zk-SNARKs: A Gentle Introduction

Anca Nitulescu

Abstract

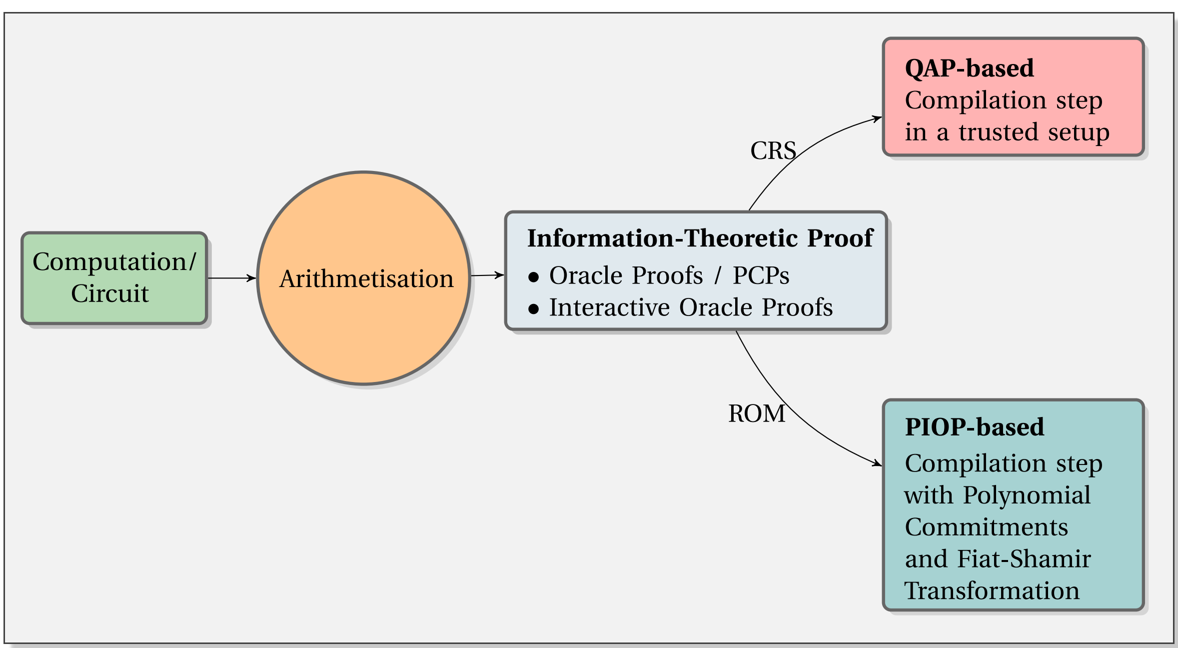
Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) are non-interactive systems with short proofs (i.e., independent of the size of the witness) that enable verifying NP computations with substantially lower complexity than that required for classical NP verification.

This is a short, gentle introduction to zk-SNARKs. It recalls some important advancements in the history of proof systems in cryptography following the evolution of the soundness notion, from first interactive proof systems to arguments of knowledge.

The main focus of this introduction is on zk-SNARKs from first constructions to recent efficient schemes. For the latter, it provides a modular presentation of the frameworks for state-of-the-art SNARKs. In brief, the main steps common to the design of two wide classes of SNARKs are:

- finding a "good" NP characterisation, or arithmetisation
- building an information-theoretic proof system
- compiling the above proof system into an efficient one using cryptographic tools

Arithmetisation is translating a computation or circuit into an equivalent arithmetic relation. This new relation allows to build an information-theoretic proof system that is either inefficient or relies on idealized components called oracles. An additional cryptographic compilation step will turn such a proof system into an efficient one at the cost of considering only computationally bounded adversaries. Depending on the nature of the oracles employed by the initial proof system, two classes of SNARKs can be considered. This introduction aims at explaining in detail the specificity of these general frameworks.



Contents

1 Introduction

The goal of this introduction is to provide a broad overview of zk-SNARKs, starting from explaining the motivations for proof systems in cryptography and giving some early context, then recalling the zk-SNARK history from the first plausible construction, through successive improvements, to today most efficient instantiations. Finally, the more technical part of will formally define the notion of zk-SNARK and the security requirements of these schemes and dive into the details of some celebrated constructions.

1.1 Proof Systems in Cryptography

A *proof system* in the cryptographical sense, is an interactive protocol by which one party (called the prover) wishes to convince another party (called the verifier) that a given statement is true. In zero-knowledge proof, we require further that the proof does not reveal anything more than the truth of the statement. At a first glimpse, it sounds counter-intuitive, being able to prove something is correct, without revealing any extra detail. Let's see that it is perfectly possible by a very simple day-to-day example:

Example 1.1 (Playing card)**.** *Imagine that we pick a card* ***A***$\diamondsuit$ *from a complete deck of playing cards and we want to prove to an adversary that we have a* ***red*** *card in our hand. We can prove that by revealing more information than expressed in the statement, just by showing our card, say it was an ace of diamonds* ***A***$\diamondsuit$*. Alternatively, we can choose to prove nothing more than the colour of our card by revealing to the adversary all the black cards* $\clubsuit$, $\spadesuit$ *left in the deck. Our opponent should now be convinced we have a red card in our hands, but it did not learn anything else about the value of our card.*

Formally, what it means for a protocol to *be zero-knowledge* is defined in terms of a simulator. The simulator is a machine that operates in a different world that is indistinguishable from the real-world for the party that is assessing whether knowledge was leaked. In this world, called the ideal world, the simulator has additional powers that allows it to compute a proof without knowing anything besides the truth of the statement.

Researches in zero-knowledge proofs have been prompted by authentication systems where one party wants to prove its identity to a second party via some secret information such as a password but does not want to disclose anything about its secret password to the second party. This is called a zero-knowledge *proof of knowledge*.

A proof of knowledge is an interactive protocol in which the prover succeeds in "convincing" a verifier that it knows something (a password, the steps of a computation, etc.) associated with the statement. For example, if the statement is "I am Alice.", the prover should show knowledge of the secret password of Alice; if the statement is "I computed the function $f(x)$ and obtained y.", then the prover must convince its verifier that it knows all the steps of this computation that lead to the result y.

What it means for a machine to *have knowledge* is defined formally in terms of an extractor. As the program of the prover does not necessarily spit out the knowledge itself (as is the case for zero-knowledge proofs), we will invoke another machine, called the knowledge extractor that, by having access to the prover, can extract this witness (the knowledge).

The next step is the introduction of non-interactive proof systems, which reduce the number of rounds of interaction between the prover and the verifier to only one. Some non-interactive protocols consist in only one message from the prover to verifier; others need the verifier to generate some setting information, called CRS, that can be made publicly available ahead of time and independently of the statement to be proved later. To enforce security, and avoid cheating from the verifier, this CRS is often generated by a third trusted party.

1.2 SNARKs

In the class of non-interactive proofs, a particularly interesting concept for proving integrity of results for large computations is that of SNARK, i.e., *succinct non-interactive argument of knowledge*. By this term, we denote a proof system which is:

***succinct*:** the size of the proof is very small compared to the size of the statement or the witness, i.e., the size of the computation itself,

***non-interactive*:** it does not require rounds of interaction between the prover and the verifier,

***argument*:** we consider it secure only for provers that have bounded computational resources, which means that provers with enough computational power can convince the verifier of a wrong statement,

***knowledge-sound*:** it is not possible for the prover to construct a proof without knowing a certain so-called witness for the statement; formally, for any prover able to produce a valid proof, there is an extractor capable of extracting a witness ("the knowledge") for the statement.

SNARK systems can be further equipped with a *zero-knowledge* property that enables the proof to be done without revealing anything about the intermediate steps (the witness). We will call these schemes zk-SNARKs.

A (zk-)SNARK protocol (as any other non-interactive proof system) is described by three algorithms that work as follows:

- Gen is the setup algorithm, generating a necessary string crs used later in the proving process and some verification key vrs, sometimes assumed to be secret to the verifier only. It is typically run by a trusted party.
- Prove is the proving algorithm that takes as input the crs, the statement u and a corresponding witness w and outputs the proof π.
- Verify is the algorithm that takes as input the verification key vrs, the statement u and the proof π, and returns 1 "accept" the proof or 0, "reject".

SNARK schemes can be used for delegating computation in the following way: a server can run a computation for a client and non-interactively prove the accuracy of the result. The client can verify the result's correctness in nearly-linear time in the size of the input (instead of running the entire computation itself).

1.2.1 Organisation

Section 2 is rather informal and it gives some high-level intuition and the historical evolution of (zero-knowledge) proofs and arguments.

Then, in Section 5 SNARKs are formally defined and the security requirements for these schemes are stated. Then the focus lies on presenting the most outstanding constructions in the recent years: PCP-based schemes in Section 6, QAP-based schemes in Section 7 and LIP-based SNARKs in Section 8. This part contains many simplifications and examples to help the reader understand step-by-step the frameworks of diverse SNARK constructions.

2 First Proofs and Arguments

Proof systems introduced by [GMR89] are fundamental building blocks in cryptography. Extensively studied aspects of proof systems are the expressivity of provable statements and their efficiency.

Complexity Classes. In order to better understand the role of proofs and their classification, we will briefly and informally introduce some basic complexity notions. The complexity classes will be defined by the type of computational problem, the model of computation, and the resource that are being bounded and the bounds. The resource and bounds are usually stated together, such as "polynomial time", "logarithmic space", "constant depth", etc. We will introduce the main two fundamental complexity classes, P and NP. They are used to classify decision problems.

P versus NP. On the one hand, the class P is the class of languages $\mathcal{L}$, such that there exists an algorithm that takes as input a bit string x and that can decide in polynomial time (in the size of x), whether $x \in \mathcal{L}$. We generally consider this class as the class of easy-to-decide languages and call them polynomial-time algorithms, efficient algorithms.

On the other hand, the class NP is the class of languages $\mathcal{L}$, such that there exists an algorithm, that takes as input two bit strings x and w and that can decide in polynomial time (in the size of x), whether w is a valid proof or witness that $x \in \mathcal{L}$. We suppose that for any statement $x \in \mathcal{L}$, there exists such a witness w, while otherwise ($x \notin \mathcal{L}$) no such witness exists. A formal definition is stated as follows:

Definition 2.1 (The Class NP)**.** *A language $\mathcal{L}$ is in the class* NP *if there exists a polynomial time algorithm $\mathcal{R}_{\mathcal{L}}$ such that*

$$\mathcal{L} = \{x | \exists\, w, |w| = \mathsf{poly}(|x|) \wedge \mathcal{R}_{\mathcal{L}}(x, w) = 1\}.$$

By restricting the definition of NP to witness strings of length zero, we capture the same problems as those in P. While the class P is clearly included in NP, finding whether NP is included in P is one of the most important open problems in computer science.

It basically asks whether being able to efficiently check a proof of a statement, is equivalent to being able to check if a statement is true or false efficiently. Even if we don't have any clear evidence for that, most researchers strongly believe that $\mathsf{P} \neq \mathsf{NP}$.

In cryptography, considerable atention is given to the NP-hard complexity class. NP-hard is the defining property of a class of problems that are, informally, "at least as hard as the hardest problems in NP".

We will often talk about NP-complete decision problems, the ones belonging to both the NP and the NP-hard complexity classes.

Example: Satisfiability Problems SAT**.** As an example for a problem in NP, let us consider the problem of boolean formula satisfiability (SAT). For that, we define a boolean formula using an inductive definition:

- any variable $x_1, x_2, x_3, \ldots$ is a boolean formula
- if f is a boolean formula, then $\neg f$ is a boolean formula (negation)
- if f and g are boolean formulas, then $(f \wedge g)$ and $(f \vee g)$ are boolean formulas (conjunction / and, disjunction / or).

The string $((x_1 \wedge x_2) \wedge \neg x_2)$ would be a boolean formula.

A boolean formula is satisfiable if there is a way to assign truth values to the variables so that the formula evaluates to true. The satisfiability problem SAT is the set of all satisfiable boolean formulas: $\mathsf{SAT}(f) := 1$ if f is a satisfiable boolean formula and 0 otherwise.

The example above, $((x_1 \wedge x_2) \wedge \neg x_2)$, is not satisfiable and thus does not lie in SAT. The witness for a given formula is its satisfying assignment and verifying that a variable assignment is satisfying is a task that can be solved in polynomial time.

The attractive property of this seemingly simple problem is that it does not only lie in NP, it is also NP-complete. It means that it is one of the hardest problems in NP, but more importantly – and that is the definition of NP-complete – an input to any problem in NP can be transformed to an equivalent input for SAT in the following sense:

For any NP-problem $\mathcal{L}$ there is a so-called reduction function f, which is computable in polynomial time such that:

$$\mathcal{L}(x) = \mathsf{SAT}(f(x)).$$

Such a reduction function can be seen as a compiler: It takes source code written in some programming language and transforms it into an equivalent machine language, which has the same semantic behaviour. Since SAT is NP-complete, such a reduction exists for any possible problem in NP.

2.1 Interactive Zero-Knowledge Proofs

By Definition 2.1, the class NP contains all languages for which an unbounded prover can compute deterministic proofs, where a proof is viewed as a string of length polynomial in the statement x.

An interactive proof relaxes these requirements in two directions: first, the prover and the verifier are allowed to use random coins, second, the output of a proof verification should only match the actual truth of the statement with some reasonable enough probability and obviously, there is interaction between parties.

First Interactive Proofs. In two independent seminal papers, that won a Gödel prize, Babai [Bab85] and Goldwasser, Micali, and Rackoff [GMR85] introduced the notion of interactive proofs also known as *Arthur-Merlin proofs*.

Both works studied complexity classes where a computationally unbounded prover must convince a polynomially bounded receiver of the truth of a statement using rounds of interactions. The main difference between the notions studied in these papers is regarding the random coins of the verifier: in the work of Babai, the verifier was required to reveal to the prover all coins that he used during the computation. Such interactive proofs are referred to as public coin interactive proofs, as opposed to private coin interactive proofs, in which the verifier might keep its internal state hidden.

The complexity classes corresponding to public coin interactive proofs were denoted $\mathsf{AM}\left[f(n)\right]$ by Babai, where AM stands for Arthur-Merlin, n is the input length, and $f(n)$ is the allowed number of rounds of interaction. The complexity classes corresponding to private coin interactive proofs were denoted $\mathsf{IP}\left[f(n)\right]$ by Goldwasser, Micali, and Rackoff.

Zero-Knowledge. As pointed out by Goldwasser, Micali, and Rackoff in their seminal paper [GMR85], an essential question about interactive proofs in cryptography is whether the prover reveals more information (or knowledge) to the verifier than the fact that $x \in \mathcal{L}$. Indeed, in cryptography, we often want to hide information. A proof that does not reveal any information to the verifier besides the membership of the statement to the language is called a zero-knowledge proof. A way to formally define this property is to consider a simulator that is able to behave exactly as the prover in the protocol and to produce a "fake" proof without knowing the witness. This should be done in a way that a verifier will not be able to tell if it interacts with the real prover or with this simulator. Intuitively, we can then argue that a honestly generated proof looks indistinguishable from a simulated value produced independently of the witness, meaning that the proof reveals as much information about the witness as this value, so basically zero-knowledge. This concept might seem very counter-intuitive and impossible to achieve. However, in [GMW86], Goldreich, Micali, and Wigderson constructed zero-knowledge proofs for any language in NP, under a very weak assumption, namely the existence of one-way functions.

Succinct Arguments. Related to efficiency and to optimization of communication complexity, it has been shown that statistically-sound proof systems are unlikely to allow for significant improvements in communication [BHZ87, GH98, GVW02, Wee05]. When considering proof systems for NP this means that, unless some complexity-theoretic collapses occur, in a statistically sound proof system any prover has to communicate, roughly, as much information as the size of the NP witness. The search for ways to beat this bound motivated the study of *computationally-sound* proof systems, also called *argument systems* [BCC88], where soundness is required to hold only against *computationally bounded* provers.

Assuming the existence of collision-resistant hash functions, Kilian [Kil92] showed a four-message interactive argument for NP. In this protocol, membership of an instance x in an NP language with NP machine M can be proven with communication and verifier's running time bounded by $p(\lambda, |M|, |x|, \log t)$, where λ is a security parameter, t is the NP verification time of machine M for the instance x, and p is an *universal* polynomial.

Such argument systems where the communication complexity (and sometimes the work of the verifier) sublinear in (or even independent of) the witness size are called *succinct*.

2.2 Non-Interactive Zero-Knowledge Proofs

As we have seen previously, interactive proofs can be understood as a relaxation of the standard non-interactive proofs (captured by the class NP), where we allow interaction (as well as random coins) between the verifier and the prover. In this section, we will focus on protocols that do not require more communication, than a sole message from prover to verifier. In a non-interactive proof or argument, the prover just sends one message (called the proof) to the verifier, and the latter can check it in order to accept it or not. This proof is similar to a witness of an NP language, except that sending a witness often gives too much information to the verifier.

