ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data

Michael Backes
CISPA, Saarland University
Germany

Manuel Barbosa
HASLab – INESC TEC
Minho University, Portugal

Dario Fiore
IMDEA Software Institute
Madrid, Spain

Raphael M. Reischuk*
ETH Zurich
Switzerland

***Abstract*—We study the problem of privacy-preserving proofs on authenticated data, where a party receives data from a trusted source and is requested to prove computations over the data to third parties in a correct and private way, i.e., the third party learns no information on the data but is still assured that the claimed proof is valid. Our work particularly focuses on the challenging requirement that the third party should be able to verify the validity with respect to the specific data authenticated by the source — even without having access to that source. This problem is motivated by various scenarios emerging from several application areas such as wearable computing, smart metering, or general business-to-business interactions. Furthermore, these applications also demand any meaningful solution to satisfy additional properties related to usability and scalability.**

In this paper, we formalize the above three-party model, discuss concrete application scenarios, and then we design, build, and evaluate ADSNARK, a nearly practical system for proving arbitrary computations over authenticated data in a privacy-preserving manner. ADSNARK improves significantly over state-of-the-art solutions for this model. For instance, compared to corresponding solutions based on Pinocchio (Oakland'13), ADSNARK achieves up to 25× improvement in proof-computation time and a 20× reduction in prover storage space.

I. Introduction

With the emergence of modern IT services, many aspects of the operation of our society have come to critically depend on the ability to share information between multiple parties, subject to complex information flow restrictions. The advance of information and communication technology has often led to the deployment of systems that offer the desired functionality, but do not offer a technical solution to enforcing the secure information flow restrictions. Instead, parties must simply trust each other, often without reasonable grounds.

The last few years have seen exciting developments in cryptography, where (quasi-)practical solutions to some of these problems were proposed, prototyped, and sometimes deployed (as we will see later in this section). In this paper, we make further progress in this direction by proposing and efficiently instantiating a new cryptographic primitive called AD-SNARK, which targets an important class of applications that is out of reach of current technology. Such applications involve a potentially large set of secret data and three parties with the following trust relationships:

- The *data owner* wishes to keep her data secret, but is forced by circumstances to reveal partial information on this data to a service provider. Typically, this is an aggregated result computed by some public function f on the secret data.
- The *service provider* does not trust the data owner to correctly compute the partial information on the data, but wants to be convinced of its validity.
- The data owner has access to a *trusted source*, who can be given local access to the data, and who is trusted by the service provider to vouch for the quality and legitimacy of the data.

For concreteness, let us look at a few applications that fall into this model, and where the public function that must be applied to the data has varying degrees of complexity.

Health Risk Assessment. A wearable biosensor [2, 3] collects fine-grained health information of an individual; the individual should give this information to a health insurance company that wants to assess her health risk in order to evaluate a corresponding premium. Privacy determines that the fine-grained health data collected by the sensor remains secret as it may reveal more about the individual's lifestyle and habits than she wishes to reveal. The computation of the premium due to the insurance company (or an aggregate, less privacy-invasive, information of the collected data) should therefore be carried out by the client. However, the client must convince the insurance company that this computation is correct *and* performed on *legitimate* data produced by the biosensor (we call this property *integrity*). In this setting, the biosensor can play the role of the trusted source, provided that it is equipped to cryptographically authenticate the individual measurements that it produces. Then the AD-SNARK primitive can be used to provide the required assurance to the health insurance company.

Smart Metering. The service provider of some commodity installs a trusted device in the facilities of the client. This trusted device periodically measures consumptions and produces a list of readings, which are delivered to the client; the client should give these readings to the service provider for billing purposes. For privacy, the client may not want to disclose these measurements as they may reveal more about the client's habits than she wishes to reveal (see, e.g., [4]). For integrity, the supplier wants to evaluate a correct bill

* Some of the results of this paper also appear in [1].

DOI 10.1109/SP.2015.24

and prevent customers from cheating. As before, the customer keeps all the readings provided by the local meter, which must be able to authenticate the data and operate as a trusted source. Then, the customer computes the amount due to the provider, and uses AD-SNARK to prove that the result is correct.

Financial Audits. Organizations are often subject to financial audits. Auditors will typically look at specific parts of the accounting data and assert that the results of relevant computations are accurate. However the accounting data should be treated as sensitive information due to its business-critical nature, and minimizing the amount of information disclosed to auditors is desirable. In this scenario, the auditor plays the role of the service provider, and the organization the role of the data owner. The natural entity to play the role of the trusted source is the person (or third party) who is legally responsible for certifying the accounts of the organization, e.g., the official bookkeeper. This entity would authenticate the accounting data, so that the organization could internally compute the audit data in a way that is verifiable by the auditors with respect to both correctness and legitimacy. As intended, using AD-SNARK in this context will transfer the responsibility of any wrongdoing to the official bookkeeper.

In the full version of this paper we present three more example applications: pay-as-you-drive insurance, loyalty cards, and health statistics. We believe that, with the rise of small computing devices and an increased awareness with respect to privacy protection, many more applications will come to fall into this three-party scenario.

Although the trust model in all of the previous applications is the same, the complexity of the associated computations varies significantly. Solutions have been proposed for smart-metering, pay-as-you-drive insurance, and loyalty cards, e.g., in [5, 6], and [7], respectively (and also for other applications of similarly low complexity). However, currently no *generic* solution is able to scale in a satisfactory way to deal with computations of arbitrary size such as those required for scenarios like the ones of financial audits or health statistics. Furthermore, although some scenarios admit to a close relation between the trusted source and the service provider that could lead to secret information being shared between the two (in the style of symmetric cryptography), other scenarios require verification for multiple parties, i.e., a form of public verifiability that is even more challenging. The AD-SNARK primitive and the efficient instantiation that we propose in this paper provides a practical solution for the moderately complex computations, even with public verifiability. Furthermore, the proposed AD-SNARK construction is as practical as the existing state of the art solutions for computations of arbitrary size on non-authenticated data.[1]

Formal Model. We now illustrate more formally the three party model we have introduced above (see Figure 1). We consider a scenario in which a prover $\mathcal{P}$ (the data owner) is requested to prove certain computations $C(D)$ on input data D to third parties $\mathcal{V}$ (one or more service providers), which we call the verifiers. Since the two parties $\mathcal{P}$ and $\mathcal{V}$ may not trust each other, we are interested in the simultaneous achievement of two main security properties: (1) *integrity*, in the sense that $\mathcal{V}$ should be convinced about the correctness of $C(D)$. In particular, in order to verify that this statement holds for some specific input D, the data is assumed to be generated and authenticated by some *trusted source* $\mathcal{S}$; and (2) *privacy*, in the sense that $\mathcal{V}$ should not learn any information about D beyond what is trivially revealed by $C(D)$.

In addition to the security requirements above, any meaningful solution has to meet the following properties that have been identified as key for practical scalability in previous work: (3) *efficiency*, meaning that $\mathcal{V}$'s verification cost should be much cheaper than the cost of computing $C(D)$; and (4) *data independence*, in the sense that the data source $\mathcal{S}$ should be independent of $\mathcal{P}$, i.e., $\mathcal{S}$ should be able to provide D without knowing in advance what computations will be executed on D (e.g., the billing function may change over time). In particular, also D's size should not be fixed in advance, i.e., $\mathcal{S}$ can continuously provide data to $\mathcal{P}$, even after some proofs have been generated.

Related Work. The simultaneous achievement of integrity and privacy is a fundamental goal that has a long research history starting with the seminal work on zero-knowledge proofs [8]. In the last years, the efficiency of zero-knowledge proofs has improved a lot, and nowadays we are on the verge of having nearly practical schemes for general-purpose computations [9–11]. Proofs on authenticated data are an important class of proofs that have been considered earlier especially in very specialized contexts such as credentials and electronic cash [12–15]. In the more general case of proving arbitrary computations over authenticated data, there is however little prior work, especially if one is concerned about achieving practical efficiency. While we review this related work later in Section VI, at this point we mention that the recent work ZQL [6] aimed to address this problem by considering a three party setting such as the one we presented above. ZQL provides an expression language for (privacy-preserving) processing of data that can be originated (i.e., authenticated) by trusted data sources, and proposes a cryptographic scheme that achieves integrity, privacy, and data independence. However, the current ZQL language has some intrinsic limitations that limit its applicability to arbitrary computations while achieving efficiency (i.e., if the verifier should perform less work than that required to generate the proof). In summary, while we do have efficient zero-knowledge proof systems for arbitrary computations, in the case of proofs on authenticated data the situation is not satisfactory.

A. Detailed Contributions

Inspired by the goals of ZQL, we formalize a cryptographic primitive for privacy-preserving proofs on authenticated data, and we propose a new realization that achieves the desired efficiency goal for arbitrary computations. We then build a system called ADSNARK and evaluate its performance in

[1]Hence the designation "nearly practical" in the title of the paper.

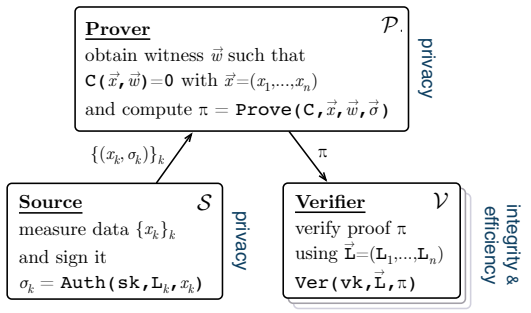


Figure 1. Three-party scenario in which a source $\mathcal{S}$ authenticates data x_k, and a prover $\mathcal{P}$ proves to a verifier $\mathcal{V}$ the satisfiability of a circuit C based on x_k. The source and the prover are interested in data privacy, whereas the verifier is interested in integrity and efficiency.

comparison with solutions based on the state of the art. More in detail, our contributions are the following.

We fully formalize a model for the above problem by defining a new cryptographic primitive that we call *Succinct Non-Interactive Arguments of Knowledge on Authenticated Data* (or AD-SNARK, for short). Succinct Non-Interactive Arguments, first introduced by Micali under the name of "CS proofs" [16], are proof systems that provide *succinct verification*, i.e., the verifier is able to check a long poly-time computation in much less time than that required to run the computation, given the witness. Our new notion of AD-SNARKs extends SNARKs to explicitly capture proofs of $\mathcal{NP}$ relations $R(x, w)$ in which the statement x (or a part of it) is *authenticated*. More precisely, the main difference between SNARKs and AD-SNARKs is that in the former, the verifier always knows the statement, whereas in the latter, the authenticated statements are not disclosed to the verifier, yet the verifier can be assured about the existence of w such that $R(x, w)$ holds for the specific x authenticated by some trusted source. Moreover, to model privacy (and looking ahead to our applications) we define the zero-knowledge property to hold not only for the witnesses of the relation, but also for the authenticated statements. In particular, our zero-knowledge definition holds also against adversaries who generate the authentication keys.

Turning our attention to realizations, we show that AD-SNARKs can be constructed in a generic fashion by embedding digital signatures into SNARKs. However, motivated by the fact that this "generic construction" is not efficient in practice, our second contribution is a *direct and more efficient realization* of AD-SNARKs, that from now on we refer to as ADSNARK. Compared to instantiating the generic construction with state-of-the-art SNARK schemes, ADSNARK performs way better on the prover side, and achieves a level of efficiency that makes it a plausible candidate for real-world deployment. In what follows we give more details on this efficiency aspect: We first discuss the efficiency of the generic construction with state-of-the-art instantiations, and then we describe our solution.

ON THE (IN)EFFICIENCY OF THE GENERIC CONSTRUCTION. The idea of the generic (not very practical) construction of AD-SNARK for an $\mathcal{NP}$ relation $R(x, w)$ is to let the prover $\mathcal{P}$ prove an extended $\mathcal{NP}$ relation R' which contains the set of tuples (x', w') with $x' = (|x|, \mathsf{pk})$, $w' = (w, x, \sigma)$, and $\sigma = (\sigma_1, \ldots, \sigma_{|x|})$, such that there is a valid signature σ_i for every statement value x_i at position i under public key pk. The problem with this generic construction is that, in practice, a proof for such extended relation R' is much more expensive than a proof for R. The issue is that R' needs to "embed" the verification algorithm of a signature scheme. If we consider very efficient SNARKs, such as the recent optimization of Pinocchio [9] proposed in [11], then embedding the verification algorithm means encoding the verification algorithm of the signature with an arithmetic circuit over a specific finite field $\mathbb{F}_p$ (where p is a large prime, the order of some bilinear groups), and then creating a Quadratic Arithmetic Program [17], a QAP for short, out of this circuit. Without going into the details of QAPs (we will review them later in Section II), we note that the efficiency of the prover in these systems depends on the size of the QAP, which in turn depends on the number of multiplication gates in the relation satisfiability circuit.

Our main observation is that the circuit resulting from expressing the verification algorithm of a digital signature scheme is very likely to be quite inefficient (from a QAP perspective), especially for the prover. Such inefficiency stems from the fact that the circuit would contain a huge number of multiplication gates. In Section III-C we discuss why this is the case for various examples of signatures in both the random oracle and the standard model, and based on different algebraic problems. Our conclusions indicate that a QAP encoding a signature verification circuit is likely to have significantly more that one thousand multiplications for *every* signature that must be checked. If, for instance, we consider smart-metering, in which the prover wants to certify about 1 000 (signed) meter readings (amounting to approximately 1 month of electricity measurements), the costs can become prohibitive!

OUR SOLUTION. In contrast, we propose ADSNARK, a new, *direct*, AD-SNARK scheme that achieves the same efficiency as state-of-the-art SNARKs, e.g., [11], yet it additionally allows for proofs on authenticated statements. Our scheme builds upon an optimized version of Pinocchio proposed and implemented in [11], and our key technical contribution is a technique (illustrated in Section I-B) for embedding the authentication verification mechanism directly in the proof system, without having to resort to extended relations that would incur the efficiency loss discussed earlier. As a result, the performance of our scheme is almost the same as that of running [11], but with the additional benefit of obtaining proofs about authenticated values.

