Pinocchio-Based Adaptive zk-SNARKs and Secure/Correct Adaptive Function Evaluation

Meilof Veeningen, meilof.veeningen@philips.com

Philips Research, Eindhoven, The Netherlands

Abstract. Pinocchio is a practical zk-SNARK that allows a prover to perform cryptographically verifiable computations with verification effort sometimes less than performing the computation itself. A recent proposal showed how to make Pinocchio adaptive (or "hash-and-prove"), i.e., to enable proofs with respect to computation-independent commitments. This enables computations to be chosen after the commitments have been produced, and for data to be shared in different computations in a flexible way. Unfortunately, this proposal is not zero-knowledge. In particular, it cannot be combined with Trinocchio, a system in which Pinocchio is outsourced to three workers that do not learn the inputs thanks to multi-party computation (MPC). In this paper, we show how to make Pinocchio adaptive in a zero-knowledge way; apply it to make Trinocchio work on computation-independent commitments; present tooling to easily program fleible verifiable computations (with or without MPC); and use it to build a prototype in a medical research case study.

1 Introduction

Recent advances in SNARKs (Succinct Arguments of Knowledge) are making it more and more feasible to outsource computations to the cloud while obtaining cryptographic guarantees about the correctness of their outputs. In particular, the Pinocchio system [GGPR13,PHGR13] managed to achieve for the first time for a practical computation a verification time of a computation proof that was actualy faster than performing the computation itself.

In Pinocchio, proofs are verified with respect to plaintext inputs and outputs of the verifier; but in many cases, it is useful to have computation proofs that also refer to committed data, e.g., provided by a third party. Ideally, such proofs should be *adaptive*, i.e., the multiple different computations can be performed on the same commitment, that are chosen after the data has been committed to; and *zero-knowledge*, i.e., the commitments and proofs should reveal no information about the committed data. This latter property allows proofs on sensitive data, and it allows extensions like Trinocchio [SVdV15] that additionally hide this sensitive data from provers by multi-party computation.

Although several approaches are known from the literature, no really satisfactory practical adaptive zk-SNARK exists. The recent "hash first" proposal [FFG+16] shows how to make Pinocchio adaptive at low overhead, but is unfortunately not zero-knowledge. On the other hand, Pinocchio's successor Geppetto [CFH+15] is zero-knowledge but not adaptive: multiple computations can

be performed on the same data but they need to be known before committing. The asymptotically best known SNARKS combining the two properties have $\Theta(n \log n)$ non-cryptographic and $\Theta(n)$ cryptographic work for the prover, a $\Theta(n)$-sized CRS, and constant-time verification (where n is the size of the computation), but with a large practical overhead: [Lip16] because it relies on the impractical subset-sum language; other constructions (e.g., [CFH+15,FFG+16]) because they rely on including hash evaluation in the computation[1]. Finally, [BBFR15] enables Pinocchio proofs on authenticated data with prover complexity as above, but verification time is linear in the number of committed inputs.

In this work, we give a new Pinocchio-based adaptive zk-SNARK that solves the above problems. We match the best asymptotic performance (i.e., $\Theta(n \log n)$ non-cryptographic work and $\Theta(n)$ cryptographic work for the prover; a $\Theta(n)$-size CRS and constant-time verification); but obtain the first practical solution by adding only minor overhead to "plain" Pinocchio (instead of relying on expensive approaches such as subset-sum or bootstrapping).

As additional contributions, we apply our zk-SNARK in the Trinocchio setting, and present tooling to easily perform verifiable computations. Trinocchio [SVdV15] achieves privacy-preserving outsourcing to untrusted workers by combining the privacy guarantees of multi-party computation with the correctness guarantees of the Pinocchio zk-SNARK. With our adaptive zk-SNARK, computation can be chosen *after* the inputs were provided and more complex functionalities can be achieved by using the output of one computation as input of another. We also improve the generality of [SVdV15] by proving security for *any* suitable MPC protocol and adaptive zk-SNARK. Our tooling consists of a Python frontend and a C++ backend. The frontend allows easy programming of verifiable computations (with libraries for zero testing, oblivious indexing and fixed-point computations), and execution either directly (for normal outsourcing sccenarios) or with MPC (for privacy-preserving outsourcing). The backend provides key generation, proving, and verification functionality for both scenarios.

2   Pinocchio/Trinocchio: zk-SNARKs from QAPs

In this section, we discuss the Pinocchio zk-SNARK [PHGR13], and the Trinocchio system [SVdV15] that uses multi-party computation to build Pinocchio proofs in a privacy-preserving way.

2.1   Modelling Computations as Quadratic Arithmetic Programs

The basic formalism used in Pinocchio to model computation is the *quadratic arithmetic program* (QAP). A QAP over a field $\mathbb{F}$ is a triple $(\mathbf{V}, \mathbf{W}, \mathbf{Y}) \in (\mathbb{F}^{d \times k})^3$, where d is called the *degree* of the QAP and k is called the *size*. A vector $\boldsymbol{x} \in \mathbb{F}^k$ is said to be a *solution* to the QAP if $(\mathbf{V} \cdot \boldsymbol{x}) \times (\mathbf{W} \cdot \boldsymbol{x}) = \mathbf{Y} \cdot \boldsymbol{x}$, where $\times$ denotes

---

[1] In practice, computing the hash is complex itself. It can be avoided with bootstrapping [Lip16], giving slightly worse asymptotics and again a large practical overhead

the pairwise product and $\cdot$ denotes normal matrix-vector multiplication. A QAP Q is said to *compute* function $f : \mathbb{F}^i \to \mathbb{F}^j$ if $\boldsymbol{b} = f(\boldsymbol{a})$ if and only if there exists a *witness* $\boldsymbol{w}$ such that $(\boldsymbol{a}; \boldsymbol{b}; \boldsymbol{w})$ is a solution to Q. For example, consider the QAP

$$\mathbf{V} = \begin{pmatrix} 1\ 1\ 0\ 0 \\ 1\ 1\ 0\ 0 \end{pmatrix}, \quad \mathbf{W} = \begin{pmatrix} 1\ 1\ 0\ 0 \\ 0\ 0\ 1\ 0 \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} 0\ 0\ 0\ 1 \\ 0\ 0\ 1\ 0 \end{pmatrix}.$$

Intuitively, the first row of this QAP represents equation $(x_1 + x_2) \cdot (x_1 + x_2) = x_4$ in variables (x_1, x_2, x_3, x_4) whereas the second row represents equation $(x_1 + x_2) \cdot x_3 = x_3$. Note that $x_3 = (x_1 + x_2)^3$ if and only if there exists x_4 satisfying the two equations, so this QAP computes function $f : (x_1, x_2) \mapsto x_3$.[2]

Fixing d distinct, public $\omega_1, \ldots, \omega_d \in \mathbb{F}$, then a QAP can equivalently be described by a collection of interpolating polynomials in these points. Namely, let $\{v_i(x)\}$ be the unique polynomials of degree $< d$ such that $v_i(\omega_j) = \mathbf{V}_{j,i}$, and similarly for $\{w_i(x)\}$, $\{y_i(x)\}$. Then $\{v_i(x), w_i(x), y_i(x)\}$ is an equivalent description of the QAP. Defining $t(x) = (x - \omega_1) \cdot \ldots \cdot (x - \omega_d) \in \mathbb{F}[x]$, note that $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ is a solution to Q if and only if, for all j, $(\sum_i \boldsymbol{x}_i \cdot v_i(\omega_j)) \cdot (\sum_i \boldsymbol{x}_i \cdot w_i(\omega_j)) = (\sum_i \boldsymbol{x}_i \cdot y_i(\omega_j))$, or equivalently, if $t(x)$ divides $p(x) := (\sum_i \boldsymbol{x}_i \cdot v_i(x)) \cdot (\sum_i \boldsymbol{x}_i \cdot w_i(x)) - (\sum_i \boldsymbol{x}_i \cdot y_i(x)) \in \mathbb{F}[x]$.

2.2 Pinocchio: A zk-SNARK from QAPs

Pinocchio is a "zero-knowledge succinct argument of knowledge" (zk-SNARK) [GGPR13] based on QAPs. A zk-SNARK consists of a key generation algorithm, that generates an evaluation key and a verification key; a prover algorithm, that uses the evaluation key to produce a proof; and a verification algorithm, that uses the verification to verify this proof. In Pinocchio the prover proves that, for given $\boldsymbol{v}$, it knows witness $\boldsymbol{w}$ such that $(\boldsymbol{v}; \boldsymbol{w})$ is a solution to a given QAP Q.

The key material of Pinocchio contains discrete logarithm groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ of order $|\mathbb{F}|$ and a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$ for which the $(4d + 4)$-PDH, d-PKE and $(8d + 8)$-SDH assumptions [PHGR13] hold, with d the degree of the QAP. Fix random, secret $s, \alpha_v, \alpha_w, \alpha_y, \beta, r_v, r_w, r_y(x) := r_v r_w$. Given generators $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $f \in \mathbb{F}[x]$, let us write $\langle f \rangle_1$ for $f(s) \cdot g_1$ and $\langle f \rangle_2$ for $f(s) \cdot g_2$.

The central idea of Pinocchio is to prove satisfaction of all QAP equations using evaluations of the interpolating polynomials in a secret point. Namely, the prover computes quotient polynomial $h = p/t$ and basically provides evaluations "in the exponent" of h, $\sum_i \boldsymbol{x}_i \cdot v_i$, $\sum_i \boldsymbol{x}_i \cdot w_i$, $\sum_i \boldsymbol{x}_i \cdot y_i$ in the point s that is unknown to him, that can then be verified using the pairing. Precisely, the prover algorithm, given solution $\boldsymbol{x} = (\boldsymbol{v}; \boldsymbol{w})$ to the QAP, generates random $\delta_v, \delta_w, \delta_y$; computes coefficients $\boldsymbol{h}$ of the polynomial $(\sum_i \boldsymbol{x}_i \cdot v_i(x) + \delta_v t(x)) \cdot (\sum_i \boldsymbol{x}_i \cdot w_i(x) + \delta_w t(x)) - (\sum_i \boldsymbol{x}_i \cdot y_i(x) + \delta_y t(x))/t(x)$ (with $\delta.$ terms added to make the proof

---

[2] In Pinocchio, the linear terms corresponding to $\mathbf{V}$, $\mathbf{W}$, $\mathbf{Y}$ can also contain constant values. This is achieved by assigning special meaning to a "constant" wire with value 1. We do not formalize this separately, instead leaving it up to the user to include a special variable and an equation $x_i \cdot x_i = x_i$ that forces this variable to be one.

zero-knowledge), and outputs (all $\sum_i$ over witness indices $|\boldsymbol{v}|+1,\ldots,|\boldsymbol{x}|$):

$$\langle V\rangle_1 = \sum_i \boldsymbol{x}_i\langle r_v v_i\rangle_1 + \delta_v\langle r_v t\rangle_1, \langle \alpha_v V\rangle_2 = \sum_i \boldsymbol{x}_i\langle \alpha_v r_v v_i\rangle_2 + \delta_v\langle \alpha_v r_v t\rangle_2,$$
$$\langle W\rangle_2 = \sum_i \boldsymbol{x}_i\langle r_w w_i\rangle_2 + \delta_w\langle r_w t\rangle_2, \langle \alpha_w W\rangle_1 = \sum_i \boldsymbol{x}_i\langle \alpha_w r_w w_i\rangle_1 + \delta_w\langle \alpha_w r_w t\rangle_1,$$
$$\langle Y\rangle_1 = \sum_i \boldsymbol{x}_i\langle r_y y_i\rangle_1 + \delta_y\langle r_y t\rangle_1, \langle \alpha_y Y\rangle_2 = \sum_i \boldsymbol{x}_i\langle \alpha_y r_y y_i\rangle_2 + \delta_y\langle \alpha_y r_y t\rangle_2.$$
$$\langle Z\rangle_1 = \sum_i \boldsymbol{x}_i\langle r_v\beta v_i + r_w\beta w_i + r_y\beta y_i\rangle_1 + \delta_v\langle r_v\beta t\rangle_1 + \delta_w\langle r_w\beta t\rangle_1 + \delta_y\langle r_y\beta t\rangle_1,$$
$$\langle H\rangle_1 = \sum_{j=0}^d \boldsymbol{h}_j\langle x^j\rangle_1.$$

The evaluation key consists of all $\langle\cdot\rangle_1$, $\langle\cdot\rangle_2$ items used in the formulas above.[3]

The verification algorithm, given statement $\boldsymbol{v}$, extends $\langle V\rangle_1$, $\langle W\rangle_1$, $\langle Y\rangle_1$ to include also the input/output wires ($\sum_i$ over I/O wire indices $1,\ldots,|\boldsymbol{v}|$): $\langle V^+\rangle_1 = \langle V\rangle_1 + \sum_i \boldsymbol{x}_i\langle r_v v_i\rangle_1$, $\langle W^+\rangle_2 = \langle W\rangle_2 + \sum_i \boldsymbol{x}_i\langle r_w w_i\rangle_2$, $\langle Y^+\rangle_1 = \langle Y\rangle_1 + \sum_i \boldsymbol{x}_i\langle r_y y_i\rangle_1$. Then, it checks (the verification key are the needed $\langle\cdot\rangle_1$, $\langle\cdot\rangle_2$ items):

$$e(\langle V\rangle_1, \langle\alpha_v\rangle_2) = e(\langle 1\rangle_1, \langle\alpha_v V\rangle_2); \tag{V}$$
$$e(\langle\alpha_w\rangle_1, \langle W\rangle_2) = e(\langle\alpha_w W\rangle_1, \langle 1\rangle_2); \tag{W}$$
$$e(\langle Y\rangle_1, \langle\alpha_y\rangle_2) = e(\langle 1\rangle_1, \langle\alpha_y Y\rangle_2); \tag{Y}$$
$$e(\langle V\rangle_1 + \langle Y\rangle_1, \langle\beta\rangle_2) \cdot e(\langle\beta\rangle_1, \langle W\rangle_2) = e(\langle Z\rangle_1, \langle 1\rangle_2); \tag{Z}$$
$$e(\langle V^+\rangle_1, \langle W^+\rangle_2) \cdot e(\langle Y^+\rangle_1, \langle 1\rangle_2)^{-1} = e(\langle H\rangle_1, \langle r_y t\rangle_2). \tag{H}$$

At a high level, checks $(\mathbf{V})$, $(\mathbf{W})$, $(\mathbf{Y})$ guarantee that the proof is a proof of knowledge of the witness $\boldsymbol{w}$; check $(\mathbf{Z})$ guarantees that the same witness $\boldsymbol{w}$ was used for $\langle V\rangle_1, \langle W\rangle_2, \langle Y\rangle_1$; and check $(\mathbf{Z})$ guarantees that indeed, $p(x) = h(x)\cdot t(x)$ holds, which implies a solution to the QAP.

2.3 Trinocchio: Distributed Proving and Multiple Inputters

In [SVdV15], Trinocchio is proposed: a Pinocchio-based system for "passively secure n-party public verifiable computation". In such a system, mutually distrusting clients can outsource a computation to a set of n workers that do not learn the data they are computing on, but cannot produce a false computation result (even if all corrupted). In Trinocchio, at a high level, clients sends secret shares of of their inputs $\boldsymbol{a}$ to the workers, who use multi-party computation to compute secret shares of output $\boldsymbol{b}$, QAP witness $\boldsymbol{w}$, and QAP proof π. The clients reconstruct their outputs and verify the QAP proof to check its correctness.

