SNARKs for Stateful Computations on Authenticated Data

Johannes Reinhart[1], Erik-Oliver Blass[2], and Bjoern Annighoefer[1]

[1]University of Stuttgart
[2]Airbus

Abstract

We introduce Authenticated Data and State Consistent SNARKs (ADSC-SNARKs), a new cryptographic primitive for proving the integrity of stateful computations over authenticated data, designed for the application domain of safety-critical control systems. ADSC-SNARKs simultaneously guarantee that a computation was performed correctly, that its inputs originate from a trusted source and that its internal state is consistent across multiple iterations. Our core contribution is the first concretely efficient construction of this primitive. The construction integrates commitments for state and authenticated data directly into the structure of a pairing-based SNARK. Our method neither requires non-degeneracy conditions on the relation nor arithmetization of cryptographic primitives. It therefore entirely avoids increasing the size of the relation to be proven, reducing prover overhead. We implement and benchmark our new SNARKs in a sample scenario of a real-time high-integrity flight control system. With our construction, prover runtime improves significantly over our baseline by a factor of 89, enabling the use of SNARKs in real-time settings previously thought impractical.

1 Introduction

In many applications, one party must demonstrate to others that they have correctly executed a certain computation. Specifically, party $\mathcal{P}$ takes input x and computes output $y = f(x)$ for some function f. To then prove correct computation of y, $\mathcal{P}$ (the prover) generates a proof π. The idea is that any verifier $\mathcal{V}$ can use π and check whether y is the correct output with respect to x and f. Two key properties towards practical efficiency are that verification of π should be more efficient than re-computing y, and that the size of π is small. The construction of such proofs is a very active area of research as indicated by the recent flurry of papers on, for example, Succinct Non-interactive ARguments of Knowledge (SNARKs) [1, 8, 9, 11, 12, 16, 22, 34, 37, 39, 41, 48, 53, 56, 57]. An additional attribute as provided by zero-knowledge SNARKs (zk-SNARKs) is that $\mathcal{V}$ does not learn anything about x besides the validity of the proof.

This paper explores a new generalization of (zk-)SNARKs for applications in digital control systems. A typical digital control system comprises a central control unit which computes control outputs for actuators to manipulate a technical system. In most such cases, the control unit must periodically process sensor input x_t, update its internal state s_t and provide output commands y_t. Many control systems are safety-critical, such that a malfunction or adversarial tampering can lead to loss of lives. Furthermore, modern control system have a large complexity and inter-connectivity, exposing many attack surfaces. Applying SNARKs in such a setting can be used to prevent cyber-attacks on control units or communication channels or help detect incorrect outputs from malfunctioning devices. These settings would in particular benefit from the ability to verify output commands of a control unit immediately in each iteration t. Our SNARKs combine two key properties to support this use case.

SNARKs over Authenticated Data If $\mathcal{V}$ must not learn x for performance or privacy reasons, the natural question is how to authenticate x, such that a malicious $\mathcal{P}$ does not choose arbitrary x as input. So, the first additional property we consider in this paper is that x be authenticated by a third party, e.g., the sensor actually measuring physical quantity x_t. Consequently, $\mathcal{P}$ must additionally prove to $\mathcal{V}$ that input x to f has been authenticated by the third party before. A crucial challenge in this context is that $\mathcal{V}$ does not have access to the third party at the time of verification.

State-Consistent SNARKs The computation of a digital controller usually depends on a changing state. That is, given initial state s_1, a function f

is applied iteratively on state s_t and input x_t producing output y_t and updated state s_{t+1}. So, in each iteration t,

$$(y_t, s_{t+1}) = f(x_t, s_t).$$

Again, the technical challenge comes in the situation where party $\mathcal{P}$ computing f is untrusted or unreliable. An adversarial $\mathcal{P}$ might arbitrarily change states s_t between iterations. Thus, the second property we target in this paper is for $\mathcal{P}$ to create a proof π_t allowing $\mathcal{V}$ to efficiently verify that y_t is the correct output when evaluating f on the state update of the previous iteration t. As the state is often large, and communication bandwidth or memory of the verifying party can be limited, we require that the size of π_t should also be smaller than the size of state s_t.

A major challenge of deploying SNARKs to digital control systems is the runtime of prover $\mathcal{P}$. Statements to be proven must be expressed (arithmetized) in an appropriate representation, such as a Quadratic-Arithmetic-Program (QAP) [37], which can become large. Prover runtime will be at least linear in the size of the arithmetized statement and in practice comes with large constants. Modifications to existing SNARKs to prove input authenticity or state consistency either significantly increase the size of the arithmetized statement, which we will refer to as arithmetization overhead, or require the prover to perform additional computations. We seek a solution that minimizes both arithmetization overhead and prover overhead for fast proving times.

This paper We introduce Authenticated Data and State Consistent SNARKs (ADSC-SNARKs) as a new, integrated cryptographic primitive for proving the integrity of stateful computations on authenticated data streams. We provide the first formal definition of this primitive and its security properties. Our main contribution is the first concretely efficient construction of an ADSC-SNARK. Our construction makes use of a novel technique for integrating state and input commitments directly into the structure of a pairing-based SNARK. The method avoids the overheads associated with the arithmetization of cryptographic primitives (e.g. hash functions or signature verifications) and non-degeneracy requirements of existing commit-and-prove schemes on the arithmetized statement to be proven. We implement our construction and show that it achieves a prover speedup of 89× over an approach with arithmetizing cryptographic primitives which provides both data authenticity and state-consistency at the same time. Compared to specialized constructions which either provide data authenticity or state-consistency, we achieve speedups of 60 % over AD-SNARK [3] and 20 % over an approach based on LegoGro [17]. The C++ implementation is open source and freely available[1].

The **technical highlights** of this paper are:

- A formal definition of the security properties for ADSC-SNARKs, including definitions for completeness, soundness, and zero-knowledge.
- The first construction of an efficient, public verifier ADSC-SNARK that natively supports state consistency and data authenticity without arithmetization overhead.

  Crucially, proof size and verifier runtime are constant in the size of witness, input, and internal state.
- An open source C++ implementation of our ADSC-SNARK based on the standard libsnark library. This is particularly suitable for practitioners in domains where C++ is predominantly used such as in embedded systems engineering.
- An evaluation demonstrating that our construction is not only asymptotically but also concretely practical, achieving up to an 89× prover speedup over folklore approaches.
- A proof-of-concept application and its evaluation in the form of a safety-critical control system, demonstrating a viable path towards a practical application of SNARKs in real-time environments.

A comparison between our construction and related work is given in Table 1. We display the additional overhead that each approach adds to the original SNARK, which the approach is based on.

Zero-Knowledge As with related work [9, 41, 53], also our ADSC-SNARK construction proves in zero-knowledge by adding randomization masks to the proof. Due to space constraints and to ease exposition, we will focus on describing ADSC-SNARKs *without* the zero-knowledge property. B presents the full construction of ADSC-SNARKs *with* zero-knowledge. We stress that our implementation and evaluation in Section 4 is performed with the full version of ADSC-SNARK, including the zero-knowledge property.

[1] https://github.com/johannes-reinhart/adsc-snark

Table 1: Comparison of our ADSC-SNARK with related approaches. $|X|$: number of private inputs, $|S|$: number of states, $|\Omega|$: size of witness, $\mathbb{F}_p$: prime field element, $\mathbb{G}_1$: curve group 1 element, $\mathbb{G}_2$: curve group 2 element, *sig*: signature, $\text{MSM}_{\mathbb{G}_1}(n)$: multiscalar multiplication of n $\mathbb{G}_1$ elements, P: pairing evaluation, SV: signature verification, PV: public verifiability, AD: proof about authenticated data, SC: state consistency, US: universal setup. Overhead is additional cost compared to base SNARK. Arithmetization overhead is increase of QAP degree of arithmetized statement. Both arithmetization and prover overhead increase prover runtime.

Approach	Base SNARK	Arith. Overhead	Prover Overhead	Verifier Overhead	Proof Overhead	PV	AD	SC	US
Folklore ("Strawman")	Groth16 [41]	2126 $+72\|X\|$ $+144\|S\|$	—	—	$1\times\mathbb{F}_p$	✓	✓	✓	—
AD-SNARK [3] (dv)	BCTV14 [9]	$\|X\|$	$1\times\text{MSM}_{\mathbb{G}_1}(\|X\|)$	$1\times\text{MSM}_{\mathbb{G}_1}(\|X\|)$ $+2\times\text{P}$	$3\times\mathbb{G}_1$	—	✓	—	—
AD-SNARK [3] (pv)	BCTV14 [9]	$\|X\|$	$1\times\text{MSM}_{\mathbb{G}_1}(\|X\|)$	$1\times\text{MSM}_{\mathbb{G}_1}(\|X\|)$ $+(\|X\|+4)\times\text{P}$ $+\|X\|\times\text{SV}$	$3\times\mathbb{G}_1$ $+\|X\|\times\mathbb{G}_2$ $+\|X\|\times sig$	✓	✓	—	—
SPHinx [32]	Marlin [22]	—	$1\times\text{MSM}_{\mathbb{G}_1}(\|\Omega\|)$ $+4\times\text{MSM}_{\mathbb{G}_1}(\|X\|)$ $+1\times\text{MSM}_{\mathbb{G}_1}(\|X\|+\|\Omega\|)$	$7\times\text{P}$ $+2\times\text{MSM}_{\mathbb{G}_1}(\|X\|)$	$11\times\mathbb{G}_1$ $+11\times\mathbb{F}_p$	✓	✓	—	✓
Geppetto [24] SC-SNARK	Pinocchio [53]	$2\times\|S\|$	$3\times\text{MSM}_{\mathbb{G}_1}(\|S\|)$	$1\times\text{P}$	$3\times\mathbb{G}_1$	✓	—	✓	—
LegoGro [17] SC-SNARK	Groth16 [41]	$2\times\|S\|$	$1\times\text{MSM}_{\mathbb{G}_1}(\|S\|)$ $+1\times\text{MSM}_{\mathbb{G}_1}(2\|S\|)$	$4\times\text{P}$	$3\times\mathbb{G}_1$	✓	—	✓	—
ADSC-SNARK **(ours)**	Groth16 [41]	—	$1\times\text{MSM}_{\mathbb{G}_1}(\|S\|)$	$3\times\text{P}$ $+1\times\text{SV}$	$2\times\mathbb{G}_1$ $+1\times sig$	✓	✓	✓	—

1.1 Our solution in a nutshell

State Consistency: To prove that the state between two computations is consistent, $\mathcal{P}$ might send constant-sized hashes of the state $H(s_t)$ and the updated state $H(s'_t)$ to $\mathcal{V}$ and augment proof π_t to show consistency of $H(s_t)$ with $H(s'_{t-1})$ from the previous iteration. This turns out to be expensive, as H incurs a large arithmetization overhead, increasing prover time significantly [7, 15, 50, 55].

A more advance approach uses commit-and-prove SNARKs [17, 24], where a proof also acts as a commitment to the state. The commitment can then be used as a link between SNARK proofs of two consecutive computational steps. However, prior constructions of pairing-based commit-and-prove SNARKs require a non-degeneracy condition on the underlying QAP, usually enforced by additional "dummy" constraints in the relation. This results in prover overhead which grows with the size of the state. Our work avoids this overhead by eliminating the need for the non-degeneracy conditions through additional randomness in the trusted setup.

Authenticated Data: To prove that the computation has been carried out on authenticated input data x, one might add a signature to x and prove correctness of the signature with respect to x. Again, this would imply an expensive arithmetization of the signature verification algorithm.

The key idea to overcome this issue is to only sign a commitment to the input data, rather than the data itself. The novelty lies in how we link the data of the SNARK proof and the data of the signed commitment. We apply the same technique of modifying the trusted setup as for state consistency to link the data without requiring non-degeneracy conditions on the QAP. The verifier can check the signature on the small, constant-size commitment and be convinced that the correct inputs were used, without needing to read the actual input.

Our construction unifies the mechanisms for state consistency and data authenticity using the same principle to create a single, efficient primitive.

1.2 Notation

In this paper, λ denotes the computational security parameter. For instantiating our scheme, we use prime-order groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, $|\mathbb{G}_1|$, $|\mathbb{G}_2|$, $|\mathbb{G}_T| \in poly(\lambda)$ together with a Type-3 bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We denote a group element α in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ by its discrete logarithm $[\alpha]_1$, $[\alpha]_2$, $[\alpha]_T$.

The group operation in $\mathbb{G}_1$ ($\mathbb{G}_2, \mathbb{G}_T$ accordingly) is represented as addition, and scalar multiplication of element α by scalar b by $b\,[\alpha]_1 = [b\alpha]_1$. The multiplication operator $(\cdot)$ applied to group elements from $\mathbb{G}_1$ and $\mathbb{G}_2$ represents applying the bilinear map e to the operands. $\mathbb{F}^*$ denotes the set of invertible elements of a field $\mathbb{F}$.

For a randomized algorithm $F(x; r_\$)$ with input x and random coins $r_\$$, $y \leftarrow F(x)$ denotes random sampling of $r_\$$ and then assigning the output of $F(x; r_\$)$ to y. We denote the assignment of multiple elements using independent random coins $r_\$$ for F and set S as $y_t \leftarrow F(x_t)$ for $t \in S$. For two randomized algorithms F and H, we write $(y; z) \leftarrow (F||H)(x)$ for picking random coin $r_\$$ and then assigning the result of applying F to x with $r_\$$ to y, and assigning the result of applying H to the same input x and the same random coin $r_\$$ to z. We write $F^{G(\cdot)}$ to denote that F has oracle access to G. To sample an element from a set S uniformly from random, we write $y \leftarrow\!\$\ S$.

For functions f and g, the approximate equal operator $\approx$ denotes that the difference $f - g$ is negligible in security parameter λ:

$$f \approx g \Leftrightarrow \forall c \in \mathbb{N} : \exists\, \lambda_0, \text{ s.t. } \forall \lambda > \lambda_0 : \\ |f(\lambda) - g(\lambda)| < \lambda^{-c}.$$

For a set S, we write $\text{pos}_S(i)$ to order elements i by size, i.e., for $S = \{3, 4\}$, $\text{pos}_S(3) = 1$, $\text{pos}_S(4) = 2$. We write PPT for a probabilistic polynomial time algorithm and Pr for probability.

2 ADSC-SNARK Background

Before our main construction, we present formal definitions of an ADSC-SNARK and its properties for a family of relations R_λ and security parameter λ.

2.1 Setup

We assume that R_λ can be generated by a generator $\text{pp} \leftarrow \mathsf{Gen}(\lambda)$ outputting public-parameters pp, such that pp include a field $\mathbb{H}$ and a relation $\text{R} \in \text{R}_\lambda$ over m variables a_i from field $\mathbb{H}$.

We require that relation R allows mapping each variable a_i into one of five different categories. These are represented by disjoint vectors ϕ, x, s, s', ω with corresponding index sets Φ, X, S, S', Ω as follows:

1. *public input-output* variables $\phi = (a_i)_{i \in \Phi}$,
2. *private input* variables $x = (a_i)_{i \in X}$,
3. *state* variables $s = (a_i)_{i \in S}$,
4. *state update* variables $s' = (a_i)_{i \in S'}$, and
5. *witness* variables $\omega = (a_i)_{i \in \Omega}$.

We say that the sequence of r variable vectors $((a_i)_{i \in [1,m],1}, (a_i)_{i \in [1,m],2}, \ldots, (a_i)_{i \in [1,m],r}) \in \text{R}^r$ with initial state s'_0 has *state consistency*, if for all $t \in [1, r]$, we have $s_t = s'_{t-1}$. We call r the number of iterations and include it in pp. For every state variable, there is one corresponding state update variable, therefore $|S| = |S'|$. In order to relate corresponding state and state update pairs, we require a bijective map $s2s$ between index sets for states $s2s : S \to S'$. For every index of a state variable $i \in S$, $s2s$ will output the index of the corresponding state update variable $i' \in S'$.

An ADSC-SNARK for R_λ consists of the following tuple of algorithms (Gen, Setup, Auth, Prove, Verify).

Algorithm $\text{pp} \leftarrow \mathsf{Gen}(\lambda)$: on input security parameter λ, Gen outputs public parameters pp including a relation $\text{R} \in \text{R}_\lambda$. All other algorithms below use pp as an additional input. For brevity we do not include pp in their description.

Algorithm $(\sigma_p, \sigma_v, \sigma_a, c_0) \leftarrow \mathsf{Setup}(s'_0)$: takes initial state s'_0 as input and computes a prover-key σ_p, a verifier-key σ_v, and an authentication-key σ_a. It also outputs a commitment to the initial state c_0.

Algorithm $\nu_t \leftarrow \mathsf{Auth}(\sigma_a, t, x_t)$: gets authentication-key σ_a, time step t, and an input x_t. It outputs a signature ν_t.