When comparing our direct construction with an instantiation of the generic scheme with [11], ADSNARK introduces a dramatic improvement (cf. Figure 2 above) in the generation of setup keys (for the relation) and proofs, which is currently

	AD-PGHR	ADSNARK	Improvement
Key Generator	299 s	16 s	18.7×
Prover	491 s	20 s	24.5×
Verifier	0.062 s	(PK) 0.61 s	0.1×
		(SK) 0.035 s	1.8×
Proving key size	319 MB	16 MB	19.9×
Verification key size	31 KB	31 KB	same
Proof size	0.3 KB	(PK) 126 KB	0.002×
		(SK) 0.4 KB	0.75×

Figure 2. Comparison between ADSNARK and the generic solution (AD-PGHR) based on the [11] SNARK considering an arithmetic circuit with 50K multiplication gates and 1000 authenticated inputs. Results obtained by running `libsnark` for AD-PGHR and our implementation (based on `libsnark`) of ADSNARK, both at a 128-bits security level.

the main bottleneck of state-of-the-art SNARKs (e.g., [9–11]). Namely, while these schemes perform excellently in terms of verification time and proof size, the performances get much worse when it comes to generating keys and proofs, especially for relations that have "unfriendly" arithmetic circuit representations, such as signature verification algorithms, as discussed earlier. This is where our technique for avoiding the explicit encoding of signature verification in the circuits allows us to use much smaller QAPs, thus saving at least one thousand multiplication gates *per* authenticated input. This improvement is clearly evident in our experimental results that show that the prover can obtain up to a 25× speed-up (20 s vs. 8 mins) and a 20× reduction in storage (16 MB vs. 320 MB). As we discuss later, on the verifier side ADSNARK allows for two different verification modes: one using the secret authentication key and one completely public. Although in the secret-key case, ADSNARK essentially achieves the *same* verification efficiency and proof size of the generic solution, our scheme pays more for public verification. However, in contrast to what happens on the prover side of the generic solution, the public verification of ADSNARK still achieves timing (0.61 s) and proof size (126 KB) that can be definitely considered practical.

B. An Intuitive Description of Our Techniques

The key idea for the construction of our AD-SNARK scheme is to build upon SNARKs based on QAPs, and in particular on the PGHR scheme in [11]. At a high level, our technique consists of extending PGHR by embedding a linearly-homomorphic MAC that forces the prover to run the PGHR's Prove algorithm on correctly authenticated statements.

More precisely, the PGHR verifier, given a statement $x = (x_1, \ldots, x_n)$, has to compute the linear combination $a_{in} = \sum_{k=1}^{n} x_k \cdot a_k(X)$ (where the $a_k(X)$ are the QAP polynomials). However, recall that in AD-SNARKs the verifier does not know the statement x, and thus is not able to compute a_{in}. Our key idea to solve this issue is to shift the computation of the linear combination a_{in} from the verifier to the prover. Then, to force a cheating prover to provide the correct a_{in}, we ask the prover to additionally show that a_{in} was indeed obtained by using authenticated values x_k. To this end, we employ another proof system, namely efficient linearly-homomorphic MACs [18, 19], that are particularly suitable for linear computations over authenticated data. Specifically, we designed a novel homomorphic MAC (which is implicitly embedded in our AD-SNARK construction) that fits the above setting.

This technique, however, does not completely solve the problem: a further complication arises from the fact that in order to achieve zero-knowledge, the value a_{in} computed by the prover must be randomized (by adding a random multiple of the QAP target polynomial $z(X)$). Unfortunately, homomorphic MACs are known to authenticate only deterministic computations. We solve this issue using the following ideas. First, we provide a novel technique to publicly re-randomize our homomorphic MACs: roughly speaking, by publicly revealing a MAC of $z(X)$. Second, we force the prover to use the same random coefficient for $z(X)$ in both a_{in} and its MAC. Intuitively, this is achieved by asking the prover to provide the linear combination a_{in} in two distinct subspaces. A final observation is that by using a MAC we only get secret-key verification. Although this may not be an issue in several applications, we also show how to further generalize these techniques to obtain public verification.

C. Organization

The paper is organized as follows. In Section II, we recall common definitions and background information on QAPs. Section III presents our definition of AD-SNARKs, the generic construction, and a discussion on the efficiency of encoding signature verification with arithmetic circuits. We describe our ADSNARK scheme in Section IV together with a theoretical evaluation and comparison to the generic solution. In Section V, we present our implementation and discuss the experimental results. Section VI discusses further related work, and finally Section VII concludes the paper. Extended proofs, and the discussion of two extensions of AD-SNARKs – handling multiple data sources, and achieving (amortized) constant-time verification – are deferred to the full version of our work [20].

II. Background

In this section, we review the notation and some basic definitions that we will use in our work.

Notation. We will denote with $\lambda \in \mathbb{N}$ a security parameter. We say that a function ϵ is *negligible* if it vanishes faster than the inverse of any polynomial. If not explicitly specified otherwise, negligible functions are negligible with respect to λ. If S is a set, $x \leftarrow_{\mathcal{R}} S$ denotes the process of selecting x uniformly at random in S. If $\mathcal{A}$ is a probabilistic algorithm, $x \leftarrow_{\mathcal{R}} \mathcal{A}(\cdot)$ denotes the process of running $\mathcal{A}$ on some appropriate input and assigning its output to x. Moreover, for a positive integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$. We denote by $\mathbb{F}$ a finite field and $\mathbb{F}_n$ is the field of size n. When n is a prime number, then elements of $\mathbb{F}_n$ are represented as integers modulo n. Elements of $\mathbb{F}$ are typically denoted by greek letters. $\mathbb{F}[X]$ denotes the field of polynomials in one variable X and coefficients in $\mathbb{F}$, while $\mathbb{F}^{\leq d}[X]$ is the subring of polynomials in $\mathbb{F}[X]$ of degree at most d.

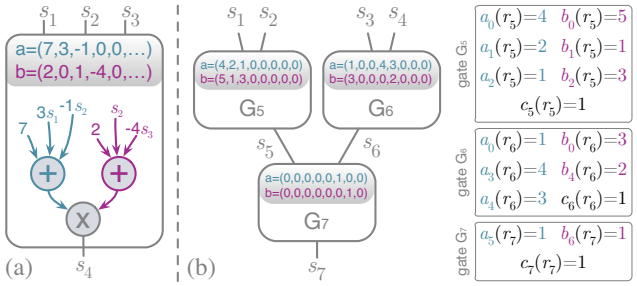


Figure 3. *Part (a):* A bilinear gate representing the arithmetic function $(7 + 3s_1 - 1s_2) \cdot (2 + s_2 - 4s_3)$ specified by coefficients a and b.
Part (b): A QAP for an arithmetic circuit with 4 input wires, 1 output wire, 3 bilinear gates. The circuit encodes the function $f(s_1, s_2, s_3, s_4) = (4 + 2s_1 + s_2) \cdot (5 + s1 + 3s_2) \cdot (1 + 4s_3 + 3s_4) \cdot (3 + 2s_4)$. The non-zero equations for the QAP polynomials are shown on the right.

Algebraic Tools. Let $\mathcal{G}(1^\lambda)$ be an algorithm that, upon input of the security parameter 1^λ, outputs the description of (asymmetric) bilinear groups $\mathsf{bgpp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2)$ where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are groups of the same prime order $p > 2^\lambda$; $\mathcal{P}_1 \in \mathbb{G}_1$ and $\mathcal{P}_2 \in \mathbb{G}_2$ are the respective generators; and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable bilinear map. We call such an algorithm $\mathcal{G}$ a *bilinear group generator*. Note that $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups, whereas $\mathbb{G}_T$ is a multiplicative group. In this work we rely on specific computational assumptions in such bilinear groups: the q-DHE [21], the q-BDHE [22], and the q-PKE [23] assumptions.

Arithmetic Circuits and QAPs. An *arithmetic circuit* C over a finite field $\mathbb{F}$ consists of addition and multiplication *gates* and of a set of *wires* between the gates. The wires carry values over $\mathbb{F}$. As in previous work [11], here we consider only arithmetic circuits with *bilinear gates*: a gate with inputs $\vec{x} = (x_1, \ldots, x_k)$ is *bilinear* if its output can be written as inner product $\langle \vec{a}, (1, x_1, \ldots, x_k) \rangle \cdot \langle \vec{b}, (1, x_1, \ldots, x_k) \rangle$ for some $\vec{a}, \vec{b} \in \mathbb{F}^{k+1}$. Note that this definition includes addition, multiplication, and constant gates (cf. Fig. 3(a) for an example).

Associated to any arithmetic circuit, we define a satisfaction problem as follows.

Definition 1 (Arith. Circuit Satisfaction [11]): The circuit satisfaction problem of a circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ with bilinear gates is defined by the relation $\mathcal{R}_C = \{(\vec{x}, \vec{w}) \in \mathbb{F}^n \times \mathbb{F}^h : C(\vec{x}, \vec{w}) = 0^l\}$ and its language is $\mathcal{L}_C = \{\vec{x} \in \mathbb{F}^n : \exists \vec{w} \in \mathbb{F}^h, C(\vec{x}, \vec{w}) = 0^l\}$.

The state-of-the-art SNARK schemes that we build on in this paper directly operate on a different model to represent computations called *quadratic arithmetic programs* (QAPs).

Definition 2 (QAP [17]): A *quadratic arithmetic program* Q of size m and degree d over $\mathbb{F}$ consists of three vectors of $m + 1$ polynomials $\vec{a}, \vec{b}, \vec{c} \in \mathbb{F}^{\leq d-1}[X]$ of degree at most $d-1$, and a target polynomial $z(X) \in \mathbb{F}[X]$ of degree exactly d.

Associated to any QAP, there is a satisfaction problem defined as follows.

Definition 3 (QAP Satisfaction): The satisfaction problem of a QAP $Q = (\vec{a}, \vec{b}, \vec{c}, z)$ of size m and degree d is the relation $\mathcal{R}_Q$ of pairs $(\vec{x}, \vec{s})$ such that:
(1) $\vec{x} \in \mathbb{F}^n$, $\vec{s} \in \mathbb{F}^m$ for some $n \leq m$;
(2) $x_i = s_i$ for $i \in [n]$, i.e., $\vec{s}$ extends $\vec{x}$;
(3) $z(X)$ divides the polynomial $p(X)$ defined as

$$p(X) = \big(a_0(X) + \sum_{i=1}^{m} s_i a_i(X)\big) \cdot \big(b_0(X) + \sum_{i=1}^{m} s_i b_i(X)\big) - \big(c_0(X) + \sum_{i=1}^{m} s_i c_i(X)\big)$$

The following result implies that one can use any QAP-based SNARK scheme as an efficient SNARK scheme taking computations more conveniently represented as arithmetic circuits.

Lemma 1 (Constructing QAPs [11, 17]): There exist two polynomial time algorithms $\mathsf{QAPInst}$ and QAPwit such that, for any circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ with u wires and v (bilinear) gates, $Q_C = (\vec{a}, \vec{b}, \vec{c}, z) = \mathsf{QAPInst}(C)$ is a QAP of size m and degree d over $\mathbb{F}$ satisfying the following properties:

Efficiency: $m = u$, and $d = v + l + 1$.
Completeness: For any $(\vec{x}, \vec{w}) \in \mathcal{R}_C$, if it holds that $\vec{s} = \mathsf{QAPwit}(C, \vec{x}, \vec{w})$ then $(\vec{x}, \vec{s}) \in \mathcal{R}_{Q_C}$.
Proof of Knowledge: For any $(\vec{x}, \vec{s}) \in \mathcal{R}_{Q_C}$, it holds $(\vec{x}, \vec{w}) \in \mathcal{R}_C$ where $\vec{w}$ is a prefix of $\vec{s}$.
Non-Degeneracy: the polynomials $a_0(X), \ldots, a_n(X)$ are all nonzero and distinct.

The very basic intuition for building a QAP according to Lemma 1 is to encode the input-output correctness for each bilinear gate in the polynomials $\vec{a}, \vec{b}, \vec{c}, z$ (see Fig. 3(b) for a simple example). Slightly more in detail, for a gate g this is done by first selecting an arbitrary value $r_g \in \mathbb{F}$ (a "root") and then, for every left wire i going to gate g, one imposes $a_i(r_g) = c$, where c is the coefficient which multiplies the value of wire i in g's left input (note that $c = 0$ if wire i is not a left input). A similar process is done for polynomials b_i and c_i w.r.t. right input and output wires respectively.[2] Once this procedure has been iterated for every bilinear gate g (selecting distinct roots r_g), one will have essentially obtained three tables of size $u \cdot v$ with entries $a_i(r_j)$, $b_i(r_j)$, and $c_i(r_j)$, respectively, where $i = 0$ to u are all the wires (where the 0 wire represents constants) and $j = 1$ to v are all the bilinear gates. The final QAP polynomials $\vec{a}, \vec{b}, \vec{c}$ are built by extending each row i of the table into a polynomial $a_i(X)$ (resp. $b_i(X), c_i(X)$) of degree $v - 1$ via interpolation in $\mathbb{F}$. The target polynomial $z(X)$ is the degree-v polynomial defined over the roots r_g of the v bilinear gates: $z(X) := \prod_{g=1}^{v}(X - r_g)$.[3] To see why the satisfiability of the QAP implies the satisfiability of the circuit, the key observation is that the third condition of Definition 3, i.e., $z(X) \mid p(X)$,

[2] The case of c_i is slightly different as coefficients are only 0 or 1.

[3] More precisely, in construction of Lemma 1 one needs to add one "artificial" bilinear gate for every output wire, plus an additional constraint to guarantee non-degeneracy: from which the final degree is $d = v + l + 1$.

means that $\langle(1,\vec{s}),\vec{a}(r_g)\rangle\cdot\langle(1,\vec{s}),\vec{b}(r_g)\rangle = \langle(1,\vec{s}),\vec{c}(r_g)\rangle$ for all roots r_g of the target polynomial $z(X)$. In other words, given the specific construction of the polynomials, the input-output correctness of every bilinear gate g of the circuit is satisfied.

