_2[j]}$
    else
        $b_2[j] \Leftarrow \mathbb{1}$
    end if
    $s[j] \Leftarrow \sqrt{b_1[j]b_2[j]}$
    ▷ This value contains contribution of $j$-th iteration of the Miller loop, with power $2^{-2}$ ◁
    $f[j+1] \Leftarrow \sqrt{f[j] \cdot s[j]}$
end for
$f \Leftarrow f[\lfloor \log(m) \rfloor]$
return $f$
```

Theorem 2. *Given input points $\mathcal{P}, \mathcal{Q} \in \mathcal{G}_1 \subset \mathbb{F} \times \mathbb{F}$ and values $p_0, p_1, p_2, p_3, v_Z, z^4 \in \mathbb{F}_r$ represented as sequences of bits, the pairing check*

$$\langle \mathcal{P}, [\gamma^{-1}]_2 \rangle + [\alpha_Z Z(\tau)]_t = \langle \mathcal{Q}, [\tau]_2 \rangle - z \cdot \langle \mathcal{Q}, [1]_2 \rangle + \sum_U v_U [\alpha_U]_t$$

can be performed by a projectively safe constraint system over $\mathbb{F}$ using the gadgets defined above.

Proof. Slightly rearrange terms:

$$\langle \mathcal{P}, [\gamma^{-1}]_2 \rangle + \langle \mathcal{Q}, [-\tau]_2 \rangle + z \cdot \langle \mathcal{Q}, [1]_2 \rangle + \sum_U v_U [\alpha_U]_t = [-\alpha_Z Z(\tau)]_t$$

Gadget 11: $\mathsf{SNARK.VerifyPairingEq}$
Builder:

```
$\ldots s_i \ldots \Leftarrow \mathsf{FindExtensionBasis}()$
$\triangleright$ Gadget 4. $\triangleleft$
For both $\mathcal{P}$ and $\mathcal{Q}$, call the Gadget 8.
$\triangleright$ Call Gadget 10 thrice (with different precomputed constants) to compute
  Miller loop results up to $\mu_{2^k}$-cofactor: $\triangleleft$
$S_1 \Leftarrow \mathsf{MillerLoop}(\mathcal{P}, [\gamma^{-1}]_2)$
$S_2 \Leftarrow \mathsf{MillerLoop}(\mathcal{Q}, [-\tau^4]_2)$
$\triangleright$ Pairing $\langle \mathcal{Q}, [1]_2 \rangle$ is shifted to compensate for exp application $\triangleleft$
$\triangleright$ Value of $z$ can be truncated up to $\lambda = 128$ bits, denote $B_z = \lambda + 2$ $\triangleleft$
$S_3' \Leftarrow \mathsf{MillerLoop}(\mathcal{Q}, [2^{B_z+2}]_2)$
$S_3 \Leftarrow \mathsf{OddGroupVarpowExp}(S_3', z)$
$S_{\mathsf{miller}} \Leftarrow \sqrt{\sqrt{S_1 \cdot S_2} \cdot \sqrt{S_3 \cdot \mathbb{1}}}$
$\triangleright$ Apply Gadget 2 to compute final exponentiation: $\triangleleft$
$S_{\mathsf{pair}} \Leftarrow \mathsf{OddGroupFixpowExp}\big(S_{\mathsf{miller}}, \big(q^{12}-1\big)/r\big).$
$\triangleright$ This result is shifted by $2^{-\sigma}$ where $\sigma = (\lfloor \log m \rfloor + 2) + 2 +$
  $\Big(\lfloor \log(\frac{q^{12}-1}{r}) \rfloor + 1\Big)$ $\triangleleft$
Get constants $[-2^{\kappa}\alpha_U]_t$ for $U \in \{A, C, H, Z\}$ where $\kappa = -\sigma + \lfloor \log r \rfloor + 3$.
$S_U \Leftarrow \mathsf{OddGroupVarpowExp}([-2^{-\kappa}\alpha_U], p_i)$ for $U \in \{A, C, H, Z\}$
$S_{\mathsf{vals}} \Leftarrow \sqrt{\sqrt{S_A S_C} \cdot \sqrt{S_H S_Z}}$
$S_{\mathsf{lhs}} \Leftarrow \sqrt{S_{\mathsf{pair}} S_{\mathsf{vals}}}$
$S_{\mathsf{rhs}} \Leftarrow [-2^{-\sigma-1}\alpha_Z Z(\tau)]_t$
$S_{\mathsf{lhs}} \cdot \mathbb{1} = S_{\mathsf{rhs}} \cdot \mathbb{1}$
```

□

4.7 Poseidon hash

Poseidon hash can be instantiated with $x \mapsto x^5$ S-box for the base field of BN254.

Gadget 12: Poseidon S-box
Functionality:
Given a variable x construct a variable $x_5 = x^5$
Variable ordering:
$\underline{x} \prec x_5$
Builder:

```
$x^2 = \mathbb{1} \cdot x_2$
$x_2 \leftarrow x^2$
$\mathbb{1} \cdot x_3 = x_2 \cdot x$
$x_3 \leftarrow x_2 \cdot x$
$\mathbb{1} \cdot x_5 = x_2 \cdot x_3$
$x_5 \leftarrow x_2 \cdot x_3$
```