A natural theoretical question is to ask whether there are zero-knowledge randomized proofs that are completely non-interactive (no round of interaction is needed). Such systems are called *non-interactive zero-knowledge proof systems* (NIZK). This question is also very interesting from a practical point of view: in the real world, interactivity means exchanging information over some network, which raises some latency issues. However, the absence of any form of trust strongly narrows the range of feasibility results: We know that there is no hope of building a zero-knowledge proof system in the standard model with a single round of interaction for non-trivial languages [GO94], and strong limitations are also known for two rounds of interaction [GO94, BLV03].

2.2.1 Common Reference String

In light of the strong limitations discussed above, an interesting research direction is to find the minimal trust assumptions one could make that lead to a model in which practically efficient NIZK proof systems can be built.

The common reference string (CRS) model, introduced by Damgård [Dam00], captures the assumption that a trusted setup in which all involved parties get access to the same string crs taken from some distribution $\mathcal{D}$ exists. Schemes proven secure in the CRS model are secure given that the setup was performed correctly. The common reference string model has proven very convenient to use for constructing a large variety of efficient primitives with strong security requirements.

This leaves the open problem of how exactly to execute the required setup. The way to perform this in practice is by organizing a trusted setup ceremony between multiple participants using multi-party computation between users who are believed not to collude.

Other recommended terminology is “Structured Reference String”, since typically the CRS used by SNARKs has some algebraic structure.

2.2.2 First NIZK Schemes

Blum et al. first study the non-interactive zero-knowledge proof system and present the common reference string model that is generally applied at present [BFM88, DMP90]. This first construction of [BFM88] is a bounded NIZK proof system, meaning that for different statements in NP language, the proof system has to use different CRSs and the length of the statement is controlled by the length of CRS. Later, Blum et al. [DMP90] presented a more general (multi-theorem) NIZK proof system for 3SAT by improving the previous one, which allows to prove many statements with the same CRS.

Both [BFM88] and [DMP90] based their NIZK systems on certain number-theoretic assumptions (specifically, the hardness of deciding quadratic residues modulo a composite number). Feige, Lapidot, and Shamir [FLS90] showed later how to construct computational NIZK proofs based on any trapdoor permutation.

Much research has been devoted to the construction of efficient NIZK proofs [Dam93, KP98, BDP00], but back then, the only known method to do so has been the ”hidden random bits” model. This hidden random bits model assumes the prover has a string of random bits, which are secret to the verifier. By revealing a subset of these bits, and keeping the rest secret, the prover can convince the verifier of the truth of the statement in question. Improvements in the efficiency of NIZK proofs have come in the form of various ways to set up a hidden random bits model and how to use it optimally.

2.2.3 Groth-Sahai Proofs

For a long time, two main types of NIZK proof systems were available: efficient but heuristically secure proof systems in the random oracle model and inefficient proof systems in the hidden random bits model [FLS90, Dam93, KP98, BDP00], which can be instantiated in the standard model, under well-studied assumptions. This changed with the arrival of pairing-based cryptography, from which a fruitful line of work (starting with the work of Groth, Ostrovsky, and Sahai [GOS06b, GOS06a]) introduced increasingly more efficient NIZK proof systems in the standard model.

The Groth-Ostrovsky-Sahai proof system was the first perfect NIZK argument system for any NP language and the first universal composable secure NIZK argument for any NP language. This resolved a central open problem concerning NIZK protocols. The mechanism was dramatically different from the previous works, such as Blum-Feldman-Micali proof system [BFM88] and Blum-Santis-Micali-Persiano proof system [DMP90].

This line of work culminated with the framework of Groth-Sahai proofs [GS08], which identified a restricted, yet very powerful class of languages for which efficient pairing-based NIZK could be designed, with security based on essentially any standard assumption on pairing-friendly groups. This framework greatly improved the efficiency and practicability of NIZK and created a new line of research on the applications of NIZK.

Nevertheless, these schemes pose a limitation on the length of the proof statement in order to achieve *adaptive soundness* against dishonest provers who may choose the target statement depending on the CRS. Since the common reference string is public, it would be more natural to define soundness adaptively.

The first adaptively-sound statistical NIZK argument for NP that does not pose any restriction on the statements to be proven requires non-falsifiable assumptions (see [AF07]). Abe and Fehr [AF07] have demonstrated also an impossibility result: no adaptively-sound statistical zero-knowledge NIZK argument for an NP-complete language can have a ”direct black-box” security reduction to a standard cryptographic assumption unless $\mathsf{NP} \subseteq \mathsf{P}/\mathsf{poly}$.

2.3 Interactive Arguments of Knowledge

2.3.1 Interactive Zero-Knowledge Proofs

A zero-knowledge proof or its relaxed version, argument, is a protocol between a prover P and a verifier V for proving that a statement x is in a language $\mathcal{L}$. Informally, such a protocol has to satisfy three properties:

Completeness. An honest verifier always accepts a proof made by an honest prover for a valid word and using a valid witness.

Soundness. No unbounded/PPT adversary can make an honest verifier accept a proof of a word $x \in \mathcal{L}$ either statistically (for zero-knowledge proofs)/computationally (for zero-knowledge arguments).

Zero-knowledge It is possible to simulate (in polynomial-time) the interaction between a (potentially malicious) verifier and an honest prover for any word $x \in \mathcal{L}$ without knowing a witness w.

Honest-Verifier Zero-Knowledge. Honest-verifier zero-knowledge arguments or proofs are similar to the ones defined above, except that we assume that the verifier is not malicious. The zero-knowledge property applies only to verifiers that behave honestly and follow the protocol. This relaxation enables to construct even more efficient schemes.

2.3.2 Knowledge Soundness

The proofs and arguments we discussed so far are tools used for membership statements, $i.e.,$ proving membership of an instance x in a language $\mathcal{L}$. Restricting our attention to NP-languages, such statements can be phrased as existential statements, of the form $\exists w, \mathcal{R}_{\mathcal{L}}(x, w) = 1$. Proofs of knowledge strengthen the security guarantee given by classical zero-knowledge proofs. While a zero-knowledge proof suffices to convince the verifier of the existence of a witness w for the statement, a proof of knowledge additionally shows that the prover knows such a witness.

Several remarks are in order here. First, we have to define what it means for a prover to know such a witness. Intuitively, to make sure a prover has used the witness, it should be possible to "extract" this knowledge from that prover. Informally, this is done as follows: we say that an (efficient) algorithm $\mathcal{A}$ knows a value w if we can build a simulator Sim that, for any such $\mathcal{A}$ that produces an accepting transcript, Sim can extract the witness w from its interaction with $\mathcal{A}$.

Second, an important property of proofs of knowledge is that they can make sense even for statements that are trivial from an existential point of view, i.e., for trivial languages for which a membership witness always exists, but can be hard to compute. We illustrate this with a classical example:

Example 2.2 (Discrete Logarithm Language)**.** *Let $\mathcal{L}_{\mathsf{DLog}}(\mathbb{G}, g)$ denote, for a cyclic group $(\mathbb{G}, \cdot)$ with a generator g, the following language:*

$$\mathcal{L}_{\mathsf{DLog}}(\mathbb{G}, g) = \{h \in \mathbb{G} | \exists\, x \in \mathbb{Z}, g^x = h\}\,.$$

As g is a generator of $\mathbb{G}$, this is a trivial language: all elements of $\mathbb{G}$ belong to $\mathcal{L}_{\mathsf{DLog}}$, $\forall\, h \in \mathbb{G}, \exists\, x \in \mathbb{Z}$ such that $g^x = h$, and this exponent x is not unique. However, computing such an integer x can be computationally infeasible (because of the discrete logarithm assumption, see Assumption 2.3). Therefore, while asking a prover to show the existence of the discrete logarithm of some word h is meaningless, convincing a verifier that a prover knows the discrete logarithm of h in base g gives him a piece of non-trivial information.

Assumption 2.3 (Discrete-Logarithm Assumption)**.** *Given a cyclic group $\mathbb{G}$ of order $n \in \mathbb{N}$ with a generator g, the discrete logarithm assumption over $\mathbb{G}$ states, informally, that it is computationally infeasible given a random group element $h \in \mathbb{G}$ to find an integer $x \in \mathbb{Z}_n$ such that $h = g^x$.*

Hardness of Discrete Logarithm. Generic algorithms to solve discrete logarithm, which are independent of the particular structure of the underlying group $\mathbb{G}$, have a running time proportional to $\sqrt{n}$. In spite of more than four decades of intense cryptanalytic effort, there exist certain groups in which we still do not know any algorithm with better efficiency than the generic algorithms.

3 Fiat-Shamir Heuristic

3.1 Sigma-Protocols

In this part we will describe a specific class of zero-knowledge proof systems to which very efficient zero-knowledge protocols from the literature belong: Σ-protocols [CDS94].

Definition 3.1 (Sigma-Protocol)**.** *A Σ-protocol for a language $\mathcal{L}$ is a public-coin honest-verifier zero-knowledge proof of knowledge, with a particular three-move structure:*

Commit Phase. *P sends to V some commitment values to some randomness,*
Challenge Phase. *V sends to P a uniformly random challenge e,*
Response Phase. *P sends to V an answer $f(w, r, e)$ where f is some public function, and w is the witness held by P*

Example: The Schnorr Protocol. In Example 2.2, we were mentioning the possibility to prove knowledge of the discrete logarithm of some group element h in some base g, where g is the generator of some group $\mathbb{G}$. We now elaborate on this example by describing a Σ-protocol for proving knowledge of a discrete logarithm. The protocol is given in Figure 1. It was first described in [Sch90], and it is commonly used as an authentication protocol: given a public value h, the prover authenticates himself by proving his knowledge of the secret value x associated to this public value (i.e., x is such that $g^x = h$ for a fixed generator g).

$$
\begin{array}{ccc}
\textbf{Prover} & & \textbf{Verifier} \\
(x = \log_g h) & & \\
r \leftarrow_{\$} \mathbb{Z}_p & & \\
a \leftarrow g^r & \xrightarrow{\quad a \quad} & \\
 & \xleftarrow{\quad e \quad} & e \leftarrow_{\$} \mathbb{Z}_p \\
\sigma \leftarrow ex + r & \xrightarrow{\quad \sigma \quad} & g^\sigma \stackrel{?}{=} h^e a
\end{array}
$$

Figure 1: Schnorr Σ-Protocol for DLog Language.

Rewinding. The standard solution to prove the security of Σ-protocols is to use a technique called rewinding. The simulator will run the code of the prover, feeding it with the verifier inputs it requires, and then rewind it to some previous state so as to feed it with different inputs. By doing so, the simulator will be able to get several outputs of the prover with respect to different verifier inputs, starting from some common state of the prover. Intuitively, this allows the simulator to cancel out some randomness that had been introduced by the prover to mask its witness.

Security Analysis (Sketch). We show that the protocol given in figure Figure 1 is perfectly complete, knowledge-extractable, and honest-verifier zero-knowledge.

Perfect completeness. It follows immediately by inspection: if $\sigma = ex + r \mod p$, then $g^\sigma = g^{ex+r} = (g^x)^e g^r = h^e a$.

Honest-verifier zero-knowledge. Let Sim be a simulator which is given the common input $(\mathbb{G}, g, h)$ and the code of the verifier. Sim selects a uniformly random tape for the verifier algorithm and runs it with this random tape on a random input message $a \in \mathbb{G}$. Once the Verifier outputs a challenge e, Sim restarts the protocol, feeding Verifier algorithm with the same random tape and setting the input message a to $g^r h^{-e}$ for a uniformly random r. Note that a is distributed exactly as in an honest execution of the protocol. After the verifier outputs the challenge e (the verifier is assumed honest, so it uses only the coins of his random tape. Hence, this challenge is the same than the one extracted by Sim in the previous run of the verifier), Sim answers with $\sigma := r$. Observe that the equation $g^\sigma = h^e a$ is satisfied for the chosen values of σ and a, and that the answer is distributed exactly as in an honest run, hence the honest-verifier zero-knowledge property.

Knowledge-extraction. Consider a prover that runs in time T and produces an accepting answer with non-negligible probability ε, and let Sim' be a simulator which is given the code of the prover as input. Once the prover outputs the first flow a, Sim' writes a random $e \in \mathbb{Z}_p$ on its message input tape, and gets an answer σ. Then, Sim' rewinds the prover to step 2 of the protocol, feeding it with a new random e' and receiving a corresponding new answer σ'. Observe that if both (σ, σ') are accepting answers, it holds that $g^\sigma = h^e a,\ g^{\sigma'} = h^{e'} a$, which gives $g^{\sigma - \sigma'} = h^{e-e'} = (g^x)^{e-e'}$. In this case, Sim' can obtain x by computing $(\sigma - \sigma')(e - e')^{-1} \pmod p$ (as we have $e \neq e'$ with overwhelming probability). We argue the simulator Sim' for a prover that runs in time T and has success probability ε runs in $\mathcal{O}(T/\varepsilon)$ (the simulator repeats the rewinding procedure at most $1/\varepsilon$ times).

Fiat-Shamir Transformation. The Fiat-Shamir transformation [FS87] is a heuristic method to convert Σ-protocols (see Section 3.1) into non-interactive zero-knowledge proofs. It proceeds as follows: to prove the membership of an instance x to a language $\mathcal{L}$ the prover first computes the first flow (the commitments) of a Σ-protocol for this statement. Let a denote this first flow. Then, the prover sets $e \leftarrow \mathsf{RO}(x, a)$, where RO is some hash function modeled by a random oracle, and computes the last flow (step 3 of the Σ-protocol), using e as the challenge. While this approach leads to very efficient NIZKs, it cannot be proven to work under any standard assumption related to hash functions. Instead, the above methodology can be proven to work only in the random oracle model.

3.2 Random Oracle Model

As already mentioned, security proofs are notoriously difficult to achieve in the standard model, so in many proofs, cryptographic primitives are replaced by idealized versions.

The random oracle model (ROM) [BR93, CGH98] is an idealised cryptographic model and it assumes the existence of a truly random function to which all parties involved in a protocol have access. Since in reality, no such ideal function exists, random oracles are instantiated with hash functions, and one heuristically assumes that a hash function behaves well enough to be a replacement for random oracles. Random oracles allow proving protocols are secure while they are still practically efficient. On the negative side, this model has its disadvantages, as it is seen more as heuristically secure since no truly random hash functions can be used in practice. Some failures of the random oracle methodology when implemented in practice are shown in [CGH98]. They show that there exist signature and encryption schemes that are secure in the ROM, but for which any implementation of the random oracle results in insecure schemes.

3.3 Fiat-Shamir-Compatible Hash Functions.

Still, an open question is whether there exist concrete hash families that are "Fiat-Shamir-compatible" (i.e., that can guarantee soundness and potentially also zero-knowledge for the transformed protocol). Initial results in this direction were negative. Indeed, Goldwasser and Kalai [GK03] (following Barak [Bar01]) demonstrated a three-round, public-coin argument scheme for which applying the Fiat-Shamir transform with any hash family never yields a sound protocol. Furthermore, Bitansky et al. [BDG+13] show that, even when starting with a three-round proof, soundness of the Fiat-Shamir transform with a concrete hash family cannot be proved via black-box reduction to standard, game-based assumptions. In contrast, a recent line of work [KRR17, CCRR18, HL18] circumvents the [BDG+13] impossibility result by using stronger than standard hardness assumptions to construct FS-compatible hash families. Kalai et al. [KRR17] gave the first construction of a hash family that is FS-compatible for arbitrary constant-round (public-coin) interactive proofs, albeit from complex obfuscation assumptions. Canetti et al. ([CCRR18]) then provide alternative families without obfuscation, but using complex KDM-security assumptions on secret-key encryption schemes. It is important to remark that the assumptions made by [KRR17, CCRR18] are non-falsifiable and highly complex in the following sense: both involve an adversary that is in part computationally unbounded.

In two recent companion articles, Canetti et al. [CCH+18, CLW18] construct explicit hash functions that are FS-compatible for a rich class of protocols, and they prove their security under assumptions that are qualitatively weaker than what was previously known. Using these hash families, new results can be obtained for delegation of computation and zero-knowledge.

4 Succinct Non-Interactive Arguments of Knowledge

Succinct proofs were already considered by Kilian [Kil92], whose four-message construction was based on probabilistically checkable proofs (PCP). Starting from Kilian's protocol, Micali [Mic94] used the Fiat-Shamir heuristic to construct a *one-message* succinct argument for NP whose soundness is set in the random oracle model.

New more efficient systems followed in the CRS model, they are called *succinct non-interactive arguments* (SNARGs) [GW11]. The area of SNARGs became quite popular some years later with the proposal of several constructions in the CRS model, some of which gained significant improvements in efficiency [Gro10, Lip12, BCCT12, GGPR13, PHGR13, BCG+13a, DFGK14, Gro16].

The approaches of [GGPR13, PHGR13], which led to concretely efficient proofs were generalized in [BCI+13] under the concept of Linear PCP (LPCP) and extractable linear-only encryption scheme, that is, an encryption scheme where a valid new ciphertext output by the adversary is an affine combination of the encryptions that the adversary sees as input.