The crucial observation of [SVdV15] is that, if Pinocchio is combined with the right multi-party computation protocols, the above construction is very efficient. Computing the vector $\boldsymbol{h}$ of coefficients of the quotient polynomial is an arithmetic circuit of multiplicative depth one. Such a circuit can be evaluated on Shamir-shared inputs without any communication, giving additive secret shares of the result. Moreover, computing the proof ($\langle V\rangle_1,\ldots,\langle H\rangle_1$) is a linear operation on the secret shares of the inputs, outputs, witness, and $\boldsymbol{h}$-coefficients. As a

---

[3] We use $\langle\alpha_v V\rangle_2$ etc. instead of $\langle\alpha_v\rangle_1$ from [SVdV15], so that we can rely on the asymmetric q-PKE assumption from [DFGK14] (which [SVdV15] did not spell out).

consequence, if the outsourced computation is performed based on Shamir secret sharing, then a Pinocchio proof can be produced for that computation without any additional communication between the workers.

3 Adaptive zk-SNARKs based on Pinocchio

In this section, we extend Pinocchio to allow proofs with respect to computation-independent commitments, turning it into an *adaptive zk-SNARK*. We now summarize the basic definitions, adapted from [Lip16] and detailed in Appendix B.

In our setting, an adaptive zk-SNARK is based on an *extractable trapdoor commitment family* $(\mathsf{G0}, \mathsf{Gc}, \mathsf{C})$. This is a family of commitment schemes with different keys based on one CRS. Here, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G0}(1^\kappa)$ outputs a system-wide CRS and a trapdoor; $(\mathsf{ck}, \mathsf{ctd}) \leftarrow \mathsf{Gc}(\mathsf{crs})$ outputs a commitment key and a trapdoor; and $c \leftarrow \mathsf{C}_{\mathsf{ck}}(m; r)$ outputs a commitment with the given key. These commitments are computationally binding (no adversary can open a commitment to two different values); trapdoor (a trapdoor allows arbitrary opening of commitments); perfect hiding (commitments are statistically independent from the underlying message); and extractable (the committed value can be extracted from an algorithm producing a correct commitment).

Given such a commitment scheme family, an *adaptive zk-SNARK* $(\mathsf{G}, \mathsf{P}, \mathsf{V})$ allows a prover to prove knowledge of a witness $\boldsymbol{w}$ such that $(\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}; \boldsymbol{w})$ satisfies a given relation $\mathcal{R}$ (e.g., it is a solution to a QAP), where $\boldsymbol{v_i}$ are committed to by $\boldsymbol{u_i} = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{v_i}; r_i)$. Here, $(\mathsf{crsp}; \mathsf{crsv}; \mathsf{tdp}) \leftarrow \mathsf{G}(\mathsf{crs}; \{\mathsf{ck}_i\})$ outputs evaluation and verification keys and a trapdoor; $\pi \leftarrow \mathsf{P}(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsp}; \boldsymbol{u}; \boldsymbol{v}; \boldsymbol{r}; \boldsymbol{w})$ outputs a proof; and $0/1 \leftarrow \mathsf{V}(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsv}; \boldsymbol{u}; \pi)$ verifies the proof. The scheme should be perfectly complete (it normally returns a verifying proof); an argument of knowledge (the commitment openings and a valid witness can be extracted from an adversary returning commitments and a verifying proof); witness indistinguishable (proofs statistically hide the commitment openings and witness), and perfectly composable zero knowledge (proofs can be simulated using the commitments and trapdoor). (This definition and our constructions do not include plain, non-committed input/output as in Pinocchio, but this is easy to add.)

3.1 A Pinocchio-Based Adaptive zk-SNARK

We obtain our Pinocchio-based adaptive zk-SNARK by generalizing the role of the $\langle Z \rangle_1$ element of the Pinocchio proof. Recall that in Pinocchio, proof elements $\langle V \rangle_1$, $\langle W \rangle_1$, and $\langle Y \rangle_1$ are essentially weighted sums $\sum_j \boldsymbol{x}_j \langle v_j \rangle_1$, $\sum_j \boldsymbol{x}_j \langle w_j \rangle_2$, $\sum_j \boldsymbol{x}_j \langle y_j \rangle_1$ over elements $\langle v_j \rangle_1$, $\langle w_j \rangle_2$, $\langle y_j \rangle_1$ from the CRS, with the weights given by the witness part of the QAP's solution vector $\boldsymbol{x}$. The $\langle Z \rangle_1$ element ensures that these weighted sums all use the same witness. This is done by forcing the prover to come up essentially with $\beta \cdot (\langle V \rangle_1 + \langle W \rangle_2 + \langle Y \rangle_1)$ given only elements $\langle \beta \cdot (v_j + w_j + y_j) \rangle_1$ in which v_j, w_j, and y_j occur together. In earlier works [CFH$^+$15,SVdV15], it was noted that a Pinocchio proof can refer to multiple "blocks" $(\langle V_i \rangle_1, \langle W_i \rangle_2, \langle Y_i \rangle_1, \langle Z_i \rangle_1)$. Each block contains the values

---

Extractable Trapdoor Commitment Scheme Family $(\mathsf{G0}^1, \mathsf{Gc}^1, \mathsf{C}^1)$:

- $\mathsf{G0}^1$: Fix $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ and random s. Return $\mathsf{crs} = (\{\langle x^i\rangle_1, \langle x^i\rangle_2\}_{i=0,\ldots,d})$, $\mathsf{td} = s$.
- Gc^1: Pick random α. Return $\mathsf{ck} = (\langle 1\rangle_1, \langle\alpha\rangle_2, \langle x\rangle_1, \langle\alpha x\rangle_2, \ldots, \langle x^d\rangle_1, \langle\alpha x^d\rangle_2)$
- C^1: Return $(r\langle 1\rangle_1 + \boldsymbol{v}_1\langle x\rangle_1 + \boldsymbol{v}_2\langle x^2\rangle_1 + \ldots, r\langle\alpha\rangle_2 + \boldsymbol{v}_1\langle\alpha x\rangle_2 + \boldsymbol{v}_2\langle\alpha x^2\rangle_2 + \ldots)$

Key generation G^1: Fix a QAP of degree at most d, and let $v_j(x), w_j(x), y_j(x)$ be as in Pinocchio. Fix random, secret $\alpha_v, \alpha_w, \alpha_y, \beta, r_v, r_w$. Let $r_y = r_v r_w$. Let $z_j(x) = x^j + r_v v_j + r_w w_j + r_y y_j$ if $j \leq W$ and $z_j(x) = r_v v_j + r_w w_j + r_y y_j$ otherwise. Evaluation key $(i = 1, \ldots, n, j = 1, \ldots, d)$:

$\langle x^j\rangle_1, \langle r_v v_j\rangle_1, \langle r_v t\rangle_1, \langle\alpha_v r_v v_j\rangle_2, \langle\alpha_v r_v t\rangle_2 \langle r_w w_j\rangle_1, \langle r_w t\rangle_1, \langle\alpha_w r_w w_j\rangle_1, \langle\alpha_w r_w t\rangle_1 \langle r_y y_j\rangle_1,$
$\langle r_y t\rangle_1, \langle\alpha_y r_y y_j\rangle_2, \langle\alpha_y r_y t\rangle_2 \langle\beta_i z_{(i-1)d+j}\rangle_1, \langle\beta_i z_{nd+j}\rangle_1, \langle\beta_i\rangle_1, \langle\beta_i r_v t\rangle_1, \langle\beta_i r_w t\rangle_1, \langle\beta_i r_y t\rangle_1$

Verification key $(i = 1, \ldots, n)$: $(\langle\alpha_v\rangle_2, \langle\alpha_w\rangle_1, \langle\alpha_y\rangle_2, \langle\beta_i\rangle_2, \langle\beta_i\rangle_1, \langle r_y t\rangle_2)$.

Proof generation P^1: Let $\boldsymbol{u}_i = \mathsf{C}^1_{\mathsf{ck}_i}(\boldsymbol{v_i}; r_i)$, and let $\boldsymbol{w}$ be the witness such that $(\boldsymbol{v}_1, \ldots, \boldsymbol{v_n}; \boldsymbol{w})$ is a solution to the QAP. Generate random $\delta_{v,i}, \delta_{w,i}, \delta_{y,i}$. Compute $\boldsymbol{h}$ as the coefficients of polynomial $((\sum_j \boldsymbol{x}_j \cdot v_j(x) + \delta_v \cdot t(x)) \cdot (\sum_j \boldsymbol{x}_j \cdot w_j(x) + \delta_w \cdot t(x)) - (\sum_j \boldsymbol{x}_j \cdot y_j(x) + \delta_y \cdot t(x)))/t(x)$. Return $(i = 1, \ldots, n; [\cdot]$ means only if $i = 1)$:

$$\langle V_i\rangle_1 = \sum_{j=1}^d \boldsymbol{v}_{i,j}\langle r_v v_{(i-1)d+j}\rangle_1 \left[+ \sum_{j=1}^N \boldsymbol{w}_j\langle r_v v_{nd+j}\rangle_1\right] + \delta_{v,i}\langle r_v t\rangle_1, \langle\alpha_v V_i\rangle_2 = \ldots$$

$$\langle W_i\rangle_1 = \sum_{j=1}^d \boldsymbol{v}_{i,j}\langle r_w w_{(i-1)d+j}\rangle_1 \left[+ \sum_{j=1}^N \boldsymbol{w}_j\langle r_w w_{nd+j}\rangle_1\right] + \delta_{w,i}\langle r_w t\rangle_1, \langle\alpha_w W_i\rangle_1 = .$$

$$\langle Y_i\rangle_1 = \sum_{j=1}^d \boldsymbol{v}_{i,j}\langle r_y y_{(i-1)d+j}\rangle_1 \left[+ \sum_{j=1}^N \boldsymbol{w}_j\langle r_y y_{nd+j}\rangle_1\right] + \delta_{y,i}\langle r_y t\rangle_1, \langle\alpha_y Y_i\rangle_2 = \ldots$$

$$\langle Z_i\rangle_1 = \sum_{j=1}^d \boldsymbol{v}_{i,j}\langle\beta_i z_{(i-1)d+j}\rangle_1 \left[+ \sum_{j=1}^N \boldsymbol{w}_j\langle\beta_i z_{nd+j}\rangle_1\right] + r_i\langle\beta_i\rangle_1 + \delta_{v,i}\langle\beta_i r_v t\rangle_1$$

$$\langle H\rangle_1 = \sum_j \boldsymbol{h}_j\langle x^j\rangle_1. \qquad\qquad\qquad + \delta_{w,i}\langle\beta_i r_w t\rangle_1 + \delta_{y,i}\langle\beta_i r_y t\rangle_1$$

Proof verification V^1: Letting $\mathsf{ck}_i = (\ldots, \langle\alpha_i\rangle_2)$, $\boldsymbol{u}_i = (\langle C_i\rangle_1, \langle\alpha_i C_i\rangle_2)$, check that:

$$e(\langle C_i\rangle_1, \langle\alpha_i\rangle_2) = e(\langle 1\rangle_1, \langle\alpha_i C_i\rangle_2); \quad e(\langle V_i\rangle_1, \langle\alpha_v\rangle_2) = e(\langle\alpha_v V_i\rangle_1, \langle 1\rangle_2); \qquad (\mathbf{C,V})$$

$$e(\langle\alpha_w\rangle_1, \langle W_i\rangle_2) = e(\langle 1\rangle_1, \langle\alpha_w W_i\rangle_2); \quad e(\langle Y_i\rangle_1, \langle\alpha_y\rangle_2) = e(\langle 1\rangle_1, \langle\alpha_y Y_i\rangle_2); \qquad (\mathbf{W,Y})$$

$$e(\langle V_i\rangle_1 + \langle Y_i\rangle_1 + \langle C_i\rangle_1, \langle\beta_i\rangle_2) \cdot e(\langle\beta_i\rangle_1, \langle W_i\rangle_2) = e(\langle Z_i\rangle_1, \langle 1\rangle_2); \qquad (\mathbf{Z})$$

$$e(\langle V\rangle_1, \langle W\rangle_2) \cdot e(\langle Y\rangle_1, \langle 1\rangle_2)^{-1} = e(\langle H\rangle_1, \langle r_y t\rangle_2). \qquad (\mathbf{H})$$

(where $\langle V\rangle_1 = \langle V_1\rangle_1 + \ldots + \langle V_n\rangle_1$, $\langle W\rangle_2 = \langle W_1\rangle_2 + \ldots + \langle W_n\rangle_2$, $\langle Y\rangle_1 = \langle Y_1\rangle_1 + \ldots)$

Fig. 1. Pinocchio-Based Adaptive zk-SNARK $(\mathsf{G}^1, \mathsf{P}^1, \mathsf{V}^1)$

of a number of variables of the QAP solution, which is enforced by providing $\langle z_j\rangle_1 = \langle\beta_i \cdot (v_j + w_j + y_j)\rangle_1$ elements only for the indices j of those variables.

Our core idea is use external commitments of the form $\sum_k \boldsymbol{v}_k \cdot \langle x^k\rangle_1$ (that can be re-used across Pinocchio computations) and link the kth component of this commitment to the jth variable of the block using a modified $\langle z_j\rangle_1 = \langle\beta_i \cdot (x^k + v_j + w_j + y_j)\rangle_1$. We use one block per external commitment that the proof refers to. The witness (which is not committed to externally) is included in

the first block, with the normal Pinocchio element $\langle z_j \rangle_1 = \langle \beta_1 \cdot (v_j + w_j + y_j) \rangle_1$ just checking internal consistency as usual. The verification procedure changes slightly: $\langle V \rangle_1$ is no longer extended to $\langle V^+ \rangle_1$ to include public I/O (which we do not have); instead, the ($\mathbf{Z}$) check ensures consistency with the corresponding commitment, for which there is a new correctness check ($\mathbf{C}$).

The precise construction is shown in Figure 1. This construction contains details on how to add randomness to make the proof zero-knowledge; and it shows how additional $\langle \alpha_i \cdot \rangle_1$ elements are added to make sure the computation-independent commitment is an extractable trapdoor commitment scheme (as shown in [Gro10,Lip16]). In Appendix C, we show that:

Theorem 1. *Under the $(4d+3)$-PDH, d-PKE, and $(8d+6)$-SDH assumptions (Appendix A), where d is the maximal degree of the QAPs used, $(\mathsf{G}^1, \mathsf{P}^1, \mathsf{V}^1)$ is an adaptive zk-SNARK.*

3.2 Smaller Proofs and Comparison to Geppetto/"Hash First"

We now present two optimization that decrease the size of the above zk-SNARK, and compare the concrete efficiency of our three proposals to two related proposals from the literature. Note that, in the above construction, seven Pinocchio proof elements $\langle V \rangle_1, \langle \alpha_v V \rangle_2, \langle W \rangle_2, \langle \alpha_w W \rangle_1, \langle Y \rangle_1, \langle \alpha_y Y \rangle_2, \langle Z \rangle_1$ are repeated for each input commitment. We show present two different (but, unfortunately, mutually incompatible) ways in which this can be avoided.

In our first optimization, inspired by a similar proposal to reduce verification work in Pinocchio ([CFH$^+$15], later corrected by [Par15]), we decrease proof size and verification time at the expense of needing a larger-degree QAP. Namely, suppose that all variables in a given commitment occur only in the right-hand side of QAP equations. In this case, $v_j(x) = w_j(x) = 0$ for all j, so proof elements $\langle V_i \rangle_1, \langle \alpha_v V_i \rangle_2, \langle W_i \rangle_2, \langle \alpha_w W_i \rangle_1, \langle Y_i \rangle_1, \langle \alpha_y Y_i \rangle_2$ contain only randomness and, setting $\delta_{v,j} = \delta_{w,j} = 0$, can be omitted. As a consequence, the marginal costs per commitment used decrease from 7 to 3; the ($\mathbf{V}$) and ($\mathbf{W}$) verification steps can be skipped and the ($\mathbf{Z}$) step simplified. To guarantee that a committed variable a only occurs in the right-hand of equations, we can introducing a witness b and equation $0 \cdot 0 = a - b$, slightly increasing the overall QAP size and degree. (This cannot be done for the first commitment since $\langle V_1 \rangle_1, \ldots$ also contain the witness, which occur in the left-hand side of equations as well.)