III. SNARKs over Authenticated Data

In this section, we define the notion of SNARKs [16, 24] on authenticated data (AD-SNARKs, for short). Let $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ be an arithmetic circuit, and let $\mathcal{R}_C = \{(\vec{x},\vec{w})\} \subseteq \mathbb{F}^n \times \mathbb{F}^h$ be the corresponding circuit satisfaction relation, where $\vec{x} \in \mathbb{F}^n$ is called the *statement*, and $\vec{w} \in \mathbb{F}^h$ is the *witness*.

In this work, we consider a setting in which (1) the statement $\vec{x}$ (or part of it) is provided to the prover by a trusted source $\mathcal{S}$, and (2) the portion of $\vec{x}$ provided by $\mathcal{S}$ is not known to $\mathcal{V}$ (see Figure 1 for illustration). Yet, $\mathcal{V}$ wants to be convinced by $\mathcal{P}$ that $(\vec{x},\vec{w}) \in \mathcal{R}_C$ holds for the specific $\vec{x}$ provided by $\mathcal{S}$, and not for some other $\vec{x}'$ of $\mathcal{P}$'s choice (which can still be in the language $\mathcal{L}_C$).

To formalize the idea that $\mathcal{V}$ checks that some values unknown to $\mathcal{V}$ have been authenticated by $\mathcal{S}$, we adopt the concept of *labeling* used for homomorphic authenticators [19, 25]. Namely, we assume that the source $\mathcal{S}$ authenticates a set of values $X_{auth} = \{x_i,\ldots,x_\ell\}$ against a set of (public) labels $\mathsf{L} = \{\mathsf{L}_i,\ldots,\mathsf{L}_\ell\}$ by using a secret authentication key (e.g., a signing key). $\mathcal{S}$ then sends the authenticated X_{auth} to $\mathcal{P}$. Later, $\mathcal{P}$'s goal is to prove to $\mathcal{V}$ that $(\vec{x},\vec{w}) \in \mathcal{R}_C$ for a statement $\vec{x}$ in which some positions have been correctly authenticated by $\mathcal{S}$, i.e., $x_i \in X_{auth}$ for some $i \in [n]$.

For such a proof system, we define the usual properties of *completeness* and *soundness*, and in addition, to model privacy, we define a *zero-knowledge* property. Moreover, since we are interested in efficient and scalable protocols, we define *succinctness* to model that the size of the proofs (and implicitly the verifier's running time) should be independent of the witness' size $h = |\vec{w}|$. Finally, we consider AD-SNARKs that can have either public or secret verifiability, the difference being in whether the adversary knows or not the verification key for the authentication tags produced by the data source $\mathcal{S}$.

A. Definition of AD-SNARKs

Here we provide the formal definition for zero-knowledge SNARKs over authenticated data.

Definition 4 (Zero-Knowledge AD-SNARK): A scheme for *Zero-Knowledge Succinct Non-interactive Arguments of Knowledge over Authenticated Data* for arithmetic circuit satisfiability consists of a tuple of algorithms $(\mathsf{Setup}, \mathsf{AuthKG}, \mathsf{Auth}, \mathsf{AuthVer}, \mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ satisfying *authentication correctness*, *completeness*, *succinctness*, *adaptive proof of knowledge*, and *zero-knowledge* (as defined below):

- $\mathsf{Setup}(1^\lambda)$: given the security parameter λ, output some common public parameters pp. The parameters also define the finite field $\mathbb{F}$ over which the circuits will be defined.
- $\mathsf{AuthKG}(\mathsf{pp})$: given the public parameters pp, the key generation algorithm outputs a secret authentication key sk, a verification key vk, and public authentication parameters pap.
- $\mathsf{Auth}(\mathsf{sk},\mathsf{L},x)$: the authentication algorithm takes as input the secret authentication key sk, a label $\mathsf{L} \in \mathcal{L}$, and a value $x \in \mathbb{F}$, and it outputs an authentication tag σ.
- $\mathsf{AuthVer}(\mathsf{vk},\sigma,\mathsf{L},x)$: the authentication verification algorithm takes as input a verification key vk, a tag σ, a label $\mathsf{L} \in \mathcal{L}$, and a value $x \in \mathbb{F}$. It outputs $\perp$ (reject) or $\top$ (accept).
- $\mathsf{Gen}(\mathsf{pap},C)$: given the public authentication parameters pap and an arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$, the algorithm outputs an evaluation key EK_C and a verification key VK_C. Gen can hence be seen as a circuit encoding algorithm.
- $\mathsf{Prove}(\mathsf{EK}_C,\vec{x},\vec{w},\vec{\sigma})$: on input an evaluation key EK_C, a statement $\vec{x} \in \mathbb{F}^n$, a witness $\vec{w} \in \mathbb{F}^h$, and authentication tags for the statement $\vec{\sigma} = (\sigma_1,\ldots,\sigma_n)$, the proof algorithm outputs a proof of membership π for $(\vec{x},\vec{w}) \in \mathcal{R}_C$. We stress that $\vec{\sigma}$ does not need to contain authentication tags for all positions: in case a value at position i is not authenticated, the empty tag $\sigma_i = \star$ is used instead.
- $\mathsf{Ver}(\mathsf{vk},\mathsf{VK}_C,\vec{\mathsf{L}},\{x_i\}_{\mathsf{L}_i=\star},\pi)$: given the verification key vk, a circuit verification key VK_C, labels $\vec{\mathsf{L}} = (\mathsf{L}_1,\ldots,\mathsf{L}_n)$ for the statement, unauthenticated statement components x_i, and a proof π, the verification algorithm outputs $\perp$ (reject) or $\top$ (accept).

AUTHENTICATION CORRECTNESS. Intuitively, an AD-SNARK scheme has authentication correctness if any tag σ generated by $\mathsf{Auth}(\mathsf{sk},\mathsf{L},x)$ authenticates x with respect to L. More formally, we say that an AD-SNARK scheme satisfies authentication correctness if for any value $x \in \mathbb{F}$, all keys $(\mathsf{sk},\mathsf{vk},\mathsf{pap})$ in the range of $\mathsf{AuthKG}(1^\lambda)$, any label $\mathsf{L} \in \mathcal{L}$, and any authentication tag σ generated by $\mathsf{Auth}(\mathsf{sk},\mathsf{L},x)$, we have that $\mathsf{AuthVer}(\mathsf{vk},\sigma,\mathsf{L},x) = \top$.

COMPLETENESS. This property aims at capturing that if the Prove algorithm produces π when run on $(\vec{x},\vec{w},\vec{\sigma})$ for some $(\vec{x},\vec{w}) \in \mathcal{R}_C$ then verification $\mathsf{Ver}(\mathsf{vk},\mathsf{VK}_C,\mathsf{L},\{x_i\}_{\mathsf{L}_i=\star},\pi)$ will accept the proof whenever $\vec{\sigma}$ authenticates $\vec{x}$ with respect to L. More formally, let us sample $(\mathsf{sk},\mathsf{vk},\mathsf{pap}) \leftarrow_{\mathcal{R}} \mathsf{AuthKG}(\mathsf{pp})$, fix a circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ and take keys $(\mathsf{EK}_C,\mathsf{VK}_C) \leftarrow_{\mathcal{R}} \mathsf{Gen}(\mathsf{pap},C)$. Let $(\vec{x},\vec{w}) \in \mathcal{R}_C$ be given; let $\vec{\mathsf{L}} = (\mathsf{L}_1,\ldots,\mathsf{L}_n) \in (\mathcal{L} \cup \{\star\})^n$ be a vector of labels, and let $\vec{\sigma} = (\sigma_1,\ldots,\sigma_n)$ be tags for the statement such that $\{\mathsf{AuthVer}(\mathsf{vk},\sigma_i,\mathsf{L}_i,x_i) = \top\}_{\mathsf{L}_i\neq\star}$. Then if $\pi \leftarrow_{\mathcal{R}} \mathsf{Prove}(\mathsf{EK}_C,\vec{x},\vec{w},\vec{\sigma})$, we require that $\mathsf{Ver}(\mathsf{vk},\mathsf{VK}_C,\vec{\mathsf{L}},\{x_i\}_{\mathsf{L}_i=\star},\pi) = \top$ with probability 1.

SUCCINCTNESS. Given a circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$, the length of the proof π is bounded by $|\pi| = \mathrm{poly}(\lambda)\mathrm{polylog}(n,h)$.

ADAPTIVE PROOF OF KNOWLEDGE. Intuitively, the adaptive proof of knowledge property captures that no malicious party can produce proofs that verify correctly for a false statement. We formalize our definition via an experiment, called $\mathbf{Exp}^{\mathsf{AD\text{-}PoK}}_{\mathcal{A},E}$, which is described in Figure 4. The experiment is parametrized by both an adversary $\mathcal{A}$ and an *extractor* E, and it works for a class $\mathcal{C}$ of circuits. Both $\mathcal{A}$ and E run on the same

$\mathbf{Exp}^{\mathsf{AD\text{-}PoK}}_{\mathcal{A},E}(1^\lambda, \mathcal{C}, z)$:
$\mathsf{pp} \leftarrow_{\mathcal{R}} \mathsf{Setup}(1^\lambda)$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{pap}) \leftarrow_{\mathcal{R}} \mathsf{AuthKG}(\mathsf{pp})$
$\mathtt{GameOutput} \leftarrow 0$
$\mathsf{S} \leftarrow \emptyset,\ \mathsf{T} \leftarrow \emptyset$
$\mathcal{A}^{\mathbf{Gen},\mathbf{Auth},\mathbf{Ver}}(\mathsf{pp}, \mathsf{pap}, z)$
$\mathtt{Return}\ \mathtt{GameOutput}$

procedure Gen(C)
$(\mathsf{EK}_C, \mathsf{VK}_C) \leftarrow_{\mathcal{R}} \mathsf{Gen}(\mathsf{pap}, C)$
$\mathsf{S} \leftarrow \mathsf{S} \cup \{(C, \mathsf{EK}_C, \mathsf{VK}_C)\}$
$\mathtt{Return}\ (\mathsf{EK}_C, \mathsf{VK}_C)$

procedure Auth(L, x)
$\mathtt{if}\ (\mathsf{L}, \cdot, \cdot) \in \mathsf{T}\ \ \mathtt{Return}\ \bot$
$\sigma \leftarrow_{\mathcal{R}} \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, x)$
$\mathsf{T} \leftarrow \mathsf{T} \cup \{(\mathsf{L}, x, \sigma)\}$
$\mathtt{Return}\ \sigma$

procedure Ver$(C, \vec{\mathsf{L}}, \{x_i\}_{\mathsf{L}_i = \star}, \pi)$
$\mathtt{if}\ (C, \cdot, \cdot) \notin \mathsf{S}\ \ \mathtt{Return}\ \bot$
$\mathtt{fetch}\ \mathsf{VK}_C\ \mathtt{with}\ (C, \cdot, \mathsf{VK}_C) \in \mathsf{S}$
$v \leftarrow \mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_C, \vec{\mathsf{L}}, \{x_i\}_{\mathsf{L}_i = \star}, \pi)$
$\mathtt{if}\ v = \top\ \mathtt{then}$
 $\mathtt{if}\ \exists\, \mathsf{L}_i \in \vec{\mathsf{L}} : (\mathsf{L}_i, \cdot, \cdot) \notin \mathsf{T}\ \mathtt{then}$
  $\mathtt{GameOutput} \leftarrow 1$ *// Type 1*
 $\mathtt{else}$
  $\mathtt{fetch}\ \vec{x} = (x_1, \ldots, x_n)\ \mathtt{with}$
   $\{(\mathsf{L}_1, x_1, \cdot), \ldots, (\mathsf{L}_n, x_n, \cdot)\} \subseteq \mathsf{T}$
   for all $\mathsf{L}_i \neq \star$
  $\vec{w} \leftarrow E(\mathsf{pp}, \mathsf{pap}, z, \mathsf{S}, aux_E)$
  $\mathtt{if}\ (\vec{x}, \vec{w}) \notin \mathcal{R}_C\ \mathtt{then}$
   $\mathtt{GameOutput} \leftarrow 1$ *// Type 2*
$\mathtt{Return}\ v$

Figure 4. Experiment for the adaptive proof of knowledge definition.

input and random tape, including some auxiliary input z. As the only difference, E takes an additional input aux_E which includes: the secret authentication keys sk, vk, and all the random coins used to run the **Auth** oracle. E is an algorithm that, for every verification query of $\mathcal{A}$ that is accepted by the Ver algorithm, outputs a witness $\vec{w}$. One should think of such E as $\mathcal{A}$ itself, and the extraction capability intuitively means that if $\mathcal{A}$ is able to produce an accepting proof, then $\mathcal{A}$ must know the corresponding witness, and thus such witness can be extracted from $\mathcal{A}$'s memory. Note also that the additional input aux_E taken by E is generated independently of the circuits in $\mathcal{C}$, and thus it cannot help the extractor to generate witnesses more than $\mathcal{A}$ can do. The three procedures **Gen**, **Auth**, and **Ver** essentially give to the adversary oracle access to the algorithms Gen, Auth, and Ver, respectively, with some additional bookkeeping information, and under the restriction that **Gen** is queried on a circuit $C \in \mathcal{C}$. It is worth noting that **Ver** returns the output of Ver, and additionally, checks whether a proof accepted by Ver (i.e., $v = \top$) proves a false statement according to $\mathcal{R}_C$. In this case, **Ver** sets $\mathtt{GameOutput} \leftarrow 1$.

Adaptive proof of knowledge is formally defined as follows. Let $\mathcal{C}$ be a class of polynomially many circuits. We say that a scheme satisfies *adaptive proof of knowledge* if for any sufficiently large $\lambda \in \mathbb{N}$, for any $\mathcal{C}$ and for every PPT adversary $\mathcal{A}$, there exists a PPT extractor E such that for every polynomial-size auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$ the probability $\Pr[\mathbf{Exp}^{\mathsf{AD\text{-}PoK}}_{\mathcal{A},E}(\mathcal{C}, 1^\lambda, z) = 1]$ is negligible in λ.