4.1 Non-Falsifiable Assumptions

Noteworthy is that all SNARG constructions are based on non-falsifiable assumptions [Nao03b], a class of assumptions that is likely to be inherent in proving the security of SNARGs (without random oracles), as stated by Gentry and Wichs in their work [GW11]. They show that no construction of SNARGs can be proven secure via a black-box reduction from any falsifiable assumption (unless that assumption is already false).

Most standard cryptographic assumptions are falsifiable (e.g., hardness of factoring, DLog, RSA, CDH, etc.) in the sense of the formal notion of cryptographic falsifiability introduced by Naor [Nao03a]. Roughly speaking, a computational hardness assumption is said to be falsifiable if it can be formulated in terms of a challenge: an interactive protocol between an adversary and a challenger (verifier), where an efficient adversary can convince the verifier to accept if and only if the assumption is false, meaning that if the assumption were false, then it would be possible to prove it.

Intuitively, assumptions that are not falsifiable are more laborious to reason about, and therefore we have significantly less confidence in them.

The knowledge assumptions are the most common non-falsifiable assumptions that we use in cryptography. They are considered non-standard assumptions. Knowledge assumptions capture our belief that certain computational tasks can be achieved efficiently only by (essentially) going through specific intermediate stages and thereby obtaining, along the way, some specific intermediate values.

A number of different knowledge assumptions exist in the literature, most of which are specific number-theoretic assumptions. Abstracting from such specific assumptions, one can formulate general notions of extractability for one-way functions and other basic primitives (see [CD09]).

4.2 Knowledge Extraction

SNARGs have also been strengthened to become SNARKs *succinct non-interactive arguments of knowledge* [BCCT12, BCC+14]. SNARKs are SNARGs where computational soundness is replaced by *knowledge soundness*. Intuitively speaking, this property says that every prover producing a convincing proof must "know" a witness. On the one hand, knowledge soundness turns out to be useful in many applications, such as delegation of computation where the untrusted worker contributes its own input to the computation, or recursive proof composition [Val08, BCCT13].

On the other hand, the formalization of knowledge soundness in non-interactive protocols is a delicate point since rewinding techniques mentioned in Section 3.1 do not apply anymore. Typically, the concept that the prover "must know" a witness is expressed by assuming that such knowledge can be efficiently extracted from the prover by means of a so-called *knowledge extractor*. In SNARKs, extractors are inherently non-black-box, and the definition of knowledge soundness requires that for every adversarial prover $\mathcal{A}$ generating an accepting proof π there must be an extractor $\mathcal{E}_{\mathcal{A}}$ that, given non-black-box access to $\mathcal{A}$ (e.g., by getting the same input, including the random coins and the code of $\mathcal{A}$), outputs a valid witness.

4.3 SNARK Frameworks

The framework for constructing SNARKs starts with finding a "good" characterization of the complexity class NP and take advantage of its specific properties for applying some compression techniques on top.

Indeed, by choosing a suitable NP-complete problem representation (see Section 2), we are able to construct generic SNARK schemes for NP-complete languages.

For example, many SNARKs have as a departure point the circuit satisfiability (Circ-SAT) problem. Circ-SAT problem is the NP-complete decision problem of determining whether a given circuit has an assignment of its inputs that makes the output true. A very important line of works focuses on building SNARKs for circuit satisfiability [GGPR13, PHGR13, Lip13, DFGK14, Gro16, GMNO18] and have as a central starting point the framework based on *quadratic span programs* introduced by Gennaro et al. in [GGPR13]. This framework will be discussed in details in (see Section 7).

Another very useful characterisation of the NP-complete class are the Probabilistically Checkable Proofs (PCP). Using this characterisation, we can give a framework for constructing SNARKs that was exploited by many works in the field [Mic94, CL08, GLR11, BCCT12, DFH12, BSBHR18].

Recent Constructions. Other more recent arithmetisation techniques lead to algebraic holographic proving systems (AHP) or (Polynomial) Interactive Oracle Proofs (P/IOPs) that are later compiled into SNARKs using polynomial commitments.

In the past few years SNARKs got a lot of attention and many efficient new constructions and implementations have emerged. We mention a list with just a few of them classified by the need for a trusted setup (pre-processing):

- pre-processing SNARKs:
  - Marlin [CHM+19] a verifier-efficient universal proving system based on AHP,
  - Sonic [MBKM19] useful in applications that use batched verifications,
  - Libra [XZZ+19] and Plonk [GWC19] using a permutation argument.
- transparent SNARKs:
  - Aurora [BSCR+18] based on Interactive Oracle Proofs (IOPs)
  - STARK [BSBHR18] and Fractal [COS19] in the Random Oracle Model
  - Spartan [Set19]
  - Halo [BGH19]
  - Hyrax [PPY19]

Other possible clasifications for SNARK frameworks can come from the building blocks used in the construction:

- PCP + Merkle Trees (e.g., CS Proofs [Mic94]), see Section 6.3
- Linear PCPs (e.g., Zaatar [SBV+12], [BCI+13, BCG+13b]), see Section 8
- IOPs/PCIPs (e.g., STARK [BSBHR18], Aurora [BSCR+18])
- MPC-Based (e.g., ZKBoo [GMO16], Ligero [AHIV17])
- Discrete-log Based (e.g., [BCC+16], Bulletproofs [BBB+18], Hyrax [PPY19])

For this gentle introduction, we will give an informal overview of some preliminary constructions and then in Section 7 we will focus on the SNARKs constructions for circuits.

Post-Quantum SNARKs. Almost all the proof systems mentioned so far[1] are based on discrete-log type assumptions, that do not hold against quantum polynomial-time adversaries [Sho99], hence the advent of general-purpose quantum computers would render insecure the constructions based on these assumptions. Efforts were made to design such systems based on quantum resilient assumptions.

Some more desirable assumptions that withstand quantum attacks are the lattice problems [Ajt96, MR04]. Nevertheless, few non-interactive proof systems are built based on these. Some recent works that we can mention are the NIZK constructions for specific languages, like [KW18, LLNW18, BBC+18] and the two designated-verifier SNARG constructions [BISW17, BISW18], designed by Boneh et al. using encryption schemes instantiated with lattices. The first lattice-based designated-verifier zk-SNARK was proposed by [GMNO18] and uses weaker assumptions than the work of Boneh et al. and additionally achieves zero-knowledge and knowledge-soundness.

[1] We note that the original protocol of Micali [Mic94] is a zk-SNARK which can be instantiated with a post-quantum assumption since it requires only a collision-resistant hash function – however (even in the best optimized version recently proposed in [BSBHR18]) the protocol does not seem to scale well for even moderately complex computations.

5 SNARKs: Definitions

In this section we provide formal definitions for the notion of succinct non-interactive arguments (SNARGs) and succinct non-interactive arguments of knowledge (SNARKs).

Let $\mathcal{R} \coloneqq \mathcal{R}_\lambda$ be a efficiently decidable binary relation for an NP language $L_\mathcal{R}$. The asymptotic security notions in this section are all quantified over λ-compatible relations $\mathcal{R}_\lambda$ and they therefore use the simplified notation $\mathcal{R}$ instead of $\mathcal{R}_\lambda$.

For pairs $(u, w) \in \mathcal{R}$ we call u the statement and w the witness.

Definition 5.1 (Argument System)**.** *A non-interactive argument for $\mathcal{R}$ is a quadruple of probabilistic polynomial algorithms $\Pi = (\mathsf{G}, \mathsf{P}, \mathsf{Ver}, \mathsf{Sim})$ such that:*

$(\mathsf{crs}, \mathsf{vrs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, \mathcal{R})$ *the CRS generation algorithm takes as input some security parameter λ, a relation $\mathcal{R}$ and outputs a common reference string* crs *and a trapdoor* td.

$\pi \leftarrow \mathsf{P}(\mathsf{crs}, u, w)$ *the prover algorithm takes as input the* crs*, a statement u, and a witness w. It outputs some argument π.*

$b \leftarrow \mathsf{Ver}(\mathsf{crs}, u, \pi)$ *the verifier algorithm takes as input a statement u together with an argument π, and* crs. *It outputs $b = 1$ (accept) if the proof was accepted, $b = 0$ (reject) otherwise.*

$\pi \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, u)$ *the simulator takes as input a simulation trapdoor* td *and a statement u together with a proof π and returns an argument π.*

5.1 SNARKs Properties

SNARKs are asked to satisfy some security properties that simultaneously protect the prover from the disclosure of the witness, and the verifier from a forged proof. We now state the security notions necessary to define a zk-SNARK:

Definition 5.2 (zk-SNARK)**.** *A non-interactive argument for $\mathcal{R}$ $\Pi = (\mathsf{G}, \mathsf{P}, \mathsf{Ver}, \mathsf{Sim})$ is a zk-SNARK if it satisfies:*

- **Completeness.** Given a true statement for the relation $\mathcal{R}$, a honest prover P with a valid witness should convince the verifier Ver. More formally, for all $\lambda \in \mathbb{N}$ and for all $(u, w) \in \mathcal{R}$:

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, u, \pi) = 1 \,\middle|\, \begin{array}{c} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, \mathcal{R}) \\ \pi \leftarrow \mathsf{P}(\mathsf{crs}, u, w) \end{array}\right] = 1.$$

- **Knowledge Soundness.** The notion of knowledge soundness implies that there is an extractor that can compute a witness whenever the adversary produces a valid argument. The extractor gets full access to the adversary's state, including any random coins. Formally, we require that for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}_\mathcal{A}$ such that

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{crs}, u, \pi) = 1 \\ \wedge\ (u, w) \notin \mathcal{R} \end{array} \,\middle|\, \begin{array}{c} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, \mathcal{R}) \\ ((u, \pi); w) \leftarrow \mathcal{A} \| \mathcal{E}_\mathcal{A}(\mathsf{crs}) \end{array}\right] = \mathsf{negl}.$$

- **Succinctness.** A non-interactive argument where the verifier runs in polynomial time in $\lambda + |u|$ and the proof size is polynomial in λ is called a pre-processing SNARK. If we also restrict the common reference string to be polynomial in λ we say the non-interactive argument is a *fully succinct* SNARK.

- **Statistical Zero-knowledge.** An argument is zero-knowledge if it does not leak any information besides the truth of the statement. Formally, if for all $\lambda \in \mathbb{N}$, for all $(u, w) \in \mathcal{R}$ and for all PPT adversaries $\mathcal{A}$ the following two distributions are statistically close:

$$D_0 = \left\{\ \pi_0 \leftarrow \mathsf{P}(\mathsf{crs}, u, w) : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, \mathcal{R})\ \right\}$$
$$D_1 = \left\{\ \pi_1 \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, u) : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, \mathcal{R})\ \right\}.$$

5.1.1 Public vs. Designated Verifier.

The arguments we defined so far use a public crs for proving and verifying statements. They are commonly known as *publicly verifiable*. To make argument systems that are *designated verifiable*, we change the setup algorithm to also outputs a secret verification state vrs which is needed only for verification. For *designated verifier* SNARKs, it is important to stress that soundness holds against adversaries that do not have access to this verification state vrs. A key question that arises in the design and analysis of designated verifier arguments is whether the same common reference string can be reused for multiple proofs. Formally, this "multi-theorem" setting is captured by requiring soundness to hold even against a prover that makes adaptive queries to a proof verification oracle (note that in this setting verification needs a secret value vrs so the prover cannot run it by itself). If the prover can choose its queries in a way that induces noticeable correlations between the outputs of the verification oracle and the secret verification state, then the adversary can potentially compromise the soundness of the scheme. Thus, special care is needed to construct designated-verifier argument systems in the multi-theorem setting.

5.2 Simulation Extractable zk-SNARK

The classic soundness or knowledge soundness definitions does not capture the fact that it is conceivable for an adversary to see a simulated proof for a false instance and might use some malleability property to modify this proof into another new proof for a different false instance. This scenario is actually very common in security models for cryptographic schemes, so it is often desirable to have some form of non-malleability that prevents cheating in the presence of simulated proofs. This kind of property is called by the so-called *simulation-extractability*. Traditionally, simulation-extractability is defined with respect to a decryption key (trapdoor) associated with the common reference string that allows the extraction of a witness from a valid proof. However, in zk-SNARKs the proofs are too small to encode the full witness. We will therefore instead define simulation-extractable zk-SNARKs using a non-black-box extractor that can deduce the witness from the internal data of the adversary.

Definition 5.3 (Simulation-Extractability)**.** *The notion of knowledge soundness can be extended to simulation-extractable knowledge soundness if for any* PPT *adversaries* $\mathcal{A}$, PPT *extractor* $\mathcal{E}_{\mathcal{A}}$ *such that such that*

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{crs}, u, \pi) = 1 \\ \wedge\ (u, w) \notin \mathcal{R} \\ \wedge\ (u, \pi) \notin Q \end{array} \;\middle|\; \begin{array}{r} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, \mathcal{R}) \\ (u, \pi) \leftarrow \mathcal{A}^{\mathsf{SimProve}}(\mathsf{crs}) \\ w \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{crs}, Q) \end{array}\right] = \mathsf{negl}$$

where the oracle for simulated proofs $\mathsf{SimProve}$ *is parametrized by* $\mathsf{crs}, \mathsf{td}$ *and defined as follows:*

$$\begin{array}{l} \mathsf{SimProve}_{\mathsf{crs},\mathsf{td}}(u_i): \\ \pi_i \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, u_i) \\ Q = Q \cup \{(u_i, \pi_i)\} \\ return\ \pi_i \end{array}$$

Simulation-Soundness. We can also have a corresponding notion of simulation soundness which is implied by white-box and black-box simulation extractability. This definition does not involve an extractor and also applies to SNARGs.

Definition 5.4 (Simulation-Soundness)**.** *We say that a SNARG/SNARK is* weakly simulation-sound *if for any PPT adversary* $\mathcal{A}$ *and* $\mathcal{R}$ *we have*

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{crs}, u, \pi) = 1 \\ \wedge u \notin \mathcal{L}_{\mathcal{R}}\ \wedge (u, \pi) \notin Q \end{array} \;\middle|\; \begin{array}{r} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, \mathcal{R}) \\ (u, \pi) \leftarrow \mathcal{A}^{\mathsf{SimProve}}(\mathsf{crs}) \end{array}\right] = \mathsf{negl}$$

where $\mathsf{SimProve} = \mathsf{SimProve}_{\mathsf{crs},\mathsf{td}}(u)$ *is a simulator oracle that calls* $\mathsf{Sim}(\mathsf{crs}, \mathsf{td}, u)$ *internally, and also records* (u, π) *into* Q*, and* $\mathcal{L}_{\mathcal{R}}$ *is the language corresponding to* $\mathcal{R}$*.*

Weaker Notions Allowing for Randomization. A weaker notion of both Simulation-Soundness and Simulation-Extractability can be defined if we change the last winning condition in the definitions above: $\mathsf{SimProve}_{\mathsf{crs},\mathsf{td}}(u)$ oracle only records queried statements u into Q, and the winning condition checks that $u \notin Q$.

This *weak* SE was first defined in [KZM+15] and allows a prover to re-randomize proofs for statements already queried to the simulation oracle.

Strong SE/SS requires $(u, \pi) \notin Q$, where SimProve records pairs of queried instances and simulated proofs. If the SNARK is randomizable, $\mathcal{A}$ can just pass re-randomized simulated proof for an instance it does not know a witness of and win the strong SE game. This is forbidden, thus the strong SE scheme must be non-malleable. Honest proofs are also non-randomizable, otherwise zero-knowledge would not hold. Weak SE relaxes this non-malleability requirement by allowing to produce $\pi' \neq \pi$ for the simulated (and thus also real) proof π.

5.3 SNARKs with Auxiliary Input and O-SNARKs.

The classical notion of knowledge soundness for SNARKs does not suffice in some applications where the security model considers adversaries with access to many functionalities in a larger protocol containing also SNARKs. These adversaries may see extra information than just the crs of the SNARK scheme, due to their interactions with other primitives involved in the protocol.

Classically, in SNARKs, we consider the extractor as a non-black-box algorithm that takes the same input as the prover, so if the prover disposes of some auxiliary information, then the extractor should include this in their inputs as well.

SNARKs with Auxiliary Input. We recall here the more general definition of SNARKs knowledge soundness property, that is stated for some auxiliary input generated using a "benign" relation generator.