Our second proposal is a modified zk-SNARK that also reduces the marginal cost per commitment from 7 to 3, but gives more efficient verification when using many commitments. The core idea is to first concatenate all commitments $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ into one "intermediate" commitment $\boldsymbol{u}'$, and then use the original zk-SNARK with respect to $\boldsymbol{u}'$. More precisely, we build intermediate commitment $\boldsymbol{u}'_1$ with the first ℓ_1 values of $\boldsymbol{u}_1$; $\boldsymbol{u}'_2$ with ℓ_1 zeros followed by the first ℓ_2 values of $\boldsymbol{u}_2$; etcetera. Then, $\boldsymbol{u}' = \sum_i \boldsymbol{u}'_i$ is a commitment to the first $\ell_1, \ldots, \ell_n$ values of the respective commitments $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$. To avoid ambiguity between normal and intermediate commitments, normal commitments include a random

Construction	Comm. Size	Proof size	Prover computation		Verif comp
			non-crypt. op.	crypt. op.	
Geppetto	3 gr. el.	8 gr. el.	$\Theta(D \log D)$	$\Theta(D)$	$4n + 12$ pair.
Hash First+Pinocchio	2 gr. el.	9n+1 gr. el.	$\Theta(d \log d)$	$\Theta(d)$	$13n + 3$ pair.
Hash First+Pinocchio*	2 gr. el.	5n+5 gr. el.	$\Theta(d' \log d')$	$\Theta(d')$	$8n + 8$ pair.
Our zk-SNARK I	2 gr. el.	7n+1 gr. el.	$\Theta(d \log d)$	$\Theta(d)$	$11n + 3$ pair.
Our zk-SNARK I*	2 gr. el.	3n+5 gr. el.	$\Theta(d' \log d')$	$\Theta(d')$	$7n + 7$ pair.
Our zk-SNARK II	2 gr. el.	3n+8 gr. el.	$\Theta(d \log d)$	$\Theta(d)$	$6n + 12$ pair.

Table 1. Comparison between Pinocchio-based SNARKs (n: number of commitments; d is QAP degree; $d' \leq d$ is QAP degree with optimization; $D \geq d$ is fixed QAP degree)

factor r_c, i.e. $(r\langle r_c \rangle_1 + \sum_i \boldsymbol{v}_i \langle r_c x^i \rangle_1, r\langle \alpha r_c \rangle_2 + \ldots)$ and intermediate commitments are as above[4]. Proving correspondence between normal and intermediate commitments is done similarly to the ($\mathbf{Z}$) check above: we generate random β_i and give $\langle \beta_i' \cdot (r_c x^j + x^{\ell_1 + \ldots + \ell_{i-1} + j}) \rangle_1$ to the prover, who needs to produce proof element $\langle Z_i' \rangle_1$ such that $\langle Z_i' \rangle_1 = (\langle C_i \rangle_1 + \langle C_i' \rangle_1)^{\beta_i'}$, which he can only do if $\langle C_i' \rangle_1$ is formed correctly. Details and the security proof appear in Appendix C.

In Table 1, we provide a detailed comparison of our zk-SNARKs with two similar constructions: the Geppetto protocol due to [CFH+15] (which is also zero-knowledge but not adaptive); and the "hash first" approach applied to Pinocchio [FFG+16] (which is adaptive but not zero-knowledge). Geppetto is Protocol 2 from [CFH+15]. We assume QAP witnesses of $\mathcal{O}(d)$. In Geppetto, a fixed set of QAPs of degree d_i are combined into one large "MultiQAP" of degree D slightly larger than $\max d_i$. As a consequence, if both small and large computations need to be applied on the same data, then the small computations take over the much worse performance of the large computations. For Hash First+Pinocchio, we took the extractable scheme $\mathsf{XP}_{\mathcal{E}}$ since the Geppetto and our construction are extractable as well. To make it work on multiple commitments (which is described for neither Hash First nor Pinocchio), we assume natural generalizations of Hash First and of Pinocchio along the lines of [CFH+15,SVdV15]. Our first optimization can be applied to this construction; we mark the result with a star and write $d' \geq d$ for the increased degree due to the use of this optimization. Finally, we show our zk-SNARK without and with the first optimization; and our second zk-SNARK construction (to which the optimization does not apply).

In conclusion, Geppetto is the most efficient construction, but apart from not being adaptive, it also requires all computations to be fixed and of the same size, making it inefficient for small computations when they are combined with large ones. Our construction outperforms Hash First+Pinocchio, essentially adding zero knowledge for free; which variant is best depends on n and $d' - d$. Note that Hash First allows using the same commitment in different zk-SNARK schemes; our scheme only allows this for zk-SNARKs based on the kind of polynomial commitments used in Pinocchio.

---

[4] Hence this construction can only handle input sof combined size at most d.

Secure adaptive function evaluation	**Correct adaptive function evaluation**
– Honest data owners send inputs $\boldsymbol{x_i}$ to trusted party	– Honest data owners send inputs $\boldsymbol{x_i}$ to trusted party
– Adversary sends inputs $\boldsymbol{x_i}$ of corrupted data owners to trusted party (active adversary may modify them)	– Adversary sends inputs $\boldsymbol{x_i}$ of corrupted data owners to trusted party (active adversary may modify them)
– Client sends function f, input $\boldsymbol{x}$ to trusted party (active: may modify)	– Client sends function f, input $\boldsymbol{x}$ to trusted party (active: may modify)
– Trusted party computes $\boldsymbol{y} = f(\boldsymbol{x_1}, \ldots, \boldsymbol{x_m}, \boldsymbol{x})$ (where $y_1 = \ldots = \bot$ if any $x_i = \bot$)	– Trusted party computes $\boldsymbol{y} = f(\boldsymbol{x_1}, \ldots, \boldsymbol{x_m}, \boldsymbol{x})$ (where $y_1 = \ldots = \bot$ if any $x_i = \bot$)
– Trusted party sends $\boldsymbol{y}$ to client, f to adversary	– Trusted party sends $\boldsymbol{x_1}, \ldots, \boldsymbol{x_m}, \boldsymbol{x}, f$ to adversary, receives r
– Client outputs received value (if honest) or $\bot$ (if corrupted); adversary chooses own output	– Trusted party sends $\boldsymbol{y}$ to client, or $\bot$ if $r = \bot$
	– Client outputs received value (if honest) or $\bot$ (if corrupted); adversary chooses own output

Fig. 2. Ideal-world executions of secure (left) and correct (right) adaptive function evaluation. The highlighted text indicates where the two differ.

4 Secure and Correct Adaptive Function Evaluation

We now show how our zk-SNARK can be used to perform "adaptive function evaluation": privacy-preserving verifiable computation on committed data. We present a construction based on multi-party computation and *any* adaptive zk-SNARK; we then show that our adaptive zk-SNARK gives a particularly efficient instantiation. This generalizes Trinocchio [SVdV15], which is not adaptive and tied to the particular Pinocchio zk-SNARK and a particular MPC protocol.

4.1 Security model

We consider a setting in which multiple mutually distrusting *data owners* want to allow privacy-preserving outsourced computations on their joint data. A *client* can choose a computation to be performed by a set of *workers*. The input data is sensitive, so the workers should not learn what data they are computing on; but the client wants to be guaranteed the computation result is correct. We demand correctness regardless of which data owners and/or workers are actively corrupted; and privacy when up to a maximum number of workers are passively corrupted. This scenario is motivated by settings where the data owners together choose the computation infrastructure (so they feel active corruption is unlikely) but need to convince an external client (e.g. a medical reviewer) of correctness.

We model security using the ideal/real paradigm, i.e., by specifying an "ideal world" in which the task at hand is carried out by an incorruptible trusted party, and demanding that real-world executions give the same result as executions in

Protocol Adaptive Trinocchio

(Data provider has $\boldsymbol{a_i} \in \mathbb{F}^d$; client has $\boldsymbol{a_c} \in \mathbb{F}^{d'}$, function $f : (\mathbb{F}^d)^n \times \mathbb{F}^{d'} \to \mathbb{F}^{d-d'}$.)

1. The trusted party generates a system-wide CRS crs of the trapdoor commitment family, and commitment keys $\mathsf{ck}_1, \ldots, \mathsf{ck}_n, \mathsf{ck}_c$ for the data owner and client. This material is distributed to all parties (formally, by a hybrid call to CommGen).
2. Each data owner computes commitment $c_i = \mathsf{C}_{\mathsf{ck}_i}(\boldsymbol{a_i}, r_i)$ to its input $\boldsymbol{a_i} \in \mathbb{F}^d$ using randomness r_i, and publishes it on a bulletin board.
3. The data owners, workers, and client use the MPC protocol to do the following:
   - Each data owner provides input $\boldsymbol{a_i}$ and randomness r_i
   - For each i, compute $c_i' = \mathsf{C}_{\mathsf{ck}_i}(\llbracket\boldsymbol{a_i}\rrbracket, \llbracket r_i\rrbracket)$; if $c_i \neq c_i'$ then abort
4. The client provides function f to the trusted party. The trusted party determines a QAP Q computing f and a function f' solving Q, and performs key generation of the adaptive zk-SNARK (where one commitment combines the client's input and output). The client gets verification key crsv; the workers get Q, f', and the corresponding evaluation key crsp (formally, by a hybrid call to CompGen).
5. The data owners, workers, and client continue with the MPC from step 3:
   - Client: provide input $\boldsymbol{a_c}$
   - Compute $(\llbracket\boldsymbol{b}\rrbracket; \llbracket\boldsymbol{w}\rrbracket) \leftarrow f'(\llbracket\boldsymbol{a_1}\rrbracket; \ldots; \llbracket\boldsymbol{a_n}\rrbracket; \llbracket\boldsymbol{a_c}\rrbracket)$
   - Compute $\llbracket c_c\rrbracket \leftarrow \mathsf{C}_{\mathsf{ck}_c}(\llbracket\boldsymbol{a_c}\rrbracket, \llbracket\boldsymbol{b}\rrbracket; \llbracket r_c\rrbracket)$ for random r_c
   - Compute $\llbracket\pi\rrbracket \leftarrow \mathsf{P}(\mathsf{crs}, \{\mathsf{ck}_i\}, \ldots, \mathsf{ck}_n, \mathsf{ck}_c, \mathsf{crsp}; c_1, \ldots, c_n, \llbracket c_c\rrbracket; \llbracket\boldsymbol{a_1}\rrbracket; \ldots; \llbracket\boldsymbol{a_n}\rrbracket; \llbracket\boldsymbol{a_c}\rrbracket, \llbracket\boldsymbol{b}\rrbracket; \llbracket r_1\rrbracket, \ldots, \llbracket r_n\rrbracket, \llbracket r_c\rrbracket; \llbracket\boldsymbol{w}\rrbracket)$
   - Open outputs $\llbracket\boldsymbol{b}\rrbracket, \llbracket r_c\rrbracket, \llbracket c_c\rrbracket, \llbracket\pi\rrbracket$ to the client
6. The client checks whether $\mathsf{V}(\mathsf{crs}, \mathsf{ck}_1, \ldots, \mathsf{ck}_n, \mathsf{ck}_c, \mathsf{crsv}; c_1, \ldots, c_n, c_c; \pi) = 1$ and $c_c = \mathsf{C}_{\mathsf{ck}_c}(\boldsymbol{a_c}, \boldsymbol{b}; r_c)$ and if so, returns computation result $\boldsymbol{b}$.

Fig. 3. The Adaptive Trinocchio protocol

this ideal world (analogously to [SVdV15]). "Secure adaptive function evaluation" (Figure 2, left) guarantees privacy *and* correctness; we will realize this ideal functionality if at most a threshold of workers are passively corrupted (but all other parties can be actively corrupted). "Correct adaptive function evaluation" (Figure 2, right) guarantees only correctness; we will realize this ideal functionality regardless of corruptions. In both models, data owners provide their inputs independently from each other and (unlike Trinocchio) from the function at hand; and corrupted inputters cannot fail selectively based on the input.

Definition 1. *A protocol securely adaptively evaluates function f if it implements the ideal functionality shown in Figure 2, left, according to the execution model from Appendix B.3 (Definition 8). A protocol correctly adaptively evaluates function f if it implements the ideal funcitonality shown in Figure 2 (right).*

4.2 General Construction

The high-level idea for obtaining secure and correct adaptive function evaluation is quite simple. To achieve secure adaptive function evaluation, the workers com-

pute the function using multi-party computation (MPC), guaranteeing privacy and correctness under certain conditions. However, when these conditions are not met, we still want to achieve correct adaptive function evaluation, i.e., we still want to ensure a correct computation result. To achieve this, the workers also produce, using MPC, a zk-SNARK proof of correctness of the result.

In our particular construction, we guarantee secure adaptive function evaluation (i.e., with privacy) as long as up to t out of m workers are passively corrupted. For this, we rely on a MPC protocol with the same privacy guarantees. More precisely, we require a MPC protocol in the outsourcing setting, i.e., with separate inputters (in our case, the data owners and the client), recipients (the client) and workers. The protocol needs to be reactive, so that the data owners can provide their input before knowing the function to be computed[5]; and secure if up to t out of m workers are passively corrupted, and any number of data owners and clients are actively corrupted. As we show below, the above requirements are met, e.g., by MPC protocols based on (t, n)-Shamir secret sharing (e.g., [dH12]) between $n = 2t + 1$ workers.

Our protocol is shown in Figure 3. It uses a MPC protocol with the above properties, a trapdoor commitment family, and an adaptive zk-SNARK, instantiated for the function to be computed. The protocol relies on a trusted party that generates the key material of the zk-SNARK, but is otherwise not involved in the computation. Each data owner has an input $\boldsymbol{a_i} \in \mathbb{F}^d$ and the client has an input $\boldsymbol{a_c} \in \mathbb{F}^{d'}$ and a function $f : (\mathbb{F}^d)^n \times \mathbb{F}^{d'} \to \mathbb{F}^{d-d'}$ that it wants to compute on the combined data. Internal variables of the MPC protocol are denoted $[\![\cdot]\!]$.

In step 1, the trusted party sets up the trapdoor commitment family, generating separate keys for data provider and the client. (This prevents parties from copying each other's input.) In step 2, each data provider publishes a commitment to his input. In step 3, each data providers inputs its data and the randomness used for the commitment to the MPC protocol. The workers re-compute the commitments based on this opening and abort in case of a mismatch. (This prevents calling P on mismatching inputs in which case it may not be zero-knowledge.) In step 4, the client chooses the function f to be computed, based on which the trusted party performs key generation. (By doing this after the data owners' inputs, we prevent a selective failure attack from their side.) In step 5, the computation is performed. Using MPC, the client's output and witness are computed; a commitment to the client's I/O is produced, and a zk-SNARK proof of correctness with respect to the commitments of the data owners and client is built.[6] The client learns the output, randomness for its commitment, the commitment itself, and the proof. In step 6, the client re-computes the commitment and verifies the proof; in case of success, it accepts the output.

In Appendix D, we prove that:

---

[5] Using non-reactive MPC requires is also possible, but then steps 3 and 4 of the protocol need to be swapped. As a consequence, data owners can abort based on the client's choice of function, leading to a weaker form of correct function evaluation.

[6] Equivalently, the workers can open c_c and π and send them to the client in the plain.

Theorem 2. *If the MPC protocol, trapdoor commitment family, and adaptive zk-SNARK are secure, then "Adaptive Trinoccio" (Figure 3) correctly adaptively evaluates f in the* $(\mathsf{CommGen}, \mathsf{CompGen})$*-hybrid model. If at most one worker is corrupted, then "Adaptive Trinoccio" securely adaptively evaluates f in the* $(\mathsf{CommGen}, \mathsf{CompGen})$*-hybrid model.*

By sharing commitment between proofs, it is possible to flexibly perform many variants of the same computation all based on the same key material. In particular, as we show in the case study, this enables computations on arbitrary-length data using the same key material (which was impossible in Trinocchio). It is also easy to support multiple clients or multiple commitments per data owner.