Algorithm $(\pi_t, c_t, p_t) \leftarrow \mathsf{Prove}(\sigma_p, a_t, \nu_t, p_{t-1})$: gets prover-key σ_p, variables $a_t = (\phi_t, x_t, s_t, s'_t, \omega_t)$, and signature ν_t. It outputs a proof π_t and a commitment c_t to state update s'_t. A prover is allowed to have some internal state p_t that is stored between iterative invocations of the proving algorithm.

Algorithm $b \leftarrow \mathsf{Verify}(\sigma_v, \phi_t, \pi_t, c_t, c_{t-1}, t)$: gets verifier-key σ_v, the public input-output ϕ_t, the proof π_t, the current commitment c_t, the commitment from the previous iteration c_{t-1} and timestep t. It either accepts the proof by outputting $b = 1$ or rejects it by outputting $b = 0$.

Discussion: To ease understanding, we briefly explain the above with the help of function f, our running example from the introduction. Relation R and the state mapping function $s2s$ represent a computation of function f such that R is satisfied if $(y_t, s'_t) = f(x_t, s_t)$. State is represented by pairs of variables, with one variable for the current state, and one for the updated state (for the next iteration). This results in the two same-sized tuples s and s'. Function $s2s$ relates the indices for the state and

the state update variables. The former is part of the input of f, the latter (the state update for that iteration) is part of the output of f. When iteratively applying f on state s, such that $s_{t+1} = s'_t$, we get $y_t, s_{t+1} = f(x_t, s_t)$. ϕ corresponds to the function's output y and private input x corresponds to the function's input x_t for iteration t.

2.2 Threat Model

ADSC-SNARK works in a setting with four parties: A data authenticator $\mathcal{D}$, a prover $\mathcal{P}$, a verifier $\mathcal{V}$ and a trusted helper $\mathcal{T}$. They engage in the following protocol: $\mathcal{T}$ runs algorithm Setup once and distributes prover-key σ_p, verifier-key σ_v and authentication-key σ_a to the other parties. There is no further interaction with $\mathcal{T}$. Subsequently, $\mathcal{D}$ starts authenticating private inputs x_t using Auth and provides these to $\mathcal{P}$. $\mathcal{P}$ runs a computation on the input data, updating its state s_t and producing public input-outputs ϕ_t. Using witness ω_t and Prove, it produces a proof-commitment pair (π_t, c_t), and sends (ϕ_t, π_t, c_t) to $\mathcal{V}$. $\mathcal{V}$ applies Verify to check whether ϕ_t is correct.

$\mathcal{V}$ does not trust $\mathcal{P}$ to provide correct ϕ_t. In this scenario, we consider an adversary that has full control over $\mathcal{P}$: It has access to σ_p and σ_v and gets to see all messages from $\mathcal{D}$. It can make $\mathcal{P}$ send any message to $\mathcal{V}$, arbitrarily deviating from the protocol. However, the adversary does neither know σ_a, nor the random choices of $\mathcal{T}$. Here, ADSC-SNARK is secure, as long as $\mathcal{V}$ does not accept incorrect ϕ_t from $\mathcal{P}$. This is formally captured by the knowledge-soundness property (see Section 2.3).

Additionally, the protocol must be efficient, which means that the amount of data sent from $\mathcal{P}$ to $\mathcal{V}$ is small and that $\mathcal{V}$ can quickly check the correctness of ϕ_t. This is formally captured by the succincntess property.

2.3 ADSC-SNARK Properties

For an ADSC-SNARK we require completeness, knowledge-soundness, and succinctness. Note that for the following definitions, the number of iterations $r \in \mathbb{N}$ is part of the public parameters pp.

Intuitively, completeness means that $\mathcal{V}$ accepts all proofs, if in each iteration: 1) relation R is satisfied, 2) the state between consecutive iterations is consistent, and 3) private inputs have been authenticated.

Definition 1 (Completeness of ADSC-SNARK)**.** *For all $\lambda \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda)$, $(a_1, ..., a_r) \in \mathrm{R}^r$, with $a_t = (\phi_t, x_t, s_t, s'_t, \omega_t)$, and $s'_0 \in \mathbb{H}^{|S|}$ such that $\bigwedge_{t=1}^{r}(s_t = s'_{t-1})$ and $p_0 = \{\}$:*

$$\Pr\left[\begin{array}{c}(\sigma_p, \sigma_v, \sigma_a, c_0) \leftarrow \mathsf{Setup}(s'_0);\\ \nu_t \leftarrow \mathsf{Auth}(\sigma_a, t, (a_{t,i})_{i\in X}) \text{ for } t \in [1,r]\,;\\ (\pi_t, c_t, p_t) \leftarrow \mathsf{Prove}(\sigma_p, a_t, \nu_t, p_{t-1}) \text{ for } t \in [1,r]:\\ \bigwedge_{t=1}^{r} \mathsf{Verify}(\sigma_v, \phi_t, \pi_t, c_t, c_{t-1}, t) = 1\end{array}\right] = 1.$$

With knowledge-soundness, intuitively, for each iteration t in which $\mathcal{V}$ accepts, relation R must be satisfied, the prover must know the witness, and the state between the previous relation and the current relation must be consistent. Additionally, the private inputs must have been authenticated. This must hold, even if an adversary prover can query Auth.

Definition 2 (Knowledge-Soundness of ADSC-SNARK)**.** *For all PPT adversaries $\mathcal{A}$ with oracle access to* Auth*, there exists a PPT algorithm* Extract*, such that*

$$\Pr\left[\begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda);\\ (\sigma_p, \sigma_v, \sigma_a, c_0) \leftarrow \mathsf{Setup}(s'_0);\\ ((\pi_t, c_t, \phi_t); (x_t, s_t, s'_t, \omega_t)) \leftarrow\\ (\mathcal{A}^{\mathsf{Auth}(\cdot)} \,||\, \mathsf{Extract})(\sigma_p, \sigma_v, t, c_0) \text{ for } t \in [1,r]:\\ \bigvee_{t=1}^{r}\left(\begin{array}{c}\left(\begin{array}{c}(\phi_t, x_t, s_t, s'_t, \omega_t) \notin \mathrm{R}\\ \vee s_t \neq s'_{t-1}\\ \vee (t, x_t) \notin \tilde{X}\end{array}\right)\\ \wedge \mathsf{Verify}(\sigma_v, \phi_t, \pi_t, c_t, c_{t-1}, t) = 1\end{array}\right)\end{array}\right] \approx 0,$$

where $\tilde{X}$ is the set of tuples $(\tilde{t}, \tilde{x}_t)$ for each query which $\mathcal{A}$ made to Auth*.*

Intuitively, succinctness implies that the proof and the commitment $(\pi,\ c)$ are asymptotically smaller than the relation (excluding public input-output). Furthermore, verifying is asymptotically cheaper than checking the relation directly. In our specific situation with ADSC-SNARKs, both proof size $|(\pi, c)|$ and runtime T_{Verify} of Algorithm Verify are even constant in the length of witness ω, the length of input x, and the size of states s. The verifier runtime is linear only in the size of statement ϕ. We formalize this with the following definition.

Definition 3 (Succinctness of ADSC-SNARK)**.** *For all $\lambda \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda)$, $(a_1, ..., a_r) \in \mathrm{R}^r$, $(\phi_t, x_t, s_t, s'_t, \omega_t) = a_t$, $s'_0 \in \mathbb{H}^{|s|}$, $(\sigma_p, \sigma_v, \sigma_a, c_0) \leftarrow \mathsf{Setup}(s'_0)$, $t \in [1,r]$, $\nu \leftarrow \mathsf{Auth}(\sigma_a, t, (a_{t,i})_{i\in X})$, $p_0 = \{\}$, $(\pi_t, c_t, p_t) \leftarrow \mathsf{Prove}(\sigma_p, a_t, \nu, p_{t-1})$:*

$$|(\pi_t, c_t)| \in O(\lambda) \text{ and } T_{\mathsf{Verify}} \in O(\lambda + |\phi|),$$

i.e., they are constant in $|\omega|, |x|, |s|$, and T_{Verify} is linear in $|\phi|$.

Discussion The definition for knowledge-soundness of ADSC-SNARK is a combination of knowledge-soundness of a regular SNARK in addition to authenticity and state-consistency. The idea is that an adversary could behave incorrectly in three different ways: First, it could try to generate a proof for a statement, which does not satisfy relation R. Second, it could use a state for the relation that is not consistent with the state-update from the previous relation. Finally, it could use private inputs that have not been authenticated by an authorized party. The definition disallows these cases by requiring the probability of any of them occurring to be negligible. For authenticity, the definition captures chosen message attacks against the authenticating party by giving $\mathcal{A}$ oracle access to Auth.

3 Main ADSC-SNARK Construction

Before presenting our main contribution, the construction of ADSC-SNARK, we start with an overview and highlight its key concepts.

3.1 Overview

Our ADSC-SNARK takes the SNARK of Groth [41] as a starting point. There, a proof comprises 3 group elements $[A]_1, [B]_2$, and $[C]_1$ that the prover derives by computing linear combinations of elements from its prover-key σ_p. Informally, the security rationale is that the prover cannot create a valid proof besides by linearly combining elements from σ_p. The verifier will check certain relations between the proof elements that depend on a set of secret values $(\alpha, \beta, \gamma, \delta)$ drawn randomly during setup. Note that the verifier does not need to see the secret values in the clear, as they can perform verification with the help of pairing e. As the prover does not know the secret values either, they can create a verifying proof only by a (linear) combination of the prover-key elements that depend on the secret values, but do not reveal them. In fact, the prover must use variables $(\phi, x, s, s', \omega) \in \mathrm{R}$ as coefficients of these linear combinations for a proof to verify. Recall that $(\phi, x, s, s', \omega) \in \mathrm{R}$ implies that the computation was carried out correctly.

Our key idea to extend the Groth SNARK such that it also provides state-consistency and data authenticity. This approach is loosely based on the observation made and used in several other works [17, 24, 33, 47]: proof elements not only allow the verifier to check the satisfaction of R, but can also act as a *commitment* to the variables satisfying R. Intuitively, if you assume that the involved prover key elements are random, any two different linear combinations of them are also different with high probability. This is similar to the standard Pedersen commitment [54]. Using the commitment property allows for checking between two different SNARK proofs whether they were carried out on the same (committed) data.

State Consistency: By defining state variables s and s' as the committed data, we can use the above technique to achieve state-consistency. More specifically, the prover outputs a proof π_{t-1} and a vector commitment c_{t-1} in step $t-1$. c_{t-1} commits to the state update s'_{t-1}. In the subsequent iteration t, the prover produces another proof π_t (and also another commitment c_t). Now, in this iteration t, the verifier can not only check whether π_t is correct, but also whether state s_t committed to by some of the elements in π_t is the same as s'_{t-1} committed to by c_{t-1}. In our ADSC-SNARK construction, we denote c_t by element $[E_t]_1$.

Authenticated Data: For proving data authenticity, we let element $[D_t]_1$ serve as commitment for private input x. First, the authenticating party signs the commitment with a regular signature scheme. Now the verifier can check whether the private input variables used for π_t match the committed input variables used for $[D_t]_1$. By also checking the signature on $[D_t]_1$ the verifier checks that the private input stems from the authenticating party.

Method: In ADSC-SNARKs, we do not use commit-and-prove schemes as a black-box, but rather design a novel strategy for checking proofs and commitments at the same time. This method avoids the non-degeneracy conditions required by earlier approaches [17, 24], which would otherwise add arithmetization overhead by forcing dummy constraints into the QAP. To verify that commitments $[D_t]_1$, $[E_t]_1$, $[E_{t-1}]_1$ and proof π_t have been computed using the same data, we let Setup generate random group elements $R_i \in \mathbb{G}_1$ for all $i \in S'$ and $T_i \in \mathbb{G}_1$ for all $i \in X$. We then subtract them from prover-key elements $(\left\{\frac{P_i(z)}{\delta}\right\}_{i\in S}, \left\{\frac{P_i(z)}{\delta}\right\}_{i\in S'}, \left\{\frac{P_i(z)}{\delta}\right\}_{i\in X})$ that correspond to the variables for the state s_t, the state update s'_t, and private input x, yielding the following new prover-key elements:

$$\left\{\frac{P_i(z) - R_{s2s(i)}}{\delta}\right\}_{i\in S}, \left\{\frac{P_i(z) - R_i}{\delta}\right\}_{i\in S'}, \left\{\frac{P_i(z) - T_i}{\delta}\right\}_{i\in X}.$$

Commitments E_{t-1}, E_t, and D_t appear as additional terms on the right hand side of the verification equation. When the prover computes proof element C_t as

$(\sigma_p, \sigma_v, \sigma_a, c_0) \leftarrow \mathsf{Setup}(s'_0)$

Parse public parameters pp to retrieve polynomials $\{u_i(Z)\}_{i\in[0,m]}, \{v_i(Z)\}_{i\in[0,m]}, \{w_i(Z)\}_{i\in[0,m]}, t(Z)$.

Randomly sample
$T_i \leftarrow\$ \mathbb{F}_p^* \text{ for } i \in X, \quad R_i \leftarrow\$ \mathbb{F}_p^* \text{ for } i \in S',$
$\alpha, \beta, \gamma, \delta, \kappa, \eta, \varepsilon, z \leftarrow\$ \mathbb{F}_p^*.$
Generate keys for the signature scheme
$(sk, pk) \leftarrow \mathsf{SigGen}()$.

Compute prover-key
$\sigma_p = ([\sigma_{p,1}]_1, [\sigma_{p,2}]_2),$

with

$$\sigma_{p,1} = \left(\begin{array}{c} \alpha, \{u_i(z)\}_{i\in[0,m]}, \left\{\frac{P_i(z)}{\delta}\right\}_{i\in\Omega}, \\ \left\{\frac{P_i(z)-\eta R_{s2s(i)}}{\delta}\right\}_{i\in S}, \left\{\frac{P_i(z)-\kappa R_i}{\delta}\right\}_{i\in S'} \\ \left\{\frac{P_i(z)-\varepsilon T_i}{\delta}\right\}_{i\in X}, \left\{\frac{z^i t(z)}{\delta}\right\}_{i\in[0,d-2]}, \{R_i\}_{i\in S'} \end{array} \right)$$

$\sigma_{p,2} = \left(\beta, \{v_i(z)\}_{i\in[0,m]}\right),$
Compute verifier-key
$\sigma_v = ([\sigma_{v,1}]_1, [\sigma_{v,2}]_2, [\alpha\beta]_T, pk)$

with

$\sigma_{v,1} = \left(\left\{\frac{P_i(z)}{\gamma}\right\}_{i\in\{0\}\cup\Phi}\right), \sigma_{v,2} = \left(\gamma, \delta, \varepsilon, \eta, \kappa\right).$

Compute authentication-key and commitment for initial state
$\sigma_a = (sk, \{[T_i]_1\}_{i\in X}),$
$c_0 = \sum_{i\in S'} s'_{0,\mathrm{pos}_{S'}(i)} [R_i]_1.$

$\nu_t \leftarrow \mathsf{Auth}(\sigma_a, t, x_t)$

Parse authentication-key σ_a to retrieve
$sk, \{[T_i]_1\}_{i\in X}.$

Parse input x_t to retrieve
$a_i \text{ for } i \in X.$

Compute
$\nu_t = ([D]_1, sig).$

with
$D = \sum_{i\in X} a_i T_i, \quad sig = \mathsf{SigSign}(sk, ([D]_1, t)).$

$(\pi_t, c_t) \leftarrow \mathsf{Prove}(\sigma_p, a_t, \nu_t)$

Parse public parameters pp to retrieve polynomials
$\{u_i(Z)\}_{i\in[0,m]}, \{v_i(Z)\}_{i\in[0,m]}, \{w_i(Z)\}_{i\in[0,m]}, t(Z).$

Parse ν_t to retrieve
$[D]_1, sig.$

Compute by polynomial division
$h(Z) =$

$$\left(\sum_{i\in[0,m]} a_{t,i} u_i(Z) \cdot \sum_{i\in[0,m]} a_{t,i} v_i(Z) - \sum_{i\in[0,m]} a_{t,i} w_i(Z)\right) / t(Z).$$

Compute proof and commitment
$\pi_t = ([A]_1, [B]_2, [C]_1, [D]_1, sig), \quad c_t = [E]_1,$

with

$$A = \alpha + \sum_{i\in[0,m]} a_{t,i} u_i(z), \quad B = \beta + \sum_{i\in[0,m]} a_{t,i} v_i(z),$$

$$C = \sum_{i\in\Omega} \frac{a_{t,i} P_i(z)}{\delta} + \sum_{i\in S} \frac{a_{t,i}\left(P_i(z) - \eta R_{s2s(i)}\right)}{\delta} + \frac{h(z)t(z)}{\delta}$$
$$+ \sum_{i\in S'} \frac{a_{t,i}\left(P_i(z) - \kappa R_i\right)}{\delta} + \sum_{i\in X} \frac{a_{t,i}\left(P_i(z) - \varepsilon T_i\right)}{\delta},$$

$$E = \sum_{i\in S'} a_{t,i} R_i.$$

$v \leftarrow \mathsf{Verify}(\sigma_v, \phi_t, \pi_t, c_t, c_{t-1}, t)$

Parse public input-output ϕ_t to retrieve
$a_{t,i} \text{ for } i \in \Phi.$

Parse proof π_t to retrieve
$[A_t]_1, [B_t]_2, [C_t]_1, [D]_1, sig.$

Parse commitments c_t and c_{t-1} to retrieve
$[E_t]_1, \quad [E_{t-1}]_1.$

Accept, iff

$$[A_t]_1 \cdot [B_t]_2 = [\alpha\beta]_T + \left(\sum_{i\in\{0\}\cup\Phi} a_{t,i} \left[\frac{P_i(z)}{\gamma}\right]_1\right) \cdot [\gamma]_2$$
$$+ [C_t]_1 \cdot [\delta]_2 + [D_t]_1 \cdot [\varepsilon]_2 + [E_{t-1}]_1 \cdot [\eta]_2 + [E_t]_1 \cdot [\kappa]_2$$

and
$\mathsf{SigVerify}(pk, sig_t, ([D_t]_1, t)) = 1.$

Figure 1: ADSC-SNARK construction (without zero-knowledge) for QAP $\{u_i(Z)\}_{i\in[0,m]}$, $\{v_i(Z)\}_{i\in[0,m]}$, $\{w_i(Z)\}_{i\in[0,m]}$, $t(Z)$. The interfaces are described in section 2.1. The differences to the Groth16 SNARK are highlighted. For brevity, $P_i(z) = \beta u_i(z) + \alpha v_i(z) + w_i(z)$ for all $i \in [0, m]$.

a linear combination of the new prover-key elements, it must use the same coefficients for E_{t-1}, E_t, and D_t. Otherwise, due to random parameters T_i and R_i, the verification equation would not be balanced with high probability. Only if the prover provided the commitments correctly, the additional linear combinations of T_i and R_i will cancel out. Recall that function $s2s$ maps indices from states S to state updates S'. Therefore, above terms for prover-key elements indexed with $i \in S$ (left side) and prover-key elements indexd with $i \in S'$ (center), contain the same random parameters $T_i, i \in S'$, allowing commitments E_t to be either matched with the state or the state update variables. We use additional random secrets $\eta, \kappa, \varepsilon$ to prevent the prover from mixing coefficients across the different commitments.