- **Knowledge Soundness with Auxiliary Input.** For all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}_\mathcal{A}$ such that for every auxiliary input $\mathsf{z} \in \{0,1\}^{poly(\lambda)}$

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{vrs}, u, \pi) = 1 \\ \wedge \\ (u, w) \notin \mathcal{R} \end{array} \middle| \begin{array}{r} (\mathsf{crs}, \mathsf{vrs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, \mathcal{R}) \\ \mathsf{z} \leftarrow \mathcal{Z}(1^\lambda) \\ ((u, \pi); w) \leftarrow \mathcal{A} \| \mathcal{E}_\mathcal{A}(\mathsf{crs}, \mathsf{z}) \end{array}\right] = \mathsf{negl}$$

About Extraction and Auxiliary Input. First, we stress that in the knowledge soundness property the extractor $\mathcal{E}_\mathcal{A}$ takes exactly the same input of $\mathcal{A}$, including its random tape. Second, the knowledge soundness definition can also be relaxed to hold with respect to auxiliary inputs from specific distributions (instead of arbitrary ones). Namely, let $\mathcal{Z}$ be a probabilistic algorithm (called the auxiliary input generator) that outputs a string z, then we say that adaptive knowledge soundness holds for $\mathcal{Z}$ if the above definition holds for auxiliary inputs sampled according to $\mathcal{Z}$, where $\mathcal{Z}$ is also a non-uniform polynomial-size algorithm.

This type of definition is certainly more useful for using SNARKs in larger cryptographic protocols, but it also introduces other subtleties. As first discussed in [HT98], extraction in the presence of arbitrary auxiliary input can be problematic, if not implausible. Bitansky et al. [BCPR14] show that, when assuming indistinguishability obfuscation, there do not exist extractable one-way functions (and thus SNARKs) with respect to arbitrary auxiliary input of unbounded polynomial length. Boyle and Pass [BP15] generalize this result showing that assuming collision-resistant hash functions and differing-input obfuscation, there is a fixed auxiliary input distribution for which extractable one-way functions do not exist.

5.3.1 SNARKs in the Presence of Oracles.

The notion of SNARK does not suffice in applications that consider a security model where adversaries who have access to oracles with a secret state, and during a security reduction one needs to run an extractor where the secret state of the oracle is not available.

Classically, in SNARKs, the extractor is considered to be a non-black-box algorithm that takes the same input as the prover and any auxiliary information that the prover also had when computing the proof.

On the other hand, extraction is not defined in a scenario in which adversarial provers are given *black-box access*, or what we call access to an *oracle*.

To address this limitation of classical SNARKs, Fiore and Nitulescu in [FN16] introduced the notion of O-SNARKs, that

In a nutshell, an O-SNARK is like a SNARK except that knowledge soundness must hold with respect to adversaries that have access to an oracle O sampled from some oracle family $\mathbb{O}$. More formally:

Definition 5.5 (O-SNARK [FN16])**.** *Adaptive knowledge soundness holds with respect to some oracle family* $\mathbb{O} = \{\mathsf{O}\}$ *if for all auxiliary information* $\mathsf{z} \leftarrow_{\$} \mathcal{Z}$ *in a given distribution, for every PPT adversary* $\mathcal{A}^{\mathsf{O}}$ *there exists a PPT extractor* $\mathcal{E}_{\mathcal{A}}$ *such that*

$$\Pr\left[\begin{array}{c|r} \mathsf{Ver}(\mathsf{crs}, u, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\lambda, \mathcal{R}) \\ \wedge & \mathsf{O} \leftarrow_{\$} \mathbb{O}, (u, \pi) \leftarrow \mathcal{A}^{\mathsf{O}}(\mathsf{crs}, \mathsf{z}) \\ (u, w) \notin \mathcal{R} & w \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{crs}, \mathsf{z}, \mathsf{qt}) \end{array}\right] = \mathsf{negl}$$

where $\mathsf{qt} = \{q_i, \mathsf{O}(q_i)\}$ *is the transcript of all oracle queries and answers made and received by* $\mathcal{A}^{\mathsf{O}}$ *during its execution.*

Candidate Constructions of O-SNARKs. The results in [FN16] relate the security of O-SNARKs to the classical SNARK security in the (better studied) case of extraction with auxiliary input, thus avoiding making further conjectures on the plausibility of extractability assumptions.

In the case of signing oracles for example, Fiore and Nitulescu [FN16] show that there are a few candidate O-SNARKs.

Computationally-sound proofs of Micali [Mic94] yield O-SNARKs for every oracle family, in the random oracle model without further restrictions.

O-SNARKs also exist for specific classes of signature oracles: If we require the underlying signatures to be hash-and-sign signatures and model the hash as a random oracle, then any classical SNARK is an O-SNARK for that oracle. In the standard model, if we require the message space of the signature scheme to be properly bounded and require the adversary to query almost the entire message space, or we require the adversary to issue oracle queries before seeing the common reference string, then all known SNARKs can be used as O-SNARKs.

6 SNARKs: Construction from PCP

In this section, we provide some high-level intuition for some notable SNARK construction in the literature, introduced by the work "The hunting of the SNARK" [BCC+14].

This methodology to construct SNARKs is based on PCP characterization of NP, and it is first achieved in the random oracle model (ROM), which gave only heuristical security. The idea is to apply the random-oracle-based Fiat-Shamir transform to Kilian's succinct PCP-based proof system [Kil92], achieving logarithmic proof size and verification time.

Later, the construction is improved by removing the use of the random oracles and replacing them with extractable collision-resistant hash functions (ECRH).

We first informally introduce Probabilistically Checkable Proofs (PCP), a characterisation of the NP-class.

6.1 Probabilistically Checkable Proofs

The original version of the PCP Theorem [ALM+98] states that proofs for any NP language can be encoded in such a way that their validity can be verified by only reading a constant number of bits, and with an error probability that is upper bounded by a constant. The class PCP is a generalization of the proof verifying system used to define NP, with the following changes:

Probabilistic Verifier. The verifier is probabilistic instead of deterministic. Hence, the verifier can have different outputs for the same inputs x.

Random Access to the Proof. The verifier has random access to the proof string π. This means each bit in the proof string can be independently queried by the verifier via a special address tape: If the verifier desires say the i-th bit in the proof of the string, it writes i in base-2 on the address tape and then receives the bit π_i.

Constant Number of Queries. We are interested in probabilistic verification procedures that access only a few locations in the proof [ALM+98], and yet are able to make a meaningful probabilistic verdict regarding the validity of the alleged proof. Specifically, the verification procedure should accept any valid proof (with probability 1) but rejects with probability at least $1/2$ any alleged proof for a false assertion.

Adaptiveness. Verifiers can be adaptive or non-adaptive. A non-adaptive verifier selects its queries based only on its inputs and random tape, whereas an adaptive verifier can, in addition, rely upon bits it has already queried in π to select its next queries.

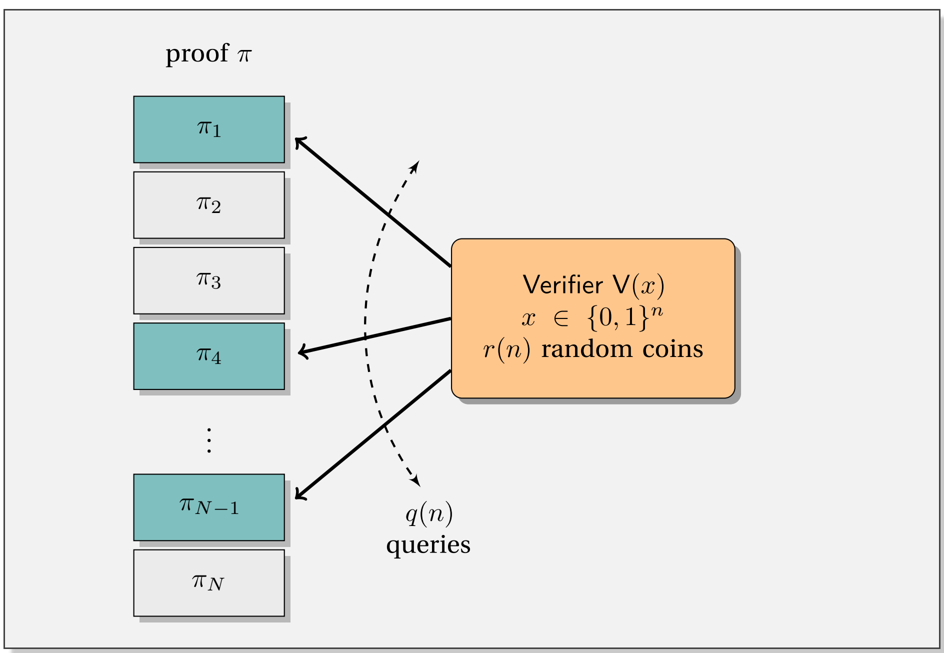


Figure 2: A PCP verifier V for a language $\mathcal{L}$ with input x and random access to a string π.

The fact that one can (meaningfully) evaluate the correctness of proofs by examining few locations in them is indeed surprizing and somewhat counter-intuitive. Needless to say, such proofs must be written in a somewhat non-standard format, because standard proofs cannot be verified without reading them in full (since a flaw may be due to a single improper inference). In contrast, proofs for a PCP system tend to be very redundant; they consist of superfluously many pieces of information (about the claimed assertion), but their correctness can be (meaningfully) evaluated by checking the consistency of a randomly chosen collection of few related pieces. NP-proofs can be efficiently transformed into a (redundant) form that offers a trade-off between the number of locations (randomly) examined in the resulting proof and the confidence in its validity. A more formal definition follows:

Definition 6.1 (Probabilistically Checkable Proofs)**.** *Let $\mathcal{L}$ be a language and $q, r : \mathbb{N} \to \mathbb{N}$. A probabilistically checkable proof system* $\mathsf{PCP}(r(n), q(n))$ *for $\mathcal{L}$ is a probabilistic polynomial-time oracle machine, called verifier and denoted V, that satisfies the following conditions:*

Efficiency. On input a string $x \in \{0,1\}^n$, and given a random access to a string π called the proof, V uses at most $r(n)$ random coins and makes at most $q(n)$ queries to locations of π (see Figure 2). Then it outputs 1 (for "accept") or 0 (for "reject").

Completeness. For every $x \in \mathcal{L}$ there exists a proof string π such that, on input x and access to oracle π, machine V always accepts x.

Soundness. For every $x \notin \mathcal{L}$ and every proof string π, on input x and access to oracle π, machine V rejects x with probability at least $1/2$.

The error probability (in the soundness condition) of PCP systems can be reduced by successive applications of the proof system. In particular, repeating the process for k times, reduces the probability that the verifier is fooled by a false assertion to 2^{-k}, whereas all complexities increase by at most a factor of k. Thus, PCP systems of non-trivial query-complexity provide a trade-off between the number of locations examined in the proof and the confidence in the validity of the assertion.

We say that a language $\mathcal{L}$ is in $\mathsf{PCP}(r(n), q(n))$ if $\mathcal{L}$ has a $(cr(n), dq(n))$-verifier V for some constants c, d.

Theorem 6.2 (PCP Theorem [ALM+98])**.**

$$\mathsf{NP} = \mathsf{PCP}(\log n, 1)$$

6.2 Merkle Trees and Hash Functions

The concept of hash (binary) trees is named after Ralph Merkle who patented it in 1979 [Mer79]. In a Merkle tree every leaf node is labelled with the hash of a data block, and every non-leaf node is labelled with the hash of the labels of its child nodes.

The structure of the tree allows for efficient mapping of arbitrarily large amounts of data and enables easy identification of where changes in that data occur. This concept enables Merkle proofs, with which, someone can verify that the hashing of data is consistent all the way up the tree and in the correct position without having to actually look at the entire set of hashes. Instead, demonstrating that a leaf node is a part of a given binary hash tree requires computing a number of hashes proportional to the logarithm of the number of leaf nodes of the tree; this contrasts with hash lists, where the number is proportional to the number of leaf nodes itself.

Collision-Resistant Hash Functions. A collision-resistant hash function (CRHF) is a function ensemble for which it is hard to find two inputs that map to the same output. Formally:

Definition 6.3 (Collision Resistant Hash Family)**.** *A collection of function families $\mathbb{H} = \{\mathcal{H}\}_\lambda$ where each $\mathcal{H}$ is a function family $\mathcal{H} = \{h : \{0,1\}^{q(\lambda)} \to \{0,1\}^{\ell(\lambda)}\}$ is collision-resistant if:*

Efficient. The functions $q(\lambda)$ and $\ell(\lambda)$ are polynomially-bounded; furthermore, given λ and $x \in \{0,1\}^{q(\lambda)}$ the value $h(x)$ can be computed in $\mathsf{poly}(\lambda)$ time.

Compressing. For all λ we have that $q(\lambda) > \ell(\lambda)$.

Collision resistant. For all PPT algorithms $\mathcal{A}$, the following probability is negligible (in λ):

$$\Pr[h \leftarrow_{\$} \mathcal{H}, (x, x') \leftarrow \mathcal{A}(1^\lambda, h) : x, x' \in \{0,1\}^{q(\lambda)} \wedge x \neq x' \wedge h(x) = h(x')] = \mathsf{negl}.$$

6.3 Kilian Interactive Argument of Knowledge from PCP

When PCP theorem [ALM+98] came out, it revolutionized the notion of "proof" – making the verification possible in time polylogarithmic in the size of a classical proof. Kilian adapted this PCP characterization

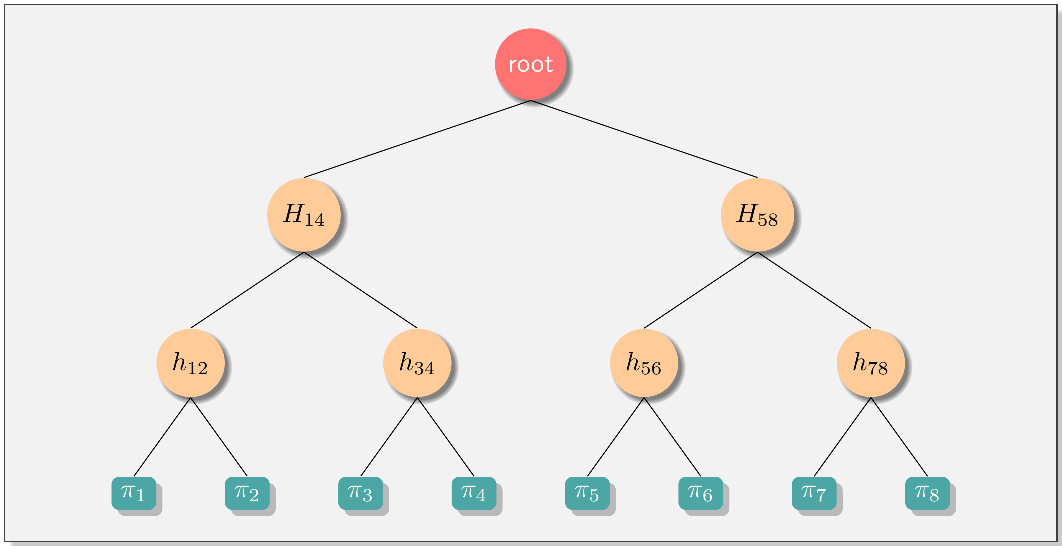


Figure 3: A Merkle tree commiting to the string π. Each inner node of the tree is the hash value of the concatenation of its two children: $h_{ij} = h(\pi_i \| \pi_j), H_{ik} = h(h_{ij} \| h_{j+1k}), \mathsf{root} = h(H_{14} \| H_{58}), i, j, k \in [8]$.

of NP to the cryptographic setting, showing that one can use PCPs to construct interactive arguments (i.e., computationally sound proof systems [BCC88]) for NP that are succinct – i.e., polylogarithmic also in their communication complexity. In his work [Kil92], Kilian presents a succinct zero-knowledge argument for NP where the prover P uses a Merkle tree (see Section 6.2) in order to provide to the verifier V "virtual access" to a PCP proof π.

Merkle tree hashing enables the prover P to use a collision resistant hash function (CRHF) H to compute a succinct commitment to the long string $\pi \in \{0,1\}^{q(\lambda)}$ and later to locally open to any bit of π (in a succinct manner). An example of such a Merkle tree is illustrated in Figure 3.

More precisely, to prove a statement $x \in \mathcal{L}$ we need the following interactions:

1. The verifier starts by sending the prover a CRHF H (in the sense of Definition 6.3).
   The prover, on private input a witness w, constructs a PCP-proof π.
   In order to yield efficient verifiability, P cannot send to V the witness w, nor π.
   Instead, it builds a Merkle tree with the proof π as the leaf values (using the CRHF H from the verifier) producing a root value.
2. The prover sends this root to V as a commitment to π.
3. V tosses a fixed polynomial number of random coins and sends them to P.
   Both the prover P and the verifier V compute the PCP queries by internally running the PCP verifier on input x and root.
4. The prover P sends back answers to those queries, together with "proofs"– called authentication paths – that each answer is consistent with the root of the Merkle tree.
   Finally, the verifier accepts if all the answers are consistent with the root value, and convinces the PCP.

Kilian's protocol is succinct, because the verifier V, invoking the PCP verifier, makes only a fixed polynomial number of queries and each query is answered with an authentication path of some fixed polynomial length, all independent of the length of the witness.

At a very high-level, the soundness follows from the fact that the Merkle tree provides the verifier "virtual access" to the PCP proof, in the sense that given the root value of the Merkle tree, for every query q, it is infeasible for a cheating prover to answer q differently depending on the queries. Therefore, interacting with the prover is "equivalent" to having access to a PCP proof oracle. Then it follows from the soundness of the PCP system that Kilian's protocol is sound.