Our definition above is inspired by the security definition for homomorphic authenticators [18, 19, 25]. The catch here is that there are essentially two ways to create a "cheating proof", and thus to break the adaptive proof of knowledge of an AD-SNARK. The first way, Type 1, is to produce an accepting proof without having ever queried an authentication tag for a label L_i. This basically captures that, in order to create a valid proof, one needs to have all authenticated parts of the statement, each with a valid authentication tag. The second way to break the security, Type 2, is the more "classical" one, i.e., generating a proof that accepts for a tuple $(\vec{x}, \vec{w})$ which

$\mathbf{Exp}^{\mathsf{Real}}_{\mathcal{D},C}(1^\lambda)$:
$\mathsf{pp} \leftarrow_{\mathcal{R}} \mathsf{Setup}(1^\lambda)$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{pap}) \leftarrow_{\mathcal{R}} \mathcal{D}(1^\lambda, \mathsf{pp})$
$(\mathsf{EK}_C, \mathsf{VK}_C) \leftarrow_{\mathcal{R}} \mathsf{Gen}(\mathsf{pap}, C)$
$(\vec{x}, \vec{\mathsf{L}}, \vec{\sigma}, \vec{w}) \leftarrow \mathcal{D}(\mathsf{EK}_C, \mathsf{VK}_C)$
$\pi \leftarrow_{\mathcal{R}} \mathsf{Prove}(\mathsf{EK}_C, \vec{x}, \vec{w}, \vec{\sigma})$
$\mathtt{if}\ (\vec{x}, \vec{w}) \notin \mathcal{R}_C\ \vee$
 $\exists i \in [n],$
  $\mathsf{AuthVer}(\mathsf{vk}, \sigma_i, \mathsf{L}_i, x_i) = \bot$
$\mathtt{then\ Return}\ 0$
$\mathtt{else\ Return}\ \mathcal{D}(\pi)$

$\mathbf{Exp}^{\mathsf{Sim}}_{\mathcal{D},C}(1^\lambda)$:
$\mathsf{pp} \leftarrow_{\mathcal{R}} \mathsf{Setup}(1^\lambda)$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{pap}) \leftarrow_{\mathcal{R}} \mathcal{D}(1^\lambda, \mathsf{pp})$
$(\mathsf{EK}_C, \mathsf{VK}_C, \mathsf{td})$
 $\leftarrow_{\mathcal{R}} \mathsf{Sim}_1(\mathsf{sk}, \mathsf{vk}, \mathsf{pap}, \mathsf{pp}, C)$
$(\vec{x}, \vec{\mathsf{L}}, \vec{\sigma}, \vec{w}) \leftarrow \mathcal{D}(\mathsf{EK}_C, \mathsf{VK}_C)$
$\pi \leftarrow_{\mathcal{R}} \mathsf{Sim}_2(\mathsf{td}, \mathsf{L}, \{x_i\}_{\mathsf{L}_i = \star})$
$\mathtt{if}\ (\vec{x}, \vec{w}) \notin \mathcal{R}_C\ \vee$
 $\exists i \in [n],$
  $\mathsf{AuthVer}(\mathsf{vk}, \sigma_i, \mathsf{L}_i, x_i) = \bot$
$\mathtt{then\ Return}\ 0$
$\mathtt{else\ Return}\ \mathcal{D}(\pi)$

Figure 5. Experiments for the zero knowledge definition.

is not the correct one, i.e., $(\vec{x}, \vec{w}) \notin \mathcal{R}_C$.

Second, we note that the above game definition captures the setting in which the verification key vk is kept secret. The definition for the publicly verifiable setting is obtained by providing vk to the adversary.

ZERO-KNOWLEDGE. Loosely speaking, a *zero-knowledge* AD-SNARK is an AD-SNARK in which the Prove algorithm generates proofs π that reveal no information: neither about the *witness*, nor about the authenticated statements. In other words, the proofs do not reveal anything beyond what is known by the verifiers when checking a proof.

Formally, let $C \in \mathcal{C}$ be an arithmetic circuit. Then an AD-SNARK is zero-knowledge if there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$, such that for all PPT distinguishers $\mathcal{D}$, the following difference is negligible $|\Pr[\mathbf{Exp}^{\mathsf{Real}}_{\mathcal{D},C}(1^\lambda) = 1] - \Pr[\mathbf{Exp}^{\mathsf{Sim}}_{\mathcal{D},C}(1^\lambda) = 1]|$, where the experiments Real and Sim are defined as in Figure 5. We note that the distinguisher $\mathcal{D}$ in these experiments has a shared state that is persistent over all invocations of $\mathcal{D}$.

We stress that the above zero-knowledge notion aims at capturing, in the strongest possible sense, that the verifier cannot learn any useful information on the inputs, *even if it knows (or chooses) the secret authentication key*. Indeed, as one can see, our definition allows the distinguisher to choose the authentication key pair as well as the authentication tags.

We also remark that the notion of AD-SNARKs immediately implies a corresponding notion of *verifiable computation on authenticated data* (similar to [19]). In [24], it is discussed how to construct a verifiable computation scheme from SNARGs for $\mathcal{NP}$ with adaptive soundness. This is simply based on the fact that the correctness of a computation can be described with an $\mathcal{NP}$ statement. It is not hard to see that, in a very similar way, one can construct verifiable computation on authenticated data from AD-SNARKs.

B. A Generic Construction

We show how to construct a zero-knowledge AD-SNARK scheme from SNARKs and digital signatures. A similar construction was informally sketched in [24][Appendix 10.1.2 of the full version]. Here we make it more formal with the main purpose of offering a comparison with our direct AD-SNARK construction proposed in the next section.

The high-level idea of the generic construction is to embed digital signatures into SNARKs. Let therefore $\Pi' =$

$(\mathsf{Gen}', \mathsf{Prove}', \mathsf{Ver}')$ be a SNARK scheme, and $\Sigma = (\Sigma.\mathsf{KG}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Ver})$ be a signature scheme.

We will use the signature scheme to sign pairs consisting of a label L and an actual message m. Although labels and messages can be arbitrary binary strings, for ease of description we assume that labels can take a special value $\star$. Also, we modify the signature scheme in such a way that $\Sigma.\mathsf{Sign}(\mathsf{sk}, \star|m) = \star$ and $\Sigma.\mathsf{Ver}(\mathsf{vk}, \star|m', \star) = 1$. Basically, we let everyone (trivially) generate a valid signature on a message with label $\star$.

We define an AD-SNARK $\Pi = (\mathsf{Setup}, \mathsf{AuthKG}, \mathsf{Auth}, \mathsf{AuthVer}, \mathsf{Prove}, \mathsf{Ver})$ as follows.

$\mathsf{Setup}(1^\lambda)$: Output $\mathsf{pp} = 1^\lambda$.

$\mathsf{AuthKG}(\mathsf{pp})$: run $(\mathsf{sk}', \mathsf{vk}') \leftarrow_{\mathcal{R}} \Sigma.\mathsf{KG}(1^\lambda)$ to generate the key pair of the signature scheme and return $\mathsf{sk} = \mathsf{sk}'$ and $\mathsf{vk} = \mathsf{pap} = \mathsf{vk}'$.

$\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, x)$: compute a signature on the concatenation of the label L and the value x, i.e., $\sigma' \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}', \mathsf{L}|x)$. Finally, output $\sigma = (\sigma', \mathsf{L})$.

$\mathsf{AuthVer}(\mathsf{vk}, \sigma, \mathsf{L}, x)$: let $\sigma = (\sigma', \mathsf{L}')$. Output the result of the signature verification algorithm $\mathsf{Ver}'(\mathsf{vk}', \mathsf{L}|x, \sigma')$.

$\mathsf{Gen}(\mathsf{pap}, C)$: for the given circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ we define C' as the circuit that outputs 0^l on all the pairs $(\vec{x}, \vec{w})$ such that $C(\vec{x}, \vec{w}) = 0^l$ and each x_i is correctly signed with respect to a set of labels and a public key. More formally, define $C' : \mathbb{F}^{n'} \times \mathbb{F}^{h'} \to \mathbb{F}^l$ as the circuit that takes as inputs pairs $(\vec{x}', \vec{w}')$ with $\vec{x}' = (y_1, \mathsf{L}_1, \ldots, y_n, \mathsf{L}_n, \mathsf{vk})$ and $\vec{w}' = (\vec{w}, z_1, \sigma_1, \ldots, z_n, \sigma_n)$ such that, by setting $x_i = y_i$ if $\mathsf{L}_i = \star$ and $x_i = z_i$ otherwise, for all $i \in [n]$, it holds: (i) $((x_1, \ldots, x_n), \vec{w}) \in \mathcal{R}_C$, and (ii) $\Sigma.\mathsf{Ver}(\mathsf{vk}, \mathsf{L}_i|x_i, \sigma_i) = 1$.
Finally, run $\mathsf{Gen}'(1^\lambda, C')$ to generate $(\mathsf{EK}'_{C'}, \mathsf{VK}'_{C'})$ and output $\mathsf{EK}_C = \mathsf{EK}'_{C'}$, $\mathsf{VK}_C = \mathsf{VK}'_{C'}$.

$\mathsf{Prove}(\mathsf{EK}_C, \vec{x}, \vec{w}, \vec{\sigma})$: Let EK_C be an evaluation key as defined above, $(\vec{x}, \vec{w}) \in \mathbb{F}^n \times \mathbb{F}^h$ be a statement-witness pair, and $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ be a tuple of authentication tags for $\vec{x} = (x_1, \ldots, x_n)$. If all the tags verify correctly, define $\vec{x}' = (y_1, \mathsf{L}_1, \ldots, y_n, \mathsf{L}_n, \mathsf{vk})$, $\vec{w}' = (\vec{w}, z_1, \sigma'_1, \ldots, z_n, \sigma'_n)$ so that for all $i \in [n]$: $z_i = x_i$, $y_i = x_i$ if $\sigma_i = \star$ and $y_i = 0$ otherwise. Next, run $\pi \leftarrow_{\mathcal{R}} \mathsf{Prove}(\mathsf{EK}'_{C'}, \vec{x}', \vec{w}')$ to generate a proof for $(\vec{x}', \vec{w}') \in \mathcal{R}_{C'}$ and return π.

$\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_C, \vec{\mathsf{L}}, \{x_i\}_{\mathsf{L}_i = \star}, \pi)$: given the verification key vk, a circuit verification key VK_C, statement labels $\vec{\mathsf{L}} = (\mathsf{L}_1, \ldots, \mathsf{L}_n)$, unauthenticated statement components x_i, and a proof π, the verification algorithm defines $\vec{x}' = (y_1, \mathsf{L}_1, \ldots, y_n, \mathsf{L}_n, \mathsf{vk})$ with $y_i = x_i$ if $\mathsf{L}_i = \star$ and $y_i = 0$ otherwise. Finally, it returns the output of $\mathsf{Ver}'(\mathsf{VK}'_{C'}, \vec{x}', \pi)$.

Note that the input size of C' is a circuit larger than C as follows: $n' = n + n \cdot |\mathsf{L}_i| + |\mathsf{vk}|$ and $h' = h + n + n \cdot |\sigma|$, where $|\mathsf{vk}|$, $|\mathsf{L}_i|$, and $|\sigma|$ represent the size, in terms of field elements, of the public key, a label, and a signature, respectively. In terms of gates and wires, C' is at least as large as C plus the circuit size of $\Sigma.\mathsf{Ver}$ for *every* signature verification, that is up to n of such circuits.

Theorem 1: If Π' is a zero-knowledge SNARK and Σ is a secure digital signature, then the scheme described above is a zero-knowledge AD-SNARK.

A proof sketch appears in the full version [20].

C. Signature Verification Overhead

We now discuss why the circuit C' resulting from explicitly encoding the verification algorithm of a digital signature scheme, as described in the generic construction, is bound to render the construction very inefficient. We consider various examples of signatures in both the random oracle and the standard model, and based on different algebraic problems.

If one considers signature schemes in the random oracle model (which include virtually all the schemes used in practice), any such scheme uses a collision-resistant hash function (e.g., SHA-1) which is thus part of the verification algorithm computation. Unfortunately, as shown also in [9], a QAP (just) for a SHA-1 computation is terribly inefficient due to the high number of multiplication gates (roughly 24 000, for inputs of 416 bits). On the other hand, if we focus on standard model signature schemes, it does not get any better: These schemes involve specific algebraic computations, and encoding these computations into an arithmetic circuit over a field $\mathbb{F}_p$ is costly. For instance, signatures based on pairings [26, 27] require pairing computations that amount to, roughly, 10 000 multiplications. RSA-based standard-model signatures (e.g., Cramer-Shoup [28]) require exponentiations over rings of large order (e.g., 3 000 bits), and simulating such computations over $\mathbb{F}_p$ ends up with thousands of multiplication gates as well. Lattice-based signatures (in the standard model), e.g., [29], can be cheaper in terms of the number of multiplications. However, such multiplications typically work over $\mathbb{Z}_q$ for a q much smaller than our p. An option would be to implement mod-q-reductions in $\mathbb{F}_p$ circuits, which is costly. Another option would be to let these schemes work over $\mathbb{Z}_p$, but then one has to work with higher dimensional lattices (or polynomial rings) for security reasons, again incurring a large number of multiplications.

This state of affairs suggests that a QAP encoding a signature verification circuit is likely to require at least (and this is a very optimistic estimate) one thousand multiplications for *every* signature that must be checked.

IV. Our Construction of Zero-Knowledge AD-SNARKs

In this section we describe our construction of an AD-SNARK scheme for the satisfiability of arbitrary arithmetic circuits. The scheme can be used with either secret or public verifiability. The main difference between the two verification modes is that the size of the proof in the secretly verifiable case is a fixed constant, whereas in the publicly verifiable case, the proof grows linearly with the number of authenticated statement values. Although we lose constant-size proofs for public verifiability, we stress that: (i) proofs are linear only in the number $N \leq n$ of authenticated values and their size

does not depend on the complexity of the circuit, and (ii) the verification algorithm runs linearly in N in any case (even in the generic construction). Furthermore, when considering concrete implementations and applications, although the proof size of ADSNARK with public verifiability is not constant, it still scales very well, e.g., the size of an ADSNARK proof for a monthly electricity bill is under 170 KB vs. a constant-size proof of 0.3 KB when using the generic scheme with [11]. In contrast, when considering the prover's performance, ADSNARK remains in the realm of practicality – 18 seconds for a monthly bill – whereas for the generic scheme the timing goes up to 10 minutes.