3.2 Technical Details

Relation R of our SNARK will be a Quadratic-Arithmetic-Program (QAP) [37]. For m variables, a QAP consists of polynomials $\{u_i(Z)\}_{i\in[0,m]}$, $\{v_i(Z)\}_{i\in[0,m]}$, $\{w_i(Z)\}_{i\in[0,m]}$, $t(Z)$. The QAP is satisfied for variables $a_i \in \mathbb{H}$ and constant $a_0 = 1$, if

there exists a polynomial $h(Z)$, such that

$$\sum_{i\in[0,m]} a_i u_i(Z) \cdot \sum_{i\in[0,m]} a_i v_i(Z) - \sum_{i\in[0,m]} a_i w_i(Z) = h(Z)t(Z). \quad (1)$$

QAPs have turned out to be useful in practice, as there exist straight-forward reductions from Arithmetic Circuits and from Rank-1-Constraint-Systems (R1CS), a popular type of relation to represent computations [6, 53].

Discussion Many of the constructions by related works ([3, 17, 24]) require a special non-degeneracy condition for the QAP, which is that the polynomials for $i \in S \cup S'$ or $i \in X$ are linearly independent. This is usually achieved by augmenting the relation by dummy constraints (see e.g. [52]), which increase degree d of the QAP. This is accounted for as arithmetization overhead in Table 1. Our construction specifically avoids this requirement.

Construction In ADSC-SNARKs, public parameters $\mathrm{pp} = (QAP, s2s, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathbb{F}_p, r)$ comprise the tuple of relation $\mathrm{R} = QAP$ and the state mapping function $s2s$, which define the computation to be proven. Furthermore, they comprise the descriptions of a pair of groups $\mathbb{G}_1, \mathbb{G}_2$ with a bilinear map e and target group $\mathbb{G}_T$, and a prime field $\mathbb{H} = \mathbb{F}_p$ such that $|\mathbb{F}_p| = p = |\mathbb{G}_1|$ and $|p| = \lambda$. Parameters pp also include the number of iterations $r \in \mathbb{N}$.

Also, ADSC-SNARKs employ a standard digital signature scheme $((sk, pk) \leftarrow \mathsf{SigGen}(\lambda)$, $sig \leftarrow \mathsf{SigSign}(sk, m)$, $v \leftarrow \mathsf{SigVerify}(pk, m))$ for computing and verifying signature sig on message m using public key pk and private key sk.

Figure 1 shows the protocol of our ADSC-SNARK. The differences to the Groth16 SNARK are highlighted. Here, we ignore prover state p_t, as it will appear only in the zero-knowledge version of our ADSC-SNARKs. To improve readability, we write $P_i(z)$ for $\beta u_i(z) + \alpha v_i(z) + w_i(z)$ for all $i \in [0, m]$.

3.3 Security Proof

We prove our construction secure in the *generic group model*. The full security proof is deferred to A. Here, we sketch the intuition behind the soundness of our ADSC-SNARK.

Knowledge-soundness requires that whenever the verifier $\mathcal{V}$ accepts a proof, the following conditions hold:

(a) the relation is satisfied ($(\phi_t, x_t, s_t, s'_t, \omega_t) \in \mathrm{R}$), and

(b) the state is consistent ($s_t = s'_{t-1}$), and

(c) the private inputs x_t have been authenticated.

Verifier $\mathcal{V}$ checks the following relation between proof elements "in the exponent":

$$A_t \cdot B_t = \alpha \cdot \beta + \sum_{i\in\{0\}\cup\Phi} a_{t,i} \frac{P_i(z)}{\gamma} \cdot \gamma + C_t \cdot \delta + D_t \cdot \varepsilon + E_{t-1} \cdot \eta + E_t \cdot \kappa. \quad (2)$$

All prover-key elements are encoded as group elements in $\mathbb{G}_1$ or $\mathbb{G}_2$. Therefore, the (adversary) prover $\mathcal{P}$ can only compute linear combinations of these elements using group operations. Although $\mathcal{P}$ may also evaluate the bilinear map, it can be shown that this yields no additional useful information due to disclosure-freeness [41].

The secret values $(\alpha, \beta, \gamma, \delta, \varepsilon, \eta, \kappa)$ appearing in (2) are sampled at random during Setup. These constrain $\mathcal{P}$ in how it can construct verifying proofs. For example, in (2), the term C_t is multiplied by δ, while no other proof element is. Therefore, in order to cancel out δ, $\mathcal{P}$ must construct C_t as a linear combination of the prover-key elements that already contain a factor of δ^{-1}, such as $\left\{\frac{P_i(z)(z)}{\delta}\right\}_{i\in\Omega}$.

Note, how the structure of (2), in particular the product on the left hand side, mirrors the quadratic constraints of the QAP (1). In fact, the only way for $\mathcal{P}$ to ensure that the proof satisfies (2), is to choose the coefficients a_i of its linear combinations so that they form a valid solution to the QAP. This establishes condition (a).

Next, consider the prover-key elements contributing to C_t. Some of these contain random parameters R_i and T_i. For $i \in S'$, the corresponding terms include an additional factor κ, which also multiplies E_t in (2). In order to cancel out R_i in the verification equation, $\mathcal{P}$ is forced to use the same coefficients $a_{t,i}$ for $i \in S'$ when constructing both C_t and E_t. Therefore, E_t commits to s'_t. The same applies to coefficients $a_{t,i}$ for $i \in S$ (the elements of s_t) and E_{t-1}. Therefore, the verification equation enforces $s_t = s'_{t-1}$, establishing condition (b).

Finally, the random parameters T_i similarly force $\mathcal{P}$ to use identical coefficients a_i for $i \in X$ when constructing C_t and D_t. Since D_t commits to the private input and $\mathcal{V}$ verifies that D_t was signed by $\mathcal{D}$, the inputs must have been authenticated by $\mathcal{D}$. This establishes condition (c).

In conclusion, except with negligible probability, any prover $\mathcal{P}$ that outputs a verifying proof must satisfy the relation, maintain state consistency, and use authenticated inputs.

3.4 Zero-Knowledge

We present the zero-knowledge variation of our main construction in B.

3.5 Extension: Multiple Authenticators

It is possible to have several independent authenticating parties J which each can only authenticate a subset X_j of the inputs, without the need of the authenticating parties to communicate with each other. In order to ensure that inputs have been provided by the correct authentication party,
$\mathsf{Setup_a}$ produces multiple key pairs for the signature scheme:

$$(sk_j, pk_j) \leftarrow \mathsf{SigGen}() \text{ for } j \in J$$

and distributes authentication key $\sigma_{a,j} = (sk_j, \{T_i\}_{i \in X_j})$ to authenticating party j.
Each authenticating party authenticates its input subset

$$\nu_{j,t} \leftarrow \mathsf{Auth}(\sigma_{a,j}, t, x_{j,t})$$

and sends it to the prover.
The prover includes all $[D_j]_1, sig_j$ from $\nu_{j,t}$ in proof π_t.
The verifier checks the signatures of every $[D_j]_1$:

$$\mathsf{SigVerify}(pk, sig_{j,t}, ([D_{j,t}]_1, t)) = 1.$$

It computes $[D_t]_1 = \sum_{j \in J} [D_{j,t}]_1$ and uses it to check the original verification equation. It rejects the proof, if any of the checks fails and accepts otherwise.

Above extension will increase the proof size by $|J|-1$ signatures and $\mathbb{G}_1$ elements. It will also increase verification time by the additional $|J|-1$ signature verifications.

4 Evaluation

We compare the performance of the full zero-knowledge version of our ADSC-SNARK against the folklore approach where a signature verification algorithm and a hash function are arithmetized for a regular SNARK (*Strawman*). We use this as the primary baseline because there exists not yet a specialized scheme, where both state-consistency and data authenticity are considered. However, to provide a fair comparison against state-of-the art techniques, we compare our ADSC-SNARK with four more recent, specialized SNARKs. First, we compare it with AD-SNARK [3], SPHinx [32] and VerITAS [25] which allow proofs for computations on authenticated data. Second, we compare it against construction that we name *LegoGro SC-SNARK*. The latter is a composition of two LegoGro16 SNARKs from the LegoSNARK framework [17] achieving state-consistency through functional composition.

We stress that neither AD-SNARKs, SPHinx, VerITAS nor LegoGro SC-SNARKs provide the same properties as the full ADSC-SNARK construction. They *either* provide authentication *or* state consistency. They also cannot be trivially combined to achieve both properties, as for example AD-SNARKs use one-time message authentication codes for the input data. This would be insecure in an iterative setting where authenticated inputs could be reused.

4.1 Implementation Details

We have implemented our ADSC-SNARK, the strawman ADSC-SNARK and the LegoGro SC-SNARK in the popular libsnark C++ library[2]. We instantiate the signature scheme by EdDSA on an Edwards curve of similar prime order group size as the SNARK curve.

The **strawman ADSC-SNARK** for relation R is a Groth16 SNARK for an augmented relation $\mathrm{R}_a = \mathrm{R} \wedge \mathrm{R}_x \wedge \mathrm{R}_s \wedge \mathrm{R}_{s'} \wedge \mathrm{R}_i$. Here, R_x encodes a signature verification for the private input x. Relations R_s and $\mathrm{R}_{s'}$ encode collision resistant hash functions applied to state s and state-update s'. Finally, R_i encodes a counter-increment preventing that signed private input can be reused across iterations. The hash outputs are made public input-output variables of the Groth16 SNARK and are considered part of the proof of the strawman ADSC-SNARK. State consistency can be checked in the verification algorithm by checking equality between corresponding state hashes additionally to the SNARK verification equation.

We base R_x on the EdDSA verification gadget in the open-source library ethsnarks[3]. To further reduce its size, we use the SNARK-friendly Poseidon hash function [40] and fixed-base scalar multiplication in affine montgomery coordinates. This results in a constant number of 2126 constraints per signature and additionally 72 constraints per private input. Note that these numbers are quite competitive in favor of the Strawman approach, as previous works estimated

[2] https://github.com/scipr-lab/libsnark
[3] https://github.com/HarryR/ethsnarks

5000 constraints per signature [32]. We use the same Poseidon hash for realizing R_s with 144 constraints per state variable. We assume that there is a single signature for all inputs when the Strawman approach is taken. This is the most conservative assumption, so speedups will likely be higher in practice than the reported numbers.

Backes et al. provide a libsnark implementation of their **AD-SNARK** construction, supporting both a designated and a public verifier version. We benchmarked against both versions.

Fiore and Tucker provide an implementation of **SPHinx** using another popular cryptographic proof systems library called "arkworks"[4]. Their implementation includes all components of SPHinx except the implementation of the base SNARK Marlin. Therefore, we separately benchmarked the Marlin implementation by Chiesa et al.. The reported numbers show the combined results.

For **VerITAS**, we benchmarked the implementation provided by Datta et al.. Their implementation uses yet another proof systems library called "plonky2"[5].

For **LegoGro SC-SNARK**, the LegoSNARK framework by Campanelli et al. [17] enables the composition of proofs including function composition. That is, for functions $y = f(x), z = h(y)$ and SNARK algorithms $SNARK_f, SNARK_h$, the framework provides a compiler to construct a new SNARK from $SNARK_f$ and $SNARK_h$ to create proofs for the relation representing $z = h(f(x))$. For our setup, we can view the iterative application of some function f on state s as a function composition of f with itself and therefore use the LegoSNARK framework to achieve state-consistency. For that, we use the LegoGro16 primitive described by Campanelli et al.. We define two index sets for state s and state-update variables s' which is equivalent to two subdomains in their paper. We implemented LegoGro16 using the libsnark library and apply their composition scheme from their Theorem 3.1 recursively to get LegoGro SC-SNARK.

4.2 Benchmarks

We have run the benchmarks on an i5-3570K Intel Core Processor with a Ubuntu 24.04 operating-system and 24 GB RAM. We have compiled the C++ code for our ADSC-SNARK, the strawman SNARK, AD-SNARK and LegoGro SC-SNARK with the GCC C++ compiler on optimization level 3, and the Rust code for SPHinx and VerITAS using rustc, also on optimization level 3. For all proof systems that use pairing-friendly elliptic curves, we selected the BN254 elliptic curve which is a standard choice. Only VerITAS does not require pairing-friendly curves but instead operates on the Goldilocks field, a prime field of order $2^{64} - 2^{32} + 1$.

We ran the benchmarks within a single thread for better comparability. We have constructed a basic R1CS relation with 2^{15} constraints for all proof systems except VerITAS. SPHinx directly proves R1CS relations. For the strawman SNARK, AD-SNARK, ADSC-SNARK and LegoGro SC-SNARK, the R1CS relation was reduced to a QAP relation. VerITAS uses an arithmetization scheme introduced in [34] for the PLONK proof system. As programs can usually be more efficiently arithmetized with this scheme, we used a relation with 2^{10} PLONK gates for benchmarking VerITAS. The relations also contain up to $|X| = 2^{20}$ private inputs and $|S| = 2^{20}$ states. We evaluated the Strawman SNARK for up to $|X| = 2^{14}$ private inputs and $|S| = 2^{14}$ states as for a larger number of inputs and states, our 24 GByte RAM machine runs out of memory when running the Strawman SNARK. The number of public input-output variables was set to 0.

Note: We stress that only ADSC-SNARK and the strawman SNARK both provide input authenticity and state consistency. We only compare against the other proof systems to put the performance of ADSC-SNARKs into perspective.

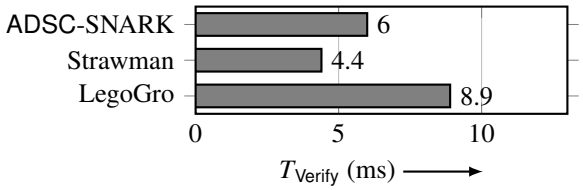

Figure 2: Verifier runtimes of our zero-knowledge ADSC-SNARK, LegoGro SC-SNARK and the strawman SNARK. All displayed SNARKs have constant verification time.