6.4 Micali's CS Proofs

Micali [Mic94] showed how to make interactive arguments non-interactive by applying the Fiat-Shamir heuristic (see Section 3) to Kilian's construction. The idea was to apply a hash function, modeled as a random oracle, to its PCP string both as a form of commitment and to non-interactively generate the verifier's PCP queries.

Private Information Retrieval (PIR). The work of [CL08] proposed the PCP+MT+PIR approach to "squash" Kilian's four-message protocol into a two-message protocol. To understand their techniques, we briefly define Private Information Retrieval (PIR) schemes.

A (single-server) polylogarithmic private information retrieval (PIR) scheme ([CMS99, Lip05, GR05, BV11]) allows a user to retrieve an item from a server in possession of a database without revealing which item is retrieved. One trivial, but very inefficient way to achieve PIR is for the server to send an entire copy of the database to the user. There are two ways to address this problem: one is to make the server computationally bounded, and the other is to assume that there are multiple non-cooperating servers, each having a copy of the database. We will consider the first one that assumes bounded running times and succinctness of the server answers. More formally:

Definition 6.4 (Private Information Retrieval)**.** *A (single-server) polylogarithmic* private information retrieval *(PIR) scheme consists of a triple of algorithms* $(\mathsf{PEnc}, \mathsf{PEval}, \mathsf{PDec})$ *that work as follow:*

$\mathsf{PEnc}(1^\lambda, i, r)$**:** outputs an encryption C_i of query i to a database DB using randomness r,
$\mathsf{PEval}(DB, C_i)$**:** outputs a succinct blob e_i "containing" the answer $DB[i]$,
$\mathsf{PDec}(e_i)$**:** decrypts the blob e_i to an answer $DB[i]$.

The three proprieties a PIR scheme should satisfy are corectness, succinctness (the running time of both $\mathsf{PEnc}, \mathsf{PEval}$ should be bounded) and semantic security, in the sense that the encryptions of indexes i with the PEnc algorithm should not reveal information about their value.

The PCP+MT+PIR Approach. We have seen that in Kilian's protocol, the verifier obtains from the prover a Merkle hash to a PCP oracle and only then asks the prover to locally open the queries requested by the PCP verifier. In [CL08]'s construction, the verifier also sends in the first message, a PIR-encrypted version of the PCP queries (the first message of a PIR scheme can be viewed as an encryption to the queries); the prover then prepares the required PCP oracle, computes and sends a Merkle hash of it, and answers the verifier's queries by replying to the PIR queries according to a database that contains the answer (as well as the authentication path with respect to the Merkle hash) to every possible verifier's query. In [CL08] the soundness of the above scheme is based on the assumption that any convincing prover P must essentially behave as an honest prover: Namely, if a proof is accepting, then the prover must have in mind a full PCP oracle, which maps under the Merkle hash procedure to the claimed root, and such a proof π can be obtained by an efficient extractor $\mathcal{E}$.

They then showed that, if this is the case, the extracted string π must be consistent with the answers the prover provides to the PCP queries, for otherwise the extractor can be used to obtain collisions of the hash function underlying the Merkle tree. Therefore, the extracted string π also passes the PCP test, where the queries are encrypted under PIR. Then, it follows from the privacy of the PIR scheme that, the string π is "computationally independent" of the query. Hence from the soundness of PCP, they conclude that the statement must be true.

6.5 SNARKs from PCP

We have mentioned two methodologies that can be applied to obtain SNARGs from PCPs, one is the Fiat-Shamir heuristic in the random oracle model, the other is the PCP+MT+PIR Approach. In both cases, we do not obtain knowledge soundness, but only plain adaptive soundness. Recent works [GLR11, BCCT12, DFH12, BCC+14] have improved Micali's construction by adding knowledge soundness and removing the random oracles, replacing them with "extractable collision-resistant hash functions" (ECRHs), a non-falsifiable extractability assumption.

Extractable Collision-Resistant Hash. We start by defining a natural strengthening of collision-resistant hash functions introduced in Definition 6.3: the extractable collision-resistant hash functions (ECRH). An ECRH function family satisfies the two following properties:

- it is collision-resistant in the standard sense of Definition 6.3,
- it is extractable in the sense that for any efficient adversary that is able to produce a valid evaluation of the function there is an extractor that is able to produce a corresponding preimage.

The ECRH+PIR Approach. The [BCC+14] construction obtains the stronger notion of knowledge soundness arguments, SNARKs, and also a pre-processed protocol rather than one-round of communication, based on the more restrictive assumption that ECRHs exist. At a very high-level, their construction modifies the PCP+MT+PIR approach by replacing the CRHF underlying the Merkle tree with an ECRH. The additional features of this modified construction are:

(a) The verifier's message can be generated offline independently of the theorem being proved and thus we refer to this message as a verifier-generated reference string (VGRS);

(b) The input can be chosen adaptively by the prover based on previous information, including the VGRS;

(c) The construction is an (adaptive) argument of knowledge;

(d) The running time of the verifier and the proof length are "universally succinct"; in particular, they do not depend on the specific NP-relation at hand.

On the other hand, the scheme is only designated-verifier.

The main challenges in [BCC+14] construction and the required modifications they make to [CL08] are briefly mentioned in the following.

Extracting a witness. To obtain knowledge soundness, they first instantiate the underlying PCP system with PCPs of knowledge, which allow for extracting a witness from any sufficiently-satisfying proof oracle.

Adaptivity. In their setting, the prover is allowed to choose the claimed theorem after seeing the verifier's first message (or, rather, the verifier-generated reference string). In order to enable the (honest) verifier to do this, they PIR-encrypt the PCP verifier's coins rather than its actual queries (as the former are independent of the instance), and require the prover to prepare an appropriate database (containing all the possible answers for each setting of the coins, rather than a proof oracle).

From local to global extraction. Unlike [CL08], which directly assumed the "global extraction" guarantees from a Merkle tree, Bitansky et al. show that the "local extraction" guarantee can be lifted using ECRH functions instead of simple CRHF, to the "global extraction" guarantee on the entire Merkle tree. The main technical challenge in their construction is establishing this "global" knowledge feature, more precisely, to obtain an extracted PCP proof π that will be sufficiently satisfying for extracting a witness from a very "local" one (namely, the fact that it is infeasible to produce images of the ECRH without actually knowing a preimage).

To achieve this, they start from the root of the Merkle tree and "work back towards the leaves"; that is, extract a candidate proof π by recursively applying the ECRH-extractor to extract the entire Merkle tree, where the leaves should correspond to π. However, recursively composing ECRH-extractors already encounters a difficulty: each level of extraction incurs a polynomial blowup in computation size. Hence, without making a very strong assumption on the amount of "blowup" incurred by the extractor, one can only apply extraction a constant number of times. This problem is solved by replacing the binary Merkle tree with a *squashed* Merkle tree, where the fan-in of each node is polynomial rather than binary as is usually the case.

Knowledge Soundness. Given the previous discussion, knowledge soundness of the entire scheme is shown in two steps:

Local Consistency. They show that whenever the verifier is convinced, the recursively extracted string contains valid answers to the verifier's PCP queries specified in its PIR queries. Otherwise, it is possible to find collisions within the ECRH as follows. A collision finder could simulate the PIR-encryption on its own, invoke both the extraction procedure and the prover, and obtain two paths that map to the same root but must differ somewhere (as one is satisfying and the other is not) and therefore obtain a collision.

From Local to Global Consistency. Next, using the privacy guarantees of the PIR scheme, they show that, whenever one extracts a set of leaves that are satisfying with respect to the PIR-encrypted queries, the same set of leaves must also be satisfying for almost all other possible PCP queries and is thus sufficient for witness-extraction. Indeed, if this was not the case then one would be able to use the polynomial-size extraction circuit to break the semantic security of the PIR.

Furthermore, the [BCC+14] construction achieves a communication complexity and a verifier's time complexity bounded by a polynomial in the security parameter, the size of the instance, and the logarithm of the time it takes to verify a valid witness for the instance, obtaining a fully-succinct SNARK.

7 SNARKs: Construction from QAP

We will present here the methodology for building SNARKs common to a family of constructions, some of which represent the state of the art in the field.

Most constructions and implementations of SNARKs [PHGR13, Lip13, DFGK14, Gro16, GMNO18] have as a central starting point the framework based on quadratic programs introduced by Gennaro et al. in [GGPR13]. This common framework allows to build SNARKs for programs instantiated as boolean or arithmetic circuits.

This approach has led to fast progress towards practical verifiable computations. For instance, using span programs for arithmetic circuits (QAPs), Pinocchio [PHGR13] provides evidence that verified remote computation can be faster than local computation. At the same time, their construction is zero-knowledge, enabling the server to keep intermediate and additional values used in the computation private.

Optimized versions of SNARKs based on QAP approach are used in various practical applications, including cryptocurrencies such as Zcash [BCG+14], to guarantee anonymity via the ZK property while preventing double-spending.

7.1 Circuits and Circ-SAT Problem

A SNARK scheme for a circuit has to enable verification of proofs for (Arithmetic or Boolean) Circ-SAT problem, i.e., a prover, given a circuit has to convince the verifier that it knows an assignment of its inputs that makes the output true. In the following definitions, we may see a circuit C as a logical specification of a satisfiability problem.

Arithmetic Circuits. Informally, an arithmetic circuit consists of wires that carry values from a field $\mathbb{F}$ and connect to addition and multiplication gates. See Figure 4 for an example.

Boolean Circuits. A boolean circuit consists of logical *gates* and of a set of *wires* between the gates. The wires carry values over $\{0, 1\}$. See Figure 5 for an example.

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

Definition 7.1 (Circuit Satisfaction Circ-SAT)**.** *The circuit satisfaction problem of a circuit* $C : I_u \times I_w \to \{0,1\}$ *is defined by the relation* $\mathcal{R}_C = \{(\mathbf{u}, \mathbf{w}) \in I_u \times I_w : C(\mathbf{u}, \mathbf{w}) = 1\}$ *and its language is* $\mathcal{L}_C = \{\mathbf{u} \in I_u : \exists\, \mathbf{w} \in I_w, C(\mathbf{u}, \mathbf{w}) = 1\}$.

Standard results show that polynomially sized circuits are equivalent (up to a logarithmic factor) to Turing machines that run in polynomial time, though of course the actual efficiency of computing via circuits versus on native hardware depends heavily on the application; for example, an arithmetic circuit for matrix multiplication adds essentially no overhead, whereas a boolean circuit for integer multiplication is far less efficient.

7.2 From Circuits to Efficient NP Characterization

Back in 2013, Gennaro, Gentry, Parno and Raykova [GGPR13] proposed a new, influential characterization of the complexity class NP using *Quadratic Span Programs* (QSPs), a natural extension of span programs defined by Karchmer and Wigderson [KW93].

Some variants and improvements of QSPs followed. In [Lip13], Lipmaa gave a class of more efficient quadratic span programs by combining the existing techniques with linear error-correcting codes.

Parno et al. [PHGR13] defined QAP, a similar notion for arithmetic circuits, namely *Quadratic Arithmetic Programs*. More recently, an improved version for boolean circuits, the *Square Span Programs* (SSP) was presented by [DFGK14]. Naturally, this led to a simplified version for arithmetic circuits in the same spirit, *Square Arithmetic Programs* (SAP), proposed in [GM17].

These are methods to compactly encode computations, so as to obtain efficient zero-knowledge SNARKs. The main idea is to represent each gate inputs and outputs as a variable. Then we may rewrite each gate as an equation in some variables representing the gate's input and output wires. These equations are satisfied only by the values of the wires that meet the gate's logic or arithmetic specification. By composing such constraints for all the gates in the circuit, a satisfying assignment for any circuit can be specified first as a set of quadratic equations, then as a constraint on the span of a set of polynomials, defining the corresponding *Quadratic/Square Span Program* for the circuit. As a consequence, the prover needs to convince the verifier that all the quadratic equations are satisfiable by finding a solution of the equivalent polynomial problem.

7.3 Quadratic Arithmetic Programs (QAPs).

Before formally defining QAPs, we walk through the steps for encoding the toy example circuit in Figure 4 into an equivalent QAP.

First, we select two arbitrary values from some field $\mathbb{F}$ of order p: $r_5, r_6 \in \mathbb{F}$ to represent the two multiplication gates (the addition gates will be compressed into their contributions to the multiplication gates).

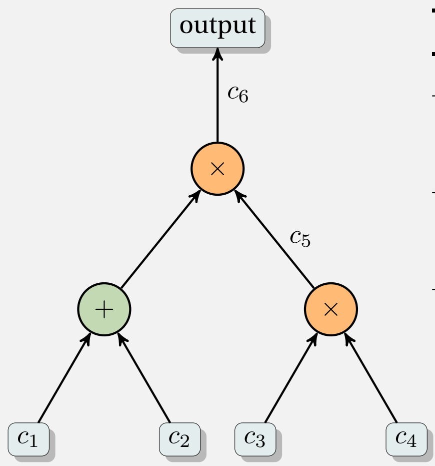
We define three sets of polynomials $\mathcal{V} = \{v_i(x)\},\ \mathcal{W} = \{w_i(x)\}$ and $\mathcal{Y} = \{y_i(x)\},\ i \in [6]$ by letting the polynomials in $\mathcal{V}$ encode the left input into each multiplication gate, the $\mathcal{W}$ encode the right input into each gate, and the $\mathcal{Y}$ encode the outputs. Thus, for the circuit in Figure 4, we define six polynomials for each set $\mathcal{V},\ \mathcal{W}$ and $\mathcal{Y}$, four for the input wires, and two for the outputs from the multiplication gates.

We define these polynomials based on each wire's contributions to the multiplication gates. Specifically all of the $v_i(r_5) = 0$, except $v_3(r_5) = 1$, since the third input wire contributes to the left input of c_5's multiplication gate. Similarly, $v_i(r_6) = 0$, except for $v_1(r_6) = v_2(r_6) = 1$, since the first two inputs both contribute to the left input of c_6's gate. For $\mathcal{W}$, we look at right inputs. Finally, $\mathcal{Y}$ represents outputs; none of the input wires is an output, so $y_i(r_5) = y_i(r_6) = 0$ for $i \in [4]$ and $y_5(r_5) = y_6(r_6) = 1$. We can use this encoding of the circuit to efficiently check that it was evaluated correctly.

More generally, we define a QAP, an encoding of an arithmetic function, as follows.

Definition 7.2 (QAP). *A Quadratic Arithmetic Program Q over the field $\mathbb{F}$ contains three sets of $m+1$ polynomials $\mathcal{V} = \{v_i(x)\},\ \mathcal{W} = \{w_i(x)\}$ and $\mathcal{Y} = \{y_i(x)\},\ i \in \{0, 1 \dots m\}$ and a target polynomial $t(x)$. Suppose F is an arithmetic function that takes as input n elements of $\mathbb{F}$ and outputs n' elements, for a total of $N = n + n'$ I/O elements. Then, $(c_1, \dots, c_N) \in \mathbb{F}^N$ is a valid assignment of F's inputs and outputs, if and only if there exist coefficients $(c_{N+1}, ..., c_m)$ such that $t(x)$ divides $p(x)$, where:*

$$p(x) \coloneqq \left(v_0(x) + \sum_{i=1}^{m} c_i v_i(x)\right) \cdot \left(w_0(x) + \sum_{i=1}^{m} c_i w_i(x)\right) - \left(y_0(x) + \sum_{i=1}^{m} c_i y_i(x)\right). \qquad (1)$$



Roots	**Polynomials in QAP** $(\mathcal{V}, \mathcal{W}, \mathcal{Y}, t(x))$		
Gates	Left inputs	Right inputs	Outputs
r_5	$v_3(r_5) = 1$ $v_i(r_5) = 0,$ $i \neq 3$	$w_4(r_5) = 1$ $w_i(r_5) = 0,$ $i \neq 4$	$y_5(r_5) = 1$ $y_i(r_5) = 0,$ $i \neq 5$
r_6	$v_1(r_6) = v_2(r_6) = 1$ $v_i(r_6) = 0,$ $i \neq 1, 2$	$w_5(r_6) = 1$ $w_i(r_6) = 0,$ $i \neq 5$	$y_6(r_6) = 1$ $y_i(r_6) = 0,$ $i \neq 6$
$t(x) = (x - r_5)(x - r_6)$			

Figure 4: Arithmetic Circuit and Equivalent QAP. The polynomials $\mathcal{V} = \{v_i(x)\}, \mathcal{W} = \{w_i(x)\}, \mathcal{Y} = \{y_i(x)\}$ and the target polynomial $t(x)$ are defined in terms of their evaluations at two random values $r_5, r_6 \in \mathbb{F}$, one for each multiplicative gate.

In other words, there must exist some polynomial $h(x)$ such that $h(x)t(x) = p(x)$. We say that the QAP Q computes F. The size of Q is m, and the degree d is the degree of $t(x)$.