4.3 Efficient Instantiation using Secret Sharing and our zk-SNARK

We now show that our zk-SNARKs and MPC based on Shamir secret sharing give a particularly efficient instantiation of the above framework. The idea is the same as for Trinocchio [SVdV15]: our zk-SNARK is essentially an arithmetic circuit of multiplicative depth 1, so given a solution to the QAP it can be done without any communication between the workers.

In more detail, we perform MPC based on Shamir secret sharing between the m workers (e.g., [dH12]). This guarantees privacy as long as at most t workers are passively corrupted, where $m = 2t + 1$. Inputs are provided by the inputters as an additive sharing between all workers: this way actively corrupted inputters cannot provide an inconsistent sharing. The workers Shamir-share and sum up the additive shares to obtain a Shamir sharing of the input. Outputs are provided to recipients either as Shamir shares or as freshly randomized additive shares: the latter allows producing our zk-SNARK proof without any communication.

Either of our zk-SNARK constructions can be used; we provide details for the first one. Below, write $[\![\cdot]\!]$ for Shamir sharing and $[\cdot]$ for additive sharing. (Note that Shamir sharings can be converted locally to additive sharings at no cost.) In step 3 of the protocol, to open c'_i, the parties apply $\mathsf{C}_{\mathsf{ck}_i}$ on their additive shares of the input and randomness, add a random additive sharing of zero (which can be generated non-interactively using pseudo-random zero sharing), and reveal the result. In step 5, $[\![\boldsymbol{b}]\!]$; $[\![\boldsymbol{w}]\!]$ are computed as Shamir secret shares. Next, $[c_c]$ is computed as an additive sharing by applying $\mathsf{C}_{\mathsf{ck}_c}$ on additive shares and adding a random sharing of zero. Next, P^1 is applied by performing the following steps:

– Generate $\delta_{v,i}, \delta_{w,i}, \delta_{y,i}$ by pseudo-random secret sharing.
– Compute $[\boldsymbol{h}] = ((\sum_j [\![\boldsymbol{x}_j]\!] \cdot v_j(x) + [\![\delta_v]\!] \cdot t(x)) \cdot (\sum_j [\![\boldsymbol{x}_j]\!] \cdot w_j(x) + [\![\delta_w]\!] \cdot t(x)) - (\sum_j [\![\boldsymbol{x}_j]\!] \cdot y_j(x) + [\![\delta_y]\!] \cdot t(x)))/t(x)$. Essentially this is done by performing the computation straight on Shamir secret shares; because there is only layer of multiplications of shares, this directly gives an additive sharing of the result. Smart use of FFTs gives time complexity $\mathcal{O}(d \cdot \log d)$ [BSCG$^+$13,SVdV15].
– All proof elements are now linear combinations of secret-shared data; compute them by taking linear combinations of the (Shamir or additive) shares and adding a random sharing of zero.

What remains is how to compute the solution of the QAP using multi-party computation. Namely, in addition to computing the function result $[\![b]\!]$, the MPC also needs to compute witness $[\![w]\!]$ to the QAP. Actually, if the function to be computed is described as an arithmetic circuit, this is very easy. Namely, in this case, the witness for the natural QAP for the function is exactly the vector of results of all intermediate multiplications; and these results are already available as Shamir secret shares as a by-product of performing the MPC. Hence, in this case, computing $[\![w]\!]$ in addition to $[\![b]\!]$ incurs no overhead.

If a custom MPC protocol for a particular subtask is used, then it is necessary to devise specific QAP equations and an MPC protocol to compute their witness. As an example, consider the MPC operation $[\![b]\!] \leftarrow [\![a \neq 0]\!]$, i.e., b is assigned 1 if $a \neq 0$ and 0 if $a = 0$. For computing $[\![b]\!]$, a fairly complex protocol is needed, cf. [dH12]. However, proving that b is correct using a QAP is simple [PHGR13]: introduce witnesses $c := (a + (1 - b))^{-1}$, $d := 1$ and equations:

$$a \cdot c = b \quad a \cdot (d - b) = 0 \quad d \cdot d = d.$$

Indeed, if $a = 0$ then the first equation implies that $b = 0$; if $a \neq 0$ then the second and third equations imply that $b = 1$. In both cases, the given c is defined and, combined with $d = 1$, makes all three equations hold. In our case study, we show similarly how, for complex MPC protocols for fixed-point arithmetic, simple QAPs proving correctness exist with easily computable witnesses.

5 Prototype and Distributed Medical Research Case

In this section, we present a proof-of-concept implementation of our second zk-SNARK construction and our Adaptive Trinocchio protocol. Computations can be specified at a high level using our Python frontend; executed either locally or in a privacy-preserving way using multi-party computation; and then automatically proven and verified to be correct by a C++ backend. We show how two different computations can be performed on the same committed data coming from multiple hospitals (with key material independent from the input length, and optionally in a privacy-preserving way): aggregate survival statistics on two patient populations, and the "logrank test": a common statistical test whether there is a statistically significant difference survival rate between the populations.

5.1 Prototype of our zk-SNARK and Adaptive Trinocchio

Our prototype is built on top of VIFF[7], a Python implementation of MPC based on Shamir secret sharing. In VIFF, computations on secret shares are specified as normal computations by means of operator overloading, e.g., assigning `c=a*b` induces a MPC multiplication protocol. We add a new runtime to VIFF that also allows computations to be performed locally without MPC.

---

[7] TUeVIFF, `http://www.win.tue.nl/~berry/TUeVIFF/`, based on `http://viff.dk`

Algorithm 1 Anonymized survival data computation

Require: $\llbracket d_1 \rrbracket, \llbracket n_1 \rrbracket, \llbracket d_2 \rrbracket, \llbracket n_2 \rrbracket$: block of survival data points for two populations
Ensure: $(\llbracket d'_1 \rrbracket, \llbracket n'_1 \rrbracket, \llbracket d'_2 \rrbracket, \llbracket n'_2 \rrbracket)$ aggegated survival data for the block
 1: **function** SUMM($\llbracket \mathbf{d}_{i,1} \rrbracket, \llbracket \mathbf{d}_{i,2} \rrbracket, \llbracket \mathbf{n}_{i,1} \rrbracket, \llbracket \mathbf{n}_{i,2} \rrbracket$)
 2:     **return** $(\sum_i \llbracket \mathbf{d}_{1,i} \rrbracket, \llbracket \mathbf{n}_{1,1} \rrbracket, \sum_i \llbracket \mathbf{d}_{2,i} \rrbracket, \llbracket \mathbf{n}_{2,1} \rrbracket)$

To support computation proofs, we developed the `viffvc` library that provides a new data type: `VcShare`, a wrapper around a secret share. Each `VcShare` represents a linear combination of QAP variables. Addition and multiplication by constants of `VcShare`s is performed locally by manipulating the linear combination. Constants v are represented as $v \cdot one$, where witness *one* satisfies $one \cdot one = one$ so $one = 1$. When two `VcShare`s $\lambda_1 x_1 + \ldots$ and $\mu_1 x_1 + \ldots$ are multiplied, a local or MPC multiplication operation is performed on the underlying data, and the result is a new `VcShare` x_k wrapping the result as a new QAP variable. QAP equation $(\lambda_1 x_1 + \ldots) \cdot (\mu_1 x_1 + \ldots) = 1 \cdot x_k$ is written to a file, and the multiplication result x_k or its secret share, when known, is written to another file. Apart from multiplication, some additional operations are supported. For the $\llbracket b \rrbracket \leftarrow \llbracket a \neq 0 \rrbracket$ operation discussed in Section 4.3, the implementation computes $\llbracket b \rrbracket$ and $\llbracket c \rrbracket = (\llbracket a \rrbracket + (1 - \llbracket b \rrbracket))^{-1}$, and writes these secret shares/values and the equations from Section 4.3 to the respective files. We also support secret indexing (e.g., [dH12]), and fixed-pont computations as discussed below.

Computations are performed by this custom VIFF-based system together with an implementation of our zk-SNARK. A first tool, `qapgen`, generates the CRS for our trapdoor commitment scheme. A second tool, `qapinput`, builds a commitment to a given input; and computes secret shares of these inputs that are used for MPC computations. Then, our Python implementation is used to compute the function, either locally or using multi-party computation. At the end of this execution, there is one file with the QAP equations, and one file with values/shares for each QAP variable. Our `qapgenf` tool uses the first file to perform key generation of the QAP; this is done only once and for next executions, previous key material is re-used. Our `qapprove` tool uses the second file to generate the zk-SNARK proof (shares) to be received by the client. Finally, a `qapver` tool verifies the proof based on the committed inputs and outputs.

5.2 Application to Medical Survival Analysis

We have applied our prototype to (adaptively) perform computations on survival data about two patient populations. In medical research, survival data about a population is a set of tuples (n_j, d_j), where n_j is the number of patients still in the study just before time j and d_j is the number of deaths at time j. We assume both populations are distributed among multiple hospitals, that each commit to their contributions $(d_{j,1}, n_{j,1}, d_{j,2}, n_{j,2})$ to the two populations at each time.

Aggregate Survival Data Our first computation is to compute an aggregate version of the survival data, where each block $\{d_{j,1}, n_{j,1}, d_{j,2}, n_{j,2}\}_{j=1}^{25}$ of 25 time

---

Algorithm 2 Logrank computation for each time step

Require: $[\![\mathbf{d}_{i,1}]\!], [\![\mathbf{d}_{i,2}]\!], [\![\mathbf{n}_{i,1}]\!], [\![\mathbf{n}_{i,2}]\!]$ survival data at time point i

Ensure: $([\![e_i]\!]^f, [\![v_i]\!]^f, [\![d_i]\!])$ contributions to $\sum_j E_{j,1}, \sum_j V_j, \sum_j d_{j,1}$ for test statistic

1: **function** BLOCK($[\![\mathbf{d}_{i,1}]\!], [\![\mathbf{d}_{i,2}]\!], [\![\mathbf{n}_{i,1}]\!], [\![\mathbf{n}_{i,2}]\!]$)
2:    $[\![ac]\!] \leftarrow [\![\mathbf{d}_{i,1}]\!] + [\![\mathbf{d}_{i,2}]\!]$
3:    $[\![bd]\!] \leftarrow [\![\mathbf{n}_{i,1}]\!] + [\![\mathbf{n}_{i,2}]\!]$
4:    $[\![frc]\!]^f \leftarrow [\![ac]\!]/[\![bd]\!]$
5:    $[\![e_i]\!]^f \leftarrow [\![frc]\!]^f \cdot [\![\mathbf{n}_{i,1}]\!]$
6:    $[\![vn]\!] \leftarrow [\![\mathbf{n}_{i,1}]\!] \cdot [\![\mathbf{n}_{i,2}]\!] \cdot [\![ac]\!] \cdot ([\![bd]\!] - [\![ac]\!])$
7:    $[\![vd]\!] \leftarrow [\![bd]\!] \cdot [\![bd]\!] \cdot ([\![bd]\!] - 1)$
8:    $[\![v_i]\!]^f \leftarrow [\![vn]\!]/[\![vd]\!]$
9:    **return** $([\![e_i]\!]^f, [\![v_i]\!]^f, [\![d_i]\!])$

---

points is summarized as $(\sum_j d_{j,1}, n_{1,1}, \sum_j d_{j,2}, n_{1,2})$. The function SUMM computing this summary is shown in Algorithm 1. Function SUMM translates into a QAP on 26 commitments: as input, for each time point j, a commitment $\sum_i c_{i,j}$ to the combined survival data ($[\![\boldsymbol{d_{i,1}}]\!], [\![\boldsymbol{n_{i,1}}]\!], [\![\boldsymbol{d_{i,2}}]\!], [\![\boldsymbol{n_{i,2}}]\!]$) from the different hospitals i at that time (using the fact that commitments are homomorphic); as output, a commitment to ($[\![d_1']\!], [\![n_1']\!], [\![d_2']\!], [\![n_2']\!]$).

Logrank test Our second computation is the so-called "Mantel-Haenzel logrank test", a statistical test to decide whether there is a significant difference in survival rate between the two populations (as implemented, e.g., in R's `survdiff` function). Given the survival data from two populations, define:

$$E_{j,1} = \frac{(d_{j,1} + d_{j,2}) \cdot n_{j,1}}{n_{j,1} + n_{j,2}}; \quad V_j = \frac{n_{j,1} n_{j,2}(d_{j,1} + d_{j,2})(n_{j,1} + n_{j,2} - d_{j,1} - d_{j,2})}{(n_{j,1} + n_{j,2})^2 \cdot (n_{j,1} + n_{j,2} - 1)};$$

$$X = \frac{\sum_j E_{j,1} - \sum_j d_{j,1}}{\sum_j V_j}.$$

The null hypothesis for the logrank test, i.e., the hypothesis that the two curves represent the same underlying "survival function", corresponds to $X \sim \chi_1^2$. This null hypothesis is rejected (i.e., the curves are different) if $1 - \mathrm{cdf}(X) > \alpha$, where cdf is the cumulative density function of the χ_1^2 distribution and, e.g., $\alpha = 0.05$. We use MPC to compute X, and then apply the cdf in the clear.

Our implementation consists of two different functions: a function BLOCK (Algorithm 2) that computes $(E_{j,1}, V_j, d_{j,1})$ given the survival data at point j; and a function FIN that, given $\sum E_{j,1}, \sum V_j$, and $\sum d_{j,1}$ computes X (Algorithm 3). As above, function BLOCK is applied to commitment $\sum_i c_{i,j}$ to the combined survival data from different hospitals at a particular time, giving output commitment c_j'. Function FIN is applied to commitment $\sum_j c_j'$ to $(\sum E_{j,1}, \sum V_j, \sum d_{j,1})$, again using the fact that commitments are homomorphic; outputting a commitment to X that is output to the client.