For verifiers that know the secret authentication key (e.g., as in a smart metering/insurance application where companies install a symmetric key in the devices), ADSNARK proofs have constant size, and – crucially – the knowledge of such a secret key by the verifier does *not* compromise privacy.

Our scheme is proven secure under two computational assumptions in bilinear groups, the q-Diffie-Hellman Exponent assumption (q-DHE) [21] and the q-Power Knowledge of Exponent assumption (q-PKE) [23]. We note that the latter one is a non-falsifiable assumption. As discussed in Section VI, this kind of assumption is likely to be inherent for SNARKs for $\mathcal{NP}$. For privacy, we show that the scheme offers statistical zero-knowledge. We stress that this property holds even against adversaries who know (and even generate) the authentication keys.

A detailed description of our scheme follows.

$\mathsf{Setup}(1^\lambda)$: On input the security parameter 1^λ, run $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2) \leftarrow_{\mathcal{R}} \mathcal{G}(1^\lambda)$ to generate a bilinear group description, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are groups of the same prime order $p > 2^\lambda$, $\mathcal{P}_1 \in \mathbb{G}_1$ and $\mathcal{P}_2 \in \mathbb{G}_2$ are the respective generators, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable bilinear map. We let the finite field $\mathbb{F}$ be the set of integers modulo p.

$\mathsf{AuthKG}(\mathsf{pp})$: Create a key pair $(\mathsf{sk}', \mathsf{vk}') \leftarrow_{\mathcal{R}} \Sigma.\mathsf{KG}(1^\lambda)$ for a regular signature scheme. Run $(S, \mathsf{prfpp}) \leftarrow_{\mathcal{R}} \mathsf{F.KG}(1^\lambda)$ to obtain the seed S and the public parameters prfpp of a pseudorandom function $\mathsf{F}_S : \{0,1\}^* \to \mathbb{F}$. Choose a random value $\kappa \leftarrow_{\mathcal{R}} \mathbb{F}$ and compute $K_1 = \kappa \mathcal{P}_1 \in \mathbb{G}_1$, $K_2 = \kappa \mathcal{P}_2 \in \mathbb{G}_2$. Return the secret key $\mathsf{sk} = (\mathsf{sk}', S, \kappa)$, the public verification key $\mathsf{vk} = (\mathsf{vk}', K_2)$, and the public authentication parameters $\mathsf{pap} = (\mathsf{pp}, \mathsf{prfpp}, K_1)$.

$\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, x)$: To authenticate a value $x \in \mathbb{F}$ with label L, generate $\phi \leftarrow \mathsf{F}_S(\mathsf{L})$ using the PRF, compute $\mu = \phi + \kappa x \in \mathbb{F}$ and $\Phi = \phi \mathcal{P}_2 \in \mathbb{G}_2$. Then compute a signature $\sigma' \leftarrow_{\mathcal{R}} \Sigma.\mathsf{Sign}(\mathsf{sk}', \Phi|\mathsf{L})$, and output the tag $\sigma = (\mu, \Phi, \sigma')$.

$\mathsf{AuthVer}(\mathsf{vk}, \sigma, \mathsf{L}, x)$: Let $\mathsf{vk} = (\mathsf{vk}', K_2)$ be the verification key. To verify that $\sigma = (\mu, \Phi, \sigma')$ is a valid authentication tag for a value $x \in \mathbb{F}$ with respect to label L, output $\top$ if $\mu \mathcal{P}_2 = \Phi + x K_2$ in $\mathbb{G}_2$, and if $\Sigma.\mathsf{Ver}(\mathsf{vk}', \Phi|\mathsf{L}, \sigma') = 1$. Output $\perp$ otherwise. In the secret key setting (i.e., if vk is replaced by sk), the tag σ can be verified by checking whether $\mu = \mathsf{F}_S(\mathsf{L}) + \kappa \cdot x$.

$\mathsf{Gen}(\mathsf{pap}, C)$: Let $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ be an arithmetic circuit. To generate the keys, proceed as follows.

1) Compute $Q_C = (\vec{a}, \vec{b}, \vec{c}, z) = \mathsf{QAPInst}(C)$ to build a QAP of size m and degree d for C. Recall that $\vec{a}, \vec{b}, \vec{c}$ are vectors of $m+1$ polynomials in $\mathbb{F}^{\leq d-1}[X]$, while the target polynomial $z \in \mathbb{F}[X]$ has degree d. Extend $\vec{a}, \vec{b}, \vec{c}$ with 3 more polynomials each, by setting:

$$\begin{array}{lll} a_{m+1} = z & a_{m+2} = 0 & a_{m+3} = 0 \\ b_{m+1} = 0 & b_{m+2} = z & b_{m+3} = 0 \\ c_{m+1} = 0 & c_{m+2} = 0 & c_{m+3} = z \end{array}$$

Let I_x, I_{mid} be the following partitions of $\{1, \ldots, m+3\}$: $I_x = \{1, \ldots, n\}$, $I_{mid} = \{n+1, \ldots, m+3\}$. In other words, we partition all the circuit wires into the n statement wires I_x, and the remaining "internal" wires I_{mid} (which include the h witness wires).

2) Pick $\rho_\mathsf{a}, \rho_\mathsf{b}, \tau, \alpha_\mathsf{a}, \alpha_\mathsf{b}, \alpha_\mathsf{c}, \beta, \gamma \leftarrow_{\mathcal{R}} \mathbb{F}$ uniformly at random, set $\rho_\mathsf{c} = \rho_\mathsf{a} \cdot \rho_\mathsf{b}$, and compute the following values:

$$\begin{array}{ll} Z = z(\tau)\rho_\mathsf{c}\,\mathcal{P}_2 & K_\mathsf{a} = z(\tau)\,\rho_\mathsf{a}\,K_1 \\ \forall k \in \{0, ..., m+3\}: & \\ A_k = a_k(\tau)\rho_\mathsf{a}\,\mathcal{P}_1 & A'_k = \alpha_\mathsf{a}\,a_k(\tau)\rho_\mathsf{a}\,\mathcal{P}_1 \\ B_k = b_k(\tau)\rho_\mathsf{b}\,\mathcal{P}_2 & B'_k = \alpha_\mathsf{b}\,b_k(\tau)\rho_\mathsf{b}\,\mathcal{P}_1 \\ C_k = c_k(\tau)\rho_\mathsf{c}\,\mathcal{P}_1 & C'_k = \alpha_\mathsf{c}\,c_k(\tau)\rho_\mathsf{c}\,\mathcal{P}_1 \\ E_k = \beta(a_k(\tau)\rho_\mathsf{a} + b_k(\tau)\rho_\mathsf{b} + c_k(\tau)\rho_\mathsf{c})\,\mathcal{P}_1 & \end{array}$$

3) Output the *evaluation key* EK_C and the *verification key* VK_C defined as follows:

$$\begin{aligned} \mathsf{EK}_C = (&\ Q_C,\ \vec{A},\ \vec{A}', \vec{B},\ \vec{B}', \vec{C},\ \vec{C}', \vec{E}, \\ &\{\tau^i\,\mathcal{P}_1\}_{i \in \{0,\ldots,d\}},\ K_\mathsf{a}\ ) \\ \mathsf{VK}_C = (&\ \mathcal{P}_1,\ \mathcal{P}_2,\ \alpha_\mathsf{a}\,\mathcal{P}_2,\ \alpha_\mathsf{b}\,\mathcal{P}_1,\ \alpha_\mathsf{c}\,\mathcal{P}_2,\ Z, \\ &\gamma\,\mathcal{P}_2,\ \beta\,\gamma\,\mathcal{P}_1,\ \beta\,\gamma\,\mathcal{P}_2,\ \{A_k\}_{k=0}^n\ ) \end{aligned}$$

$\mathsf{Prove}(\mathsf{EK}_C, \vec{x}, \vec{w}, \vec{\sigma})$: Let EK_C be an evaluation key defined as above, $(\vec{x}, \vec{w}) \in \mathbb{F}^n \times \mathbb{F}^h$ be a statement-witness pair, and $\sigma = (\sigma_1, \ldots, \sigma_n)$ be a tuple of authentication tags for x such that, for any $i \in [n]$, either $\sigma_i = (\mu_i, \Phi_i, \sigma'_i)$ or $\sigma_i = \star$. We define $I_\sigma = \{i \in I_x : \sigma_i \neq \star\} \subseteq I_x$ as the set of indices for which there is an authenticated statement value, and let $I_\star = I_x \setminus I_\sigma$ be its complement. To produce a proof for the satisfiability of $C(\vec{x}, \vec{w}) = 0^l$ proceed as follows.

1) Compute $\vec{s} = \mathsf{QAPwit}(C, \vec{x}, \vec{w}) \in \mathbb{F}^m$ (and recall that $s_i = x_i$ for all $i \in [n]$).

2) Randomly sample $\delta^\sigma_\mathsf{a}, \delta^{mid}_\mathsf{a}, \delta_\mathsf{b}, \delta_\mathsf{c} \leftarrow_{\mathcal{R}} \mathbb{F}$, and set $\delta_\mathsf{a} = \delta^\sigma_\mathsf{a} + \delta^{mid}_\mathsf{a}$. Also, define the vector $\vec{u} = (1, \vec{s}, \delta_\mathsf{a}, \delta_\mathsf{b}, \delta_\mathsf{c}) \in \mathbb{F}^{m+4}$.

3) Solve the QAP Q_C by computing the coefficients $(h_0, \ldots, h_d) \in \mathbb{F}^{d+1}$ of the polynomial $h \in \mathbb{F}[X]$ such that $h(X)z(X) = a(X)b(X) - c(X)$, where $a, b, c \in \mathbb{F}[X]$

are

$$\begin{aligned} a(X) &= a_0(X) + \sum_{k\in[m]} s_k \cdot a_k(X) + \delta_{\mathsf{a}} \cdot z(x) \\ b(X) &= b_0(X) + \sum_{k\in[m]} s_k \cdot b_k(X) + \delta_{\mathsf{b}} \cdot z(x) \\ c(X) &= c_0(X) + \sum_{k\in[m]} s_k \cdot c_k(X) + \delta_{\mathsf{c}} \cdot z(x) \end{aligned}$$

Then compute $H = h(\tau)\,\mathcal{P}_1$ using the values $\tau^i\,\mathcal{P}_1$ contained in the evaluation key EK_C. Note that we have $a(X) = \langle \vec{u}, \vec{a} \rangle$, $b(X) = \langle \vec{u}, \vec{b} \rangle$ and $c(X) = \langle \vec{u}, \vec{c} \rangle$.

4) Compute the following values:

$$\begin{aligned} \pi_{\mathsf{b}} &= \langle \vec{u}, \vec{B} \rangle & \pi_{\mathsf{c}} &= \langle \vec{u}, \vec{C} \rangle \\ \pi'_{\mathsf{b}} &= \langle \vec{u}, \vec{B'} \rangle & \pi'_{\mathsf{c}} &= \langle \vec{u}, \vec{C'} \rangle \\ \pi_\sigma &= \langle \vec{u}, \vec{A} \rangle_{I_\sigma} + \delta^\sigma_{\mathsf{a}}\, A_{m+1} \\ \pi'_\sigma &= \langle \vec{u}, \vec{A'} \rangle_{I_\sigma} + \delta^\sigma_{\mathsf{a}}\, A'_{m+1} \\ \pi_{mid} &= \langle \vec{u}, \vec{A} \rangle_{I_{mid}} - \delta^\sigma_{\mathsf{a}}\, A_{m+1} \\ \pi'_{mid} &= \langle \vec{u}, \vec{A'} \rangle_{I_{mid}} - \delta^\sigma_{\mathsf{a}}\, A'_{m+1} \\ \pi_E &= \langle \vec{u}, \vec{E} \rangle \end{aligned}$$

5) Authenticate the value π_σ by computing

$$\pi_\mu = \langle \vec{\mu}, \vec{A} \rangle_{I_\sigma} + \delta^\sigma_{\mathsf{a}}\, K_{\mathsf{a}}$$

6) Construct and return proof π as the tuple $(\pi_\mu, \pi_\sigma, \pi'_\sigma, \pi_{mid}, \pi'_{mid}, \pi_{\mathsf{b}}, \pi'_{\mathsf{b}}, \pi_{\mathsf{c}}, \pi'_{\mathsf{c}}, \pi_E, H)$. To make the proof publicly verifiable, include also $\{\Phi_k, \sigma'_k\}_{k\in I_\sigma}$.

$\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_C, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \pi)$: Let VK_C be the verification key for the circuit C, $\vec{\mathsf{L}} = (\mathsf{L}_1, \ldots, \mathsf{L}_n)$ be a vector of labels, and let π be a proof as defined above. In a similar way as in Prove, we define $I_\sigma = \{i \in I_x : \mathsf{L}_i \neq \star\} \subseteq I_x$ and $I_\star = I_x \setminus I_\sigma$. The verification algorithm computes $A_\star = A_0 + \langle \vec{x}, \vec{A} \rangle_I$ and proceeds as follows:

($A.1^{secret}$) If verification is done using the secret key $\mathsf{sk} = (S, \kappa)$, check the authenticity of π_σ against the labels $\vec{\mathsf{L}}$ by checking whether the following equation holds in $\mathbb{G}_1$:[4]

$$\pi_\mu = \langle \mathsf{F}_S(\vec{\mathsf{L}}), \vec{A} \rangle_{I_\sigma} + \kappa\, \pi_\sigma$$

($A.1^{public}$) If the verification is performed using the public verification key $\mathsf{vk} = (\mathsf{vk}', K_2)$: first, check the validity of all Φ_k by verifying that $\Sigma.\mathsf{Ver}(\mathsf{vk}', \Phi_k | \mathsf{L}_k, \sigma'_k) = 1$ for all $k \in I_\sigma$; second, check the authenticity of π_σ by verifying that the following equation is satisfied over $\mathbb{G}_T$:

$$e(\pi_\mu, \mathcal{P}_2) = \prod_{k\in I_\sigma} e(A_k, \Phi_k) \cdot e(\pi_\sigma, K_2)$$

($A.2$) Check the validity of knowledge commitments for the authenticated values:

$$e(\pi'_\sigma, \mathcal{P}_2) = e(\pi_\sigma, \alpha_{\mathsf{a}}\mathcal{P}_2)$$

[4]The expansion of $\langle \mathsf{F}_S(\vec{\mathsf{L}}), \vec{A} \rangle_I$ is defined as the component-wise application of F, i.e., $\prod_{i\in I} \mathsf{F}_S(\mathsf{L}_i) \cdot A_i$.