In [PHGR13], the authors show that for any arithmetic circuit with d multiplication gates and N I/O elements, one can construct an equivalent QAP with degree (the number of roots) d and size (number of polynomials in each set) $m = d + N$. Note that addition gates and multiplication-by-constant gates do not contribute to the size or degree of the QAP. Thus, these gates are essentially "free" in QAP-based SNARKS.

Building a QAP Q for a general arithmetic circuit $\mathtt{C}$ is fairly straightforward:

We pick an arbitrary root $r_g \in \mathbb{F}$ for each multiplication gate g in $\mathtt{C}$ and define the target polynomial to be $t(x) = \prod_g (x - r_g)$.

We associate an index $i \in [m]$ to each input of the circuit and to each output from a multiplication gate.

Finally, we define the polynomials in $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{Y}$ by letting the polynomials in $\mathcal{V}$, $\mathcal{W}$ encode the left/right input into each gate, and $\mathcal{Y}$ encode the outputs of the gates: $v_i(r_g) = 1$ if the i-th wire is a left input to gate g, and $v_i(r_g) = 0$ otherwise. Similarly, we define the values of polynomials $w_i(r_g)$ and $y_i(r_g)$.

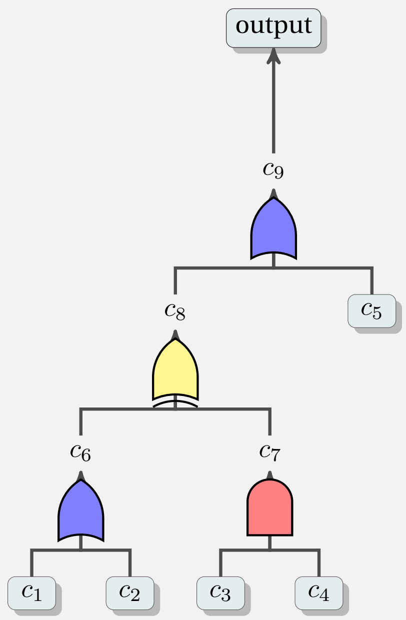
Thus, if we consider a particular gate g and its root r_g, Equation (1) and the constraint $p(r_g) = t(r_g)h(r_g) = 0$ just says that the output value of the gate is equal to the product of its inputs, the very definition of a multiplication gate.

For example, in the QAP for the circuit in Figure 4, if we evaluate $p(x)$ at r_5, we get $c_3c_4 = c_5$, which directly encodes the first multiplication gate, and similarly, at r_6, $p(x)$ simplifies to $(c_1 + c_2)c_5 = c_6$, that is, an encoding of the second multiplication gate.

In short, the divisibility check that $t(x)$ divides $p(x)$ decomposes into $d = \deg(t(x))$ separate checks, one for each gate g and root r_g of $t(x)$, that $p(r_g) = 0$.

7.4 Square Span Programs (SSPs)

Danezis et al. [DFGK14] found a way to *linearize* all logic gates with fan-in 2 in a boolean circuit. This starts from the observation that any 2-input binary gate $g(a, b) = c$ with input wires a, b and output c can be specified using an affine combination $L = \alpha a + \beta b + \gamma c + \delta$ of the gate's input and output wires that take exactly two values, $L = 0$ or $L = 2$, when the wires meet the gate's logical specification. This leads to



Linearization of logic gates

OR ($c_1 \vee c_2 = c_6$)	AND ($c_3 \vee c_4 = c_7$)
c_1 c_2 c_6	c_3 c_4 c_7
0 0 0	0 0 0
0 1 1	0 1 0
1 0 1	1 0 0
1 1 1	1 1 1
$-c_1 - c_2 + 2c_6 \in \{0,1\}$	$c_3 + c_4 - 2c_7 \in \{0,1\}$

XOR gate, input and output bits

XOR ($c_6 \oplus c_7 = c_8$)	IN ($\{c_i\}_{i=1}^{8}$)	OUT (c_9)
c_6 c_7 c_8	c_i	c_9
0 0 0		
0 1 1	$\in \{0,1\}$	1
1 0 1		
1 1 0		
$c_6 + c_7 + c_8 \in \{0,2\}$	$2c_i \in \{0,2\}$	$3 - 3c_9 \in \{0,2\}$

Figure 5: Boolean circuit and the linearization of its logic gates.

an equivalent single ”square” constraint $(L-1)^2 = 1$. We refer to Figure 5 for the truth table and simple linearization of some gates in a toy example.

Composing such constraints, a satisfying assignment for any binary circuit can be specified first as a set of affine map constraints, then as a constraint on the span of a set of polynomials, defining the square span program for this circuit.

Due to their conceptual simplicity, SSPs offer several advantages over previous constructions for binary circuits. Their reduced number of constraints lead to smaller programs, and to lower sizes and degrees for the polynomials required to represent them, which in turn reduce the computation complexity required in proving or verifying SNARKs.

Let $\mathtt{C}$ be a boolean circuit with m wires and n fan-in 2 gates. We formally define SSPs ([DFGK14]):

Definition 7.3 (SSP). *A Square Span Program (SSP) S over the field $\mathbb{F}$ is a tuple consisting of $m+1$ polynomials $v_0(x), \ldots, v_m(x) \in \mathbb{F}[x]$ and a target polynomial $t(x)$ such that $\deg(v_i(x)) \leqslant \deg(t(x))$ for all $i = 0, \ldots, m$. We say that the square span program SSP has size m and degree $d = \deg(t(x))$. We say that SSP accepts an input $c_1, \ldots, c_N \in \{0,1\}$ if and only if there exist $c_{N+1}, \ldots, c_m \in \{0,1\}$ such that $t(x)$ divides $p(x)$, where:*

$$p(x) := \left(v_0(x) + \sum_{i=1}^{m} c_i v_i(x)\right)^2 - 1.$$

We say that SSP S verifies a boolean circuit $\mathtt{C} : \{0,1\}^N \to \{0,1\}$ if it accepts exactly those inputs $(c_1, \ldots, c_N) \in \{0,1\}^N$, satisfying $\mathtt{C}(c_1, \ldots c_N) = 1$.

Theorem 7.4 ([DFGK14, Theorem 2]). *For any boolean circuit $\mathtt{C}$ of m wires and n fan-in 2 gates and for any prime $p \geq \max(n, 8)$, there exist polynomials $v_0(x), \ldots, v_m(x)$ such that, for any distinct roots*

$r_1, \ldots, r_d \in \mathbb{F}$, *C is satisfiable if and only if:*

$$\prod_{i=1}^{d}(x - r_i) \text{ \textit{divides} } p(x) := \left(v_0(x) + \sum_{i=1}^{m} c_i v_i(x)\right)^2 - 1,$$

where $c_1, \ldots, c_m \in \{0, 1\}$ *correspond to the values on the wires in a satisfying assignment for the circuit.*

Define $t(x) := \prod_{i=1}^{d}(x - r_i)$*, then for any circuit* C *of* m *wires and* n *gates, there exists a degree* $d = m + n$ *square span program* $S = (v_0(x), \ldots, v_m(x), t(x))$ *over a field* $\mathbb{F}$ *of order* p *that verifies* C*.*

Building a SSP S for a general boolean circuit $\mathtt{C} : \{0, 1\}^N \to \{0, 1\}$ with m wires and n fan-in 2 gates follows some simple steps (See Figure 5 for a toy example).

First, we represent an assignment to the wires of C as a vector $\mathbf{c} \in \{0, 1\}^m$. The assignment is a satisfying witness for the circuit if and only if the inputs belong to $\{0, 1\}$, the inputs respect all gates, and the output wire is 1. It is easy to impose the condition $c_i \in \{0, 1\}, \ \forall i \in [m]$ by requiring $2c_i \in \{0, 2\}$. Scaling some of the gate equations from Figure 5 by a factor 2, we can write all gate equations in the form $L = \alpha c_i + \beta c_j + \gamma c_k + \delta \in \{0, 2\}$. We want the circuit output wire c_{out} to have value 1. We do that by adding the condition $3 - 3c_{\mathsf{out}}$ to the linearization of the output gate.

We further define a matrix $\mathbf{V} \in \mathbb{Z}^{m \times d}$ and $\mathbf{b} \in \mathbb{Z}^d$ such that $\mathbf{cV} + \mathbf{b} \in \{0, 2\}^d$ corresponds to the linearization of the gates and of inputs/outputs as described above. The existence of $\mathbf{c}$ such that $\mathbf{cV} + \mathbf{b} \in \{0, 2\}^d$ is equivalent to a satisfying assignment to the wires in the circuit. We can rewrite this condition as

$$(\mathbf{cV} + \mathbf{b}) \circ (\mathbf{cV} + \mathbf{b} - \mathbf{2}) = 0 \iff (\mathbf{cV} + \mathbf{b} - \mathbf{1}) \circ (\mathbf{cV} + \mathbf{b} - \mathbf{1}) = \mathbf{1}, \tag{2}$$

where $\circ$ denotes the Hadamard product (entry-wise multiplication).

Next step consists in defining the polynomials $\{v_i(x)\}_{i=0}^{m}$. Let $r_1, \ldots r_d$ be d distinct elements of a field $\mathbb{F}$ of order p for a prime $p \geq \max(d, 8)$. Define $v_0(x), v_1(x), \ldots v_m(x)$ as the degree $d - 1$ polynomials satisfying $v_0(r_j) = b_j - 1$ and $v_i(r_j) = V_{i,j}$.

We can now reformulate condition 2 again: The circuit C is satisfiable if and only if there exists $\mathbf{c} \in \mathbb{F}^m$ such that for all r_j : $\left(v_0(r_j) + \sum_{i=1}^{m} c_i v_i(r_j)\right)^2 = 1.$

Since the evaluations in $r_1, \ldots r_d$ uniquely determine the polynomial $v_{\mathbf{c}}(x) = v_0(x) + \sum_{i=1}^{m} c_i v_i(x)$ we can rewrite the condition 2:

$$\prod_{i=1}^{d-1}(x - r_i) \text{ divides } \left(v_0(x) + \sum_{i=1}^{m} c_i v_i(x)\right)^2 - 1.$$

Proving on Top of Quadratic and Square Programs. Once we have stated the corresponding quadratic/square span program associated to the (boolean or arithmetic) circuit, the steps in building a proof protocol from this polynomial problem are the following:

Prover. The prover has to solve a SSP (or a QAP) that consists of a set of polynomials $\{v_i(x)\}$ (or respectively $\{v_i(x)\}, \{w_i(x)\}, \{y_i(x)\}$). In both cases, the task is to find a linear combination $\{c_i\}$ of its input polynomials – $v_{\mathbf{c}}(x) = v_0(x) + \sum_i c_i v_i(x)$ (and $w_{\mathbf{c}}(x), y_{\mathbf{c}}(x)$ for QAP) – in such a way that the polynomial $p(x)$ defined by the program is a multiple of another given polynomial $t(x)$.

For a given input, the worker evaluates the circuit C directly to obtain the output and the values of the internal circuit wires. These values correspond to the coefficients $\{c_i\}_{i=1}^{m}$ of the quadratic/square program.

Verifier. From the other side, the verification task consists of checking whether one polynomial divides another polynomial. This can be facilitated by the prover if it sends the quotient polynomial $h(x)$

such that $t(x)h(x) = p(x)$, which turns the task of the verifier into checking a polynomial identity $t(x)h(x) = p(x)$. Put differently, verification consists into checking that $t(x)h(x) - p(x) = 0$, i.e., checking that a certain polynomial is the zero polynomial.

Efficiency. Since the size of these polynomials is very large, the verifier will need a more efficient way to check the validity of the proof, than to multiply such big polynomials. Also, from the point of view of succinctness, sending the polynomials $h(x), v_{\mathbf{c}}(x)$ (and $w_{\mathbf{c}}(x), y_{\mathbf{c}}(x)$ for QAP), each of degrees proportional with the number of gates in the original circuit, is not optimal for our purposes.

Evaluation in a Random Point. So, instead of actually computing polynomial products, the verifier chooses a secret random point s and ask the prover to send the evaluations $h(s), v_{\mathbf{c}}(s)$ (and $w_{\mathbf{c}}(s), y_{\mathbf{c}}(s)$ for QAP) instead of the full polynomials and only checks that $t(s)h(s) = p(s)$. So the polynomial operations are simplified to field multiplications and additions independent of the degree of those polynomials.

Soundness. Checking a polynomial identity only at a single point instead of at all points reduces the security, but according to Schwartz–Zippel lemma any two distinct polynomials of degree d over a field $\mathbb{F}$ can agree on at most a $d/|\mathbb{F}|$ fraction of the points in $\mathbb{F}$. So, if we choose the field $\mathbb{F}$ carefully, $s \leftarrow_{\$} \mathbb{F}$ is assumed to be picked at random and since $t(x)h(x), p(x)$ are non-zero polynomials, the possibility of a false proof to verify is bounded by a negligible fraction (where the evaluations $h(s), p(s)$ are part of, or can be computed from the proof elements). Of course, the point s should be not known in advance by the prover when it generates its polynomials. This is essential to avoid cheating strategies that lead to proofs of false statements.

Encoding the Random Point. We have concluded that the key factor for the soundness to hold is the secrecy of the evaluation point s. The prover should not know in advance this value when computing the solution to SSP $v_{\mathbf{c}}(x), h(s)$ (respectevely $v_{\mathbf{c}}(x), w_{\mathbf{c}}(x), y_{\mathbf{c}}(x), h(s)$ for QAP). Nevertheless, the prover should be allowed to compute the evaluation of its polynomials in s. Finding a method of hiding s that, at the same time, allows the prover to perform linear operations over this hidden value and the verifier to check the proof, is the key trick in order to build a SNARK.

7.5 Encoding Schemes

The main ingredient for an efficient preprocessing SNARK is an encoding scheme Enc over a field $\mathbb{F}$ that hides the evaluation point s and has important properties that allow proving and verifying on top of encoded values.

A formalisation of these encoding schemes for SNARKs was initially introduced in [GGPR13]:

Definition 7.5 (Encoding Scheme)**.** *An encoding scheme* Enc *over a field* $\mathbb{F}$ *is composed of the following algorithms:*

$\mathsf{K}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ a key generation algorithm that takes as input some security parameter and outputs some secret state sk together with some public information pk.

$\mathsf{Enc}(s) \to z$ an encoding algorithm mapping a field element s to some encoding value. Depending on the encoding algorithm, Enc will require either the public information pk generated from K, or the secret state sk. To ease notation, we will omit this additional argument.

The above algorithms must satisfy the following properties:

- ***additively homomorphic:*** Intuitively, we want the encoding scheme to behave well when applying linear operations $\mathsf{Enc}(x + y) = \mathsf{Enc}(x) + \mathsf{Enc}(y)$.
- ***quadratic root detection:*** There exists an efficient algorithm that, given $\mathsf{Enc}(a_0), \ldots, \mathsf{Enc}(a_t)$, and the quadratic polynomial $\mathsf{pp} \in \mathbb{F}[x_0, \ldots, x_t]$, can distinguish if $\mathsf{pp}(a_1, \ldots, a_t) = 0$. We will use an informal notation for this check

$$\mathsf{pp}(\mathsf{Enc}(a_0), \ldots, \mathsf{Enc}(a_t)) \stackrel{?}{=} 0.$$

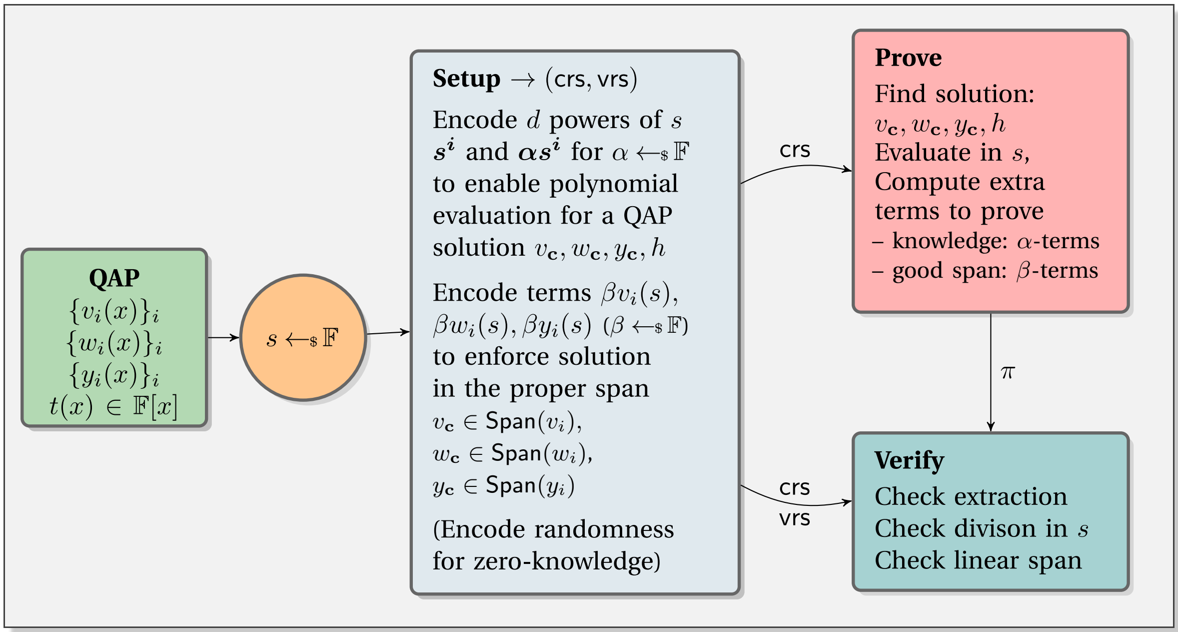


Figure 6: Simplified roadmap from QAP to a SNARK protocol.