Algorithms 2 and 3 use fixed-point numbers $[\![x]\!]^f$, representing value $x \cdot 2^{-\mathrm{k}}$ where we use precision $\mathrm{k} = 20$. We use the fixed-point multiplication $[\![c]\!]^f \leftarrow$

Algorithm 3 Logrank final computation

Require: $[\![es]\!], [\![vs]\!], [\![ds]\!]$: summed-up values required to compute X
Ensure: $[\![chi]\!]^f$ test statistic comparing two curves; supposedly $chi \sim \chi_1^2$
1: **function** FIN($[\![es]\!], [\![vs]\!], [\![ds]\!]$)
2:     $[\![ds]\!]^f \leftarrow [\![ds]\!] \ll$ PRECISION
3:     $[\![dmi]\!]^f \leftarrow [\![ds]\!]^f - [\![vs]\!]^f$
4:     $[\![chi0]\!]^f \leftarrow [\![dmi]\!]^f / [\![vs]\!]^f$
5:     $[\![chi]\!]^f \leftarrow [\![chi0]\!]^f \cdot [\![dmi]\!]^f$
6:     **return** $[\![chi]\!]^f$

	Computation (function): 0.0s (w/o MPC) / 0.1s (w/ MPC)		
	Computation (function+witness): 0.0s (w/o MPC) / 0.1s (w/ MPC)		
	BS=1	**BS=25**	**BS=175**
Aggregate	QAP degree: 3	QAP degree: 3	QAP degree: 57
	Prover: 0.3s/0.4s	Prover: 0.1s/0.1s	Prover: 0.0s/0.0s
	Verifier: 1.2s/1.5s	Verifier: 0.2s/0.2s	Verifier: 0.0s/0.0s
	Computation (function): 0.2s (w/o MPC) / 190.5s (w/ MPC)		
	Computation (function+witness): 0.6s (w/o MPC) / 235.2s (w/ MPC)		
	BS=1	**BS=25**	**BS=175**
Logrank	QAP deg (block): 173	QAP deg (block): 4304	QAP deg (block): 30104
	QAP deg (fin): 85	QAP deg (fin): 85	QAP deg (fin): 85
	Prover: 13.9s/78.5s	Prover: 16.2s/81.0s	Prover: 9.8s/73.5s
	Verifier: 3.9s/4.9s	Verifier: 0.2s/0.3s	Verifier: 0.0s/0.0s

Table 2. Performance: computation/proving/verification; with/without MPC

$[\![a]\!]^f \cdot [\![b]\!]^f$ and division $[\![c]\!]^f \leftarrow [\![a]\!]/[\![b]\!]$, $[\![c]\!]^f \leftarrow [\![a]\!]^f/[\![b]\!]^f$ protocols due to [dH12]. To prove that $[\![c]\!]^f \leftarrow [\![a]\!]^f \cdot [\![b]\!]^f$ is correct, note that we need to show that $c - a \cdot b \in [-2^k, 2^k]$, or equivalently, that $\alpha := c - a \cdot b + 2^k \geq 0$ and $\beta := 2^k - (c - a \cdot b) \geq 0$. We prove this by computing, using MPC, bit decompositions [dH12] $\alpha = \alpha_0 + \alpha_1 \cdot 2 + \ldots + \alpha_k \cdot 2^k$ and $\beta = \beta_0 + \beta_1 \cdot 2 + \ldots + \beta_k \cdot 2^k$ (indeed, α and β are $\leq k + 1$ bits long); these α_i, β_i are the witnesses to QAP equations:

$$\forall i : \alpha_i \cdot (1 - \alpha_i) = 0 \quad c - a \cdot b + 2^k = \alpha_0 + \alpha_1 \cdot 2 + \ldots + \alpha_k \cdot 2^k$$

$$\forall i : \beta_i \cdot (1 - \beta_i) = 0 \quad \beta = 2^k - (c - a \cdot b) = \beta_0 + \beta_1 \cdot 2 + \ldots + \beta_k \cdot 2^k.$$

Similarly, note that $[\![c]\!]^f \leftarrow [\![a]\!]^f/[\![b]\!]^f$ is correct if and only if $a \cdot 2^k - b \cdot c \in [-a, a]$, i.e., $\gamma := d + c \cdot 2^k - d \cdot e \geq 0$ and $\delta := d + d \cdot e - c \cdot 2^k \geq 0$. If a has bitlength at most K (i.e., the represented number has absolute value $\leq 2^{K-k}$), then γ and δ have at most $K + 1$ bits. As above, we prove correctness by determining $(K + 1)$-length bit decompositions of γ and δ and proving them correct. Proving correctness of $[\![c]\!]^f \leftarrow [\![a]\!]/[\![b]\!]$ is analogous.

Performance Table 2 shows the performance of our proof-of-concept implementation for computing aggregate survival data and the logrank test (on a modern laptop). As input, we used the "btrial" data set included in R's "kmsurv" package (on which we indeed repdocue R's `survdiff` result) of 175 data points. Apart

from having one data point per commitment, we also experiment with having a "block size" of 25 or 175 data points. For the logrank test, we use one QAP per block; larger blocks mean less work for the verifier (since there are fewer proofs) but, in theory, more work for the prover (since the proving algorthm is superlinear in the QAP size). For aggregation, we use one QAP per 25 data points or per commitment, whichever is more.

We time the performance of running the computation, producing the proof, and verifying it, with or without MPC. As expected, MPC induces a large overhead for the computation, especially for the logrank test (due to the many fixed-point computations). MPC also incurs an overhead for proving: this is because of the many exponentiations with $|\mathbb{F}|$-sized secret shares rather than small witnesses. Note that proving is faster than computing with MPC: the underlying operations are slower [SVdV15], but the QAP proof is in effect on a verification circuit that is smaller than the circuit of the computation itself [dHSV16]. Proving is faster for block size 175 than block size 25, which is unexpected; this may be because our FFT subroutine rounds up QAP degrees to the nearest power of two, which is favourable in the 175-sized case but not in the 25-sized case. As expected, verification is faster for larger block sizes. (The overhead of MPC here is due to recombing the proof shares into one overall proof.)

6   Conclusion

In this work, we have given the first practical Pinocchio-based adaptive zk-SNARK; applied it in the privacy-presering outsourcing setting; and presented a proof-of-concept implementation. We mention a few promising directions for follow-ups. First, aside from providing a non-adaptive zk-SNARK, Geppetto also introduces the interesting idea of proof bootstrapping, where the verification procedure of the zk-SNARK itself can be performed by means of a verifiable computation, so multiple related proofs can be verified in constant time. Applying this technique in our setting should combine our flexibility with their constant-time verification. Second, it is interesting to see if, apart from secret sharing plus our SNARK, there are other appealing instantiations of our privacy-preserving outsourcing framework. Finally, the combination of MPC and verifiable computation raises the question of constructing efficient QAPs for specific operations and building efficient MPC protocols for computing their witnesses. We have presented zero testing and fixed-point computations as examples, but the same idea is applicable to many other operations as well. More generally, extending our zk-SNARK prototype with more basic operations, and improving its user-friendliness, would help bring the techniques closer to practice.

References

[BBFR15]   M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 271–286, 2015.

[BSCG+13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *Proceedings of CRYPTO*. 2013.

[Can98] R. Canetti. Security and Composition of Multi-party Cryptographic Protocols. *Journal of Cryptology*, 13:2000, 1998.

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

[DFGK14] G. Danezis, C. Fournet, J. Groth, and M. Kohlweiss. Square Span Programs with Applications to Succinct NIZK Arguments. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 532–550, 2014.

[dH12] S. de Hoogh. *Design of large scale applications of secure multiparty computation: secure linear programming.* PhD thesis, Eindhoven University of Technology, 2012.

[dHSV16] S. de Hoogh, B. Schoenmakers, and M. Veeningen. Certificate Validation in Secure Computation and Its Use in Verifiable Linear Programming. In *Progress in Cryptology - AFRICACRYPT 2016 - 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings*, pages 265–284, 2016.

[FFG+16] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno. Hash First, Argue Later: Adaptive Verifiable Computations on Outsourced Data. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1304–1316, 2016.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. In *Proceedings of EUROCRYPT*. 2013.

[Gro10] J. Groth. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, pages 321–340, 2010.

[Lip16] H. Lipmaa. Prover-Efficient Commit-and-Prove Zero-Knowledge SNARKs. In *Progress in Cryptology - AFRICACRYPT 2016 - 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings*, pages 185–206, 2016.

[Par15] B. Parno. A Note on the Unsoundness of vnTinyRAM's SNARK. Cryptology ePrint Archive, Report 2015/437, 2015. `http://eprint.iacr.org/`.

[PHGR13] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *Security and Privacy, 2013 IEEE Symposium on*, pages 238–252, 2013.

[SVdV15] B. Schoenmakers, M. Veeningen, and N. de Vreede. Trinocchio: Privacy-Friendly Outsourcing by Distributed Verifiable Computation. Cryptology ePrint Archive, Report 2015/480, 2015. `http://eprint.iacr.org/`.

A Cryptographic Assumptions

In this section, we outline the cryptographic assumptions on which our protocols are based. We use the formulations from [DFGK14]: they are generalizations of the definitions from [PHGR13] to the asymmetric pairing setting that we use.

Suppose $(p, \mathbb{G}, \mathbb{G}', \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\kappa)$ outputs three multiplicative cyclic groups of prime order p, and an efficient bilinear map $e : \mathbb{G} \times \mathbb{G}' \to \mathbb{G}_T$, where, for any generators $g \in \mathbb{G}, g' \in \mathbb{G}'$, $e(g, g') \neq 1$ and $e(g^a, g'^b) = e(g, g')^{ab}$. Then:

Definition 2. *The q-power Diffie Hellman (q-PDH) assumption holds for $\mathcal{G}$ if, for any NUPPT adversary $\mathcal{A}$ we have:*

$$\Pr[(p, \mathbb{G}, \mathbb{G}', \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\kappa); \; g \in_R \mathbb{G}^*; g' \in_R \mathbb{G}'^*; \; s \in_R \mathbb{Z}_p^*;$$
$$y \leftarrow \mathcal{A}(p, \mathbb{G}, \mathbb{G}', \mathbb{G}_T, e, \{g^{s^i}, g'^{s^i}\}_{i=1,\ldots,q,q+2,\ldots,2q}) : y = g^{s^{q+1}}] \approx_\kappa 0.$$

Definition 3. *The q-power knowledge of exponent (q-PKE) assumption holds for $\mathcal{G}$ and a class $\mathcal{Z}$ of auxiliary input generators if, for every NUPPT auxiliary input generator $Z \in \mathcal{Z}$ and any NUPPT adversary $\mathcal{A}$ there exists a NUPPT extractor $\mathcal{E}_\mathcal{A}$ such that:*

$$\Pr[crs := (p, \mathbb{G}, \mathbb{G}', \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\kappa); \; g \in_R \mathbb{G}^*; \; s \in_R \mathbb{Z}_p^*; \; z \leftarrow Z(crs, g, \ldots, g^{s^q});$$
$$g' \in_R \mathbb{G}'^*; (c, c' || a_0, \ldots, a_q) \leftarrow (\mathcal{A} || \mathcal{E}_\mathcal{A})(crs, \{g^{s^i}, g'^{s^i}\}_{i=0,\ldots,q}, z) :$$
$$e(c, g') = e(g, c') \; \wedge \; c \neq \prod_{i=0}^q g_1^{a_i s^i}] \approx_\kappa 0.$$

Here, $(a || b) \leftarrow (\mathcal{A} || \mathcal{E}_\mathcal{A})(c)$ denotes running both algorithms on the same inputs and random tape, and assigning their results to a respectively b. For certain auxiliary input generators, the q-PKE assumption does not hold, so we have to conjecture that our auxiliary input generators are "benign", cf. [DFGK14].

Definition 4. *The q-target group strong Diffie Hellman (q-SDH) assumption holds for $\mathcal{G}$ if, for any NUPPT adversary $\mathcal{A}$,*

$$\Pr[crs := (p, \mathbb{G}, \mathbb{G}', \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\kappa); g \in_R \mathbb{G}^*; g' \in_R \mathbb{G}'^*; s \in_R \mathbb{Z}_p$$
$$(r, Y) \leftarrow \mathcal{A}(crs, \{g^{s^i}, g'^{s^i}\}_{i=0,\ldots,q}) : r \in \mathbb{Z}_p \setminus \{s\} \wedge Y = e(g, g')^{\frac{1}{s-r}}] \approx_\kappa 0.$$

B Definitions

B.1 Extractable Trapdoor Commitment Family

We first define an extractable trapdoor commitment family. This definition is a straightforward generalisation of the standard notion of an extractable trapdoor commitment scheme (e.g., [Lip16]), in which we explicitly capture multiple commitment keys generated from the same CRS.

Definition 5. *Let* $(\mathsf{G0}, \mathsf{Gc}, \mathsf{C})$ *be a scheme where* $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G0}(1^\kappa)$ *outputs a system-wide CRS and a trapdoor;* $(\mathsf{ck}, \mathsf{ctd}) \leftarrow \mathsf{Gc}(\mathsf{crs})$ *outputs a commitment key and a trapdoor; and* $c \leftarrow \mathsf{C}_{\mathsf{ck}}(m; r)$ *outputs a commitment with the given key. Such a scheme is called an* extractable trapdoor commitment family *if:*

- *(Computationally binding) For every NUPPT committer* $\mathcal{A}$,

$$\Pr[(\mathsf{crs}, \cdot) \leftarrow \mathsf{G0}(1^\kappa); (\mathsf{ck}, \cdot) \leftarrow \mathsf{Gc}(\mathsf{crs}); (v; r; v'; r') \leftarrow \mathcal{A}(\mathsf{crs}; \mathsf{ck}) :$$
$$\mathsf{C}_{\mathsf{ck}}(v; r) = \mathsf{C}_{\mathsf{ck}}(v'; r')] \approx 0.$$

- *(Perfectly hiding) Letting* $(\mathsf{crs}, \cdot) \leftarrow \mathsf{G0}(1^\kappa)$; $(\mathsf{ck}, \cdot) \leftarrow \mathsf{Gc}(\mathsf{crs})$, *for all* v, v', $\mathsf{C}_{\mathsf{ck}}(v; r)$ *and* $\mathsf{C}_{\mathsf{ck}}(v'; r')$ *are identically distributed given random* r, r'
- *(Trapdoor) There exists a NUPPT algorithm* $\mathcal{T}$ *such that if* $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G0}(1^\kappa)$; $(\mathsf{ck}; \mathsf{ctd}) \leftarrow \mathsf{Gc}(\mathsf{crs})$; $(u; t) \leftarrow \mathcal{T}(\mathsf{crs}; \mathsf{td}; \mathsf{ck}; \mathsf{ck})$; $r \leftarrow \mathcal{T}(t; u; v)$, *then* u *is distributed identically to real commitments and* $\mathsf{C}_{\mathsf{ck}}(v; r) = u$.
- *(Extractable) For every NUPPT committer* $\mathcal{A}$, *there exists a NUPPT extractor* $\mathcal{E}_{\mathcal{A}}$ *such that*

$$\Pr[(\mathsf{crs}; \cdot) \leftarrow \mathsf{G0}(1^\kappa); (\mathsf{ck}; \cdot) \leftarrow \mathsf{Gc}(\mathsf{crs}); (u||v; r) \leftarrow (\mathcal{A}||\mathcal{E}_{\mathcal{A}})(\mathsf{crs}; \mathsf{ck}) :$$
$$u \in Range(\mathsf{C}_{\mathsf{ck}}) \ \wedge \ u \neq \mathsf{C}_{\mathsf{ck}}(v; r)] \approx 0.$$

B.2 Adaptive zk-SNARK

Given a relation $\mathcal{R}$ and a number of commitment keys $\mathsf{ck}_1, \ldots, \mathsf{ck}_n$ from the same commitment family, we define:

$$\mathcal{R}_{\mathsf{ck}_1, \ldots, \mathsf{ck}_n} := \{(\boldsymbol{u}; \boldsymbol{v}, \boldsymbol{r}, \boldsymbol{w}) : \boldsymbol{u_i} = \mathsf{C}_{\mathsf{ck}_i}(\boldsymbol{v_i}; \boldsymbol{r_i}) \wedge (\boldsymbol{v}; \boldsymbol{w}) \in \mathcal{R}\}.$$

Intuitively, an adaptive zk-SNARK is a zk-SNARK for relation $\mathcal{R}_{\mathsf{ck}_1, \ldots, \mathsf{ck}_n}$.