Figures 2 and 3 compare the verifier runtimes T_{Verify} of the different proof systems. The verifier runtimes of our ADSC-SNARK, the strawman SNARK, and LegoGro SC-SNARK are constant with respect to input size $|X|$, state size $|S|$ and size of the relation. The ADSC-SNARK verifier needs 54 % less runtime than VerITAS, 70 % less than the designated verifier AD-SNARK, 75 % less than the public verifier AD-SNARK and 84 % less than SPHinx in the worst case of $|X| = 1$. As the designated-verifier AD-SNARK iterates over all private input labels, the verifier is

[4] https://github.com/arkworks-rs
[5] https://github.com/0xPolygonZero/plonky2

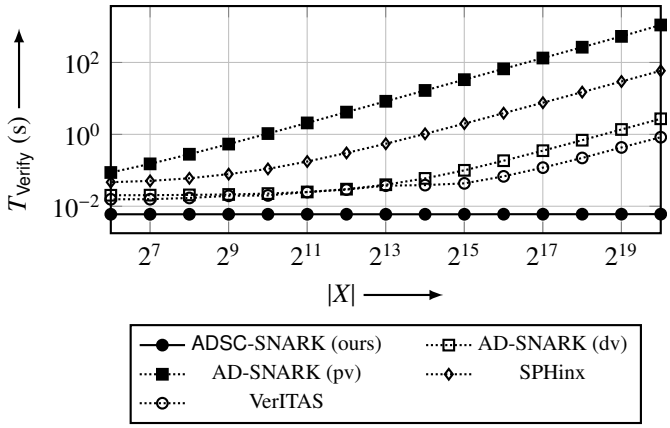

Figure 3: Verifier runtimes of our zero-knowledge ADSC-SNARK, AD-SNARK designated verifier (dv) and public verifier (pv), SPHinx and VerITAS. Verifier runtimes of all but our ADSC-SNARK increase with the number of private inputs $|X|$.

Table 2: Proof size comparison for SNARKs with constant-sized proofs, for curve BN254

System	Elements	Size
Strawman SNARK	$2 \times \mathbb{G}_1, 1 \times \mathbb{G}_2, 1 \times \mathbb{F}_r$	166 B
LegoGro SC-SNARK	$5 \times \mathbb{G}_1, 1 \times \mathbb{G}_2$	236 B
AD-SNARK (dv)	$10 \times \mathbb{G}_1, 1 \times \mathbb{G}_2$	406 B
SPHinx	$24 \times \mathbb{G}_1, 19 \times \mathbb{F}_r$	1343 B
ADSC-SNARK	$5 \times \mathbb{G}_1, 1 \times \mathbb{G}_2, 1 \times sig$	271 B

asymptotically linear with respect to the number of private inputs and therefore will increase in settings with many inputs. This is even more significant for the public verifier version, where the verifier needs to compute a linear number of pairing evaluations with respect to $|X|$ (see Table 1 for details). For $|X| = 2^{15}$ private inputs, T_{Verify} is 101.4 ms for the designated verifier version and 34.9 s for the public verifier version. Also, both the verification runtimes of VerITAS and SPHinx increase with $|X|$. The strawman SNARK has the fastest verifier runtime as it invokes the unmodified Groth16 verifier with a hash digest as a public input.

Our ADSC-SNARK is faster than the LegoGro SC-SNARK (33 % less runtime), because the LegoGro16 verifier has additional overhead to allow checking for state-consistency between different proof systems, while our construction is optimized to the setting of a single proof system and a single relation.

Table 2 presents the proof sizes for each proof system with constant-sized proofs. The center column shows the elements of each proof, the right column shows the actual size after serialization when using the BN254 curve. The serialization routines compress elliptic curve points by storing just one coordinate and a sign bit. However, there is additional overhead for paddings to align objects to 8-bit data chunks and for additional separation tokens between data objects. While the strawman SNARK achieves the smallest proofs, our ADSC-SNARKs and LegoGro SC-SNARK are still smaller than the designated verifier AD-SNARK. The proofs of VerITAS and the public-verifier AD-SNARK are significantly larger and increase with $|X|$ (Figure 4).

The left plot in Figure 5 compares the prover runtime with respect to the number of private inputs. The prover runtime for the designated and public AD-SNARK verifier are the same. For 2^{14} inputs, our ADSC-SNARK is 29× faster than the strawman SNARK and 1.6× faster than AD-SNARK. Both SPHinx and VerITAS have larger prover runtimes. The right plot compares the prover runtime with respect to the number of states. For 2^{14} states, our zero-knowledge ADSC-SNARK is 39× faster than the strawman SNARK and 1.2× faster than LegoGro SC-SNARK. Note that AD-SNARK, SPHinx and VerITAS do not support state-consistency and LegoGro SC-SNARK does not support input authenticity. Table 3 shows the prover runtimes of the strawman SNARK and our zero-knowledge ADSC-SNARK as

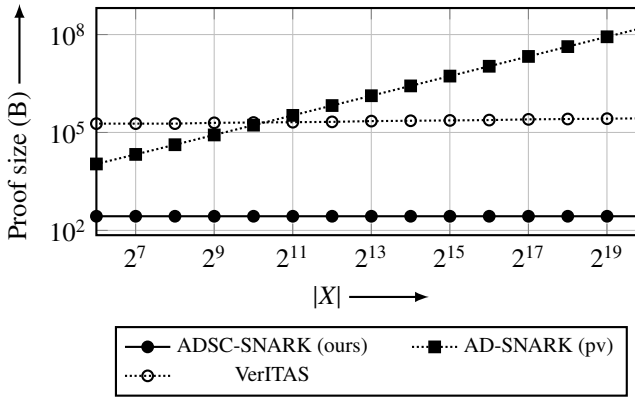

Figure 4: Proof sizes of ADSC-SNARK, public-verifier AD-SNARK and VerITAS. The public verifier AD-SNARK proof size depends on $|X|$, the proof size of VerITAS depends both on $|X|$ and the size of the relation, and was evaluated for 2^{10} PLONK gates.

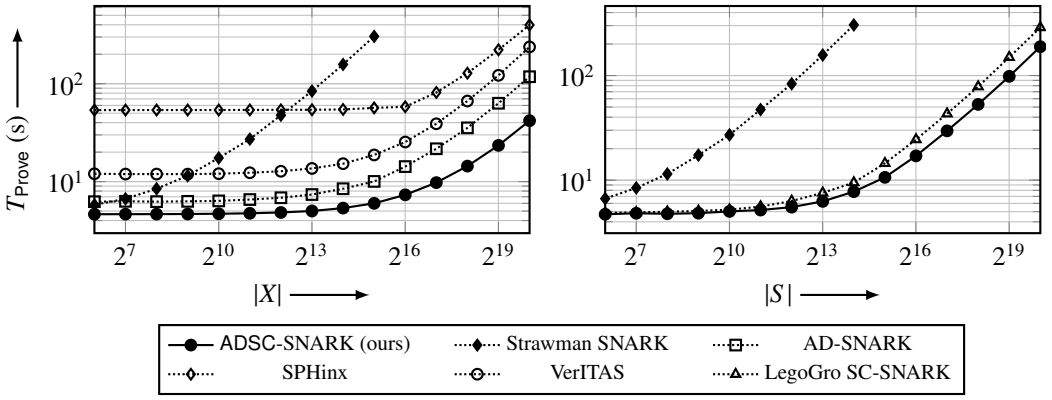

Figure 5: Prover runtime. Left: varying private inputs. Right: varying number of states. Public-verifier and designated-verifier AD-SNARK have the same proving time.

well as the ratio between the runtimes with respect to both the number states and private inputs. The number of private inputs and states is at most 2^{14} due to the heavy memory requirements of the strawman SNARK. To set the effects of the input and state size on the prover runtime in relation to the constraint size of the base relation, we chose a smaller relation with 2^{10} constraints. We do not include the other baselines, as they only support either states or private inputs but not both. The table shows a significant speedup in prover time that increases with the number of states and inputs, for 2^{14} inputs and states, it faster by a factor of $\times 89$.

Summarizing, our construction scales well with the number of private inputs and states and has good concrete prover performance. This comes at the cost of a small increase in proof size and verifier time. This is a deliberate design trade-off. The constant-time verification of 6 ms and constant proof size below 300 B are viable parameters for many practical applications, while the significant reduction in prover time makes otherwise infeasible real-time applications possible.

5 Applications

ADSC-SNARKs have been designed for the use in digital control systems. A typical digital control system comprises sensors that measure some state of the system to be controlled, a control unit that uses sensor measurements to compute control outputs, and actuators that manipulate the system accordingly. In most cases, control units have an internal state that is periodically updated and compute outputs at a fixed frequency. Therefore, the typical computation of a digital control unit is stateful and applied to many inputs.

Prominent examples are flight control in aircraft,

Table 3: Prover runtime for strawman SNARK vs. ADSC-SNARK and speedup factor for varying states and inputs for a small relation with 2^{10} constraints. $|X|$: number of inputs, $|S|$: number of states.

$\|S\|$	2^0			2^7			2^{14}		
$\|X\|$	Strawman	Ours	Speedup	Strawman	Ours	Speedup	Strawman	Ours	Speedup
2^0	0.64 s	0.25 s	×2.5	3.80 s	0.32 s	×11.9	303 s	4.38 s	×69
2^7	2.46 s	0.27 s	×9.1	5.05 s	0.33 s	×15	303 s	4.30 s	×70
2^{14}	154 s	1.04 s	×147	155 s	1.11 s	×140	447 s	5.03 s	×89

electrical power grid control, power generation plant control or control in automated industrial processes. Many of these systems are safety critical, i.e., a malfunction or adversarial tampering can lead to loss of lives or cause severe economic consequences. In order to meet rising demands for efficiency, autonomy, and adaptability of such systems, their complexity and interconnectivity increases. This results in new attack surfaces for malicious actors and also introduces new sources for non-malicious but unintended system failures due to design errors or random faults in components. The discovery of sophisticated cyber-attacks on industrial control systems, including Stuxnet [29] or Duqu [21], demonstrates the importance of securing these systems.

Applying ADSC-SNARK in such a setting is beneficial for several reasons. First, SNARKs could be used as a security measure to prevent cyber-attacks on control units or communication channels. The idea would be to require the commands from a control unit to carry a proof about their correctness. Second, in highly critical areas such as aviation, hardware failures of control units can lead to complex and unpredictable failure behaviour which must be mitigated [26]. Current solutions to detect these failures require using redundant computers and comparing their outputs. SNARKs could be used as an alternative for detecting malfunctions reducing the number of redundant components [55]. We further discuss the application of ADSC-SNARK for securing a power grid control system and for safety in a flight control system.

5.1 Power Grid Control

A power grid enables the distribution of electric power from producers such as power plants, wind- or solar farms to power consumers, such as factories or individual households. Power grid control ensures an equilibrium between power demand and power production. In this setting, sensors measure loads and AC-frequency at various locations in the grid. Power producers act as the control system's actuators. A control center (the control unit) commands producers to increase or decrease power production depending on the demand of consumers and current loads on the transmission lines [4]. For example, the Polish power grid has 114 generators that are controlled automatically and in real-time by a central control unit over a wide area network spanning the entire country and connecting to neighboring states [42]. For these systems, many local and remote access points exist, creating a large attack surface [28]. SNARKs could be beneficial in this setting. The idea would be to require commands from the control unit to carry a proof which asserts that the commands are the output of the correct global control function that stabilizes the power grid. In case of an invalid proof, a power producer could request commands from a backup control unit or enter a fail-safe mode. This measure would limit the capabilities of an attacker, even if the attacker had full authority over the central control unit.

5.2 Flight Control System

Fly-by-wire control is standard equipment of modern large aircraft. It is realized by a flight control system consisting of a set of redundant control units (flight control computers), actuators manipulating various control surfaces (rudders, elevators, ailerons) and sensors measuring flight conditions (aircraft velocity and attitude) and pilot commands.

A malfunction of the flight control system is considered catastrophic, as it can result in uncontrollability and therefore loss of the airplane. As a result, reliable fault detection and mitigation techniques need to be employed. The currently prevalent method in safety-critical avionics to ensure computational integrity is replicating control computers, which compare each other's outputs to detect hardware faults [59, 63]. Replicating computers for ensuring computational integrity results in more weight for electronics and cabling and in high development cost. For example, the flight control system of Airbus A340 consists of 5 duplex (10 in total) flight control computers [59], and the flight control system of Boeing 777 consists of 3 triplex flight control computers (9 in total) [63].

We suggest to employ ADSC-SNARKs and demonstrate their practicality in a flight control system, not to protect against cyber-attacks, but against hardware faults. With ADSC-SNARK, commands from such a possibly faulty control computer can be verified before being accepted by actuator electronics. As a consequence, the number of required redundant computers and thus total cost can be reduced. Note that this approach is not supposed to reduce computational power but rather the number of individual computers. Furthermore, ADSC-SNARKs cannot reduce the number of computers to one, as there still need to be redundant units to meet hardware availability requirements.

Threat model: While some hardware faults in avionics systems can be trivially detected, some can exhibit unexpected and seemingly implausible behavior [26]. For demonstrating compliance to regulatory aircraft safety requirements, it is standard practice to assume worst-case "byzantinian" behavior of faulty computers. Such a conservative approach is much more economically viable than empirically evaluating the unmanageably large space of possible failure modes of a digital computer [46]. We can follow this paradigm in our setting by modeling a faulty computer of the flight control system as a malicious adversary, deliberately trying the actuator electronics to accept invalid control commands. Note, however, that the computational resources of the adversary are severely limited: It must periodically provide a verifying proof every few milliseconds and has only the computational power of the available hardware. This limitation allows to drastically reduce the security parameter in this setup.

To illustrate the concept, we arithmetized the computation of a flight control law inspired by the Airbus A320 aircraft and simulate the generation and verification of ADSC-SNARK proofs thereof.

5.2.1 Simulation

Our scenario (Fig. 6) consists of four sensor units. There are two air data and inertial reference units (ADIRUs) that measure airflow (velocity V, angle-of-attack α, side-slip angle β), attitude (roll angle Φ, pitch angle Θ), rotation (roll rate p, pitch rate q, yaw rate r), and vertical acceleration (n). In addition there are two transducer units that measure the position of the pilot's sidestick controls for pitch (q_c) and roll (p_c) commands, and the pedals for sideslip commands β_c. In each time-step, every sensor unit authenticates their measurements and sends them to the central flight control computer. The flight control computer evaluates a stateful flight control law, taking the measurements from the sensors as inputs. The resulting control commands are sent to the actuators for elevators (elevator deflection Δq), ailerons (aileron deflection Δp), rudder (rudder deflection Δr) and trimmable horizontal stabilizer (THS, pitch trim $\Delta\tau$). They also produce a proof-commitment pair (π, c) for each iteration that allow the actuator units to verify the correctness of the commands. We do not simulate the dynamics and kinematics of the aircraft. Also, we do not simulate an additional backup flight control computer, which would be required for increased availability, once a failure of the first control computer has been detected.

Discussion For this setting, the succinctness properties of ADSC-SNARK are particularly important. While the flight control computer can be expected to have a decent amount of computational power and access to all the required input data from various sensors and other systems (either trough connections via a central aircraft data network or a multitude of several field busses), the actuator electronics can be expected to be fairly computationally weak and to have low bandwidth data connections. Any solution where verification time and proof size increase with the size of the state, (private) input data or the size of the control law would be unsuitable, when the number of inputs, states, and the size of the law are large. This is why in our solution verification time and proof size are constant in these metrics. However, the concrete computational cost for verification is still challenging, due to the costly pairing evaluations. Practical experiments by Andreica et al. demonstrated feasibility of computing pairings on an automotive-grade microcontroller, where a pairing on a 158-bit Barreto-Naehring curve was reportedly evaluated in 40.9 ms, requiring 27 kB of flash program memory.

5.2.2 Control Law

The specific control law evaluated and proven by the control unit is shown as a block diagram in Figure 7. It contains major elements of the Airbus A320 normal flight control law described in references [23, 30, 31] and the A320 flight crew operating manual: Lon and Lat Control are the basic control laws for the longitudinal and lateral aircraft motions. Lon Control integrates the error between pitch command and measured pitch for stationary accuracy and mixes the measurement signals corresponding to the pitch movement (n, q). Lat Control integrates the roll command and mixes the signals corresponding to the roll and yaw movements (p, r, β, Φ). The Trim block integrates the pitch command resulting in a THS de-

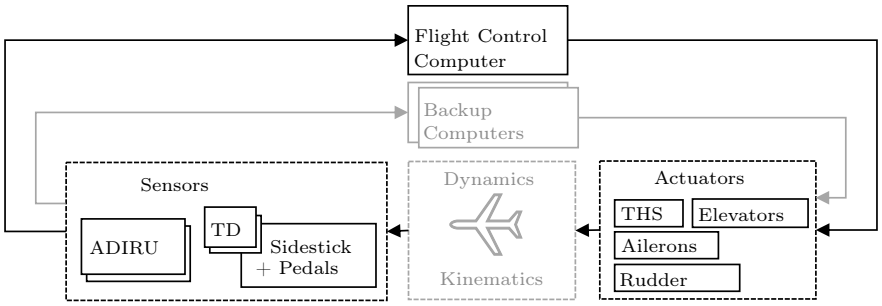

Figure 6: Flight-control scenario depicting signal flows between sensors, control unit and actuators.