($P.1$) Check the satisfiability of the QAP:

$$e(A_\star + \pi_\sigma + \pi_{mid}, \pi_{\mathsf{b}}) = e(H, Z) \cdot e(\pi_{\mathsf{c}}, \mathcal{P}_2)$$

($P.2$) Check the validity of knowledge commitments:

$$\begin{aligned} e(\pi'_{mid}, \mathcal{P}_2) &= e(\pi_{mid}, \alpha_{\mathsf{a}}\mathcal{P}_2) \wedge \\ e(\pi'_{\mathsf{b}}, \mathcal{P}_2) &= e(\alpha_{\mathsf{b}}\mathcal{P}_1, \pi_{\mathsf{b}}) \wedge \\ e(\pi'_{\mathsf{c}}, \mathcal{P}_2) &= e(\pi_{\mathsf{c}}, \alpha_{\mathsf{c}}\mathcal{P}_2) \end{aligned}$$

($P.3$) Check that all the QAP linear combinations use the same coefficients:

$$e(\pi_E, \gamma\mathcal{P}_2) = e(A_\star + \pi_\sigma + \pi_{mid} + \pi_{\mathsf{c}}, \beta\gamma\mathcal{P}_2) \cdot e(\beta\gamma\mathcal{P}_1, \pi_{\mathsf{b}})$$

If all checks above are satisfied, return $\top$; otherwise $\bot$.

$\mathsf{ReRand}(\mathsf{EK}_C, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \pi)$: The scheme also allows for perfect re-randomization of an existing proof, say π given by tuple $(\pi_\mu, \pi_\sigma, \pi'_\sigma, \pi_{mid}, \pi'_{mid}, \pi_{\mathsf{b}}, \pi'_{\mathsf{b}}, \pi_{\mathsf{c}}, \pi'_{\mathsf{c}}, \pi_E, H)$. If π verifies for a set of labels L and a set of non-authenticated values $\{x_i\}_{\mathsf{L}_i=\star}$, then π can be re-randomized as follows. First, choose random values $\tilde{\delta}^\sigma_{\mathsf{a}}, \tilde{\delta}^{mid}_{\mathsf{a}}, \tilde{\delta}_{\mathsf{b}}, \tilde{\delta}_{\mathsf{c}} \leftarrow_{\mathcal{R}} \mathbb{F}$, and set $\tilde{\delta}_{\mathsf{a}} = \tilde{\delta}^\sigma_{\mathsf{a}} + \tilde{\delta}^{mid}_{\mathsf{a}}$. Second, compute

$$\begin{aligned} \tilde{\pi}_{\mathsf{b}} &= \pi_{\mathsf{b}} + \tilde{\delta}_{\mathsf{b}}\, B_{m+2} & \tilde{\pi}'_{\mathsf{b}} &= \pi'_{\mathsf{b}} + \tilde{\delta}_{\mathsf{b}}\, B'_{m+2} \\ \tilde{\pi}_{\mathsf{c}} &= \pi_{\mathsf{c}} + \tilde{\delta}_{\mathsf{c}}\, C_{m+3} & \tilde{\pi}'_{\mathsf{c}} &= \pi_{\mathsf{c}} + \tilde{\delta}_{\mathsf{c}}\, C'_{m+3} \\ \tilde{\pi}_\sigma &= \pi_\sigma + \tilde{\delta}^\sigma_{\mathsf{a}}\, A_{m+1} & \tilde{\pi}'_\sigma &= \pi'_\sigma + \tilde{\delta}^\sigma_{\mathsf{a}}\, A'_{m+1} \\ \tilde{\pi}_{mid} &= \pi_{mid} + \tilde{\delta}^{mid}_{\mathsf{a}}\, A_{m+1} \\ \tilde{\pi}'_{mid} &= \pi'_{mid} + \tilde{\delta}^{mid}_{\mathsf{a}}\, A'_{m+1} \\ \tilde{\pi}_E &= \pi_E + \tilde{\delta}_{\mathsf{a}}\, E_{m+1} + \tilde{\delta}_{\mathsf{b}}\, E_{m+2} + \tilde{\delta}_{\mathsf{c}}\, E_{m+3} \\ \tilde{\pi}_\mu &= \pi_\mu + \tilde{\delta}^\sigma_{\mathsf{a}}\, K_{\mathsf{a}} \\ \tilde{H} &= H + \tilde{\delta}_{\mathsf{a}}\pi_{\mathsf{b}} + \tilde{\delta}_{\mathsf{b}}\pi_{\mathsf{a}} + \tilde{\delta}_{\mathsf{a}}\tilde{\delta}_{\mathsf{b}} z(\tau)\mathcal{P}_1 - \tilde{\delta}_{\mathsf{c}}\,\mathcal{P}_1 \end{aligned}$$

where $z(\tau)\mathcal{P}_1$ can be included in EK_C. Finally, output the re-randomised proof $\tilde{\pi}$ as
$(\tilde{\pi}_\mu, \tilde{\pi}_\sigma, \tilde{\pi}'_\sigma, \tilde{\pi}_{mid}, \tilde{\pi}'_{mid}, \tilde{\pi}_{\mathsf{b}}, \tilde{\pi}'_{\mathsf{b}}, \tilde{\pi}_{\mathsf{c}}, \tilde{\pi}'_{\mathsf{c}}, \tilde{\pi}_E, \tilde{H})$.
It is not hard to check that $\tilde{\pi}$ is identically distributed as a fresh proof π generated by Prove.

The following theorem shows that the scheme ADSNARK described above is a zero-knowledge AD-SNARK.

Theorem 2: If F is a pseudorandom function, and the q-PKE [23] and the q-DHE [21] assumptions hold, then ADSNARK is a secretly-verifiable zero-knowledge AD-SNARK. Furthermore, if additionally Σ is a secure signature scheme, then ADSNARK is a publicly-verifiable zero-knwoledge AD-SNARK.

Security Intuition. A proof of the theorem appears in the full version [20]. Here we provide only an intuition. The completeness of the scheme follows from the properties of the QAP and from the construction of the authentication mechanism. As an intuition for seeing why the scheme has adaptive proof of knowledge, note that the adversary can fool the verification equation in three ways: (i) a false proof involving a label that was never queried during the game; (ii) a false proof

involving labels $\mathsf{L}_1, \ldots, \mathsf{L}_n$ that were all queried during the experiment for values $x_1, \ldots, x_n$, but such that the element π_σ does not encode all $x_1, \ldots, x_n$; (iii) a false proof in which all labels $\mathsf{L}_1, \ldots, \mathsf{L}_n$ were queried and the element π_σ encodes exactly the same $x_1, \ldots, x_n$ authenticated with the respective labels during the security game. The hardness of breaking the security in case (i) follows by the pseudorandomness of the PRF: equation $(A.1)$ will indeed use a value $\mathsf{F}_S(\mathsf{L}_k)$ that was never seen by the adversary and that is pseudorandomly distributed in $\mathbb{F}$; roughly speaking, this means that $(A.1)$ can be satisfied only with negligible probability $1/p \approx 2^{-\lambda}$. Breaking the security in case (ii) reduces to breaking the security of the linearly-homomorphic MAC, i.e., the adversary should be able to come up with a π_μ and an incorrect π_σ that satisfy equation $(A.1)$. Producing such incorrect values intuitively requires to know the secret key κ: we formally prove this by showing that any adversary breaking the security in this case can be reduced to an adversary breaking the q-DHE assumption [21] in the underlying bilinear groups. Breaking security in the last case (iii) essentially means that the adversary is providing a false proof using the same values that were correctly authenticated. For this case the intuition is that the adversary must break the PGHR SNARK; we formally prove this argument by relating the security of the two schemes accordingly.

A very similar proof applies to the publicly verifiable setting where the main difference is that the hardness of case (i) stems from the hardness of breaking the signature scheme. Note that in the publicly verifiable case, our AD-SNARK satisfies a relaxed notion of succinctness in which the proof does not depend on the complexity of the computation but may depend linearly on the number of authenticated inputs.

Finally, the intuition behind the zero-knowledge property of ADSNARK is that the proof elements $\pi_\sigma, \pi_{mid}, \pi_\mathsf{b}, \pi_\mathsf{c}$ are statistically randomly distributed, while the remaining elements are uniquely determined by the verification equations once the above elements are fixed.

Performance and Comparison. Here we pause to discuss the performance of our scheme ADSNARK in comparison with the PGHR SNARK [9]. More precisely, we consider its optimization proposed by Ben-Sasson et al. [11].

First, we note that the Gen algorithm is virtually the same in both schemes except that in ADSNARK we have one more exponentiation[5] in $\mathbb{G}_1$ to generate $K_\mathsf{a} = z(\tau)\,\rho_\mathsf{a}\,K_1$. Also, from a bandwidth point of view, the evaluation key of EK_C of ADSNARK contains only one more $\mathbb{G}_1$ element, K_a, compared to the evaluation key of PGHR. The verification key instead is the same in both schemes.

Second, let us focus on the differences in the Prove algorithm. ADSNARK's Prove has to compute three more $\mathbb{G}_1$ elements: π_σ, π'_σ, and π_μ. Generating these elements amounts to performing three multi-exponentiations that involve $N = |I_\sigma|$ terms each. When looking at the proof size, ADSNARK's proof contains such three additional elements in the group $\mathbb{G}_1$, plus the signatures $\{\sigma_k\}_{k \in I_\sigma}$ in the publicly verifiable setting.

Third, we analyze the differences between ADSNARK and PGHR in the Ver algorithm. The equations $(P.1)$, $(P.2)$, and $(P.3)$ are identical in both schemes and thus require the same computational effort. In PGHR one computes $A_x = A_0 + \sum_{k=1}^{n} x_k A_k \in \mathbb{G}_1$, whereas in ADSNARK we compute a similar value $A_\star = A_0 + \sum_{k \in I} x_k A_k \in \mathbb{G}_1$ which involves fewer terms: precisely $|I_\star| = n - N$. Then, ADSNARK has to perform some additional computation for verifying equations $(A.1)$ and $(A.2)$. $(A.2)$ costs only two pairings – a constant overhead. The first equation instead requires different computations according to whether we are in the secretly verifiable case ($(\text{A.1})^{secret}$) or in the publicly verifiable case ($(\text{A.1})^{public}$). $(\text{A.1})^{secret}$ requires one multi-exponentiation with $N = |I_\sigma|$ terms (plus the cost of running the PRF which is unnoticeable compared to the multi-exponentiation). Hence, considering the cost of computing A_x in PGHR and the total cost of computing $A_\star$ and $(\text{A.1})^{secret}$ in ADSNARK, these are essentially the same. In other words, ADSNARK's secretly verifiable case is slightly slower than PGHR for the cost of computing two pairings in $(A.2)$.

In the publicly verifiable case, equation $(\text{A.1})^{public}$ requires to check a total of N signatures, $\{\sigma_k\}_{k \in I_\sigma}$, and then to compute $e(\pi_\mu, \mathcal{P}_2)e(\pi_\sigma, -K_2)$ and $\prod_{k \in I_\sigma} e(A_k, \Phi_k)$. In general, note that the verification of such N signatures can be done by using batching techniques, and the "multi pairings" can also be computed efficiently. In particular, as we show in our instantiation, this cost is close to the cost of computing A_x in PGHR. In other words, ADSNARK's publicly verifiable case is slightly slower than PGHR for the cost of computing the pairings in $(A.1)$ and $(A.2)$ and for checking the signatures.

In Section V, we give concrete comparisons resulting from our experiments, which are consistent with the analysis above. Indeed, we show based on concrete timings that ADSNARK performs almost as PGHR used *without* authenticated data. These results conclude that our technique added an important property to the SNARK at almost no cost.

However, for the sake of fairness, we should also consider a comparison of the two protocols when they are used to provide equivalent guarantees, i.e., when proving statements on authenticated data. To this end, we now compare ADSNARK against the best possible instantiation of the generic construction of Section III-B, which we take to be PGHR working with the "extended" circuit C'. We call this scheme AD-PGHR. In our analysis, we assume that the verification of every signature requires an arithmetic circuit with c multiplication gates, and also assume (very optimistically) that this is the only additional cost for the design of C'. This means that: if C yields a QAP of size m and degree d, then C' yields a QAP of, at least, size $m' = m + cN$ and degree $d' = d + cN$.

In AD-PGHR, the performance of Ver remains the same as the one of Ver in PGHR discussed above. On the other hand, the Prove algorithm of AD-PGHR heavily depends on the QAP size m' and degree d'. Precisely, Prove performs

[5]We use the term "exponentiation" only for 'historical' reasons, as $\mathbb{G}_1$ is actually an additive group.

multi-exponentiations with m' and d' terms, and a polynomial division operation whose cost is $O(d' \log^2 d')$.

In conclusion, if we fix a circuit C and a number N of authenticated values, and we compare ADSNARK for circuit C against AD-PGHR for the same C (i.e., PGHR with the extended circuit C'), then we obtain:

For secret verification, both schemes perform almost the same, the only difference being that we need to perform two more pairings; for public verification, ADSNARK has an additional cost of one multi-pairing computation with N terms plus the signature verification. For proof generation, AD-PGHR has to perform additional operations that involve a factor at least linear in $c \cdot N$. We recall from the discussion in Section III-C that such c is likely to be larger than 1000. Therefore, one can see that while our solution charges a little more to the verifier (and only in the public verification case), the costs of our scheme on the prover side can be much cheaper, at least by a factor $c \cdot N$. We confirm the above asymptotic comparison in Section V by showing the experimental results obtained by running our implementation.

V. Evaluation

We now describe our implementation of the ADSNARK scheme proposed in Section IV and then present the experimental results we obtained to support the efficiency and practical applicability claims for our construction.