Definition 6. *An adaptive zk-SNARK for extractable trapdoor commitment family* $(\mathsf{G0}, \mathsf{Gc}, \mathsf{C})$ *and relation* $\mathcal{R}$ *is a scheme* $(\mathsf{G}, \mathsf{P}, \mathsf{V})$ *where:*

- $(\mathsf{crsp}; \mathsf{crsv}; \mathsf{tdp}) \leftarrow \mathsf{G}(\mathsf{crs}; \{\mathsf{ck}_i\})$, *given a CRS and commitment keys, outputs evaluation and verification keys, and a trapdoor;*
- $\pi \leftarrow \mathsf{P}(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsp}; \boldsymbol{u}; \boldsymbol{v}; \boldsymbol{r}; \boldsymbol{w})$, *given a CRS; commitment keys; an evaluation key; commitments; openings; and a witness, outputs a proof;*
- $0/1 \leftarrow \mathsf{V}(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsv}; \boldsymbol{u}; \pi)$, *given a CRS; commitment keys; a verification key; commentments; and a proof, verifies the proof,*

satisfying the following properties:

- Perfect completeness*:*

$$\Pr[(\mathsf{crs}; \cdot) \leftarrow \mathsf{G0}(1^\kappa); \forall i : (\mathsf{ck}_i; \cdot) \leftarrow \mathsf{Gc}(\mathsf{crs}); (\mathsf{crsp}; \mathsf{crsv}; \cdot) \leftarrow \mathsf{G}(\mathsf{crs}; \{\mathsf{ck}_i\});$$
$$(\boldsymbol{u}; \boldsymbol{v}; \boldsymbol{r}; \boldsymbol{w}) \leftarrow \mathcal{R}_{\{\mathsf{ck}_i\}} :$$
$$\mathsf{V}(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsv}; \boldsymbol{u}; \mathsf{P}(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsp}; \boldsymbol{u}; \boldsymbol{v}; \boldsymbol{r}; \boldsymbol{w})) = 1] = 1.$$

- Argument of knowledge*: for every NUPPT $\mathcal{A}$ there exists NUPPT extractor $\mathcal{E}_{\mathcal{A}}$ such that, for every auxiliary information* $\mathsf{aux} \in \{0,1\}^{poly(\kappa)}$:

$$\Pr[(\mathsf{crs};\mathsf{td}) \leftarrow \mathsf{G0}(1^\kappa); \forall i : (\mathsf{ck}_i;\mathsf{ctd}_i) \leftarrow \mathsf{Gc}(\mathsf{crs}); (\mathsf{crsp};\mathsf{crsv};\mathsf{tdp}) \leftarrow \mathsf{G}(\mathsf{crs}, \{\mathsf{ck}_i\});$$
$$(\boldsymbol{u};\pi||\boldsymbol{v};\boldsymbol{r};\boldsymbol{w}) \leftarrow (\mathcal{A}||\mathcal{E}_{\mathcal{A}})(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsp}; \mathsf{aux}||\ldots; \mathsf{td}; \mathsf{ctd}_1; \ldots; \mathsf{ctd}_n; \mathsf{tdp}) :$$
$$(\boldsymbol{u};\boldsymbol{v};\boldsymbol{r};\boldsymbol{w}) \notin \mathcal{R}_{\{\mathsf{ck}_i\}} \wedge \mathsf{V}(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsv}; \boldsymbol{u};\pi) = 1] \approx_\kappa 0.$$

  Here, $(\mathcal{A}||\mathcal{E}_{\mathcal{A}})(\cdot||\ldots;\cdot')$ is parallel execution as above; $\mathcal{E}_{\mathcal{A}}$ has extra input $\cdot'$.
- Witness indistinguishability*: if $(\mathsf{crs};\cdot) \leftarrow \mathsf{G0}(1^\kappa)$; $\forall i : (\mathsf{ck}_i;\cdot) \leftarrow \mathsf{Gc}(\mathsf{crs})$; $(\mathsf{crsp};\mathsf{crsv};\cdot) \leftarrow \mathsf{G}(\mathsf{crs}, \{\mathsf{ck}_i\})$ and $(\boldsymbol{u};\boldsymbol{v},\boldsymbol{r},\boldsymbol{w}),(\boldsymbol{u};\boldsymbol{v}',\boldsymbol{r}',\boldsymbol{w}') \in \mathcal{R}_{\{\mathsf{ck}_i\}}$ with $\boldsymbol{r},\boldsymbol{r}' \in_R \mathfrak{R}$, then, as probability distributions,*

$$\mathsf{P}(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsp}; \boldsymbol{u};\boldsymbol{v};\boldsymbol{r};\boldsymbol{w}) \approx_\kappa \mathsf{P}(\mathsf{crs}; \{\mathsf{ck}_i\}; \mathsf{crsp}; \boldsymbol{u};\boldsymbol{v}';\boldsymbol{r}';\boldsymbol{w}').$$

- Perfectly composable zero knowledge*: there exists a PPT simulator $\mathcal{S}$ such that, for all stateful NUPPT adversaries $\mathcal{A}$,*

$$\Pr[(\mathsf{crs}, \cdot) \leftarrow \mathsf{G0}(1^\kappa); \forall i : (\mathsf{ck}_i, \cdot) \leftarrow \mathsf{Gc}(\mathsf{crs}); (\mathsf{crsp}, \mathsf{crsv}, \cdot) \leftarrow \mathsf{G}(\mathsf{crs}, \{\mathsf{ck}_i\}),$$
$$(\boldsymbol{u};\boldsymbol{v};\boldsymbol{r};\boldsymbol{w}) \leftarrow \mathcal{A}(\mathsf{crs}, \{\mathsf{ck}_i\}, \mathsf{crsp}); \pi \leftarrow \mathsf{P}(\mathsf{crs}, \{\mathsf{ck}_i\}, \mathsf{crsp}; \boldsymbol{u};\boldsymbol{v};\boldsymbol{r};\boldsymbol{w}) :$$
$$(\boldsymbol{u},\boldsymbol{v},\boldsymbol{r},\boldsymbol{w}) \in \mathcal{R}_{\{\mathsf{ck}_i\}} \wedge \mathcal{A}(\pi) = 1] =$$
$$\Pr[(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G0}(1^\kappa); \forall i : (\mathsf{ck}_i, \mathsf{ctd}_i) \leftarrow \mathsf{Gc}(\mathsf{crs}); (\mathsf{crsp}, \mathsf{crsv}, \mathsf{tdp}) \leftarrow \mathsf{G}(\mathsf{crs}, \{\mathsf{ck}_i\}),$$
$$(\boldsymbol{u};\boldsymbol{v};\boldsymbol{r};\boldsymbol{w}) \leftarrow \mathcal{A}(\mathsf{crs}, \{\mathsf{ck}_i\}, \mathsf{crsp}); \pi \leftarrow \mathcal{S}(\mathsf{crs}, \{\mathsf{ck}_i\}, \mathsf{crsp}; \boldsymbol{u}; \mathsf{td}, \{\mathsf{ctd}_i\}, \mathsf{tdp}) :$$
$$(\boldsymbol{u},\boldsymbol{v},\boldsymbol{r},\boldsymbol{w}) \in \mathcal{R}_{\{\mathsf{ck}_i\}} \wedge \mathcal{A}(\pi) = 1]$$

Our definition differs from the one in [Lip16] on a few minor points:

- We generalize the definition to use a commitment family instead of a commitment scheme. This is because, in Adaptive Trinocchio, we need every party to provide inputs using a different commitment key to ensure that inputs are chosen independently. (Note that this is a generalization of the original definition. Alternatively, as in [SVdV15], an additional input commitment round could be used. In this case, the generalization is not needed.)
- In our definition, we have included a witness that is not committed to using the computation-independent commitment scheme. As a consequence, the witness does not have to be committed to, which makes the solution slightly more efficient. Again, this generalization could be avoided by letting Adaptive Trinocchio produce an additional commitment to the witness.
- In our definition, the extractor gets access to the trapdoor. Also, we guarantee statistical witness indistinguishability as opposed to perfect witness indistinguishability. Both are needed because of the use of Pinocchio [GGPR13].

B.3 Execution Model for Adaptive Function Evaluation

In this section, we provide more details on the ideal-world execution model for the ideal functionalities from Figure 2, and the real-world execution model for the

protocol from Figure 3. Our execution model is based on Canetti's traditional standalone execution model [Can98], of which we now give an overview. We assume static, non-uniform probabilistic polynomial time (NUPPT) adversaries.

A *distribution ensemble* $X = \{X(\lambda, a)\}_{\lambda \in \mathbb{N}, a \in D}$ is an infinite sequence of probability distributions, indexed over a security parameter λ and inputs a from a given domain D.

Definition 7 (Computational indistinguishability ([Can98])). *Let $\delta : \mathbb{N} \to \{0, 1\}$. We say that distribution ensembles X and Y have* computational distance at most δ *if for every algorithm $\mathcal{D}$ that is probabilistic polynomial-time in its first input, for all sufficiently large λ, all a and all auxiliary information $w \in \{0, 1\}^*$ we have:*

$$\left| Prob(\mathcal{D}(1^\lambda, a, w, x) = 1) - Prob(\mathcal{D}(1^\lambda, a, w, y) = 1) \right| < \delta(\lambda),$$

where x is chosen from distribution $X(\lambda, a)$, y is chosen from distribution $Y(\lambda, a)$, and the probabilities are taken over the choices of x, y, and the random choices of $\mathcal{D}$. If ensembles X and Y have computational distance at most λ^{-c} for all $c > 0$ then we say that X and Y are computationally indistinguishable.

At a high level, a protocol is said to "implement" an ideal functionality if real-life executions of the protocol and ideal-world executions of the ideal functionality are computationally indistinguishable.

In the real-life model, parties P_i perform protocol π together with adversary $\mathcal{A}$. Parties are NUPPT Turing machines that receive input x_i. The adversary is a NUPPT machine that receives as input the identities of the corrupted parties and their inputs; an auxiliary input; and a value λ for the security parameter. Computation takes place in a fully connected, ideally authenticated, synchronous network with rushing. Moreover, we assume access to a secure bulletin board, i.e., with guaranteed and authenticated delivery. The adversary sees all communication of passively corrupted parties and controls all communication of actively corrupted parties. In the $(g_1, \ldots, g_k)$-hybrid model, parties have access to an incorruptible trusted party that evaluates the functions $g_1, \ldots, g_k$. At the end of the computation, uncorrupted parties output whatever is specified in the protocol, corrupted parties output $\perp$, and the adversary outputs an arbitrary function over its view. We write $(y_1, \ldots, y_m, a) = \text{EXEC}_{\pi, \mathcal{A}}(\lambda; x_1, \ldots, x_m; z)$ for the random variable consisting of the outputs y_i of parties P_i and output a of the adversary, and $\text{EXEC}_{\pi, \mathcal{A}}$ for the distribution ensemble $\{\text{EXEC}_{\pi, \mathcal{A}}(\lambda; x_1, \ldots, x_m; z)_{\lambda \in \mathbb{N}; x_1, \ldots, x_m, z \in \{0, 1\}^*}$. When working in the $(g_1, \ldots, g_k)$-hybrid model, we write $\text{EXEC}_{\pi, \mathcal{A}}^{(g_1, \ldots, g_k)}(\lambda; x_1, \ldots, x_m; z)$ and $\text{EXEC}_{\pi, \mathcal{A}}^{(g_1, \ldots, g_k)}$, respectively.

In the ideal-world model, parties P_i and adversary $\mathcal{S}$ interact with an incorruptible trusted party. As above, P_i and adversary $\mathcal{A}$ are NUPPT machines; P_i have input x_i and the adversary has input the identities of the corrupted parties and their inputs; an auxiliary input; and the security parameter λ. The ideal process then proceeds as indicated by the ideal functionality $\mathcal{F}$. We write

$(y_1, \ldots, y_m, a) = \text{IDEAL}_{\mathcal{F},\mathcal{S}}(\lambda; x_1, \ldots, x_m; z)$ for the random variable consisting of the outputs y_i of parties P_i and output a of the adversary, and $\text{IDEAL}_{\mathcal{F},\mathcal{A}}$ for the distribution ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{A}}(\lambda; x_1, \ldots, x_m; z)\}_{\lambda \in \mathbb{N}; x_1, \ldots, x_m, z \in \{0,1\}^*}$.

Definition 8. *Protocol π implements ideal functionality $\mathcal{F}$ (in the $(g_1, \ldots, g_l)$-hybrid model) if for every NUPPT adversary $\mathcal{A}$ there exists a NUPPT simulator $\mathcal{S}$ such that the distribution ensembles $\text{EXEC}_{\pi,\mathcal{A}}$ (resp. $\text{Exec}_{\pi,\mathcal{A}}^{(g_1, \ldots, g_l)}$) and $\text{IDEAL}_{\mathcal{F},\mathcal{S}}$ are computationally indistinguishable.*

C   Security Proofs for our zk-SNARKs

We now prove that the constructions from Section 3 are adaptive zk-SNARKs.

Proof (of Theorem 1). We show each of the properties in turn:

Perfect completeness By inspection.

Argument of knowledge Our proof follows the structure of [PHGR13]. Let $\mathcal{A}$ be a NUPPT algorithm. We construct extractor $\mathcal{E}_{\mathcal{A}}$ as follows. Suppose $\mathcal{A}$ returns a verifying proof, consisting of a sequence of tuples:

$$(\langle C_i \rangle_1, \langle \alpha_i C_i \rangle_2, \langle V_i \rangle_1, \langle \alpha_v V_i \rangle_2, \langle W_i \rangle_2, \langle \alpha_w W_i \rangle_1, \langle Y_i \rangle_1, \langle \alpha_y Y_i \rangle_2, \langle Z_i \rangle_1$$

and a proof element $\langle H \rangle_1$.

Extractor $\mathcal{E}_{\mathcal{A}}$ runs the extractor of the commitment family (i.e., the d-PKE extractor) to obtain, for each i, openings $\boldsymbol{v_i}, r_i$ of commitment $\langle C_i \rangle_1, \langle \alpha_i C_i \rangle_2$. (Indeed, set-up of the extractable commitment scheme family is exactly ths set-up of the d-PKE assumption; and key generation of the commitment scheme family and zk-SNARK together form the auxiliary input generator.) Note that, letting $c_i(x) := r_i + \boldsymbol{v_i}_{,1} x + \ldots + \boldsymbol{v_i}_{,d} x^d$, we have $\langle C_i \rangle_1 = \langle c_i(x) \rangle_1$.

At the same time, we can use the d-PKE extractor to extract polynomials $v_i(x)$, $w_i(x)$, and $y_i(x)$ of degree at most d such that $\langle V_i \rangle_1 = \langle r_v \cdot v_i(x) \rangle_1$, $\langle W_i \rangle_2 = \langle r_w \cdot w_i(x) \rangle_2$, $\langle Y_i \rangle_1 = \langle r_y \cdot y_i(x) \rangle_1$. (E.g., for $\langle V_i \rangle_1$, the q-PKE set-up generates $\langle r_v \rangle_1$ and random s; the auxiliary input generator uses this to perform set-up and key generation of the commitment famly, and key generation not involving $\langle r_v x^i \rangle_1$; the d-PKE adversary uses $\{\langle r_v x^i \rangle_1, \langle \alpha_v r_v x^i \rangle_2\}$ to generate the appropriate evauation key elements, runs $\mathcal{A}$, and returns $\langle V_i \rangle_1, \langle \alpha_v V_i \rangle_2$.)

Next, the extractor computes $r_v' := r_v(s^{d+1})^{-1}$, $r_w' := r_w(s^{2d+2})^{-1}$, $r_y' := r_y(s^{3d+2})^{-1}$, and determines $P_i(x) = c_i(x) + r_v' x^{d+1} v_i(x) + r_w' x^{2d+2} w_i(x) + r_y' x^{3d+3} y_i(x)$. In the QAP solution, let $W+1, \ldots, W+N$ be the indices of the witness. Let $p_j(x) = r_v' x^{d+1} v_j(x) + r_w' x^{2d+2} w_j(x) + r_y' x^{3d+3} y_j(x)$. The extractor finds $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N, \delta_{v,1}, \delta_{w,1}, \delta_{y,1}$ such that

$$P_1(x) = r_1 + \sum_{j=1}^{d} \boldsymbol{v_1}_{,j} \cdot (x^j + p_j(x)) + \sum_{j=1}^{N} \boldsymbol{w_j} p_{W+j}(x) +$$
$$\delta_{v,1} r_v' x^{d+1} t + \delta_{w,1} r_w' x^{2d+2} t + \delta_{y,1} r_y' x^{3d+3} t,$$

otherwise it fails. For all other i, the extractor finds $\delta_{v,i}, \delta_{w,i}, \delta_{y,i}$ such that

$$P_i(x) = r_i + \sum_{j=1}^{d} \boldsymbol{v_{i,j}} \cdot (x^j + p_{(i-1)d+j}(x)),$$

otherwise it fails. Finally, the extractor returns $\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}; r_1, \ldots, r_n; \boldsymbol{w}$.

We need to show that, if $\mathcal{A}$ produces a verifying proof, the extractor fails only with negligible probability and the returned values are in $\mathcal{R}$.