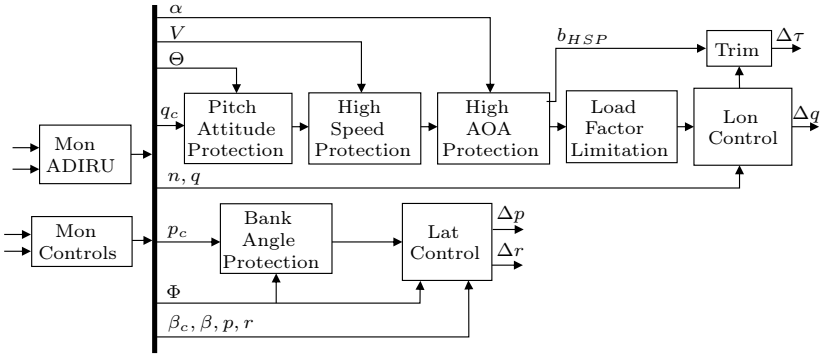

Figure 7: Block diagram of demo flight control law inspired by Airbus A320 flight control.

flection command which results in a steady pitch moment. Due to the integrators, all three blocks are stateful. The pitch attitude protection limits the pilots control inputs, if the pitch angle exceeds 30° nose up or 15° nose down. The high-speed protection adds a proportional pitch-up command to reduce aircraft speed close to the structural speed limits. A hysteresis (activation of the protection is at higher speeds than deactivation) makes this block stateful. The high angle of attack protection changes the pilot input command from a vertical acceleration command to a direct angle-of-attack command in case of a high angle-of-attack. It also limits the pitch command to prevent a stall condition. It is only deactivated when the pilot pushes the stick by more than 8° pitch down or by 0.5° for more than 0.5 s. This condition makes this block also stateful. The load factor limitation limits the pitch command to prevent maneuvers that would exceed the structural limits of the aircraft. Finally, the bank angle protection changes a roll rate command from the side stick to a bank angle command, if the aircraft exceeds a bank angle of 33° and limits the bank angle to at most 67°. We do not claim that this represents the complete flight control function of a commercial transport aircraft, in fact, there are various additional modes for other operational conditions, such as a ground mode, flare mode, abnormal attitude laws and alternate laws with reduced protections. We also do not consider spoiler, engine, slat and flap commands. We use fixed constants for control gains that do not reflect the actual gains, which are not publicly available.

5.2.3 Evaluation

Each physical quantitiy is represented as a finite field element, resulting in $|X| = 24$ private input variables (2×9 signals from ADIRU $+2 \times 3$ pilot control signals), $|S| = 6$ states (3 discrete integrators, 3 states for hysteresis in protection blocks), $|\Phi| = 4$ public input-output variables and $|\Omega| = 1506$ witness variables. The resulting QAP is of degree $d = 1593$.

We have measured I) the runtime for the control unit to process the input data and to create a proof for a valid computation, and II) for the actuation units to verify the commands from the control unit. This includes serializing, writing, reading and deserializing exchanged data in addition to executing the ADSC-SNARK algorithms. Additionally, the size of the proof including commitment is reported.

For sake of simplicity, the simulation does not run on different computers sending the data over a network, but instead runs the logic of the sensors, control unit and actuation system on the same machine.

Table 4: Runtime and communication of digital control system demo. Proof size includes proof and commitment.

Elliptic Curve	Control Unit runtime	Actuation System runtime	Proof size
BN254	102 ms	8.9 ms	568 B
BN183	50.4 ms	4.3 ms	432 B
BN124	41.4 ms	3.7 ms	296 B
GMV181	39.6 ms	3.8 ms	448 B
GMV97	29.6 ms	2.6 ms	304 B
GMV58	17.3 ms	1.3 ms	160 B

The numbers reported include the walltime time passed including reading and writing in- and output, not just the processor time as in Section 4. It does not include one-time setup and initialization computations, such as deserializing and preprocessing the verifier-key. The reported proof size includes overhead such as separation characters used in the serialization routines

As the untrusted control computer has limited computational resources, one may significantly reduce the security parameter of SNARKs [55]. Consequently, we present benchmarks for a choice of different elliptic curves and security parameters. The smaller curves are presented strictly for the hardware fault-tolerant use case. For applications requiring protection against intentional tampering, standard cryptographic security levels must be selected.

- BN254, BN183, BN124: Barreto-Naehring elliptic curves [5] with embedding degree 12 and 254, 183, and 124 bits prime group order,
- GMV181, GMV97, GMV58: Galbraith-McKee-Valença elliptic curves [35] with embedding degree 6 and 181, 97, and 58 bits prime group order.

Curve BN254 was constructed by Ben-Sasson et al. [9] and is the default curve in libsnark, also referred to as `alt_bn128`. GMV181 is the curve used by Ben-Sasson et al. [8]. We constructed and implemented the other curves according to the methods of Barreto and Naehrig as well as Galbraith et al..

Table 4 shows runtime measurements and the communication overhead for sensors (authentication tag size) and control unit (proof size). The simulation has been run with the same settings as the benchmarks in Section 4. The code is compiled with GCC at optimization level 3 and executed on an Intel Core i5-3570K Processor.

The long runtime for the control unit when using elliptic curves with a large prime group order restricts the update rate in this scenario to frequencies between 1 and 10 Hz, while smaller prime group elliptic curves allow for up to 50 Hz update frequency. Again, we stress that smaller prime group curves do not provide security against a powerful malicious adversary, but can be employed to improve the integrity of computations susceptible to random hardware malfunctions.

Typical update rates for flight control in large aircraft lie in the range of 25 Hz (Space Shuttle [49]) and 100 Hz (Boeing Fly-by-Wire system [13]).

With previous approaches which require arithmetizing signature verifications or hash functions, the additional overhead in proving time would be prohibitive for this application. Other, more specialized approaches, such as AD-SNARK, LegoGro SC-SNARK, SPHinx or VerITAS yield better proving times than arithmetization-based approaches. However, they cannot be directly applied to our setting, as they do not provide both state-consistency and proofs over authenticated data. Our construction is specifically designed for this application setting and provides the lowest proving time as well as very small, constant-sized proofs.

We conclude that for simple control laws, reductions in prover runtime by ADSC-SNARK over previous approaches enable a usage in real-time systems with moderate timing requirements for protection against random hardware faults. For more complex control laws and with requirements to protect against intentional tampering, the prover runtimes currently still limit the applicability of this approach to systems with small update frequencies.

6 Related Work

This work is the first to explicitly consider proofs of stateful computations and on authenticated data, both at the same time without arithmetizing signatures or hashes. In this section, we discuss methods that achieve either one of these properties.

Starting with proofs on authenticated data, some works describe the folklore approach of arithmetizing an entire signature verification algorithm [24, 43] or parts thereof [62]. The first SNARK-based solution that does not require arithmetizing a signature verification algorithm is AD-SNARK by Backes et al. [3]. AD-SNARK comes in two flavors: A designated verifier version and a public verifier version. The latter is, unfortunately, not succinct, as proof size and verifier runtime are linear in the size of signed input

data. More recently, SPHinx [32] was designed, which is a publicly verifiable SNARK for proofs on authenticated data with proof size constant in the size of signed input data. While our construction requires a circuit-specific trusted setup, SPHinx allows for a universal setup, which means that there is only one trusted setup required for all possible circuits up to a specified size. However, verification time is linear in the number of signed data items. VerITAS [25] aims to prove that only a set of allowed transformations was applied to a signed image. They present two modes, where the first mode is a very efficient hashing algorithm to be arithmetized. The second mode is similar to our construction, where the signer signs a commitment to the image. The prover then applies a modified SNARK proving algorithm eliminating the need to arithmetize the commitment algorithm. However, they base their construction on the PLONK [34] proof system, which results in a larger proof size compared to our construction.

A different line of works on proofs of computations over authenticated data are homomorphic message authentication codes [18, 19, 36] and their publicly verifiable counterpart: homomorphic signatures [14, 20]. However, these constructions are either concretely inefficient as they build on fully homomorphic encryption ([36]), or on cryptographic multi-linear maps ([19, 20]), or they are not succinct ([14, 18]). So, either the proof size or the verification time increase at least linearly with the size of the computation. Furthermore, the class of supported computations is more restricted than SNARK-based solutions.

Next, there exist several works on proofs of stateful computations. Our notion of state-consistency stipulates schemes facilitating multiple proofs over a set of shared data which we refer to as the state. One line of work follows the folklore approach of arithmetizing collision-resistant hash functions [15] or more complex structures such as Merkle-trees [7, 10] or RSA acumulators [50]. These approaches significantly increase the size of the relation and therefore prover runtime.

The state-consistency property is also given in the setting of incrementally verifiable computation (IVC) [45, 51, 60], which can be achieved with recursive SNARKs [10] or folding schemes [27, 44, 45]. These allow a verifier to check an entire iterative computation up to the current step with a single proof. However, this is a stronger requirement compared to our setting, where we only check a single iteration at once. IVC provers incur an overhead of proving either a proof verification of the previous iteration or the correctness of compressing (folding) two computational steps.

Gepetto [24] introduces an approach that avoids arithmetizing hash functions. It extends the Pinocchio SNARK [53] to a commit-and-prove scheme, enabling proofs on relations with a shared state. Campanelli et al. [17] extend this idea by making the commitment independent of the relation and the proof system, allowing proofs from different proving schemes to share state. These fit well in our setting, because they can be used to prove stateful computations without an additional recursion or folding overhead. Therefore, our construction closely follows their approaches. However, we do not use commit-and-prove SNARKs as a black box but rather optimize the internals for our specific setup. This allows us to avoid the special non-degeneracy condition of the QAP, required by previous commit-and-prove schemes. There is a large line of work on various other commit-and-proof schemes that could be used for state consistency as well [33, 47, 61].

Combining both state-consistency and proofs over authenticated data can be non-trivial. For example the designated-verifier SNARK in [3] uses one-time message authentication codes for the input data, which would be insecure in an iterative setting without additional measures.

7 Conclusion and Outlook

In this paper, we have introduced ADSC-SNARKs, a generalization of SNARKs that add input data authentication and state consistency over multiple executions. Including these two properties in SNARKs is important for applications in the domain of cyber-physical control systems. ADSC-SNARKs combine authentication of input data with state consistency in an efficient manner by careful modifications and optimizations to previous works on achieving the properties separately. Key novelties include a new technique that entirely avoids increasing the size of the arithmetized relation, the formalization of security properties for the new setting of proofs for stateful computations on authenticated data, and the introduction of a new application domain for SNARKs. Compared to a naïve solution, ADSC-SNARK achieve a 89× reduction in prover time. Compared to recent more efficient approaches, ADSC-SNARK achieve better prover time, verification time, and smaller proof size.

Several avenues warrant further investigation to improve applicability for and beyond real-time control. Future research should address the resilience of implementations on embedded hardware, for example taking into account unreliable communication net-

works and providing concrete timing guarantees. Furthermore, validating ADSC-SNARKs across a wider range of elliptic curves and security parameters is essential to optimize trade-offs between latency and security in various industrial applications. Finally, one could further improve the efficiency of ADSC-SNARKs for scenarios with distributed authentication, where inputs are authenticated by multiple independent entities.

References

[1] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2087–2104. ACM, 2017.

[2] Tudor Andreica, Bogdan Groza, and Pal-Stefan Murvay. Applications of pairing-based cryptography on automotive-grade microcontrollers. In Barbara Gallina, Amund Skavhaug, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, volume 11094 of *Lecture Notes in Computer Science*, pages 331–343. Springer International Publishing, Cham, 2018.

[3] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. ADSNARK: Nearly practical and privacy-preserving proofs on authenticated data. In *2015 IEEE Symposium on Security and Privacy*, pages 271–286. IEEE, 2015.

[4] B. H. Bakken and O. S. Grande. Automatic generation control in a deregulated power system. *IEEE Transactions on Power Systems*, 13 (4):1401–1406, 1998.

[5] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *International workshop on selected areas in cryptography*, pages 319–331. Springer, Berlin, Heidelberg, 2005.

[6] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. Circom: A circuit description language for building zero-knowledge applications. *IEEE Transactions on Dependable and Secure Computing*, 20 (6):4733–4751, 2023.

[7] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 401–413. ACM Press, 2013.

[8] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Proceedings of the 33rd Annual International Cryptology Conference*, pages 90–108. Springer, Berlin, Heidelberg, 2013.

[9] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *23rd USENIX Security Symposium*, pages 781–796. USENIX Association, 2014.

[10] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 276–294. Springer Berlin Heidelberg, 2014.

[11] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ACM Conferences, pages 326–349. ACM, 2012.

[12] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography: 10th Theory of Cryptography Conference*, pages 315–333. Springer, Berlin, Heidelberg, 2013.

[13] Robert Bleeg. Commercial jet transport fly-by-wire architecture considerations. In *Digital Avionics Systems Conference*, pages 399–406. American Institute of Aeronautics and Astronautics, 1988.

[14] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *Advances in Cryptogoly - EUROCRYPT 2011*, pages 149–168. Springer, Berlin, Heidelberg, 2011.

[15] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 341–357. ACM, 2013.

[16] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, Springer eBook Collection, pages 677–706. Springer International Publishing and Imprint Springer, 2020.

[17] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2075–2092. ACM, 2019.

[18] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In *Advances in Cryptology - EUROCRYPT 2013*, pages 336–352. Springer, Berlin, Heidelberg, 2013.

[19] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In Hugo Krawczyk, editor, *Public-key cryptography*, Lecture notes in computer science Security and cryptology, pages 538–555. Springer, 2014.

[20] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014*, Lecture notes in computer science Security and cryptology, pages 371–389. Springer, 2014.

[21] Eric Chien, Liam O' Murchu, and Nicolas Falliere. W32.Duqu: The precursor to the next Stuxnet. In *5th USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2012.

[22] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, Lecture Notes in Computer Science, pages 738–768. Springer, Cham, 2020.

[23] S. G. Corps. Airbus A320 side stick and fly by wire - an update. In *SAE Technical Paper Series*, pages 1263–1275. SAE International, 1986.

[24] Craig Costello, Cedric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy*, pages 253–270. IEEE, 2015.

[25] Trisha Datta, Binyi Chen, and Dan Boneh. VerITAS: Verifying Image Transformations at Scale. *Cryptology ePrint Archive*, 2024.

[26] Kevin Driscoll, Brendan Hall, Michael Paulitsch, Phil Zumsteg, and Håkan Sivencrona. The real Byzantine generals. In *The 23rd Digital Avionics Systems Conference*, pages 6.D.4–61–11. IEEE Operations Center, 2004.

[27] Liam Eagen and Ariel Gabizon. ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances. *Cryptology ePrint Archive*, pages 1–18, 2023.

[28] Göran N. Ericsson. Cyber security and power system communication—essential parts of a smart grid infrastructure. *IEEE Transactions on Power Delivery*, 25(3):1501–1507, 2010.

[29] Nicolas Falliere, Liam O' Murchu, and Eric Chien. W32.Stuxnet dossier, 2011.

[30] J. Farineau and X. Letron. Qualités de vol latéral d'un avion de transport civil équipé de commandes de vol électriques: expérience de l'Airbus A320. In *AGARD Conference Proceedings No.508*, pages 6.1–6.7, 1990.

[31] C. Favre. Fly-by-wire for commercial aircraft: The Airbus experience. *International Journal of Control*, 59(1):139–157, 1994.

[32] Dario Fiore and Ida Tucker. Efficient Zero-Knowledge Proofs on Signed Data with Applications to Verifiable Computation on Data Streams. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022.

[33] Dario Fiore, Cedric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash first, argue later. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1304–1316. ACM, 2016.

[34] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, pages 1–34, 2019.

[35] S. D. Galbraith, J. F. McKee, and P. C. Valença. Ordinary abelian varieties having small embedding degree. *Finite Fields and Their Applications*, 13(4):800–814, 2007.

[36] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, Lecture Notes in Computer Science, pages 301–320. Springer, 2013.

[37] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology – EUROCRYPT 2013*, pages 626–645. Springer, Berlin, Heidelberg, 2013.

[38] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*. ACM, 2011.

[39] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, Lecture Notes in Computer Science, pages 193–226. Springer Nature Switzerland and Imprint Springer, 2023.

[40] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *Proceedings of the 30th USENIX Security Symposium*, pages 519–535. USENIX Association, 2021.

[41] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326. Springer Berlin Heidelberg, 2016.

[42] Jacek Jarmakiewicz, Krzysztof Parobczak, and Krzysztof Maślanka. Cybersecurity protection for power grid control infrastructures. *International Journal of Critical Infrastructure Protection*, 18:20–33, 2017.

[43] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, T-H. Hubert Chan, Charalampos Papamanthou, Rafael Pass, Abhi Shelat, and Elaine Shi. CØCØ: A framework for building composable zero-knowledge proofs. *Cryptology ePrint Archive*, 2015.

[44] Abhiram Kothapalli and Srinath Setty. HyperNova: Recursive arguments for customizable constraint systems. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, Lecture Notes in Computer Science, pages 345–379. Springer Nature Switzerland and Imprint Springer, 2024.

[45] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Advances in Cryptology - CRYPTO 2022*, pages 359–388. Springer, Cham, 2022.

[46] Jaynarayan H. Lala and Richard E. Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, 1994.