- ***image verification:*** There exists an efficiently computable algorithm ImVer that can distinguish if an element c is a correct encoding of a field element ($\mathsf{ImVer}(c) \rightarrow 0/1$).

Publicly vs Designated-Verifier Encoding. In some instantiations, the encoding algorithm will need a secret state sk to perform the *quadratic root detection*. Then, in the resulting SNARK, the verification algorithm does need a verification key $\mathsf{vrs} = \mathsf{sk}$.

If such a secret state is not needed, we will consider $\mathsf{sk} = \perp$ and call it "one-way" or *publicly-verifiable* encoding.

Bilinear Groups. At present, the only candidates for such a "one-way" encoding scheme that we know of are based on bilinear groups, where the the bilinear maps support efficient testing of quadratic degrees without any additional secret information. A symmetric bilinear group is given by a description $(p, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$, where:

- p is a λ-bit prime
- $\mathbb{G}, \mathbb{G}_T$ are cyclic groups of order p
- $\mathsf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear map: $\forall a, b \in \mathbb{Z}_p : \mathsf{e}(g^a, g^b) = \mathsf{e}(g, g)^{ab}$
- if $\langle g \rangle = \mathbb{G}$, then $\langle \mathsf{e}(g, g) \rangle = \mathbb{G}_T$

Example 7.6 (Pairing-based encoding scheme)**.** *Consider a symetric bilinear group of prime order q described by $\mathsf{gk} := (p, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$. Let g be a generator of $\mathbb{G}$. We can implement an encoding scheme with the previous properties as:*

$$\mathsf{Enc}(a) = g^a.$$

- *additively homomorphic:* To compute en encoding of a sum $g^{(a_1+a_2)}$, we just multiply the respective group elements $g^{a_1} g^{a_2} = g^{a_1+a_2}$.
  For a known polynomial $h(x)$, this property can be used to compute an encoding of an evaluation $h(s) = \sum_{i=0}^{d} h_i s^i$ in some point s, given the coefficients $\{h_i\}_{i=1}^{d}$ and the encodings of powers of s,

$\{g^{s^i}\}_{i=1}^d$. This is the linear combination

$$\prod_{i=0}^{d}(g^{s^i})^{h_i} = g^{\sum_{i=0}^{d} h_i s^i} = g^{h(s)}$$

- *quadratic root detection:* Given for example the following quadratic polynomial $\mathsf{pp}_0 = x_1x_2 + x_3^2$ and some encodings $(g^{a_1}, g^{a_2}, g^{a_3})$ use the bilinear map to check the equality:

$$\mathsf{e}(g,g)^{\mathsf{pp}_0(a_1,a_2,a_3)} = \mathsf{e}(g,g)^{a_1a_2+a_3^2} = \mathsf{e}(g^{a_1}, g^{a_2}) \cdot \mathsf{e}(g^{a_3}, g^{a_3}) \stackrel{?}{=} \mathsf{e}(g,g)^0.$$

- *image verification:* Typically, it is straightforward to determine whether an element is in the group $\mathbb{G}$, and all elements of $\mathbb{G}$ are valid encodings.

Remark 7.7. *Remark that none of the three properties requires any secret state, this leads to a publicly-verifiable SNARK, to perform the checks; the verification algorithm does not need anything else than the pairing function* e *that is public. Note also that this encoding scheme is deterministic.*

For instance, the family of elliptic curves $\mathbb{G} := E(\mathbb{F}_q)$. described in [BF01] satisfies the above description.

7.6 Pairing-Based Assumptions

The q-type Assumptions. Non-static q*-type* assumptions are parametrized by q, and they, are actually, a family of assumptions. They may be used in a static way for a fixed q, but if a security proof relies on the non-static version, then q is usually related to the number of oracle queries an adversary makes, to the size of input or to the number of computational steps necessary in a protocol.

The q–Power Diffie-Hellman (q-PDH). Let the generator $\mathcal{G}$ denote the algorithm by which bilinear groups are generated. $\mathcal{G}$ inputs a security parameter λ and outputs a description of a bilinear group $\mathsf{gk} := (p, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \leftarrow_\$ \mathcal{G}(1^\lambda)$. Roughly speaking, the q-PDH assumption says that given $g, g^s, \dots g^{s^q}, g^{s^{q+2}}, \dots g^{s^{2q}}$ it is hard to compute $y = g^{s^{q+1}}$. A heuristic argument for believing in the q-PDH hardness is given by Groth in [Gro10].

Assumption 7.8 (q-PDH)**.** *The q-Power Diffie-Hellman (q-PDH) assumption holds for the bilinear group generator $\mathcal{G}$ if for all* PPT *adversaries $\mathcal{A}$ we have, on the probability space* $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $g \leftarrow_\$ \mathbb{G}$ *and* $s \leftarrow_\$ \mathbb{Z}_p$*:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{q\text{-}pdh}} := \Pr\left[q\text{-}\mathsf{PDH}_{\mathcal{A}}(\mathsf{gk}, 1^\lambda) = 1\right] = \mathsf{negl}.$$

where q-$\mathsf{PDH}_{\mathcal{A}}$ is defined as in Figure 7.

q-$\mathsf{PDH}_{\mathcal{A}}(\mathsf{gk}, 1^\lambda)$	q-$\mathsf{SDH}_{\mathcal{A}}(\mathsf{gk}, 1^\lambda)$
$g \leftarrow_\$ \mathbb{G}$	$g \leftarrow_\$ \mathbb{G}$
$s \leftarrow_\$ \mathbb{Z}_p$	$s \leftarrow_\$ \mathbb{Z}_p$
$\tau \leftarrow (g, g^s, \dots g^{s^q}, g^{s^{q+2}}, \dots g^{s^{2q}})$	$\sigma \leftarrow (g, g^s, \dots g^{s^q})$
$y \leftarrow \mathcal{A}(\mathsf{gk}, \tau)$	$(r, y) \leftarrow \mathcal{A}(\mathsf{gk}, \sigma)$
return $(y = g^{s^{q+1}})$	**return** $(y = g^{1/(s-r)})$

Figure 7: Games for q-PDH and q-SDH assumptions.

The q–Strong Diffie-Hellman Assumption (q-SDH). The Strong Diffie-Hellman assumption [BB08] says that given gk, $g \leftarrow_{\$} \mathbb{G}$ and a set of powers $(g, g^s, \ldots g^{s^q})$ for a random exponent $s \leftarrow_{\$} \mathbb{Z}_p$, it is infeasible to compute $y = g^{\frac{1}{s-r}}$ for a chosen $r \in \mathbb{Z}_p$.

Assumption 7.9 (q-SDH)**.** *The q–Strong Diffie-Hellman assumption holds relative to a bilinear group generator $\mathcal{G}$ if for all* PPT *adversaries $\mathcal{A}$ we have, on the probability space $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $g \leftarrow_{\$} \mathbb{G}$ and $s \leftarrow_{\$} \mathbb{Z}_p$:*

$$\mathsf{Adv}^{\mathsf{q\text{-}sdh}}_{\mathcal{A}} := \Pr\left[q\text{-}\mathsf{SDH}_{\mathcal{A}}(\mathsf{gk}, 1^\lambda) = 1\right] = \mathsf{negl}.$$

where q-$\mathsf{SDH}_{\mathcal{A}}$ is the experiment depicted in Figure 7.

7.6.1 Knowledge Assumptions

All SNARK constructions are inherently based on non-falsifiable assumptions [Nao03b], as stated by Gentry and Wichs in their work [GW11].

The framework of such an assumption is as follows: a knowledge assumption considers any PPT algorithm M that, on input a security parameter λ returns a secret output and a public output. Then, the assumption states that if M satisfies certain efficiency or hardness properties (to be defined later), then for any adversary algorithm $\mathcal{A}$ trying to simulate M, there exists an efficient algorithm $\mathcal{E}_{\mathcal{A}}$ that, given the security parameter, $\mathcal{A}$'s public output and random bits, can compute a matching secret output.

The q-Power Knowledge of Exponent Assumption. This class of assumptions have the following flavor: if an efficient algorithm, given the description of a finite group along with some other public information, computes a list of group elements that satisfies a certain algebraic relation, then there exists a knowledge extractor that outputs some related values that "explain" how the public information was put together to satisfy the relation. The knowledge of exponent (KEA) assumption was the first of this type, introduced by Damgard [Dam92]. It says that given g, g^α in a group $\mathbb{G}$ it is infeasible to create $c, \widehat{c}$ so $\widehat{c} = c^\alpha$ without knowing a such that $c = g^a$ and $\widehat{c} = (g^\alpha)^a$.

$$\begin{array}{l}
\underline{q\text{-}\mathsf{PKE}_{\mathsf{gk},\mathcal{Z},\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda)} \\
g \leftarrow_{\$} \mathbb{G}, \;\; s \leftarrow_{\$} \mathbb{Z}_p \\
\sigma \leftarrow (g, g^s, \ldots g^{s^q}, g^\alpha, g^{\alpha s} \ldots g^{\alpha s^q}) \\
z \leftarrow \mathcal{Z}(\mathsf{gk}, \sigma) \\
(c, \widehat{c}; \{a_i\}_i^q) \leftarrow (\mathcal{A} \| \mathcal{E}_{\mathcal{A}})(\sigma, z) \\
\mathbf{return}\ (\widehat{c} = c^\alpha) \;\wedge\; c \neq \prod_i^q (g^{s^i})^{a_i}
\end{array}$$

Figure 8: Game for q-PKE assumption.

The q-power knowledge of exponent assumption (q-PKE) is a generalization of KEA. It says that given the successive powers of some random value $s \in \mathbb{Z}_p$ encoded in the exponent $\{g, g^s, g^{s^2}, \ldots, g^{s^q}, g^\alpha, g^{\alpha s} \ldots g^{\alpha s^q}\}$, it is infeasible to create $c, \widehat{c}$ where $\widehat{c} = c^\alpha$ without knowing $a_0, a_1, \ldots a_q$ that satisfy $c = \prod_{i=0}^q (g^{s^i})^{a_i}$. This is more formally defined (in the symmetric case) by the existence of an extractor:

Assumption 7.10 (q-PKE)**.** *The q-Power Knowledge of Exponent (q-PKE) assumption holds relative to a bilinear group given by the description* gk *and for the class* Z *of auxiliary input generators if, for every non-uniform* PPT *auxiliary input generator $\mathcal{Z} \in$* Z *and non-uniform* PPT *adversary $\mathcal{A}$, there exists a non-uniform* PPT *extractor $\mathcal{E}_{\mathcal{A}}$ such that:*

$$\mathsf{Adv}^{\mathsf{q\text{-}pke}}_{\mathsf{Enc},\mathcal{Z},\mathcal{A},\mathcal{E}_{\mathcal{A}}} := \Pr\left[\mathsf{q\text{-}PKE}_{\mathsf{Enc},\mathcal{Z},\mathcal{A},\mathcal{E}_{\mathcal{A}}} = \mathbf{true}\right] = \mathsf{negl},$$

$\mathsf{Gen}(1^\lambda, \mathtt{C})$

$\mathsf{gk} := (p, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$

$s, \alpha, \beta_v, \beta_w, \beta_y \leftarrow_\$ \mathbb{Z}_q$

$Q := \big(\{v_i, w_i, y_i\}_{i\in[m]}, t\big)$

$\mathcal{I}_{\mathsf{mid}} = \{N+1, \ldots m\}$

$\mathsf{crs} := \Big(Q, \mathsf{gk},$

$\{g^{s^i}, g^{\alpha s^i}\}_{i=0}^{d}$

$g^{\beta_v}, \{g^{\beta_v v_i(s)}\}_{i\in\mathcal{I}_{\mathsf{mid}}}$

$g^{\beta_w}, \{g^{\beta_w w_i(s)}\}_{i\in[m]}$

$g^{\beta_y}, \{g^{\beta_y y_i(s)}\}_{i\in[m]}\Big)$

return crs

$\mathsf{Prove}(\mathsf{crs}, u, w)$

$u := (c_1, \ldots, c_N)$

$w := (\{c_i\}_{i\in\mathcal{I}_{\mathsf{mid}}})$

$v_{\mathsf{mid}} := \sum_{i\in\mathcal{I}_{\mathsf{mid}}} c_i v_i(x)$

$H := g^{h(s)}, \quad \widehat{H} := g^{\alpha h(s)}$

$V_{\mathsf{mid}} := g^{v_{\mathsf{mid}}(s)}, \ \widehat{V}_{\mathsf{mid}} := g^{\alpha v_{\mathsf{mid}}(s)}$

$W := g^{w_{\mathbf{c}}(s)}, \quad \widehat{W} := g^{\alpha w_{\mathbf{c}}(s)}$

$Y := g^{y_{\mathbf{c}}(s)}, \quad \widehat{Y} := g^{\alpha y_{\mathbf{c}}(s)}$

$B := g^{\beta_v v_{\mathbf{c}}(s)+\beta_w w_{\mathbf{c}}(s)+\beta_y y_{\mathbf{c}}(s)}$

$\pi := (H, \widehat{H}, V_{\mathsf{mid}}, \widehat{V}_{\mathsf{mid}}, W, \widehat{W}, Y, \widehat{Y}, B)$

$\mathsf{Ver}(\mathsf{crs}, u, \pi)$

Extractability check:

$\mathsf{e}(H, g^\alpha) = \mathsf{e}(g, g^{\widehat{H}})$

$\mathsf{e}(V_{\mathsf{mid}}, g^\alpha) = \mathsf{e}(g, g^{\widehat{V}_{\mathsf{mid}}})$

$\mathsf{e}(W, g^\alpha) = \mathsf{e}(g, g^{\widehat{W}})$

$\mathsf{e}(Y, g^\alpha) = \mathsf{e}(g, g^{\widehat{Y}})$

Divisibility check:

$\mathsf{e}(H, g^{t(s)}) = \mathsf{e}(V, W)/\mathsf{e}(Y, g)$

Linear span check:

$\mathsf{e}(B, g) = \mathsf{e}(V, g^{\beta_v}) \cdot \mathsf{e}(W, g^{\beta_w}) \cdot \mathsf{e}(Y, g^{\beta_y})$

Figure 9: SNARK from QAP.

where $\mathsf{q\text{-}PKE}_{\mathsf{Enc},\mathcal{Z},\mathcal{A},\mathcal{E}_\mathcal{A}}$ *is the game depicted in Figure 8.*

These assumptions can be reformulated in terms of encodings in the sense of Section 7.5, as a generalization of the exponential function in the bilinear group.

7.7 SNARKs from QAP

In what follows, we will present the celebrated SNARK construction of Parno et al. [PHGR13].

Equipped with the encoding tool we have defined above, we are ready to construct a QAP-based SNARK scheme. A very high-level overview of the SNARK from QAP construction is provided in Figure 6. This diagram gives some intuition, but hides a lot of important details of the scheme.

For the sake of the presentation, the description of the protocol is given for a general encoding scheme Enc. In Figure 9 there is a SNARK construction for an instantiation based on bilinear group encodings (see Example 7.6).