A. Implementation

We have implemented our ADSNARK scheme as an extension to the libsnark library[6] [10, 11]. Our scheme extends the PGHR SNARK implementation offered by this library and supports the same class of statements expressed in the NP-complete language R1CS (rank-1 constraint systems), which is similar to arithmetic circuit satisfiability. The resulting implementation is totally generic, following the libsnark code writing policies, and can be instantiated with arbitrary digital signatures and PRF constructions (in addition to the various parameterization options already offered by the libsnark library). The source code is available upon request.

The modifications to the original PGHR SNARK implementation required by our extensions were relatively small.[7] In the global parameter generation algorithm, the modifications were limited to one additional exponentiation. In the symmetric verification algorithm, we replaced the computations performed on the (known) inputs with (essentially equivalent) computations on the corresponding authentication elements.[8] In the prover algorithm, the extra code comprises the three multi-exponentiations required to compute the extra authentication elements. Finally, our extensions are most visible in the public verification algorithm where, in addition to the digital signature verification operations, the number of pairings to be computed also increases linearly with the number of authenticated inputs. Our implementation strategy was to employ the optimizations available in the libsnark codebase whenever possible, taking advantage of the existing multi- and batch-exponentiation algorithms. The additional pairing computations required in public verification are performed two-by-two, exploiting the available *double Miller loop* optimization.

For the extra cryptographic components required by our construction, i.e., the generic signature scheme and the PRF mapping labels to field elements, we have turned to the state-of-the-art implementations offered by the most recent version of the Supercop framework.[9] For the signature scheme, we have used the ed25519[10] implementation described in [30], which offers extremely fast batch verification that we incorporated in the ADSNARK public verification algorithm (recall that one signature per input must be verified). For the PRF implementation, we have fixed labels to be 128-bit binary strings and the PRF key to be a 256-bit string partitioned as two AES keys. The PRF construction uses one AES computation to map the input label to a 128-bit pseudorandom seed, applies an independent instance of AES in counter mode to expand the seed to 384 pseudorandom bits, and then uses modular reduction to obtain a pseudorandom 254-bit field element.[11] To select the best ed25519 and AES implementations, we have simply run Supercop on our target machine to exhaustively evaluate all available implementations, and then used the recommendations that this framework produced for the fastest implementations and corresponding compilation options.

Microbenchmarks. All measurements were taken in a modest machine with two Dual-Core AMD Opteron 2218 processors clocked at 1 GHz, with 12 GB RAM. The reported values for every parameter correspond to the median of measurements computed over at least 100 runs. Following the original implementation of the libsnark library, we have equipped our implementation of the verification algorithm with the capability to perform part of the computation off-line. However, all our results pessimistically report the full verification time. The security level was set at 128-bits.

B. Experiments Setup

We have conducted experiments to carry out two types of performance evaluation: the first targeting general circuits, and the second focusing on a concrete application.

General circuits. To obtain our first set of experimental results, we have relied on the libsnark functionality that permits generating random instances of constraint systems of

[6] libsnark is available from https://github.com/scipr-lab/libsnark. Details on the publication of our libsnark extension can be found in the full version.

[7] This would be expected from the theoretical description of our scheme, but praise should also go to the developers of the libsnark library, who produced a nice, modular and well documented implementation on which it was easy to build upon.

[8] We deviate slightly from the original implementation in the way we store these input authentication elements. We use a simple (dense) vector representation as opposed to the more elaborate (sparse) map representation in the original. This originated a slight improvement in verification times in the experiments we conducted, but this is simply due to the fact that we did not explore more complex input handling scenarios, where our representation of inputs data might prove less adequate.

[9] http://bench.cr.yp.to/supercop.html

[10] http://ed25519.cr.yp.to/

[11] It is straightforward to prove that this construction yields a secure PRF, assuming that AES is itself a secure PRF.

arbitrary sizes. This allowed us to evaluate the performance of our protocol when dealing with proof goals corresponding to computations of growing complexity and with a varying number of inputs. Our goal here was to corroborate the theoretical analysis presented in Section IV, by benchmarking our protocol against both the original (unauthenticated) PGHR SNARK protocol and the generic AD-SNARK construction described in Section III-B instantiated with PGHR, that we call AD-PGHR.

We have arbitrarily fixed the complexity of the computation associated with the proof goal to involve 50K restrictions (or equivalent, roughly 50K multiplication gates), which typically corresponds to a computation of intermediate complexity according to the state of the art (see for example [9]). The concrete size of the computation is not important, since we will be concerned with the relative degradation of the performance of the various protocols, as we gradually increase the number of (possibly authenticated) inputs to the computation from 100 to 1000. For the generic construction AD-PGHR, we have (very optimistically) taken the penalty for including the signature verification circuit in the proof goal to be only of 1000 multiplications per signature. The fact that, in practice, the cost will probably be higher only strengthens our claims.

Concrete Application. Our second set of experimental results targets a real-world scenario, where the security guarantees provided by an AD-SNARK are highly relevant: a concrete smart-metering application like the one described in the introduction. Analogous results can be obtained for similar applications such as the pay-as-you-drive insurance or the health risk assessment. Our goal here is to indeed demonstrate the practical applicability of our ADSNARK implementation and to show that the overhead incurred by the generic construction can be prohibitive in practice, as it may lead to a significant increase in the complexity of the proof goal. This is particularly true if the proof goal is reasonably simple to start with, as is the case in the application that follows.

We focus on the smart-metering application described in [5, 6] where a (non-linear) cumulative price function is applied to the consumption measurements in order to determine the aggregated cost. The idea here is that the smart meter is able to authenticate the measurements, and that the client locally computes the monetary value corresponding to the measured consumption. The client can then use an AD-SNARK protocol to demonstrate to the supplier that the computation is correct and based on legitimate measurements, without divulging the details of the individual values. As a simple example of a cumulative policy [5], one may think of a non-linear function defined by the following list of threshold/price pairs: $[(0, 2), (3, 5), (7, 8)]$. This policy establishes four consumption intervals and their corresponding prices, as follows: $[0, 3] \rightarrow 2$, $(3, 7] \rightarrow 5$, $(7, \infty) \rightarrow 8$. For a measured consumption of 9, the price due is $3 \times 2 + 4 \times 5 + 2 \times 8 = 42$.

In this application, the complexity of the price computation depends on both the number of measurements and the number of intervals prescribed by the cost function.

We have implemented a generator of R1CS statements that, for a specified number of measurements and a concrete cumulative cost function, is able to construct a constraint system for an arithmetic circuit that checks the correctness of the computed cost, for any given set of measurements. The number of multiplication gates in (i.e., the number of constraints associated to) the resulting circuits is $36 \times \#\mathsf{measurements} \times \#\mathsf{intervals} + 1$.[12] For the generic construction AD-PGHR, we have again used the estimate of 1000 additional multiplications per signature verification. We set the number of thresholds to 5 (a coarse level of granularity in specifying the non-linear policy) so that we obtain a moderately sized circuit even for a month's worth of readings. We then take the indicative value of 48 measurements per day, and vary the number of days separating the price computation to be 1, 7, 14, 21, and 28 days. The policy is defined by thresholds 5, 10, 15, 20, and 25. The measurement values were sampled at random in the range 0 to 100.

C. Performance for General Circuits

Figure 6 shows the results we obtained in terms of execution time. It is clear from the graphs the rapid degradation of the global generation and proving times in the case of AD-PGHR. This is a direct consequence of increasing the size of the circuit and corresponding increase in the size of the proving key, which for 1000 inputs in AD-PGHR approaches 320 MB, as opposed to 15 MB for ADSNARK and PGHR.[13] The (relatively) small penalty payed for using public verification in ADSNARK is visible in the verification times. Furthermore, it is interesting to observe that the secret-key verification of ADSNARK is as fast as the one of AD-PGHR or the (unauthenticated) PGHR. The size of the proof is under 500 bytes for all protocols except the public verification version of AD-PGHR, where the authentication data takes an additional 128 bytes per input. Even so, for 1000 inputs, the proof size is under 126Kbytes.[14]

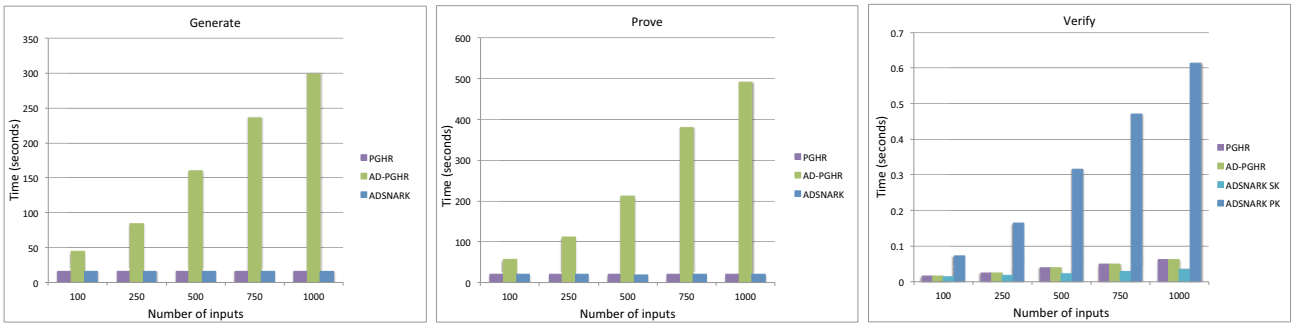
D. Performance for Smart Metering Billing

Figure 7 shows the results we obtained in terms of execution time. It is clear from the graphs that ADSNARK yields proving times that are compatible with real-world deployment: even for one month's worth of measurements, the proving time is around 18 seconds, the proof size is under 0.5 KB for secret verification and under 170 KB for public verification. The contrast to AD-PGHR is evident, where the proof size is essentially the same as ADSNARK with secret verification, but the running time of the AD-PGHR's prover goes up to over 10 minutes. Moreover, even for a month's worth of readings, ADSNARK would pay little more time for public verification (around 0.8 seconds vs. 0.08 seconds of AD-PGHR). Although

[12]The circuit implementation assumes that measurements and thresholds are represented as 32-bit integer values.

[13]For PGHR and ADSNARK, the variations in generation and proving times with the increasing number of inputs are barely visible due to the fact that the number of constraints in the circuit is fixed at 50K.

[14]In our implementation, each signature and public key takes 64 bytes, and the group element takes 64 bytes per input.



	Generation Time (seconds)			Proving Time (seconds)			Verification Time (seconds)			
Inputs	PGHR	AD-PGHR	ADSNARK	PGHR	AD-PGHR	ADSNARK	PGHR	AD-PGHR	ADSNARK SK	ADSNARK PK
100	16.259	44.441	16.269	19.600	56.349	19.558	0.017	0.017	0.014	0.073
250	16.312	84.695	16.358	19.651	111.008	19.597	0.025	0.025	0.017	0.165
500	16.317	159.943	16.335	19.561	212.162	19.473	0.038	0.038	0.023	0.316
750	16.344	236.379	16.307	19.602	380.563	19.672	0.050	0.050	0.029	0.470
1 000	16.350	299.314	16.276	19.513	490.852	19.612	0.062	0.062	0.035	0.613

	Proving Key Size (KBytes)			Verification Key Size (KBytes)			Proof size (Kbytes)			
Inputs	PGHR	AD-PGHR	ADSNARK	PGHR	AD-PGHR	ADSNARK	PGHR	AD-PGHR	ADSNARK SK	ADSNARK PK
100	15 650	45 944	15 657	3.5	3.5	3.5	0.3	0.3	0.4	12.9
250	15 640	91 885	15 657	8.2	8.2	8.2	0.3	0.3	0.4	31.6
500	15 622	167 092	15 657	16.0	16.0	16.0	0.3	0.3	0.4	62.9
750	15 605	250 459	15 657	23.8	23.8	23.8	0.3	0.3	0.4	94.1
1 000	15 587	318 590	15 657	31.5	31.5	31.5	0.3	0.3	0.4	125.4

Figure 6. Experimental results showing generation, proving and verification times for random constraint systems of size 50K and varying number of inputs. For AD-PGHR, the number of multiplication gates is $50K + 1000 \times \#\mathsf{inputs}$. For ADSNARK in the public verification variant, the proof size is equal to the SNARK proof size plus the size of the authentication data, which is 128 bytes per input.

this may not be very important for smart-metering, it shows, once more, that the public verification time scales very well.

VI. Further Related Work

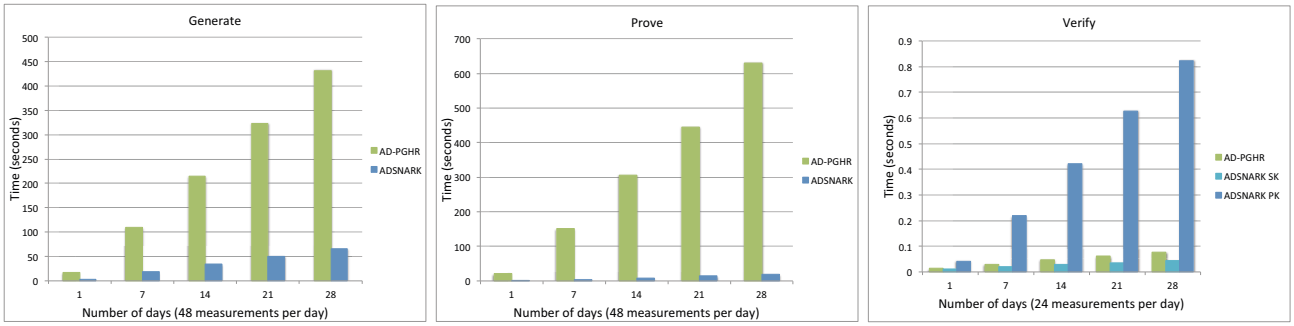
As we mentioned earlier, our work extends the notion of succinct non-interactive arguments of knowledge (SNARKs) [16, 24], which in turn build on (succinct) interactive proofs [8] and interactive arguments [31, 32]. In particular, we focus on the so-called *preprocessing model* where the verifier is required to run an expensive but re-usable key generation phase. In this preprocessing model, several works [17, 23, 33, 34] proposed efficient realizations of SNARKs, and more recent works [9–11] have shown efficient, highly-optimized, implementations that support general-purpose computations. These schemes can also support zero-knowledge proofs. It is worth mentioning that all known SNARKs are either in the random oracle model or rely on non-standard non-falsifiable assumptions [35]. Assumptions from this class have been shown [36] likely to be inherent for SNARKs for $\mathcal{NP}$.