[47] Helger Lipmaa. Prover-efficient commit-and-prove zero-knowledge SNARKs. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in cryptology - AFRICACRYPT 2016*, Lecture notes in computer science Security and cryptology, pages 185–206. Springer, 2016.

[48] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, editor, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ACM Digital Library, pages 2111–2128. Association for Computing Machinery, 2019.

[49] G. M. Minott, J. B. Peller, and K. J. Cox. Space Shuttle digital flight control system. *Advanced Control Technology and its Potential for Future Transport Aircraft*, pages 271–294, 1976.

[50] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. Scaling verifiable computation using efficient set accumulators. In *Proceedings of the 29th USENIX Security Symposium*, pages 2075–2092. USENIX Association, 2020.

[51] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *2022 IEEE 63rd Annual Symposium*

on Foundations of Computer Science, pages 1045–1056. IEEE, 2022.

[52] Bryan Parno. A note on the unsoundness of vnTinyRAM's SNARK. *Cryptology ePrint Archive*, pages 1–4, 2015.

[53] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, 2013.

[54] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, SpringerLink Bücher, pages 129–140. Springer-Verlag Berlin Heidelberg, 1992.

[55] Johannes Reinhart, Bastian Luettig, Nicolas Huber, Julian Liedtke, and Bjoern Annighoefer. Verifiable computing in avionics for assuring computer-integrity without replication. In *Digital Avionics Systems Conference 2023*, pages 1–10, 2023.

[56] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, volume 12172 of *Springer eBook Collection*, pages 704–737. Springer International Publishing and Imprint Springer, 2020.

[57] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. *Cryptology ePrint Archive*, 2020.

[58] Victor Shoup. Lower bounds for discrete logarithms and related problems. pages 256–266. Springer, Berlin, Heidelberg, 1997.

[59] Pascal Traverse, Isabelle Lacaze, and Jean Souyris. Airbus fly-by-wire: A total approach to dependability. In *Building the Information Society*, pages 191–212. Springer, Boston, MA, 2004.

[60] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography: Fifth Theory of Cryptography Conference*, pages 1–18. Springer, Berlin, Heidelberg, 2008.

[61] Meilof Veeningen. Pinocchio-based adaptive zk-SNARKs and secure/correct adaptive function evaluation. In Marc Joye and Abderrahmane Nitaj, editors, *Progress in Cryptology - AFRICACRYPT 2017*, Lecture notes in computer science Security and cryptology, pages 21–39. Springer, 2017.

[62] Zhiguo Wan, Zhangshuang Guan, Yan Zhou, and Kui Ren. zk-AuthFeed: How to Feed Authenticated Data into Smart Contract with Zero Knowledge. In *2019 IEEE International Conference on Blockchain*, pages 83–90. IEEE, 2019.

[63] Y. C. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307, 1996.

A Security Proof

We prove that our construction satisfies the three properties completeness, knowledge-soundness, and succinctness following definitions 1, 2, and 3 in the *generic group model*. That is, we assume that an adversary can only perform group operations on group elements (including the pairing operation). In particular, the adversary cannot exploit any structure or representation of the group. Within the generic group model, computing a discrete logarithm is hard [58] and the Groth16 SNARK, which our construction is based on, is knowledge-sound [41]. While the generic group model is admittedly a strong restriction to an adversary, we point out that there exist no SNARKs which can be proven secure via a black-box reduction from falsifiable assumptions [38]. Our proof follows the same type of blueprint and arguments as Groth [41].

Before diving into formal details, we start by sketching the intuition behind our proof. Note that we defer the treatment of zero-knowledge to B.

A.1 Proof Intuition

Verifier $\mathcal{V}$ checks the following relation between proof elements "in the exponent":

$$A_t \cdot B_t = \alpha \cdot \beta + \sum_{i \in \{0\} \cup \Phi} a_{t,i} \frac{P_i(z)}{\gamma} \cdot \gamma + C_t \cdot \delta + D_t \cdot \varepsilon + E_{t-1} \cdot \eta + E_t \cdot \kappa \quad (3)$$

Completeness is given by construction, which can be verified by inserting the terms for proof elements $A_t, B_t, C_t, ...$ in (3), and by comparing the result with QAP relation (1).

We show *knowledge-soundness* in two steps:
First, prover $\mathcal{P}$ is forced to build a proof from prover-key σ_p, as Equation 3 checks a relation depending on

secret values $\alpha, \beta, z, \ldots$. The only information that $\mathcal{P}$ has on these secrets are the prover-key elements. As operations take place "in the exponent", $\mathcal{P}$ is restricted to only linear operations on prover-key elements σ_p, which are encoded as elements in $\mathbb{G}_1$ or $\mathbb{G}_2$. While it is possible to also evaluate the bilinear map on these elements, one can show that $\mathcal{P}$ does not gain any useful information from doing so (disclosure-freeness[41]).
Second, as $\mathcal{P}$ can only compute linear combinations of the prover-key, we can parameterize the set of all possible proofs it can produce. By inserting into (3) and comparing coefficients on both sides, one can show that all possible proofs satisfying (3) contain proof elements with coefficients a_i that also satisfy the original QAP. Hence, $\mathcal{P}$ must know these a_i that satisfy QAP. Similarly, (3) can only be satisfied, if those coefficients representing the state $(a_i)_{i\in S}$ in the current proof and coefficients representing the state update $(a_i)_{i\in S'}$ are equal to the coefficients in commitments c_{t-1} (proof element E_{t-1}) and c_t (E_t), which ensures state consistency. Finally, for (3) to hold, coefficients $(a_i)_{i\in X}$ must equal the committed coefficients in proof element D. By letting the verifier check the signature on proof element D, authenticity is ensured.

A.2 Preliminaries

We require a few security definitions for proving security of our ADSC-SNARKs. First, the construction of our authentication function (and therefore soundness) relies on digital signatures. Recall the standard security definition of a digital signature scheme with generator $(sk, pk) \leftarrow \mathsf{SigGen}(\lambda)$, signing algorithm $sig \leftarrow \mathsf{SigSign}(sk, m)$ and verification algorithm $b \leftarrow \mathsf{SigVerify}(pk, sig, m)$ for message m, private key sk, public key pk, and signature sig, such that:

Definition 4 (Correctness of Digital Signature Scheme)**.** *For all security parameters λ and messages m:*

$$\Pr\left[\begin{array}{c}(sk,pk)\leftarrow\mathsf{SigGen}(\lambda):\\ \mathsf{SigVerify}(pk,\mathsf{SigSign}(sk,m),m)=1\end{array}\right]=1.$$

Definition 5 (Security of Digital Signature Scheme)**.** *A digital signature scheme* (SigGen, SigSign, SigVerify) *is secure, if for all PPT adversaries $\mathcal{A}$ with oracle access to* SigSign*:*

$$\Pr\left[\begin{array}{c}(sk,pk)\leftarrow\mathsf{SigGen}(\lambda);(m,sig)\leftarrow\mathcal{A}^{\mathsf{SigSign}(\cdot)}():\\ \mathsf{SigVerify}(pk,sig,m)=1\wedge m\notin\tilde{M}\end{array}\right]\approx 0,$$

where $\tilde{M}$ is the set of queried messages by $\mathcal{A}$.

Second, we make use of a Pedersen vector commitment scheme for vector v of size n with binding properties:

Setup: $\sigma_c \leftarrow \mathsf{PedSetup}(\lambda)$:
Setup group $\mathbb{G}_1$ with size $|\mathbb{G}_1| = p$ and $|p| = \lambda$.
Pick: $T_i \leftarrow\!\$\, \mathbb{G}_1$ for $i \in [0, n]$, Set $\sigma_c = (T_i)_{i\in[0,n]}$

Commit: $(c, o) \leftarrow \mathsf{PedCommit}(\sigma_c, v)$:
Parse T_i from σ_c.
Pick: $o \leftarrow\!\$\, \mathbb{F}_p^*$, Compute $c = o \cdot T_0 + \sum_{i\in[1,n]} v_i \cdot T_i$.

Verify: $b \leftarrow \mathsf{PedVerify}(\sigma_c, c, v, o)$:
Parse T_i from σ_c.
Output $b = 1$ (accept), iff $c = o \cdot T_0 + \sum_{i\in[1,n]} v_i \cdot T_i$

A Pedersen vector commitment is correct, iff:

Definition 6 (Commitment Correctness)**.** *For all security parameters λ and vectors v:*

$$\Pr\left[\begin{array}{c}\sigma_c\leftarrow\mathsf{PedSetup}(\lambda);\\ (c,o)\leftarrow\mathsf{PedCommit}(\sigma_c,v):\\ \mathsf{PedVerify}(\sigma_c,c,v,o)=1\end{array}\right]=1.$$

A Pedersen vector commitment is binding:

Definition 7 (Security of Commitment)**.** *For every PPT adversary $\mathcal{A}$:*

$$\Pr\left[\begin{array}{c}\sigma_c\leftarrow\mathsf{PedSetup}(\lambda);(c,v,o,\tilde{v},\tilde{o})\leftarrow\mathcal{A}(\sigma_c):\\ \mathsf{PedVerify}(\sigma_c,c,v,o)=1\\ \wedge\,\mathsf{PedVerify}(\sigma_c,c,\tilde{v},\tilde{o})=1\\ \wedge v\neq\tilde{v}\end{array}\right]\approx 0.$$

A.3 Proof Details

We proceed by proving the properties of our ADSC-SNARK from section 3.

A.3.1 Completeness

Theorem 1. *Our* ADSC*-SNARK is complete.*

Proof. We show that for all $t \in [1, r]$, Verify will accept given outputs from Setup, Auth, and Prove for a satisfied relation R with state-consistency.
Step 1.1. Inserting A_t, B_t, C_t, D_t, E_t, and E_{t-1} in (3), multiplying out γ, δ, η, κ, and ε, and subtracting $\alpha\beta$ from both sides, we get

$$\begin{aligned}&\sum_{i\in[0,m]} a_{t,i}u_i(z)\cdot\sum_{i\in[0,m]} a_{t,i}v_i(z)\\ &+\beta\sum_{i\in[0,m]} a_{t,i}u_i(z)+\alpha\sum_{i\in[0,m]} a_{t,i}v_i(z)\\ &=\sum_{i\in[0,m]} a_{t,i}P_i(z)+h_t(z)t(z)-\sum_{i\in S} a_{t,i}\eta R_{s2s(i)}+\sum_{i\in S'} a_{t-1,i}R_i\eta.\end{aligned}$$

Recall that function $s2s$ maps indices from states S to state updates S'. Condition $s_t = s'_{t-1}$ translates to $a_{t,i} = a_{t-1,s2s(i)}$ for $i \in S$. Therefore, the latter two sums are equal and cancel out. Writing out $P_i(z)$ as $\beta u_i(z) + \alpha v_i(z) + w_i(z)$, we get

$$\begin{aligned}
&\sum_{i\in[0,m]} a_{t,i}u_i(z) \cdot \sum_{i\in[0,m]} a_{t,i}v_i(z) \\
&+ \beta \sum_{i\in[0,m]} a_{t,i}u_i(z) + \alpha \sum_{i\in[0,m]} a_{t,i}v_i(z) \\
&= \sum_{i\in[0,m]} a_{t,i}\left(\beta u_i(z) + \alpha v_i(z) + w_i(z)\right) + h_t(z)t(z).
\end{aligned}$$

We further simplify to

$$\begin{aligned}
\sum_{i\in[0,m]} a_{t,i}u_i(z) \cdot \sum_{i\in[0,m]} a_{t,i}v_i(z) - \sum_{i\in[0,m]} a_{t,i}w_i(z) \\
= h_t(z)t(z).
\end{aligned}$$

This equation holds when the QAP is satisfied. Note that we do not need to consider a separate case for E_0, as E_0 in Setup is computed in the same way as E_t for $t > 0$ in Prove.

Step 1.2. According to the correctness property (Definition 4) of a signature scheme, a valid signature verifies:

$$\begin{aligned}
&\mathsf{SigVerify}(pk, sig_t, ([D_t]_1, t)) \\
&= \mathsf{SigVerify}(pk, \mathsf{SigSign}(sk, ([D_t]_1, t)), ([D_t]_1, t)) = 1.
\end{aligned}$$

Step 1.3. As both conditions which the verifier checks are satisfied in every case, it will always accept. □

A.3.2 Knowledge Soundness

To prove knowledge-soundness, we assume the generic group model. In this model, an adversary $\mathcal{A}$ can only manipulate group elements via oracle access to the group operations and the bilinear map. For example, when querying $([\gamma]_2, [\delta]_2, +)$ the oracle returns $[\gamma + \delta]_2$. Although our construction uses asymmetric pairings with $\mathbb{G}_1 \neq \mathbb{G}_2$, knowledge-soundness also holds for symmetric pairings or under the assumption, that there exists an efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. To capture this stronger setting, we grant $\mathcal{A}$ oracle access to such an isomorphism in both directions. For example, when querying the first element of the prover-key $[\alpha]_1$ the oracle returns $[\alpha]_2$.

Theorem 2. *In the generic group model, our* ADSC-*SNARK is knowledge-sound.*

Proof. First (step 2.1), following Groth we show that the reference-string (σ_p, σ_v) is disclosure-free implying that $\mathcal{A}$ does not gain advantage from applying the bilinear map for constructing a convincing proof. Next (steps 2.2 - 2.4), we argue that any adversary producing accepting proofs with non-negligible probability must follow a linear prover strategy. We show that a witness can be extracted, which satisfies state-consistency (step 2.5) and input data authenticity (steps 2.6 - 2.7).

Step 2.1. The reference-string in our construction is disclosure-free (see Definition 4 in [41]): The reference-string elements can be regarded as multivariate polynomials in the randomly sampled secrets $\alpha, \beta, z, \ldots$. Any quadratic test on them (applying the bilinear map) evaluating to zero is either due to one of the following cases:

1. The corresponding polynomial is the zero polynomial. This does not reveal information as the same test on any independently generated reference-string will also evaluate to zero.

2. The corresponding polynomial evaluates to zero only for specific values $\alpha, \beta, z, \ldots$. This occurs with negligible probability by the Schwartz-Zippel lemma.

Hence, $\mathcal{A}$ cannot learn any additional information from applying the bilinear map with non-negligible probability.

Step 2.2. The verifier checks an algebraic relation between the proof elements involving randomly sampled secrets $\alpha, \beta, z, \ldots$. The only information about these secrets available to $\mathcal{A}$ consists of the reference-string (σ_p, σ_v) and the encodings of random parameters $\{[T_i]_1\}_{i\in X}$ (as $\mathcal{A}$ has oracle access to Auth, allowing it to query arbitrary linear combinations of $[T_i]_1$). By the generic group model and the disclosure-freeness of the reference-string, it follows that any adversary that outputs accepting proofs with non-negligible probability must use a linear prover strategy. That is, all proof elements and commitments $[A]_1, [B]_2, [C]_1, [D]_1, [E]_1$ are computed as linear combinations of elements from (σ_p, σ_v) and $\{[T_i]_1\}_{i\in X}$.

We now characterize the set of all expressions that

$\mathcal{A}$ can form under such a strategy. Let

$$\mathbb{P} = \left\{ \begin{array}{c} P_\alpha \alpha \\ +P_\beta \beta \\ +\sum_{i\in[0,m]} P_{(u_i(z))} u_i(z) \\ +\sum_{i\in[0,m]} P_{(v_i(z))} v_i(z) \\ +\sum_{i\in\Omega} P_{(P_i(z)/\delta)} \frac{P_i(z)}{\delta} \\ +\sum_{i\in S} P_{(P_i(z)-\eta R_{s2s(i)})/\delta} \frac{P_i(z)-\eta R_{s2s(i)}}{\delta} \\ +\sum_{i\in S'} P_{(P_i(z)-\kappa R_i)/\delta} \frac{P_i(z)-\kappa R_i}{\delta} \\ +\sum_{i\in X} P_{(P_i(z)-\varepsilon T_i)/\delta} \frac{P_i(z)-\varepsilon T_i}{\delta} \\ +\sum_{i\in[0,d-2]} P_{z^i t(z)/\delta} \frac{z^i t(z)}{\delta} \\ +\sum_{i\in S'} P_{R_i} R_i \\ +\sum_{i\in X} P_{T_i} T_i \end{array} \middle| \begin{array}{c} P_* \\ \in \mathbb{F}_p \end{array} \right\}$$

The coefficients P_* are indexed by the corresponding reference-string terms to simplify notation. Under a linear prover strategy, $\mathcal{A}$ selects proof elements $[A]_1, [B]_2, [C]_1, [D]_1, [E]_1$ such that

$$A, B, C, D, E \in \mathbb{P}.$$

The elements of $\mathbb{P}$ depend on the randomness of the setup procedure, as they involve the secret parameters $\alpha, \beta, \gamma, \delta, \kappa, \eta, \varepsilon, z$, as well as $\{T_i\}_{i\in X}$ and $\{R_i\}_{i\in S'}$. Accordingly, rather than treating them as concrete group elements, we interpret them as multivariate Laurent polynomials in these indeterminates.