So suppose the extractor fails, i.e., it cannot write polynomial $P_i(x)$ as the linear combination specified above, with nonneglible probability. We construct an adversary $\mathcal{B}$ for the $(4d+3)$-PDH assumption as follows. $\mathcal{B}$ receives a $(4d+3)$-PDH challenge, i.e., values $\{\langle x^i \rangle_1, \langle x^i \rangle_2\}_{i=0,1,\ldots,4d+3,4d+5,\ldots,8d+6}$. $\mathcal{B}$ performs $\mathsf{G0}, \mathsf{Gc}$ as usual using $\{\langle x^i \rangle_1, \langle x^i \rangle_2\}_{i=0,\ldots,d}$. For G, rather than picking r_v, r_w at random, it picks $r_v', r_w', r_y' = r_v' r_w'$ at random and defines $r_v(x) = r_v' x^{d+1}$, $r_w(x) = r_w' x^{2d+2}$, $r_y(x) = r_y' x^{3d+3}$ implicitly. Instead of generating β_i at random, it generates it as follows. Let $\mathcal{P}$ be the collection of polynomials that P_i should be a linear combination of. That is, for $i = 1$, take polynomials 1, $x^j + p_j(x)$, $p_{W+j}(x)$, $r_v' x^{d+1} t(x)$, $r_w' x^{2d+2} t(x)$, and $r_y' x^{3d+3} t(x)$; for other j, take 1, $x^j + p_j(x)$, $r_v' x^{d+1} t(x)$, $r_w' x^{2d+2} t(x)$, and $r_y' x^{3d+3} t(x)$. Choose polynomial $\beta_i'(x)$ of degree $\leq 4d+3$ uniformly at random such that, for every $p(x) \in \mathcal{P}$, $\beta_i'(x) \cdot p(x)$ has zero coefficient at $4d+4$. Let $\beta_i = \beta_i'(s)$. Observe that, by construction, the PDH adversary can generate all needed terms without ever knowing s.

Now, $\mathcal{B}$ performs the argument of knowledge experiment with extractor $\mathcal{E_A}$. Using the d-PKE extractor, it obtains polynomial $P_i(x)$ satisfying the $(\mathbf{Z})$-check, so $\langle P(x) \cdot \beta_i'(x) \rangle_1 = \langle Z_i \rangle_1$, but with nonneglible probability, $P_i \notin \text{span}(\mathcal{P})$. In looking at how $\mathcal{B}$ selected $\beta_i'(x) = \beta^0 + \beta^1 x + \ldots + \beta^{4d+3} x^{4d+3}$, observe that for every nonconstant $p \in \mathcal{P}$, the zeroness requirement on the $(4d+4)$th cofficient of $\beta_i'(x)$ induces a linear constraint on coefficients $\beta^1, \ldots, \beta^{4d+3}$. Let B be the vector space of $\beta^1, \ldots, \beta^{4d+3}$ satisfying these constraints. A priori this vector space has dimension $4d+3$, and every linearly independent non-constant polynomial in $\mathcal{P}$ reduces this dimension by one. Now consider the subspace $B' \subset B$ of vectors $\beta^1, \ldots, \beta^{4d+3}$ for which additionally $P_i(x) \cdot \beta_i(x)$ has zero coefficient at $4d+4$. Since $P_i \notin \text{span}(\mathcal{P})$, this induces a new constraint, so $\dim B' = \dim B - 1$. Now, a random polynomial $\beta_i'(x)$ can be chosen by choosing β^0 uniformly random from $\mathbb{F}$ and $\boldsymbol{\beta} = \beta^1, \ldots, \beta^{4d+3}$ uniformly random from B. Note that, from the point of view of the $\mathcal{A}$, every choice from B is equally likely. This is because $\mathcal{A}$ only sees $\beta_i(s)$ which includes the uniformly random constant term β^0. So, the chance that $\beta_i'(x)$ lies in the set B' defined by the adversary's $p(x)$ is equal to the chance that any random $\boldsymbol{b}$ lies in B', i.e., $1/|\mathbb{F}|$. So, with probability $1 - 1/|\mathbb{F}|$, the coefficient of $P_i(x) \cdot \beta_i'(x)$ at $4d+4$ is nonzero. But in this case, from $\langle Z_j \rangle_1 = \langle P_i(x) \cdot \beta_i(x) \rangle_1$ and $\langle x^j \rangle_1$ $(j \neq 4d+4)$, $\mathcal{B}$ can compute $\langle x^{4d+4} \rangle_1$. This contradicts the $(4d+3)$-PDH assumption, so in fact, the extractor fails only with negligible probability.

It remains to show that the values returned by the extractor are in $\mathcal{R}$. By the properties of the PKE-extractor we know that $\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}, r_1, \ldots, r_n$ correctly open the given commitments; it remains to show that $\boldsymbol{x} := (\boldsymbol{v_1}; \ldots; \boldsymbol{v_n}; \boldsymbol{w}) \in \mathcal{R}$. Suppose this is not the case, then we build an adversary to the $(8d+6)$-SDH

assumption as follows. Given a $(8d+6)$-SDH challenge $\{\langle x^i \rangle_1, \langle x^i \rangle_2\}_{i=0,1,\ldots,8d+6}$ it uses this challenge to perform key generation (again generating r_v, r_w, r_y implicitly as polynomials), and with these keys performs the argument of knowledge experiment. Suppose the experiment returns values such that $\boldsymbol{x}$ is not in $\mathcal{R}$. Let $V'(x) = \sum_{j=1}^{W+N} \boldsymbol{x_i} \cdot v_j(x) + (\sum_i \delta_{v,i}) \cdot t(x)$, $W'(x) = \sum_{j=1}^{W+N} \boldsymbol{x_i} \cdot w_j(x) + (\sum_i \delta_{w,i}) \cdot t(x)$, $Y'(x) = \sum_{j=1}^{W+N} \boldsymbol{x_i} \cdot y_j(x) + (\sum_i \delta_{y,i}) \cdot t(x)$. Since $\boldsymbol{x} \notin \mathcal{R}$, by definition of $v_j(x)$, $w_j(x)$, and $y_j(x)$, we have that $t(x)$ does not divide $p(x) = V'(x)W'(x) - Y'(x)$. Let $(x-r)$ be a monomial dividing $t(x)$ but not $p(x)$. The adversary uses the extended Euclidean algorithm to compute $d(x) = \gcd(p(x), t(x))$, $a(x)$ of degree $\leq 2d-1$ and $b(x)$ of degree $\leq d-1$ such that $a(x)t(x) + b(x)p(x) = d(x)$. Multiplying the left-hand side and right-hand side by $1/((x-r)d(x))$ and re-arranging, we have that

$$a(x) \cdot \frac{t(x)}{(x-r)d(x)} + b(x) \cdot \frac{t(x)}{(x-r)d(x)} \cdot \frac{p(x)}{t(x)} = \frac{1}{x-r}.$$

Note that $(x-r)d(x)$ dividies $t(x)$, so $t(x)/((x-r)d(x))$ is a polynomial. Hence, the adversary can evaluate $\langle a(x) \cdot \frac{t(x)}{(x-r)d(x)} \rangle_1$ and hence $e(g,g)^{a(s) \cdot \frac{t(s)}{(s-r)d(s)}}$. For the same reason, the adversary can compute $\langle b(x) \cdot \frac{t(x)}{(x-r)d(x)} \rangle_1$. Translated to the present situation, the proof check $(\mathbf{H})$ states that:

$$e(\langle r_v V'(x) \rangle_1, \langle r_w W'(x) \rangle_2) = e(\langle H \rangle_1, \langle r_v r_w t \rangle_2) \cdot e(\langle r_v r_w Y'(x) \rangle_1, \langle 1 \rangle_2).$$

By the properties of the pairing, this implies that $g^{r_v V'(s) \cdot r_w W'(s) - r_v r_w Y'(s)} = \langle H \rangle_1^{r_v r_w t(s)}$, so $\langle H \rangle_1 = g^{(V'(s)W'(s) - Y'(s))t(s)^{-1}} = \langle p(x)/t(x) \rangle_1$. Hence, the adversary can also evaluate $e(g,g)^{b(s) \cdot \frac{t(s)}{(s-r)d(s)} \cdot \frac{p(s)}{t(s)}}$. But then, as established above, the adversary can evaluate $e(g,g)^{\frac{1}{s-r}}$. This contradicts the $(8d+6)$-SDH assumption, so in fact the extractor returns values in $\mathcal{R}$.

Perfectly composable zero-knowledgeness The existence and correctness of our simulator $\mathcal{S}$ follow the analogous result from [GGPR13].

Suppose the simulator is given commitments $\boldsymbol{u_i} = (\langle C_i \rangle_1, \langle \alpha_c C_i \rangle_2)$. The simulator generates, for each i, random $\delta_{v,i}, \delta_{w,i}, \delta_{y,i}$, and sets $\langle V_i \rangle_1 = \langle r_v \delta_{v,i} t \rangle_1$, $\langle W_i \rangle_2 = \langle r_w \delta_{w,i} t \rangle_2$, $\langle Y_i \rangle_1 = \langle r_y \delta_{y,i} t \rangle_1$, which is statistically identical to the real $\langle V_i \rangle_1$, $\langle W_i \rangle_2$, $\langle Y_i \rangle_1$ as long as $t(s) \neq 0$. Note that verification relations $(\mathbf{V})$–$(\mathbf{H})$ now fix the remaining proof elements, and moreover, the simulator can generate remaining proof elements satisfying these relations. Namely, it computes $\langle \alpha_v V_i \rangle_2 = \langle \alpha_v r_v \delta_{v,i} t \rangle_2$, $\langle \alpha_w W_i \rangle_1 = \langle \alpha_w r_w \delta_{w,i} t \rangle_1$, $\langle \alpha_y Y_i \rangle_2 = \langle \alpha_y r_y \delta_{y,i} t \rangle_2$; $\langle Z_i \rangle_1 = \langle \beta_i \cdot (r_v \delta_{v,i} t + r_w \delta_{w,i} + r_y \delta_{y,i} t) \rangle_1 + \beta_i \cdot \langle C_i \rangle_1$, and $\langle H \rangle_1 = \langle \delta_{v,i} \delta_{w,i} - \delta_{y,i} \rangle_1$. One verifies that these elements indeed satisfy the verification relations. Since $\langle V_i \rangle_1$, $\langle W_i \rangle_2$, $\langle Y_i \rangle_1$ are distributed as in the real proof and the other proof elements follow from them (so also have the same distribution), the simulator has produced a proof that is indistinguishable from real, as required.

Witness indistinguishability Follows from our proof for perfectly composable zero-knowledgeness: both the left-hand side and the right-hand side of the equation are statistically indistinguishable from the simulated distribution. $\qquad\square$

Extractable Trapdoor Commitment Scheme Family $(\mathsf{G0}^2, \mathsf{Gc}^2, \mathsf{C}^2)$:

- $\mathsf{G0}^2$: Fix $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ and random s, r_c. Return $\mathsf{crs} = (\{\langle x^i\rangle_1, \langle x^i\rangle_2\}_{i=0,\ldots,d},$ $\langle r_c\rangle_1, \langle r_c x\rangle_1, \ldots, \langle r_c x^d\rangle_1), \mathsf{td} = (s, r_c)$.
- Gc^2: For $\alpha \in_R \mathbb{F}$, $\mathsf{ck} = (\langle r_c\rangle_1, \langle r_c x\rangle_1, \ldots, \langle r_c x^d\rangle_1, \langle \alpha r_c\rangle_2, \langle \alpha r_c x\rangle_2, \ldots, \langle \alpha r_c x^d\rangle_2)$
- C^2: Return $(r\langle r_c\rangle_1 + \boldsymbol{v}_1\langle r_c x\rangle_1 + \boldsymbol{v}_2\langle r_c x^2\rangle_1 + \ldots, r\langle \alpha r_c\rangle_2 + \boldsymbol{v}_1\langle \alpha r_c x^2\rangle_2 + \ldots)$

Key generation G^2: Generate random α_c, let $\mathsf{ck}_c = (\langle 1\rangle_1, \langle \alpha_c\rangle_2, \langle x\rangle_1, \langle \alpha_c x\rangle_2, \ldots,$ $\ldots, \langle x^d\rangle_1, \langle \alpha_c x^d\rangle_2)$. Generate $(\mathsf{crsp}, \mathsf{crsv}, \mathsf{tdp}) \leftarrow \mathsf{G}^1(\mathsf{crs}; \mathsf{ck}_c)$. Generate random $\beta'_1, \ldots, \beta'_n$. Return evaluation key $(i = 1, \ldots, n, j = 1, \ldots, \ell_i)$:

$$(\mathsf{ck}_c, \mathsf{crsp}, \langle \beta'_i r_c\rangle_1, \langle \beta'_i\rangle_1, \langle \beta'_i \cdot (r_c x + x^{\ell_1 + \ldots + \ell_{i-1} + j})\rangle_1)$$

and verification key $(i = 1, \ldots, n)$: $(\mathsf{crsv}, \langle \alpha_c\rangle_2, \langle \beta'_i\rangle_2)$

Proof generation P^2: Let $\boldsymbol{u}_i = \mathsf{C}_{\mathsf{ck}_i}(\boldsymbol{v}_i; r_i)$ and let $\boldsymbol{w}$ be the witness such that $\boldsymbol{v} := (\boldsymbol{v}_{1,1}, \ldots, \boldsymbol{v}_{1,\ell_1}; \ldots; \boldsymbol{v}_{1,1}, \ldots, \boldsymbol{v}_{1,\ell_n}; \boldsymbol{w})$ is a solution to the QAP. Generate random $r'_1, \ldots, r'_n$ and build commitments and correspondence proofs $(i = 1, \ldots, n)$:

$$\boldsymbol{u}'_i = (r'_i\langle 1\rangle_1 + \sum_{j=1}^{\ell_i} \boldsymbol{v}_{i,j}\langle x^{\ell_1 + \ldots + \ell_{i-1} + j}\rangle_1, r'_i\langle \alpha_c\rangle_2 + \sum_{j=1}^{\ell_i} \boldsymbol{v}_{i,j}\langle \alpha_c x^{\ell_1 + \ldots + \ell_{i-1} + j}\rangle_2)$$

$$\langle Z'_i\rangle_1 = (r_i\langle \beta'_i r_c\rangle_1 + r'_i\langle \beta'_i\rangle_1 + \sum_{j=1}^{\ell_i} \boldsymbol{v}_{2,j}\langle \beta'_i \cdot (r_c x + x^{\ell_1 + \ldots + \ell_{i-1} + j})\rangle)_1$$

Call P_1 on $(\boldsymbol{u}'_1 + \ldots + \boldsymbol{u}'_n; \boldsymbol{v}; r'_1 + \ldots + r'_n; \boldsymbol{w})$ and return along with the $\boldsymbol{u}'_i, \langle Z'_i\rangle_1$.

Proof verification V^2: Letting $\mathsf{ck}_i = (\ldots, \langle \alpha_i\rangle_2)$, $\boldsymbol{u}_i = (\langle C_i\rangle_1, \langle \alpha_i C_i\rangle_2)$, $\boldsymbol{u}'_i = (\langle C'_i\rangle_1, \langle \alpha_c C'_i\rangle_2)$ check that $e(\langle C_i\rangle_1, \langle \alpha_i\rangle_2) = e(\langle 1\rangle_1, \langle \alpha_i C_i\rangle_1)$, $e(\langle C'_i\rangle_1, \langle \alpha_c\rangle_2) = e(\langle 1\rangle_1, \langle \alpha_c C'_i\rangle_2), e(\langle C_i\rangle_1 + \langle C'_i\rangle_1, \langle \beta'_i\rangle_2) = e(\langle Z'_i\rangle_1, \langle 1\rangle_2)$. Call V_1 on the supplied proof with respect to commitment $\boldsymbol{u}'_1 + \ldots + \boldsymbol{u}'_n$.