The notion of SNARKs is also related to *verifiable computation* [37], in which a (computationally weak) client delegates the computation of a function to a powerful server and wants to verify the result efficiently. As noted in previous work, by using SNARKs for $\mathcal{NP}$, it is possible to construct a verifiable computation scheme, and several works [9, 10, 17] indeed follow this approach. However, alternative approaches to realizing verifiable computation have been proposed, notably based on fully homomorphic encryption [37–39] or attribute-based encryption [40].

The Pantry system [41] considers an extension of verifiable computation to a setting similar to ours, where the verifier may not know the full input. The Pantry solution in this model combines memory-checking techniques with verifiable computation. In particular, Pantry heavily relies on proving the correctness of hash computations which suffer the same efficiency problems as those discussed in Section III-C.

Another line of work which is closely related to ours is the one on *homomorphic authentication* (comprising both homomorphic/malleable signatures [42–45] and MACs [18, 19, 25]). The main idea of homomorphic authenticators is that, given a set of messages $(\sigma_1, \ldots, \sigma_n)$ authenticated using a secret key sk, anyone can evaluate a program P on such authenticated messages in a way that the result $\sigma \leftarrow P(\{\sigma_i\})$ is again authenticated with respect to the same key sk (or some public key vk in the case of signatures). Some works in this area [44, 45] considered various privacy notions (called context-hiding) to model that signatures on the outputs of a computation should not reveal information about the inputs. In this sense, AD-SNARKs are closely related to the notion of multi-input malleable signatures [45]. However, to the best of our knowledge, none of these schemes achieves practical efficiency for arbitrary computations.

The recent work $Z\emptyset$ [7] aimed to combine the best of different zero-knowledge proof systems by doing an efficiency cost analysis to use the best one for every application. In particular, $Z\emptyset$ relies on both ZQL and Pinocchio [9]. However, when using Pinocchio with authenticated data, $Z\emptyset$ does not provide any guarantee on the integrity of this data, i.e., on the



		Generation Time (seconds)		Proving Time (seconds)		Verification Time (seconds)		
Days	Mgates	AD-PGHR	ADSNARK	AD-PGHR	ADSNARK	AD-PGHR	ADSNARK SK	ADSNARK PK
1	8 641	17.929	3.262	21.760	0.622	0.013	0.013	0.042
7	60 481	110.164	18.296	151.146	4.463	0.030	0.020	0.219
14	120 961	214.457	34.507	306.705	9.078	0.047	0.028	0.421
21	181 441	213.647	50.770	444.592	14.314	0.062	0.037	0.628
28	241 921	431.341	65.539	629.003	18.426	0.077	0.043	0.823

		Proving Key Size (KBytes)		Verification Key Size (KBytes)		Proof size (Kbytes)		
Days	Mgates	AD-PGHR	ADSNARK	AD-PGHR	ADSNARK	AD-PGHR	ADSNARK SK	ADSNARK PK
1	8 641	17 463	2 500	1.9	1.9	0.3	0.4	6.4
7	60 481	124 274	17 641	10.9	10.9	0.3	0.4	42.4
14	120 961	248 547	35 282	21.3	21.4	0.3	0.4	84.4
21	181 441	364 661	52 923	31.8	31.8	0.3	0.4	126.4
28	241 921	497 094	70 563	42.2	42.3	0.3	0.4	168.4

Figure 7. Experimental results showing generation, proving, and verification times for the smart metering application, with the number of measurements varying from 1 day to 28 days (with 48 measurements per day). For AD-PGHR, the number of multiplication gates is $\#\mathsf{Mgates} + 1000 \times \#\mathsf{days} \times 48$. For ADSNARK in the public verification variant, the proof size is equal that of the SNARK proof plus the size of the authentication data (128 bytes per input).

validity of the corresponding signatures.

VII. Conclusions

This paper presents and addresses the problem of enabling privacy-preserving (aka zero-knowledge) data processing with a specific focus on the case where the input data is authenticated, and solely the authentication guarantees "percolate" to the resulting proof, without disclosing information on the original data. Current approaches to solve this problem are limited in either the class of computations that can be supported [6], or in the prover's scalability (as we show in our experiments).

In this paper, we propose a formal approach to this three-party problem via a new cryptographic primitive, AD-SNARK, of which we propose an efficient realization. Starting from our realization, we build and evaluate a nearly practical system, ADSNARK, for proving arbitrary computations over authenticated data in a privacy-preserving way.

Our experimental evaluations show that ADSNARK performs essentially as well as non-authenticated state of the art solutions [9, 11], which means that it scales excellently for modest computations. Moreover, ADSNARK dramatically improves over generic solutions to the input authentication problem. Furthermore, since ADSNARK leverages the recent developments in zero-knowledge proof systems, it permits handling arbitrary computations in an easy and usable way. Indeed, any of the available compilers (e.g., [9]) can be used as a front-end tool for translating from high-level languages (e.g., C++) into arithmetic circuit satisfaction problems that can later be passed to the zero-knowledge backend, in our case to ADSNARK.

ADSNARK also inherits some of the limitations of existing SNARKs, such as the use of the circuit computation model. Recent work [11] have shown how to move to more efficient representations such as RAM. We leave it as future work to study the extension of AD-SNARKs to more convenient and efficient computation models.

Acknowledgment

The research of Dario Fiore has been partially supported by the European Commission's Seventh Framework Programme Marie Curie Cofund Action AMAROUT II (grant no. 291803).

Manuel Barbosa was partially supported by: i. the BEST CASE project (NORTE-07-0124-FEDER-000056) financed by the North Portugal Regional Operational Programme (ON.2 - O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by national funds, through the Foundation for Science and Technology (FCT); and ii. the European Union's Seventh Framework Program (FP7/2007-2013) grant agreement n. 609611 (PRACTICE).

Michael Backes is supported by the BMBF competence center CISPA.

References

[1] R. M. Reischuk, "Declarative design and enforcement for secure cloud applications," Ph.D. dissertation, Saarland University, Saarbrücken, Germany, 2014.

[2] Vitalconnect, "Healthpatch," http://www.vitalconnect.com, 2014.

[3] BBC, "Google unveils 'smart contact lens' to measure glucose levels," http://www.bbc.com/news/technology-25771907, 2014.

[4] R. Anderson and S. Fuloria, "On the security economics of electricity metering," in *9th Annual Workshop on the Economics of Information Security, WEIS 2010, Harvard University, Cambridge, MA, USA, June 7-8, 2010*, 2010.

[5] A. Rial and G. Danezis, "Privacy-preserving smart metering," in *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, ser. WPES '11. New York, NY, USA: ACM, 2011, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/2046556.2046564

[6] C. Fournet, M. Kohlweiss, G. Danezis, and Z. Luo, "ZQL: A compiler for privacy-preserving data processing," in *Proceedings of the 22Nd USENIX Conference on Security*, ser. SEC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 163–178.

[7] M. Fredrikson and B. Livshits, "ZO: An optimizing distributing zero-knowledge compiler," in *USENIX Security*, 2014.

[8] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[9] B. Parno, C. Gentry, J. Howell, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *IEEE Symposium on Security and Privacy, Oakland*, 2013, corrected version (13 May 2013): http://eprint.iacr.org/2013/279.

[10] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *CRYPTO 2013, Part II*, ser. LNCS, R. Canetti and J. A. Garay, Eds., vol. 8043. Springer, Aug. 2013, pp. 90–108.

[11] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von Neumann architecture," in *USENIX Security*, 2014, pp. 781–796.

[12] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *Commun. ACM*, vol. 28, no. 10, pp. 1030–1044, Oct. 1985. [Online]. Available: http://doi.acm.org/10.1145/4372.4373

[13] I. Damgård, "Payment systems and credential mechanisms with provable security against abuse by individuals," in *CRYPTO'88*, ser. LNCS, S. Goldwasser, Ed., vol. 403. Springer, Aug. 1988, pp. 328–335.

[14] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf, "Pseudonym systems," in *SAC 1999*, ser. LNCS, H. M. Heys and C. M. Adams, Eds., vol. 1758. Springer, Aug. 1999, pp. 184–199.

[15] S. Meiklejohn, C. C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya, "Zkpdl: A language-based system for efficient zero-knowledge proofs and electronic cash," in *Proceedings of the 19th USENIX Conference on Security*, ser. USENIX Security'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 13–13. [Online]. Available: http://dl.acm.org/citation.cfm?id=1929820.1929838

[16] S. Micali, "CS proofs (extended abstracts)," in *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 436–453.

[17] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *EUROCRYPT 2013*, ser. LNCS, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, May 2013, pp. 626–645.

[18] D. Catalano and D. Fiore, "Practical homomorphic MACs for arithmetic circuits," in *EUROCRYPT 2013*, ser. LNCS, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, May 2013, pp. 336–352.

[19] M. Backes, D. Fiore, and R. M. Reischuk, "Verifiable delegation of computation on outsourced data," in *ACM CCS 13*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM Press, Nov. 2013, pp. 863–874.

[20] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk, "ADSNARK: nearly practical and privacy-preserving proofs on authenticated data," Cryptology ePrint Archive, Report 2014/617, 2014.

[21] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *PKC 2009*, ser. LNCS, S. Jarecki and G. Tsudik, Eds., vol. 5443. Springer, Mar. 2009, pp. 481–500.

[22] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *EUROCRYPT 2005*, ser. LNCS, R. Cramer, Ed., vol. 3494. Springer, May 2005, pp. 440–456.

[23] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *ASIACRYPT 2010*, ser. LNCS, M. Abe, Ed., vol. 6477. Springer, Dec. 2010, pp. 321–340.

[24] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *ITCS 2012*, S. Goldwasser, Ed. ACM, Jan. 2012, pp. 326–349.

[25] R. Gennaro and D. Wichs, "Fully homomorphic message authenticators," in *ASIACRYPT 2013, Part II*, ser. LNCS, K. Sako and P. Sarkar, Eds., vol. 8270. Springer, Dec. 2013, pp. 301–320.

[26] D. Boneh and X. Boyen, "Short signatures without random oracles," in *EUROCRYPT 2004*, ser. LNCS, C. Cachin and J. Camenisch, Eds., vol. 3027. Springer, May 2004, pp. 56–73.

[27] B. R. Waters, "Efficient identity-based encryption without random oracles," in *EUROCRYPT 2005*, ser. LNCS, R. Cramer, Ed., vol. 3494. Springer, May 2005, pp. 114–127.

[28] R. Cramer and V. Shoup, "Signature schemes based on the strong RSA assumption," in *ACM CCS 99*. ACM Press, Nov. 1999, pp. 46–51.

[29] X. Boyen, "Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more," in *PKC 2010*, ser. LNCS, P. Q. Nguyen and D. Pointcheval, Eds., vol. 6056. Springer, May 2010, pp. 499–517.

[30] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang, "High-speed high-security signatures," *J. Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012. [Online]. Available: http://dx.doi.org/10.1007/s13389-012-0027-1

[31] J. Kilian, "A note on efficient zero-knowledge proofs and arguments (extended abstract)," in *24th ACM STOC*. ACM Press, May 1992, pp. 723–732.

[32] ——, "Improved efficient arguments (preliminary version)," in *CRYPTO'95*, ser. LNCS, D. Coppersmith, Ed., vol. 963. Springer, Aug. 1995, pp. 311–324.

[33] H. Lipmaa, "Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments," in *TCC 2012*, ser. LNCS, R. Cramer, Ed., vol. 7194. Springer, Mar. 2012, pp. 169–189.

[34] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth, "Succinct non-interactive arguments via linear interactive proofs," in *TCC 2013*, ser. LNCS, A. Sahai, Ed., vol. 7785. Springer, Mar. 2013, pp. 315–333.

[35] M. Naor, "On cryptographic assumptions and challenges (invited talk)," in *CRYPTO 2003*, ser. LNCS, D. Boneh, Ed., vol. 2729. Springer, Aug. 2003, pp. 96–109.

[36] C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *43rd ACM STOC*, L. Fortnow and S. P. Vadhan, Eds. ACM Press, Jun. 2011, pp. 99–108.

[37] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *CRYPTO 2010*, ser. LNCS, T. Rabin, Ed., vol. 6223. Springer, Aug. 2010, pp. 465–482.

[38] K.-M. Chung, Y. Kalai, and S. P. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *CRYPTO 2010*, ser. LNCS, T. Rabin, Ed., vol. 6223. Springer, Aug. 2010, pp. 483–501.

[39] B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: Efficient verification via secure computation," in *ICALP 2010, Part I*, ser. LNCS, S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, Eds., vol. 6198. Springer, Jul. 2010, pp. 152–163.

[40] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *TCC 2012*, ser. LNCS, R. Cramer, Ed., vol. 7194. Springer, Mar. 2012, pp. 422–439.

[41] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish, "Verifying computations with state," in *ACM Symposium on Operating Systems Principles, SOSP 2013*, 2013.

[42] R. Johnson, D. Molnar, D. X. Song, and D. Wagner, "Homomorphic signature schemes," in *CT-RSA 2002*, ser. LNCS, B. Preneel, Ed., vol. 2271. Springer, Feb. 2002, pp. 244–262.

[43] D. Boneh and D. M. Freeman, "Homomorphic signatures for polynomial functions," in *EUROCRYPT 2011*, ser. LNCS, K. G. Paterson, Ed., vol. 6632. Springer, May 2011, pp. 149–168.

[44] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, a. shelat, and B. Waters, "Computing on authenticated data," in *TCC 2012*, ser. LNCS, R. Cramer, Ed., vol. 7194. Springer, Mar. 2012, pp. 1–20.

[45] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Malleable signatures: New definitions and delegatable anonymous credentials," in *Computer Security Foundation (CSF)*, 2014.