For example, the polynomial corresponding to $[C]_1$ is:

$$\begin{aligned} C(\alpha,\beta,\gamma,\delta,\kappa,\eta,\varepsilon,z,\{T_i\}_{i\in X},\{R_i\}_{i\in S'}) = \\ C_\alpha \alpha + C_\beta \beta + \sum_{i\in[0,m]} C_{(u_i(z))} u_i(z) + \\ \sum_{i\in[0,m]} C_{(v_i(z))} v_i(z) + \sum_{i\in\Omega} C_{(P_i(z)/\delta)} \frac{P_i(z)}{\delta} + \dots \ . \end{aligned}$$

For A and B, we factor out the leading coefficients A_α and B_β and mark the scaled coefficients with a dash ($A'_\beta = A_\beta/A_\alpha$, $B'_\alpha = B_\alpha/B_\beta$, $\dots$) yielding:

$$A(\alpha,\beta,\dots) = A_\alpha(\alpha + A'_\beta \beta + \sum_{i\in[0,m]} A'_{(u_i(z))} u_i(z) + \dots).$$

An analogous transformation is applied to B. We will later show that $A_\alpha \neq 0$ and $B_\beta \neq 0$.

Substituting these polynomials into the verification equation yields polynomial identities over the indeterminates. If the resulting polynomials on both sides differ in at least one coefficient, then the probability that they evaluate to the same value under random instantiation of the indeterminates is negligible by the Schwartz-Zippel lemma. Therefore, except with negligible probability, all corresponding coefficients must be equal.

Step 2.3. We start with (3) for timestep t and look at proof elements A_t, B_t, C_t, D_t, E_t and E_{t-1}. To improve readability, we omit index t when it can be determined from the context.

Comparing coefficients of α^2, we get $A_\alpha B_\beta B'_\alpha = 0$, therefore one of the term's factors must be zero. Comparing coefficients of $\alpha\beta$, we get $A_\alpha B_\beta + A_\alpha A'_\beta B_\beta B'_\alpha = A_\alpha B_\beta (1 + A'_\beta B'_\alpha) = 1$. Therefore, neither A_α nor B_β can be zero. Thus, $B'_\alpha = 0$ and $A_\alpha B_\beta = 1$. Comparing coefficients of β^2, we get $A_\alpha A'_\beta B_\beta = A'_\beta = 0$. The polynomial from the left-hand side of the first verification equation then simplifies to:

$$\begin{aligned} LHS = (\alpha + \sum_{i\in[0,m]} A'_{(u_i(z))} u_i(z) \\ + \sum_{i\in[0,m]} A'_{(v_i(z))} v_i(z) \dots) \\ \times (\beta + \sum_{i\in[0,m]} B'_{(u_i(z))} u_i(z) \\ + \sum_{i\in[0,m]} B'_{(v_i(z))} v_i(z) \dots). \end{aligned}$$

Step 2.4. We now demonstrate that the remaining terms in A, which are not multiples of α, $u_i(z)$, $v_i(z)$, δ, ε, η or κ, must be zero, due to the left-hand side product resulting in terms with multiples of β that do not appear as indeterminates on the right-hand side: Comparing coefficients β/γ, we get $\sum_{i\in\{0\}\cup\Phi} A'_{(P_i(z)/\gamma)} P_i(z)/\gamma = 0$. Terms are zero similarly for coefficients β/δ, $\beta\gamma$, βT_i for any $i \in X$ and βR_i for any $i \in S'$. The same holds for the terms in B when comparing coefficients α/γ, α/δ, $\alpha\gamma$, αT_i for any $i \in X$ and αR_i for any $i \in S'$.

We now have

$$\begin{aligned} LHS = \\ (\alpha + \sum_{i\in[0,m]} A'_{(u_i(z))} u_i(z) \\ + \sum_{i\in[0,m]} A'_{(v_i(z))} v_i(z) \\ + A'_\delta \delta + A'_\varepsilon \varepsilon + A'_\eta \eta + A'_\kappa \kappa) \\ \times (\beta + \sum_{i\in[0,m]} B'_{(u_i(z))} u_i(z) + \sum_{i\in[0,m]} B'_{(v_i(z))} v_i(z) \\ + B'_\delta \delta + B'_\varepsilon \varepsilon + B'_\eta \eta + B'_\kappa \kappa). \end{aligned}$$

Comparing the remaining terms with α on both sides, we get (recall that $P_i(z)$ contains terms $\alpha v_i(z)$):

$$
\begin{aligned}
&\alpha\left(\sum_{i\in[0,m]} B'_{(u_i(z))}u_i(z) + \sum_{i\in[0,m]} B'_{(v_i(z))}v_i(z)\right)\\
&= \sum_{i\in\{0\}\cup\Phi} a_{t,i}\cdot\alpha v_i(z) + \sum_{i\in\Omega} C_{t,P_i(z)/\delta}\cdot\alpha v_i(z)\\
&+ \sum_{i\in S} C_{t,(P_i(z)-\eta R_{s2s(i)})/\delta}\cdot\alpha v_i(z)\\
&+ \sum_{i\in S'} C_{t,(P_i(z)-\kappa R_i)/\delta}\cdot\alpha v_i(z)\\
&+ \sum_{i\in X} C_{t,(P_i(z)-\varepsilon T_i)/\delta}\cdot\alpha v_i(z)
\end{aligned}
$$

Renaming coefficients $C_{t,*}$ to $a_{t,i}$, we get:

$$
\begin{aligned}
&\alpha\left(\sum_{i\in[0,m]} B'_{(u_i(z))}u_i(z) + \sum_{i\in[0,m]} B'_{(v_i(z))}v_i(z)\right)\\
&\qquad = \sum_{i\in[0,m]} a_i\alpha v_i(z).
\end{aligned}
$$

We compare similarly the remaining terms with β on both sides. Then the left hand side simplifies to:

$$
\begin{aligned}
LHS = &\begin{pmatrix} \alpha + \sum_{i\in[0,m]} a_i u_i(z) \\ +A'_\delta\delta + A'_\varepsilon\varepsilon + A'_\eta\eta + A'_\kappa\kappa \end{pmatrix}\\
&\times \begin{pmatrix} \beta + \sum_{i\in[0,m]} a_i v_i(z) \\ +B'_\delta\delta + B'_\varepsilon\varepsilon + B'_\eta\eta + B'_\kappa\kappa \end{pmatrix}.
\end{aligned}
$$

Finally, comparing all terms with powers of z, we get:

$$
\begin{aligned}
&\left(\sum_{i\in[0,m]} a_{t,i}u_i(z)\right)\cdot\left(\sum_{i\in[0,m]} a_{t,i}v_i(z)\right)\\
&\quad = \sum_{i\in[0,m]} a_{t,i}w_i(z) + \sum_{i\in[0,d-2]} C_{t,(z^i t(z)/\delta)}z^i t(z)
\end{aligned}
$$

Writing the last sum as $h(z)t(z)$, one can see that the $a_{t,i}$ for $i \in [0,m]$ satisfy the QAP, i.e. $(a_{t,i})_{i\in[1,m]} = (\phi_t, x_t, s_t, s'_t, \omega_t)$, $(\phi_t, x_t, s_t, s'_t, \omega_t) \in$ R.

Next, for any PPT adversary $\mathcal{A}$ which outputs an accepting proof, we construct a PPT algorithm $\mathsf{Extract}$ that outputs these $a_{t,i}$ for $i \in [0,m]$. Per Definition 2, $\mathsf{Extract}$ gets the same input, including random coins as $\mathcal{A}$. As shown above, $\mathcal{A}$ will follow a linear prover strategy and construct proof elements $[A_t]_1, [B_t]_2, [C_t]_1, [D_t]_1, [E_t]_1, [E_{t-1}]_1$ as linear combinations of the reference-string elements and $\{T_i\}_{i\in X}$. For computing these elements, $\mathcal{A}$ invokes the group operation oracle, as required by the generic group model. $\mathsf{Extract}$ follows the same strategy as $\mathcal{A}$, but whenever $\mathcal{A}$ queries the oracle for group operations, $\mathsf{Extract}$ additionally stores the query. This allows $\mathsf{Extract}$ to obtain all coefficients $C_{t,*}$ used by $\mathcal{A}$ to construct $[C_t]_1$. These are identical to $a_{t,i}$ for indices $i \in \Omega \cup S \cup S' \cup X$ (they were simply renamed above). The remaining $a_{t,i}$ for $i \in \Phi$ are in the public input-output ϕ_t provided by $\mathcal{A}$ alongside proof π_t and commitment c_t. By definition, $a_{t,0} = 1$.

In conclusion, for every PPT adversary $\mathcal{A}$ which outputs an accepting proof, there exists a PPT algorithm $\mathsf{Extract}$ which outputs $a_{t,i}$ for $i \in [0,m]$ satisfying the QAP.

Step 2.5. We now show that $\mathcal{V}$ accepting at iteration t implies $s_t = s'_{t-1}$.

Comparing all remaining terms with ηR_i for each $i \in S$, we get:

$$
\begin{aligned}
\forall i \in S : E_{t-1,R_{s2s(i)}}\eta R_{s2s(i)} - a_{t,i}\eta R_{s2s(i)} = 0\\
\Rightarrow \forall i \in S : E_{t-1,R_{s2s(i)}} = a_{t,i}.
\end{aligned}
$$

Similarly, when comparing terms with κR_i for each $i \in S$, we get:

$$
\begin{aligned}
&\forall i \in S : E_{t,R_{s2s(i)}}\kappa R_{s2s(i)} - a_{t,s2s(i)}\eta R_{s2s(i)} = 0\\
&\Rightarrow \forall i \in S : E_{t,R_{s2s(i)}} = a_{t,s2s(i)}\\
&\Rightarrow \forall i \in S : E_{t-1,R_{s2s(i)}} = a_{t-1,s2s(i)}.
\end{aligned}
$$

Combining both results gets us:

$$
\forall i \in S : a_{t-1,s2s(i)} = a_{t,i} \Leftrightarrow s'_{t-1} = s_t.
$$

Step 2.6. Next, we show that an accepting verifier $\mathcal{V}$ implies that the private input must have been authenticated. We first demonstrate that coefficients in proof element D_t equal coefficients $a_{t,i}$ for $i \in X$.

Comparing all remaining terms with εT_i for each $i \in X$, we get:

$$
\begin{aligned}
&\forall i \in X : D_{t,T_i}\varepsilon T_i - a_{t,i}\varepsilon T_i = 0\\
&\Rightarrow \forall i \in X : D_{t,T_i} = a_{t,i}.
\end{aligned}
$$

Step 2.7. As shown in step 2.4, if $\mathsf{Verify} = 1$, then $(\phi_t, x_t, s_t, s'_t, \omega_t) \in$ R, and as shown in step 2.5, if $\mathsf{Verify} = 1$ then $s_t = s'_{t-1}$. For knowledge-soundness it remains to be shown that if $\mathsf{Verify} = 1$, then the tuple (t, x_t) must be part of the set of queries to $\mathsf{Auth}(\cdot)$: $(t, x_t) \in \tilde{X}$. We do so by assuming that $\mathsf{Extract}$ can extract an x_t, such that $(t, x_t) \notin \tilde{X}$ and $\mathsf{Verify} = 1$. From that we construct another adversary $\mathcal{B}$ with oracle access to $\mathsf{SigSign}$, which either breaks security of the signature scheme or the binding property of the pedersen vector commitment scheme. We conclude that such an adversary does not exist.

We define the success probability of $\mathcal{B}$ as:

$$\Pr_{\mathcal{B}} = \Pr\left[\begin{array}{c} (\sigma_a, \sigma_{aux}) \leftarrow \mathsf{Setup_a}(); \\ T_0 \leftarrow\$ \mathbb{F}_p; T_i \leftarrow \mathsf{Parse}(\sigma_a); \\ \sigma_c = (T_0, \{T_i\}_{i\in X}); \\ (t, D, sig, x_t, \tilde{x}, o, \tilde{o}) \leftarrow \mathcal{B}^{\mathsf{SigSign}(\cdot)}(\sigma_c, \sigma_{aux}) : \\ \left(\begin{array}{c} \mathsf{PedVerify}(\sigma_c, D, x_t, o) = 1 \\ \wedge \mathsf{PedVerify}(\sigma_c, D, \tilde{x}, \tilde{o}) = 1 \\ \wedge x_t \neq \tilde{x} \end{array}\right) \\ \vee \left(\begin{array}{c} \mathsf{SigVerify}(pk, sig, (D,t)) = 1 \\ \wedge (D,t) \notin \tilde{M} \end{array}\right) \end{array}\right],$$

where $\tilde{M}$ is the set of queries by $\mathcal{B}$ made to $\mathsf{SigSign}$.

$(t, D, sig, x_t, \tilde{x_t}, o, \tilde{o}) \leftarrow \mathcal{B}^{\mathsf{SigSign}(\cdot)}(\sigma_c, \sigma_{aux})$ works as follows:

Pick any $s'_0 \in \mathbb{F}_p^{|S|}$.
Compute

$$(\sigma_p, \sigma_v, c_0) \leftarrow \mathsf{Setup_{pv}}(s'_0, \sigma_{aux}).$$

Compute

$$((\pi_t, c_t, \phi_t); (x_t, s_t, s'_t, \omega_t)) \leftarrow (\mathcal{A}^{\mathsf{Auth}(\cdot)} \,||\, \mathsf{Extract})(\sigma_p, \sigma_v, t, c_0)$$

for any t, such that $\mathsf{Verify}(\sigma_v, \phi_t, \pi_t, c_t, c_{t-1}, t) = 1$ and $x_t \notin \tilde{X}$, where $\tilde{X}$ is the set of queries given to Auth.

Note that $\mathcal{B}^{\mathsf{SigSign}(\cdot)}$ can answer the queries of $\mathcal{A}^{\mathsf{Auth}(\cdot)}$ to Auth by using the information from σ_{aux} and the oracle access to $\mathsf{SigSign}$. Let $\tilde{M}$ be the set of queries made by $\mathcal{B}$ to the signing oracle.

Now, $\mathcal{B}$ distinguishes two cases:
a.) If there exists $\tilde{x} \in \tilde{X}$, such that $x_t \neq \tilde{x}$ and $\sum_{i\in X} x_{t,i}T_i = \sum_{i\in X} \tilde{x}_i T_i$, output:

$$t, D = \sum_{i\in X} x_{t,i}T_i, \quad o = \tilde{o} = 0, \quad sig = \{\}.$$

b.) Otherwise, parse sig from π_t and output

$$t, D = \sum_{i\in X} x_{t,i}T_i, \quad sig, \quad x_t = \tilde{x_t} = o = \tilde{o} = \{\}.$$

We see that if $\mathcal{A}$ is successful, either of the cases a.) or b.) occurs, resulting in either

$$\mathsf{PedVerify}(\sigma_c, D, x_t, o) = 1 \wedge \mathsf{PedVerify}(\sigma_c, D, \tilde{x}, \tilde{o}) = 1 \wedge x_t \neq \tilde{x_t}$$

to hold (case a), or

$$\mathsf{SigVerify}() = 1$$

to hold (case b).
Therefore, by construction, $\Pr_{\mathcal{B}} \approx \Pr_{\mathcal{A}}$.

It is now easy to see that we can derive from $\mathcal{B}$ another adversary $\mathcal{C}$, which tries to break the binding property of a pedersen commitment of Definition 7. Its success probability is $\Pr_{\mathcal{C}}$.
$(c, v, o, \tilde{v}, \tilde{o}) \leftarrow \mathcal{C}(\sigma_c)$ works as follows:
Parse $\{T_i\}$ from σ_c
Compute σ_{aux} as in $\mathsf{Setup_a}$, but use T_i from σ_c, instead of sampling them.
Compute $(t, D, sig, x_t, \tilde{x_t}, o, \tilde{o}) \leftarrow \mathcal{B}^{\mathsf{SigSign}(\cdot)}(\sigma_c, \sigma_{aux})$.
Set $c = D, v = x_t, \tilde{v} = \tilde{x_t}$.
Similarly, we can derive from $\mathcal{B}$ an adversary $\mathcal{D}$ with oracle access to $\mathsf{SigSign}$, which tries to break the security of the signature scheme of Definition 5. It's success probability is $\Pr_{advd}$.
$(m, sig) \leftarrow \mathcal{D}^{\mathsf{SigSign}(\cdot)}()$ works as follows: Compute

$$(\sigma_a, \sigma_{aux}) \leftarrow \mathsf{Setup_a}()$$
$$T_0 \leftarrow\$ \mathbb{F}_p, T_i \leftarrow \mathsf{Parse}(\sigma_a)$$
$$\sigma_c = (T_0, \{T_i\}_{i\in X}).$$

Run $\mathcal{B}$

$$(t, D, sig, x_t, \tilde{x_t}, o, \tilde{o}, t) \leftarrow \mathcal{B}^{\mathsf{SigSign}(\cdot)}(\sigma_c, \sigma_{aux}).$$

Set $m = (D, t)$.
Output m, sig.