Fig. 4. Pinocchio-Based Adaptive zk-SNARK $(\mathsf{G}^2, \mathsf{P}^2, \mathsf{V}^2)$

In Figure 4 we give the details of zk-SNARK $(\mathsf{G}^2, \mathsf{P}^2, \mathsf{V}^2)$, the second optimization presented in from Section 3.2.

Theorem 3. *Under the $(4d + 3)$-PDH, d-PKE, and $(8d + 6)$-SDH assumptions (Section A), where d is the maximal degree of the QAPs used, $(\mathsf{G}^2, \mathsf{P}^2, \mathsf{V}^2)$ is an adaptive zk-SNARK.*

Proof. We show each of the properties in turn:

Perfect completeness By inspection.

Argument of knowledge Suppose $\mathcal{A}$ returns commitments $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$, where $\boldsymbol{u}_i = (\langle C_i\rangle_1, \langle \alpha_i C_i\rangle_2)$, and verifying proof $\pi = (\langle C'_i\rangle_1, \langle \alpha_c C'_i\rangle_2, \langle Z'_i\rangle_1, \pi')$. Extractor $\mathcal{E}_{\mathcal{A}}$ works as follows. Analogously to above, it uses the d-PKE extractor, it extracts openings $\boldsymbol{v}_i, r_i$ to commitments $\boldsymbol{u}_i$ (so $\langle C_i\rangle_1 = \langle r_c \cdot c_i(x)\rangle_1$, where $c_i(x) = r_i + \boldsymbol{v}_{i,1} \cdot x + \ldots$). It also extracts openings $\boldsymbol{v}'_i, r'_i$ to intermediate commitments $\boldsymbol{u}'_i = (\langle C'_i\rangle_1, \langle \alpha_c C'_i\rangle_2)$ (so $\langle C'_i\rangle_1 = \langle c'_i(x)\rangle_1$, where $c'_i(x) = r'_i + \boldsymbol{v}_{i,1}' x + \ldots$). Using the extractor of the zk-SNARK from the preceding construction, it further obtains

opening v', r' of the intermediate commitment $u' := u'_1 + \ldots + u'_n$ and witness w. (Note that $\mathsf{G0}^2$ consists of performing setup $\mathsf{G0}^1$ and generating additional CRS elements based on its output.) If any v'_i is incorrect, i.e., it does not consists of $\ell_1 + \ldots + \ell_{i-1}$ zeros, followed by $v_{i,1}, \ldots, v_{i,\ell_i}$, followed by zeros, it fails. If v' is not equal to $v'_1 + \ldots + v'_n$, it also fails. Otherwise it returns $(v_1, \ldots, v_n; r_1, \ldots, r_n; w)$.

We need to show that, if $\mathcal{A}$ produces a verifying proof, the extractor fails only with negligible probability and the returned values are in $\mathcal{R}$. By the argument of knowledge property of the used zk-SNARK, we know that except with negligible probability, v', r' correctly open u', and (v', w) are in the relation of the QAP. By the properties of the d-PKE extractor, the returned $v_1, \ldots, v_n; r_1, \ldots, r_n$ are also openings to the commitments $u_1, \ldots, u_n$. Moreover, if the extractor does not fail, $v_1, \ldots, v_n$ and $v'_1 + \ldots + v'_n$ relate to each other in the way intended by the computation, hence the returned values are in $\mathcal{R}$. So we are only left to show that the extractor fails with negligible probability.

First, suppose the extractor fails because v' is not equal to $v'_1 + \ldots + v'_n$. Note that (v', r') and $(v'_1 + \ldots + v'_n, r'_1 + \cdots + r'_n)$ are two different openings to commitment u'. But this contradicts the binding property of the intermediate commitment scheme. Indeed, given a $(4d + 3)$-PDH challenge, we can run the above extractor to obtain, with nonneglible probability, two openings to commitment u', from which by the argument of [Gro10] we can compute $\langle x^{4d+3} \rangle_1$, contradicting the $(4d + 3)$-PDH assumption. The other possibility is that the extractor fails because any v'_i was incorrectly formed. If this is the case, then we use this to construct an adversary $\mathcal{A}$ to the $(4d+3)$-PDH assumption. Given a $(4d+3)$-PDH challenge $\{\langle x^i \rangle_1, \langle x^i \rangle_2\}_{i=0,1,\ldots,4d+3,4d+5,\ldots,8d+6}$, $\mathcal{A}$ runs the above extractor, but in doing so chooses β'_i implicitly as a polynomial such that, for every polynomial $p_j(x) := r'_c x^{d+1+j} + x^{\ell_1 + \ldots + \ell_{l-1} + j}$, $\beta'_i(x) \cdot p_j(x)$ has zero coefficient at $4d + 4$. Since the extractor fails, by assumption the polynomial $P_i(x) = r_c c_i(x) + c'_i(x)$ does not lie in the span of the $\{p_j(x)\}$. Following exactly the same reasoning as in the proof for the argument of knowledge property of our previous construction, with probability $1 - 1/\mathbb{F}$, this implies that $P_i(x) \cdot \beta'_i(x)$ has nonzero coefficient at $4d + 4$, enabling $\mathcal{A}$ to compute $\langle x^{4d+4} \rangle_1$ and contradict $(4d + 3)$-PDH.

Perfectly composable zero-knowledgeness Our simulator samples random r'_i, lets $\langle C'_i \rangle_1 = \langle r'_i \rangle_1$, $\langle \alpha_c C'_i \rangle_2 = \langle \alpha_c r'_i \rangle_2$, and $\langle Z_i \rangle_1 = \beta'_i \cdot (\langle C_i \rangle_1 + \langle C'_i \rangle_1)$, and then uses the simulator for the above zk-SNARK with commitment $(\langle C'_1 \rangle_1 + \ldots + \langle C'_n \rangle_1, \langle \alpha_c C'_1 \rangle_2 + \ldots + \langle \alpha_c C'_n \rangle_2)$. These proofs clearly have the correct distribution.

Witness indistinguishability Follows from our proof for perfectly composable zero-knowledgeleness: both the left-hand side and the right-hand side of the equation are statistically indistinguishable from the simulate distribution. $\square$

D Security Proof for Adaptive Trinocchio

D.1 Correct Function Evaluation

Figure 5 shows the simulator used to prove that our protocol implements correct function evaluation.

Simulator: Adaptive Trinocchio ("correct" case)

(Given NUPPT adversary $\mathcal{A}$ actively corrupting all workers and subset $\mathcal{C} \cap \mathcal{I}$ of inputters $\mathcal{I}$.)

1. Perform CRS and key generation of the commitment scheme, keeping trapdoors $\mathsf{crs}, \mathsf{ctd}_1, \ldots, \mathsf{ctd}_n, \mathsf{ctd}_o$. Simulate step 1 of the protocol with respect to $\mathcal{A}$ by sending this material.

2. Simulate step 2 of the protocol: provide trapdoor commitments c_i for the honest parties, and use the extractor of the commitment scheme family to extract openings $(\boldsymbol{a_i}, r_i)$ of the provided commitments c_i. Send these $\boldsymbol{a_i}$ to the trusted party, and get $\boldsymbol{a_1}, \ldots, \boldsymbol{a_n}, \boldsymbol{a_c}, f$ back.

3. Simulate step 3 of the protocol: using the trapdoors of the commitment scheme family, compute randomness r_i that opens the honest parties' trapdoor commitments c_i to the received values $\boldsymbol{a_i}$. Simulate a regular execution of the MPC protocol with respect to the adversary.

4. Simulate step 4 of the protocol: handle calls to $\mathsf{CompGen}$ by performing the key generation of the adaptive zk-SNARK for QAP Q and keeping the trapdoor tdp.

5. Simulate step 5 of the protocol: take the algorithm that simulates a regular execution of the MPC protocol with respect to the adversary and returns the reconstructed computation result $\boldsymbol{b}, r_c, c_c, \pi$ for the client. Use the argument of knowledge extractor of the adaptive zk-SNARK to extract $\boldsymbol{a'_1}; \ldots; \boldsymbol{a'_n}; \boldsymbol{a'_c}, \boldsymbol{b'}$ and the randomness, proof, and witness.

6. Simulate step 6 of the protocol: send $\top$ to the trusted party if: (1) the received commitment c_c matches the received opening $\boldsymbol{b}, r_c$ and the received proof verifies; (2) the values $\boldsymbol{a'_1}, \ldots$ and randomness extracted in the previous step match those extracted in step 2, and $\boldsymbol{a'_c}, \boldsymbol{b'}, r_c$ match the MPC result; (3) the commitments and extracted values are in $\mathcal{R}_{\mathsf{ck}_1, \ldots, \mathsf{ck}_n, \mathsf{ck}_c}$; otherwise send $\bot$.

Fig. 5. Simulator for Adaptive Trinocchio ("correct" case)

Proof (Theorem 2, "correct" case). We need to show that, for every NUPPT adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that, for all inputs $\boldsymbol{a_1}, \ldots, \boldsymbol{a_n}, \boldsymbol{a}$ and functions f, real-world execution of the protocol in Figure 3 and ideal-world execution of $\mathcal{S}$ give indistinguishable results. We claim that the simulator in Figure 5 does the job. We show this by, starting from the the ideal-world execution (i.e., the algorithm consisting of the code of the ideal-world trusted party, honest parties, simulator and their orchestration), changing the execution in indistiguishable steps until we arrive at the real-world execution (i.e., the algorithm consisting of the code of the real-world trusted party, honest parties, adversary, and their orchestration):

From Ideal to YAD_1 In step 2, instead of using trapdoor commitments for the honest parties' inputs, we use the actual inputs given as argument to YAD_1. In step 4, we use the actual openings of these commitments.

Indistinguishability of Ideal and YAD₁ Follows directly from the trapdoor property of the commitment family.

From YAD₁ to YAD₂ If the computation has succeeded, instead of letting the trusted party evaluate f and using that as the client's output, use the result $\boldsymbol{b}$ of the MPC protocol.

Indistinguishability of YAD₁ and YAD₂ The computation only succeeds if the extracted values are in $\mathcal{R}$ and match the inputs $\boldsymbol{a_1}, \ldots, \boldsymbol{a_n}, \boldsymbol{a_c}$ to f. By definition of $\mathcal{R}$ this implies that evaluation of f gives $\boldsymbol{b}$.

From YAD₂ to YAD₃ In step 6, skip check (2).

Indistinguishability of YAD₂ and YAD₃ The only difference is if checks (1) and (3) are satisfied, but check (2) is not, i.e., there is a mismatch in extracted values. Suppose $\boldsymbol{a_i} \neq \boldsymbol{a_i'}$. We know that $c_i = \mathsf{C}_{\mathsf{ck}_i}(\boldsymbol{a_i}; r_i)$ (because we extracted those values earlier). However, we also know that $(c_1, \ldots, c_n, c_c; \boldsymbol{a_1'}, \ldots, \boldsymbol{w'}) \in \mathcal{R}_{\mathsf{ck}_1, \ldots, \mathsf{ck}_n, \mathsf{ck}_c}$, so also $c_i = \mathsf{C}_{\mathsf{ck}_i}(\boldsymbol{a_i'}; r_i')$. Note that in YAD₃ we are not using the commitment trapdoors anywhere. So YAD₃ can be used to construct an adversary that opens commitments in two different ways, contradicting the computational binding property of the commitment family. This is possible only with negligible probability, proving indistinguishability in this case.

The cases $\boldsymbol{a_c} \neq \boldsymbol{a_c'}$ or $\boldsymbol{b} \neq \boldsymbol{b'}$ are analogous, except we know the openings not because they were extracted but because they were output by the MPC protocol.

From YAD₃ to Real In step 5, directly execute the MPC protocol instead of using the zk-SNARK knowledge extractor. In step 6, skip check (3). Observe that what is left is simply a regular execution of the protocol.

Indistinguishability of YAD₃ and Real The only difference is if check (1) is satisfied but check (3) is not. The argument of knowledge property of the zk-SNARK states that this probability is negligible. □

D.2 Private Function Evaluation

Proof (Theorem 2, "private" case). Analogously to the above proof for correct function evaluation, we claim that the simulator from Figure 6 gives indistinguishable results to a real protocol execution, and prove this via a sequence of intermediate probability distributions:

From Ideal to YAD₁ In step 2, for c_i instead of using commitments to zero, we use commitments to the actual inputs of the honest parties. (Note that we can do this without changing anything else in the simulator; in particular, in step 3 the MPC simulator uses c_i without requiring its opening.)

Indistinguishability of Ideal and YAD₁ Follows directly from the perfect hiding property of the commitment family.

Simulator: Adaptive Trinocchio ("private" case)

(Given NUPPT adversary $\mathcal{A}$ passively corrupting at most one worker, and actively corrupting a subset of inputters and possibly the client.)

1. Perform CRS and key generation of the commitment scheme, keeping trapdoors $\mathsf{crs}, \mathsf{ctd}_1, \ldots, \mathsf{ctd}_n, \mathsf{ctd}_o$. Simulate step 1 of the protocol with respect to $\mathcal{A}$ by sending this material.
2. Simulate step 2 of the protocol: send commitments c_i to zero on behalf of the honest parties, keeping the randomness r_i used. Obtain commitments c_i provided by the adversary on behalf of corrupted inputters.
3. Simulate step 3 of the protocol using the MPC simulator. For the c_i' output on behalf of the honest inputters, output c_i. For the c_i' output on behalf of the corrupted inputters, output $\mathsf{C}_{\mathsf{ck}_i}(\boldsymbol{a_i'}, r_i')$, where $\boldsymbol{a_i'}, r_i'$ are the values input to the MPC protocol by the inputter. Whenever $c_i \neq c_i'$, set $\boldsymbol{a_i'} = \bot$ and stop after this step. Send $\boldsymbol{a_i'}$ of corrupted inputters to the trusted party.
4. Simulate step 4 of the protocol: if the client is corrupted, learn f from its call to $\mathsf{CompGen}$; otherwise, receive f from the ideal-world trusted party. Handle the adversary's calls to $\mathsf{CompGen}$ by performing key generation on this function f.
5. Simulate step 5 of the protocol. If the client is not corrupted, then the adversary does not learn any outcome of the MPC, so no information is needed to perform the simulation. If the client is corrupted, then get value $\boldsymbol{a_c'}$ input by the client from the simulator. Send $f, \boldsymbol{a_c'}$ to the trusted party, and receive computation result $\boldsymbol{b}$. Generate random r_c and compute $c_c = \mathsf{C}_{\mathsf{ck}_c}(\boldsymbol{a_c'}, \boldsymbol{b}; r_c)$. Apply the zero-knowledge simulator of the zk-SNARK on the CRS, commitment keys, $c_1, \ldots, c_n, c_c$, and the trapdoors, to get a proof π. Simulate the remainder of the MPC such that $\boldsymbol{b}, r_c, c_c, \pi$ is output to the corrupted client.
6. Simulate step 6 of the protocol: let a corrupted client handle its received result.

Fig. 6. Simulator for Adaptive Trinocchio ("private" case)

From YAD_1 to YAD_2 In step 5, instead of using the zk-SNARK simulator, use P on the commitments $c_1, \ldots, c_n, c_c$, inputs $\boldsymbol{a_1}, \ldots, \boldsymbol{a_n}, \boldsymbol{a_c'}, \boldsymbol{b}$, randomness $r_1, \ldots, r_n, r_c$, and witness $\boldsymbol{w}$ available from the MPC simulator.

Indistinguishability of YAD_1 and YAD_2 Follows directly by the perfectly composable zero-knowledge property of the zk-SNARK.

Indistinguishability of YAD_2 and Real Note that the only remaining difference between the YAD_2 and a real-world execution is that the MPC protocol is simulated in YAD_2 and executed in Real. Both for the MPC execution in step 3 and in step 5, the output of the computation is the same: in step 3 it is the commitments of the values provided by the honest and corrupted inputters; in step 5, it is the computation result and proof based on those values and the client's input. Hence, by the security of the MPC protocol, the two distributions are indistiguishable. □