```
▷ At this point, everything is constrained but nothing is certified. ◁
$x_2^2 = x \cdot x_3$
$x_3^2 = x \cdot x_5$
▷ And certify $x_2, x_3$ ◁
```

Gadget 13: Poseidon permutation
Functionality:
Given a state $(t_0, \ldots, t_n)$ construct $(t'_0, \ldots, t'_n) = \mathsf{PoseidonPermutation}(t_0, \ldots, t_n)$.

Variable ordering:
$\underline{t_i} \prec t'_i$
Builder:
Apply *Gadget* 12 and linear layers according to the Poseidon hash algorithm. Each S-box gadget certifies it's inputs and linear operations are free, so in effect it certifies everything but the outputs.

One important thing is that the outputs of this Poseidon permutation gadget are left uncertified, so if they are dropped they need to be certified separately.

This could be circumvented by using a different kind of nonlinear S-box that certifies its output. For example, we can use $x \mapsto \sigma(\sigma x)^{1/2}$ where $\sigma = x^{(p-1)/2}$ and the root is uniquely chosen to be a square. This kind of arithmetic hash would require a separate cryptanalysis.

4.8 Non-native arithmetic

The purpose of this section is to perform the check $v_A^2 - v_C = v_H v_Z + u$ of Algorithm 6 SNARK.Verify. It is best done in the exponent. Denote $B_r = \lfloor \log(r) \rfloor + 2$

Gadget 14: SNARK.VerifyArith
Builder:

```
Input: $v_A, v_C, v_Z, v_H, u$ (all bit sequences representing elements of $\mathbb{F}_r$)
$[2^{B_r+1} v_A]_t \Leftarrow \mathsf{OddGroupVarpowExp}([2^{2B_r+1}]_t, v_A)$
$[2v_A^2]_t \Leftarrow \mathsf{OddGroupVarpowExp}([2^{B_r+1} v_A]_t, v_A)$
$[-2v_C]_t \Leftarrow \mathsf{OddGroupVarpowExp}([-2^{B_r+1}]_t, v_C)$
▷ This is a normal sqrt gate, just in the additive notation: ◁
$[v_A^2 - v_C]_t \Leftarrow \frac{1}{2}\left([2v_A^2]_t + [-2v_C]_t\right)$
$[2^{B_r+1} v_Z]_t \Leftarrow \mathsf{OddGroupVarpowExp}([2^{2B_r+1}]_t, v_Z)$
$[2v_H v_Z]_t \Leftarrow \mathsf{OddGroupVarpowExp}([2^{B_r+1} v_Z]_t, v_H)$
$[2u]_t \Leftarrow \mathsf{OddGroupVarpowExp}([-2^{B_r+1}]_t, u)$
$[v_H v_Z + u]_t \Leftarrow \frac{1}{2}\left([2v_A^2]_t + [2u]_t\right)$
$[v_A^2 - v_C]_t = [v_H v_Z + u]_t$
```

4.9 Final gadget

Gadget 15: SNARK.VerifyFull

Builder:

```
**Input:** Proof $= (u, \mathcal{P}, \mathcal{Q}, v_U \text{ for } U \in \{A, C, H, Z\})$
$z \Leftarrow \mathsf{Hash}(\mathsf{pp}, \mathcal{P}, u)[0..\lambda]$
**Bitcheck:** $u$, $v_U$-s, all outputs of the Poseidon permutation.
$\dots s_i \dots \Leftarrow \mathsf{FindExtensionBasis}()$
$\mathsf{SNARK.VerifyArith}(u, v_U)$
$\mathsf{SNARK.VerifyPairingEq}(\mathcal{P}, \mathcal{Q}, v_U)$
```

4.10 Cost estimates

A python script (approximately) modeling the gadgets above gives the following results (with v denoting variable count and g denoting gate count):

Subroutine	Variables	Gates
Extension	11	22
Glue code	1265	1270
Hash	1017	1192
Arithmetic	3578	3580
Pairing equation	8212	8307
Totals	14083	14371

Substituting the ciphertext size formula $v(2g+1)^2|\mathbb{F}|$, we get 338 TB size of the ciphertext.

These numbers should not be fully trusted - after all, we do not have a full implementation and only a modeling script which could very well miss some important subroutine completely. On the other hand, due to the cubic scaling of the ciphertext size even marginal improvements in the circuit size translate into very significant improvements in the ciphertext size.

Size of the ciphertext might possibly be decreased further - for example, by picking a better hash with a certification-friendly S-box or choosing a curve with smaller 2-adicity of $\mathbb{F}^d$.

Overall, we conclude that this approach to witness encryption, while very costly, is realistically implementable.

References

AKL$^+$10. Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio López. Faster explicit formulas for computing pairings over ordinary curves. Cryptology ePrint Archive, Paper 2010/526, 2010. 25

BIJ$^+$20. James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation and witness encryption. Cryptology ePrint Archive, Paper 2020/889, 2020. 1, 2, 6, 7, 8, 9, 11, 12, 14

DMS24. Michel Dellepere, Pratyush Mishra, and Alireza Shirzad. Garuda and pari: Faster and smaller SNARKs via equifficient polynomial commitments. Cryptology ePrint Archive, Paper 2024/1245, 2024. 2, 18

GGSW13. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. Cryptology ePrint Archive, Paper 2013/258, 2013. 1, 7

KS99. Aviad Kipnis and Adi Shamir. Cryptanalysis of the hfe public key cryptosystem by relinearization. In *Advances in Cryptology — CRYPTO' 99*, pages 19–30, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. 14