As we assume that the pedersen vector commitment is binding (Definition 7), $\mathcal{C}$ has negligible chance of success: $\Pr_{\mathcal{C}} \approx 0$. As we also assume that the signature scheme is secure (Definition 5), $\mathcal{D}$ has also negligible chance of success: $\Pr_{\mathcal{D}} \approx 0$. Both $\mathcal{C}$ and $\mathcal{D}$ having negligible chance of success means that also $\Pr_{\mathcal{B}} \approx 0$. It follows that $\Pr_{\mathcal{A}} \approx \Pr_{\mathcal{B}} \approx 0$. □

Theorem 3. *Our* ADSC-*SNARK is succinct.*

Proof. A proof π consists of a fixed number of 6 group elements and a signature. The algorithm Verify does $1 + |\phi|$ scalar multiplications, 6 pairing operations, 5 field multiplications, a signature verification and 2 comparisons. □

B Zero-Knowledge

We present a variation of our main ADSC-SNARK construction which is also zero-knowledge.

B.1 Zero-Knowledge Definition

We let algorithm Setup output a trapdoor τ which is passed to a simulator Sim. The simulator does not receive any of the private input x, states s and s' or

witness ω. If adversary $\mathcal{A}$ will not be able to distinguish simulated from real proofs, no information about x, s, s' or ω is leaked by a real proof.

Definition 8 (Zero-knowledge of ADSC-SNARK)**.** *For all $\lambda \in \mathbb{N}$, $\mathrm{pp} \leftarrow \mathsf{Gen}(\lambda)$, there exists a PPT algorithm $(\pi_t, c_t) \leftarrow \mathsf{Sim}(\tau, \phi_t, c_{t-1}, t)$, such that for all $(a_1, ..., a_r) \in \mathrm{R}^r$, with $a_t = (\phi_t, x_t, s_t, s'_t, \omega_t)$, and $s'_0 \in \mathbb{H}^{|S|}$ such that $\bigwedge_{t=1}^{r}(s_t = s'_{t-1})$, and for all adversaries $v \leftarrow \mathcal{A}(\sigma_p, \sigma_v, \sigma_a, \tau, \Pi, C)$:*

$$\Pr\left[\begin{array}{c}(\sigma_p, \sigma_v, \sigma_a, c_0, \tau) \leftarrow \mathsf{Setup}(s'_0);\\ \nu_t \leftarrow \mathsf{Auth}(\sigma_a, t, (a_{t,i})_{i\in X}) \text{ for } t \in [1,r]\,;\\ (\pi_t, c_t, p_t) \leftarrow \mathsf{Prove}(\sigma_p, a_t, \nu_t, p_{t-1})\\ \text{for } t \in [1,r]:\\ \mathcal{A}(\sigma_p, \sigma_v, \sigma_a, \tau, (\pi_1, \pi_2, ..., \pi_t),\\ (c_0, c_1, ..., c_t)) = 1\end{array}\right]$$

$$= \Pr\left[\begin{array}{c}(\sigma_p, \sigma_v, \sigma_a, c_0, \tau) \leftarrow \mathsf{Setup}(s'_0);\\ (\pi_t, c_t) \leftarrow \mathsf{Sim}(\tau, \phi_t, c_{t-1}, t) \text{ for } t \in [1,r]:\\ \mathcal{A}(\sigma_p, \sigma_v, \sigma_a, \tau, (\pi_1, \pi_2, ..., \pi_t),\\ (c_0, c_1, ..., c_t)) = 1\end{array}\right].$$

B.2 Zero-Knowledge Construction

To make proofs zero-knowledge, the authenticating party and the prover add randomization masks b_A, b_B, b_D, b_E to the proof. They are randomly drawn from $\mathbb{F}_p^*$, multiplied with $[\delta]_1$ and added to elements $[A]_1 = [A' + b_A\delta]_1, [B]_2 = [B' + b_B\delta]_2, [D]_1 = [D' + b_D\delta]_1, [E]_1 = [E' + \delta]_1$, where $A', B', ...$ denote the proof elements from the non-zero-knowledge version. The distribution of $([A]_1, [B]_2, [D]_1, [E]_1)$ generated by the probabilistic prover and authenticating party will therefore be identical to the distribution of randomly drawn elements $\left[\tilde{A}\right]_1, \left[\tilde{B}\right]_2, \left[\tilde{D}\right]_1, \left[\tilde{E}\right]_1$, where $\tilde{A}, \tilde{B}, \tilde{D}, \tilde{E} \leftarrow\!\!\$\ \mathbb{F}_p$. For a valid proof, the remaining element $[C]_1$ is completely determined by the verification equation. The signature ν, which is part of the proof, only depends on $[D]_1$. Therefore, an adversary cannot extract any more information from a valid proof than from a randomly drawn (simulated) proof.

For the proof to still verify with randomization masks added, the proof element C must be modified, such that the additional randomization masks cancel out in the verification equation. The product $A \cdot B$ from the left hand side evaluates to

$$A' \cdot B' + b_B A'\delta + b_A B'\delta + b_A b_B \delta^2 = A' \cdot B' + b_B A\delta + b_A A\delta - b_A b_B \delta^2.$$

The resulting additional term $b_B A + b_A A - b_A b_B \delta$ is added to C' in order to balance the equations. Similarly, the additional terms $b_D\varepsilon$, $b_{t-1,E}\eta$, and $b_{t,E}\kappa$, which appear on the right hand side of the verification equation, are subtracted from C'. The prover stores the mask for the E' term (b_E) between iterations, as it is needed for balancing the equation in the next iteration.

The zero-knowledge ADSC-SNARK construction is shown in Figure 8. Differences to the version without the zero-knowledge property are highlighted.

B.3 Security Proof

We show that the zero-knowledge ADSC-SNARK is complete, knowledge-sound, succinct, and zero-knowledge following definitions 1, 2, 3, and 8.

Theorem 4. *The zero-knowledge* ADSC-*SNARK is complete.*

Proof. When comparing the zero-knowledge variant to the main variant, it is easy to see, that the additional randomization masks b_A, b_B, b_C, b_D, b_E, that are added to the proof elements by Auth and Prove cancel out in the verification equations. Then, completeness of the zero-knowledge ADSC-SNARK follows from completeness of the main ADSC-SNARK (Theorem 1). □

Theorem 5. *The zero-knowledge* ADSC-*SNARK is zero-knowledge.*

Proof. We provide a simulator and show that both real and simulated proofs satisfy the verification equations and demonstrate that real and simulated proofs are distributed equally.

Simulate: $(\pi_t, c_t) \leftarrow \mathsf{Sim}(\tau, \phi_t, c_{t-1}, t)$
Set $E_{t-1} = c_{t-1}$.
Pick $A_t, B_t, D_t, E_t \leftarrow\!\!\$\ \mathbb{F}_p$.
Compute $C_t = \frac{1}{\delta}\left(A_t B_t - \alpha\beta - \sum_{i\in\{0\}\cup\Phi} a_{t,i}P_i(z) - D_t\varepsilon - E_{t-1}\eta - E_t\kappa\right)$.
Compute $sig = \mathsf{SigSign}(sk, ([D_t]_1, t))$.
Set $\pi_t = ([A_t]_1, [B_t]_2, [C_t]_1, [D_t]_1, sig)$, $c_t = [E_t]_1$.

Step 5.1. Real proofs verify due to completeness (Theorem 4). From construction, it is straight forward to see that simulated proofs verify.
Step 5.2. The elements A_t, B_t, D_t, E_t for $t \in [1, r]$ of both real and simulated proofs are uniformly and randomly distributed. The remaining proof element C_t is uniquely determined by the first verification equation and sig was derived from (D_t, t) using SigSign in both the real and simulated proofs. Therefore, the probability for adversary $\mathcal{A}$ in Definition 8 of outputting 1 is the same for real and simulated proofs. □

Theorem 6. *In the generic group model, our zero-knowledge* ADSC-*SNARK is knowledge-sound.*

Proof. We show that if a PPT adversary $\mathcal{A}_{zk}$ can break knowledge-soundness of the zero-knowledge ADSC-SNARK, there exists a PPT adversary $\mathcal{A}_m$, which can break knowledge soundness of the main ADSC-SNARK. Then, we can conclude that the zero-knowledge ADSC-SNARK is knowledge-sound, because the main ADSC-SNARK is knowledge-sound according to Theorem 2.

First, we observe that the verifier algorithm of the zero-knowledge ADSC-SNARK is the same as the verifier algorithm of the main ADSC-SNARK. Second, $\mathcal{A}_{zk}$ has the following additional information compared to $\mathcal{A}_m$: elements $[\beta]_1, [\delta]_1, \{[v_i(z)]_1\}_{i\in[0,m]}, [\epsilon]_1, [\eta]_1, [\kappa]_1$ from prover-key σ_p.

Recall from the proof of Theorem 2, that the main ADSC-SNARK is knowledge-sound, even if there existed an efficient isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. Assuming the existence of such a isomorphism, $\mathcal{A}_m$ can easily compute all the additional elements $[\beta]_1, [\delta]_1, ...,$ from $[\beta]_2, [\delta]_2, ...$ It can then invoke $\mathcal{A}_{zk}$ to find a verifying proof without knowing a valid witness. This proof will also satisfy the verification equation of the zero-knowledge variant. This is only possible with negligible probability, due to knowledge-soundness. We conclude, that the zero-knowledge variant is knowledge-sound. □

Theorem 7. *The zero-knowledge* ADSC-*SNARK is succinct.*

Proof. The proof size and the verifier algorithm of the zero-knowledge ADSC-SNARK are the same as the proof size and the verifier algorithm of the main ADSC-SNARK. From Theorem 3 it follows that also the zero-knowledge variant is succinct. □

$(\sigma_p, \sigma_v, \sigma_a, c_0, \tau) \leftarrow \mathsf{Setup}(s'_0)$

Parse public parameters pp to retrieve polynomials
$\{u_i(Z)\}_{i\in[0,m]}, \{v_i(Z)\}_{i\in[0,m]}, \{w_i(Z)\}_{i\in[0,m]}, t(Z)$.

Randomly sample
$T_i \leftarrow\$ \mathbb{F}_p^* \text{ for } i \in X, \quad R_i \leftarrow\$ \mathbb{F}_p^* \text{ for } i \in S',$
$\alpha, \beta, \gamma, \delta, \kappa, \eta, \varepsilon, z \leftarrow\$ \mathbb{F}_p^*.$
Generate keys for the signature scheme
$(sk, pk) \leftarrow \mathsf{SigGen}()$.

Compute prover-key
$\sigma_p = ([\sigma_{p,1}]_1, [\sigma_{p,2}]_2),$

with

$$\sigma_{p,1} = \begin{pmatrix} \alpha, \beta, \delta, \{u_i(z)\}_{i\in[0,m]}, \{v_i(z)\}_{i\in[0,m]}, \\ \left\{\frac{P_i(z)}{\delta}\right\}_{i\in\Omega}, \left\{\frac{P_i(z)-\eta R_{s2s(i)}}{\delta}\right\}_{i\in S}, \\ \left\{\frac{P_i(z)-\kappa R_i}{\delta}\right\}_{i\in S'}, \left\{\frac{P_i(z)-\varepsilon T_i}{\delta}\right\}_{i\in X}, \\ \left\{\frac{z^i t(z)}{\delta}\right\}_{i\in[0,d-2]}, \{R_i\}_{i\in S'}, \varepsilon, \eta, \kappa \end{pmatrix}$$

$\sigma_{p,2} = \left(\beta, \delta, \{v_i(z)\}_{i\in[0,m]}\right).$
Compute verifier-key
$\sigma_v = ([\sigma_{v,1}]_1, [\sigma_{v,2}]_2, [\alpha\beta]_T, pk)$

with

$$\sigma_{v,1} = \left(\left\{\frac{P_i(z)}{\gamma}\right\}_{i\in\{0\}\cup\Phi}\right), \quad \sigma_{v,2} = (\gamma, \delta, \varepsilon, \eta, \kappa).$$

Compute authentication-key and commitment for initial state
$\sigma_a = (sk, \{[T_i]_1\}_{i\in X}, [\delta]_1),$
$c_0 = \sum_{i\in S'} s'_{0,\mathrm{pos}_{S'}(i)} [R_i]_1.$
Compute trapdoor
$\tau = (\alpha, \beta, \gamma, \delta, \kappa, \eta, \varepsilon, z, \{R_i\}_{i\in S'}).$

$\nu_t \leftarrow \mathsf{Auth}(\sigma_a, t, x_t)$

Parse authentication-key σ_a to retrieve
$sk, \{[T_i]_1\}_{i\in X}, [\delta]_1.$

Parse input x_t to retrieve
$a_i \text{ for } i \in X.$

Sample randomization mask
$b_D \leftarrow\$ \mathbb{F}_p.$

Compute
$\nu_t = \left([D]_1, b_D, sig\right)$

with
$D = \sum_{i\in X} a_i T_i + b_D\delta, \quad sig = \mathsf{SigSign}(sk, ([D]_1, t)).$

$(\pi_t, c_t, p_t) \leftarrow \mathsf{Prove}(\sigma_p, a_t, \nu_t, p_{t-1})$

Parse public parameters pp to retrieve polynomials
$\{u_i(Z)\}_{i\in[0,m]}, \{v_i(Z)\}_{i\in[0,m]}, \{w_i(Z)\}_{i\in[0,m]}, t(Z).$

Parse ν_t to retrieve
$[D]_1, sig, b_D.$

Retrieve the randomization mask $b_{t-1,E}$ of the previous iteration from prover state p_{t-1}
$b_{t-1,E} = \begin{cases} 0, \text{if } p_{t-1} = \{\} \\ p_{t-1}, \text{else.} \end{cases}$
Sample remaining randomization masks
$b_A, b_B, b_{t,E} \leftarrow\$ \mathbb{F}_p$

Compute by polynomial division
$h(Z) =$

$$\left(\sum_{i\in[0,m]} a_{t,i}u_i(Z) \cdot \sum_{i\in[0,m]} a_{t,i}v_i(Z) - \sum_{i\in[0,m]} a_{t,i}w_i(Z)\right)/t(Z).$$

Compute proof and commitment
$\pi_t = ([A]_1, [B]_2, [C]_1, [D]_1, sig), \quad c_t = [E]_1,$

with

$$A = \alpha + \sum_{i\in[0,m]} a_i u_i(z) + b_A\delta, B = \beta + \sum_{i\in[0,m]} a_i v_i(z) + b_B\delta,$$

$$C = \sum_{i\in\Omega} \frac{a_i P_i(z)}{\delta} + \sum_{i\in S} \frac{a_i\left(P_i(z) - \eta R_{s2s(i)}\right)}{\delta} + \frac{h(z)t(z)}{\delta}$$
$$+ \sum_{i\in S'} \frac{a_i\left(P_i(z) - \kappa R_i\right)}{\delta} + \sum_{i\in X} \frac{a_i\left(P_i(z) - \varepsilon T_i\right)}{\delta}$$
$$+ b_B A + b_A B - b_A b_B \delta - b_D\varepsilon - b_{t-1,E}\eta - b_{t,E}\kappa,$$

$$E = \sum_{i\in S'} a_i R_i + b_{t,E}\delta.$$

Set prover state update
$p_t = b_{t,E}.$

$v \leftarrow \mathsf{Verify}(\sigma_v, \phi_t, \pi_t, c_t, c_{t-1}, t)$

Parse public input-output ϕ_t to retrieve
$a_{t,i} \text{ for } i \in \Phi.$

Parse proof π_t to retrieve
$[A_t]_1, [B_t]_2, [C_t]_1, [D]_1, sig.$

Parse commitments c_t and c_{t-1} to retrieve
$[E_t]_1, \quad [E_{t-1}]_1.$

Accept, iff

$$[A_t]_1 \cdot [B_t]_2 = [\alpha\beta]_T + \left(\sum_{i\in\{0\}\cup\Phi} a_{t,i}\left[\frac{P_i(z)}{\gamma}\right]_1\right) \cdot [\gamma]_2$$
$$+ [C_t]_1 \cdot [\delta]_2 + [D_t]_1 \cdot [\varepsilon]_2 + [E_{t-1}]_1 \cdot [\eta]_2 + [E_t]_1 \cdot [\kappa]_2$$

and
$\mathsf{SigVerify}(pk, sig_t, ([D_t]_1, t)) = 1.$

Figure 8: Zero-Knowledge ADSC-SNARK construction for QAP $\{u_i(Z)\}_{i\in[0,m]}$, $\{v_i(Z)\}_{i\in[0,m]}$, $\{w_i(Z)\}_{i\in[0,m]}$, $t(Z)$. The interfaces are described in section 2.1. The differences to the non zero-knowledge version are highlighted. For brevity, $P_i(z) = \beta u_i(z) + \alpha v_i(z) + w_i(z)$ for all $i \in [0, m]$.