Generation Algorithm $\mathsf{Gen}(1^\lambda, \mathtt{C}) \to (\mathsf{crs}, \mathsf{vrs})$

The setup algorithm Gen takes as input 1^λ and the circuit $\mathtt{C}$ with N input/output values. It generates a QAP Q of size m and degree d over a field $\mathbb{F}$, that verifies $\mathtt{C}$. It defines $\mathcal{I}_{\mathsf{mid}} = \{N+1, \ldots m\}$. Then, it runs the setup for an encoding scheme Enc (with secret state sk, or without, $\mathsf{sk} = \perp$). Finally, it samples $\alpha, \beta_v, \beta_w, \beta_y, s \leftarrow \mathbb{F}$ such that $t(s) \neq 0$, and returns $(\mathsf{vrs} = \mathsf{sk}, \mathsf{crs})$ where crs is:

$$\begin{aligned}\mathsf{crs} := \Big(Q, \mathsf{Enc}, &\{\mathsf{Enc}(1), \mathsf{Enc}(s), \ldots, \mathsf{Enc}(s^d), \mathsf{Enc}(\alpha), \mathsf{Enc}(\alpha s), \ldots, \mathsf{Enc}(\alpha s^d)\},\\ &\{\mathsf{Enc}(\beta_v), \mathsf{Enc}(\beta_w), \mathsf{Enc}(\beta_y)\}\\ &\{\mathsf{Enc}(\beta_v v_i(s))\}_{i\in\mathcal{I}_{\mathsf{mid}}}, \{\mathsf{Enc}(\beta_w w_i(s))\}_{i\in[m]}, \{\mathsf{Enc}(\beta_y y_i(s))\}_{i\in[m]}\Big)\end{aligned} \tag{3}$$

Prover $\mathsf{Prove}(\mathsf{crs}, u, w) \to \pi$

The prover algorithm Prove, on input some statement $u := (c_1, \ldots, c_N)$, computes a witness $w := (c_{N+1} \ldots c_m)$ and $v_{\mathsf{mid}}(x) = \sum_{i \in I_{\mathsf{mid}}} c_i v_i(x), v_{\mathbf{c}}(x) = \sum_{i \in [m]} c_i v_i(x)$, $w_{\mathbf{c}}(x) = \sum_{i \in [m]} c_i w_i(x), y_{\mathbf{c}}(x) = \sum_{i \in [m]} c_i y_i(x)$ such that :

$$t(x) \text{ divides } p(x) = v_{\mathbf{c}}(x) w_{\mathbf{c}}(x) - y_{\mathbf{c}}(x).$$

Then, it computes the quotient polynomial $h(x)$:

$$h(x) := \frac{p(x)}{t(x)}. \tag{4}$$

By using the additively homomorphic property of the encoding scheme Enc and the values in the crs, the prover computes encodings of the following polynomial evaluations in s:

$$\begin{aligned}
H &:= \mathsf{Enc}(h(s)), & \widehat{H} &:= \mathsf{Enc}(\alpha h(s)), \\
V_{\mathsf{mid}} &:= \mathsf{Enc}(v_{\mathsf{mid}}(s)), & \widehat{V}_{\mathsf{mid}} &:= \mathsf{Enc}\left(\alpha v_{\mathsf{mid}}(s)\right), \\
W &:= \mathsf{Enc}(w_{\mathbf{c}}(s)), & \widehat{W} &:= \mathsf{Enc}\left(\alpha w_{\mathbf{c}}(s)\right), \\
Y &:= \mathsf{Enc}(y_{\mathbf{c}}(s)), & \widehat{Y} &:= \mathsf{Enc}\left(\alpha y_{\mathbf{c}}(s)\right), \\
B &:= \mathsf{Enc}\left(\beta_v v_{\mathbf{c}}(s) + \beta_w w_{\mathbf{c}}(s) + \beta_y y_{\mathbf{c}}(s)\right).
\end{aligned} \tag{5}$$

Where the polynomial $v_{\mathsf{mid}}(x) := \sum_{i \in \mathcal{I}_{\mathsf{mid}}} c_i v_i(x)$. Since the values of c_i, where $i \in [N]$ correspond to the input u (which is also known to the verifier), the verifier can compute the missing part of the full linear combination $v_{\mathbf{c}}(x)$ for $\{v_i(x)\}$ and encode it by itself

$$V := \mathsf{Enc}(v_{\mathbf{c}}(s)).$$

The proof π consists of elements $(H, \widehat{H}, V_{\mathsf{mid}}, \widehat{V}_{\mathsf{mid}}, W, \widehat{W}, Y, \widehat{Y}, B)$.

Verifier $\mathsf{Ver}(\mathsf{vrs}, u, \pi) \to 0/1$

Upon receiving a proof π and a statement u, the verifier, uses the quadratic root detection algorithm of the encoding scheme Enc to verify that the proof satisfies:

Extractability terms. $\widehat{H} \stackrel{?}{=} \alpha H, \qquad \widehat{V}_{\mathsf{mid}} \stackrel{?}{=} \alpha V_{\mathsf{mid}}, \qquad \widehat{W} \stackrel{?}{=} \alpha W, \qquad \widehat{Y} \stackrel{?}{=} \alpha Y.$
The above terms are engineered to allow extractability using a knowledge assumption.

Divisibility check. $H \cdot T \stackrel{?}{=} V \cdot W - Y$ where $T = \mathsf{Enc}(t(s)),\ V := \mathsf{Enc}(v_{\mathbf{c}}(s))$ and can be computed using crs. This corresponds to the polynomial division constraint.

Linear span check. $B \stackrel{?}{=} \beta_v V + \beta_w W + \beta_y Y$. This check makes sure that the polynomials $v_{\mathbf{c}}(x), w_{\mathbf{c}}(x)$ and $y_{\mathbf{c}}(x)$ are indeed linear combinations of the initial set of polynomials $\{v_i\}_i, \{w_i\}_i, \{y_i\}_i$.

7.7.1 Knowledge Soundness

The intuition is that it is hard for the prover, who knows the CRS but not α, to output any pair $(H, \widehat{H})$ where H encodes some value h and $\widehat{H} = \mathsf{Enc}(\alpha h)$ unless the prover knows a representation $h = \sum_{i \in [d]} h_i s^i$ and applies the same linear combination to αs^i in order to obtain $\widehat{H} = \sum_{i \in [d]} h_i \alpha s^i$. Knowledge of exponent assumptions (PKE defined in Assumption 7.10) formalize this intuition; it says that for any algorithm that outputs a pair of encoded elements with ratio α, there is an extractor that "watches" the algorithm's computation and outputs the corresponding representation (the coefficients of a linear combination).

We will give an overview of the proof in tree steps, one for each of the three checks in the verification algorithm:

Extractability terms. From the pairs of encodings $(H, \widehat{H}), (V_{\mathsf{mid}}, \widehat{V}_{\mathsf{mid}}), (W, \widehat{W}), (Y, \widehat{Y})$, based on the q-PKE assumption (see Assumption 7.10) we can extract out coefficients for polynomials $v_{\mathsf{mid}}(x), w_{\mathbf{c}}(x)$, $y_{\mathbf{c}}(x), h(x)$.

Divisibility check. If the check $H \cdot T \stackrel{?}{=} V \cdot W - Y$ where $T = \mathsf{Enc}(t(s))$ verifies, then $h(s)t(s) = v_{\mathbf{c}}(s)w_{\mathbf{c}}(s) - y_{\mathbf{c}}(s)$. If indeed $h(x)t(x) = v_{\mathbf{c}}(x)w_{\mathbf{d}}(x) - y_{\mathbf{c}}(x)$ as polynomials, the soundness of our QAP implies that we have extracted a true proof. Otherwise, $h(x)t(x) - v_{\mathbf{c}}(x)w_{\mathbf{d}}(x) - y_{\mathbf{c}}(x)$ is a nonzero polynomial having s as a root, which allows us to solve a q-type assumption instance.

Linear span check. In the scheme, the α-terms $\widehat{V}_{\mathsf{mid}}, \widehat{W}$ and $\widehat{Y}$ are used only to extract representations of the encoded terms with respect to the power basis, and not as a linear combination in the set of polynomials $\{v_i(x), w_i(x), y_i(x)\}_{i\in[m]}$. This extraction does not guarantee that the polynomials $v_{\mathsf{mid}}(x), w_{\mathbf{c}}(x), y_{\mathbf{c}}(x)$ lie in the appropriate spans. Therefore, the final proof term B is needed to enforce this. B can only be computed by the prover from the crs by representing $v_{\mathbf{c}}, w_{\mathbf{c}}, y_{\mathbf{c}}$ as a linear combination of corresponding $\{v_i(x), w_i(x), y_i(x)\}_{i\in[m]}$. This is then checked in the verification algorithm $B \stackrel{?}{=} \beta_v V + \beta_w W + \beta_y Y$. If this final check passes, but polynomials $v_{\mathbf{c}}, w_{\mathbf{c}}, y_{\mathbf{c}}$ lie outside their proper span, then the one can solve d-power Diffie-Hellman problem (see Assumption 7.8 for $q = d$).

7.7.2 Adding Zero-Knowledge

The construction we just described is not zero-knowledge, since the proof terms are not uniformly distributed and may reveal information about the witness, i.e., about the polynomials $v_{\mathbf{c}}(x) = \sum_i c_i v_i(x)$, $w_{\mathbf{c}}(x),\ y_{\mathbf{c}}(x),\ h(x)$. To make this proof statistically zero-knowledge, we will randomize the polynomials $v_{\mathbf{c}}(x), w_{\mathbf{c}}(x), y_{\mathbf{c}}(x), h(x)$ by adding a uniformly sampled value, while keeping the divisibility relation between them. The idea is that the prover just uses some random values $\delta_v, \delta_w, \delta_y \in \mathbb{F}$ and performs the following replacements in the proof to randomize its original polynomials from above:

- $v'_{\mathsf{mid}}(x) \coloneqq v_{\mathsf{mid}}(x) + \delta_v t(x)$,
- $w'_{\mathbf{c}}(x) \coloneqq w_{\mathbf{c}}(x) + \delta_w t(x)$,
- $y'_{\mathbf{c}}(x) \coloneqq y_{\mathbf{c}}(x) + \delta_y t(x)$,
- $h'(x) = h(x) + \delta_v w_{\mathbf{c}}(x) + \delta_w v_{\mathbf{c}}(x) + \delta_v \delta_w t^2(x) - \delta_y t(x)$.

By these replacements, the values V_{mid}, W and Y, which contain an encoding of the witness factors, basically become indistinguishable from randomness and thus intuitively they are zero-knowledge. For this modification to be possible, additional terms containing the randomness $\delta_v, \delta_w, \delta_y$ should be added to the crs to enable the prover to mask its proof and the verifier to check it. For a formal proof and other details, we refer the reader to the original work [PHGR13].

7.7.3 Groth16: SNARK in the Generic Group Model

The state-of-the-art for SNARK for QAPs is the celebrated result of Groth [Gro16] that achieves proof size of three group elements in the Generic Group Model.

This construction is simplified compared to the one in [PHGR13]. We observe that the proof elements are not duplicated with respect to some random factor and this is not needed for the extraction, since the security relies on a stronger model, the GGM and not on q-PKE assumption.

Generic Group Model. The generic group model [Sho97, Mau05] is an idealised cryptographic model, where algorithms do not exploit any special structure of the representation of the group elements and can thus be applied in any cyclic group. In this model, the adversary is only given access to a randomly chosen encoding of a group, instead of efficient encodings, such as those used by the finite field or elliptic curve groups used in practice. The model includes an oracle that executes the (additive) group operation. Therefore, one can efficiently extract the coefficients used to express an output of the oracle as a linear combination of initial group elements.

To describe Groth's construction, we consider the relation $\mathcal{R}$ implemented as an arithmetic circuit for a total number of wires m. We denote by $\mathcal{I}_{\mathsf{io}} = \{1, 2, \dots \ell\}$ the indices corresponding to the public input and public output values of the circuit wires (corresponding to the statement u) and by $\mathcal{I}_{\mathsf{mid}} = \{\ell + 1, \dots m\}$, the wire indices corresponding to private input and non-input, non-output intermediate values (for the witness w).

$\mathsf{Groth.Gen}(1^\lambda, \mathtt{C})$

$\alpha, \beta, \gamma, \delta \leftarrow_\$ \mathbb{Z}_p^*, \quad s \leftarrow_\$ \mathbb{Z}_p^*,$

$\mathsf{crs} = \Big(QAP, g^\alpha, g^\beta, g^\delta, \{g^{s^i}\}_{i=0}^{d-1}, \Big\{g^{\frac{\beta v_k(s)+\alpha w_k(s)+y_k(s)}{\gamma}}\Big\}_{k=0}^{\ell}, \Big\{g^{\frac{\beta v_k(s)+\alpha w_k(s)+y_k(s)}{\delta}}\Big\}_{k>\ell},$

$\Big\{g^{\frac{s^i t(s)}{\delta}}\Big\}_{i=0}^{d-2}, h^\beta, h^\gamma, h^\delta, \{h^{s^i}\}_{i=0}^{d-1}\Big)$

$\mathsf{vk} := \Big(P = g^\alpha, Q = h^\beta, \Big\{S_k = g^{\frac{\beta v_k(s)+\alpha w_k(s)+y_k(s)}{\gamma}}\Big\}_{k=0}^{\ell}, H = h^\gamma, D = h^\delta\Big)$

$\mathsf{td} = (s, \alpha, \beta, \gamma, \delta)$

return $(\mathsf{crs}, \mathsf{td})$

$\mathsf{Groth.Prove}(\mathsf{crs}, u, w)$

$u = (a_1, \ldots, a_\ell),\ a_0 = 1$

$w = (a_{\ell+1}, \ldots, a_m)$

$v(x) = \sum_{k=0}^{m} a_k v_k(x)$

$v_{mid}(x) = \sum_{k \in I_{mid}} a_k v_k(x)$

$w(x) = \sum_{k=0}^{m} a_k w_k(x)$

$w_{mid}(x) = \sum_{k \in I_{mid}} a_k w_k(x)$

$y(x) = \sum_{k=0}^{m} a_k y_k(x)$

$y_{mid}(x) = \sum_{k \in I_{mid}} a_k y_k(x)$

$h(x) = \frac{(v(x)w(x)-y(x))}{t(x)}$

$f_{mid} = \frac{\beta v_{mid}(s)+\alpha w_{mid}(s)+y_{mid}(s)}{\delta}$

$r, r' \leftarrow_\$ \mathbb{Z}_p^*$

$a = \alpha + v(s) + r\delta,$

$b = \beta + w(s) + r'\delta$

$c = f_{mid} + \frac{t(s)h(s)}{\delta} + r'a + rb - r'r\delta$

return $\pi : (A = g^a, B = h^b, C = g^c)$

$\mathsf{Groth.Ver}(\mathsf{vk}, u, \pi)$

$\pi = (A, B, C)$

$v_{io}(x) = \sum_{i=0}^{\ell} a_i v_i(x)$

$w_{io}(x) = \sum_{i=0}^{\ell} a_i w_i(x)$

$y_{io}(x) = \sum_{i=0}^{\ell} a_i y_i(x)$

$f_{io} = \frac{\beta v_{io}(s)+\alpha w_{io}(s)+y_{io}(s)}{\gamma}$

Check

$e(A, B) = e(g^\alpha, h^\beta) e(g^{f_{io}}, h^\gamma) e(C, h^\delta)$

$\mathsf{Groth.Sim}(\mathsf{td}, u)$

$a, b \leftarrow_\$ \mathbb{Z}_p^*$

$c = \frac{ab - \alpha\beta - \beta v_{io}(s) + \alpha w_{io}(s) + y_{io}(s)}{\delta}$

return $\pi : (A = g^a, B = h^b, C = g^c)$

Figure 10: Groth16 Construction from QAP.

Groth16 realizes the divisibility and linear span check in a single pairing product equation. The role of α and β is to ensure A, B, and C are consistent with each other in the choice of $a_1, \ldots, a_m$. The role of δ and γ is to make the different products of the verification equation independent of each other. The role of r and r' is to randomizes the proof to get zero-knowledge.

7.8 SNARKs from SSP

Another notable SNARK that achieves reduced computation complexity for proving evaluation of boolean circuits represented as SSPs is the construction of Danezis et al. [DFGK14]. In this short survey, we will restrict ourself to a sketched description of their scheme prioritizing intuition to rigurosity. We refer the reader to [Nit19], Chapter 4, for a more technical presentation of a version of SSP-based SNARKs.

The key element of Danezis' et al. SNARK is the use of SSP language. The square span program require only a single polynomial $v_{\mathbf{c}}(x)$ to be evaluated for verification (instead of two for earlier QSPs, and three for QAPs) leading to a simpler and more compact setup, smaller keys, and fewer operations required for proof and verification. The resulting, SSP-based SNARK may be the most compact construction to date. The proof consists of just 4 group elements; they can be verified in just 6 pairings, plus one multiplication for each (non-zero) bit of input, irrespective of the size of the circuit $\mathtt{C} : \{0,1\}^N \to \{0,1\}$.

$\underline{\mathsf{Gen}(1^\lambda, \mathtt{C})}$

$\mathsf{gk} := (p, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$
$S := (v_0, \ldots, v_m, t)$
$\mathcal{I}_{\mathsf{mid}} := \{N+1, \ldots m\}$
$g \leftarrow_\$ \mathbb{G},\ \ s, \alpha, \beta \leftarrow_\$ \mathbb{Z}_q$
$\mathsf{crs} := \big(S,\ \mathsf{gk},$
$\{g^{s^i}, g^{\alpha s^i}\}_{i=0}^{d},$
$g^\beta, \{g^{\beta v_i(s)}\}_{i \in \mathcal{I}_{\mathsf{mid}}}, g^{\beta t(s)} \big)$
return crs

$\underline{\mathsf{Prove}(\mathsf{crs}, u, w)}$

$u := (c_1, \ldots, c_N) \in \{0,1\}^N$
$w := (\{c_i\}_{i \in \mathcal{I}_{\mathsf{mid}}})$
$v_{\mathsf{mid}} := \sum_{i \in \mathcal{I}_{\mathsf{mid}}} c_i v_i(x)$
$H := g^{h(s)}, \quad V_{\mathsf{mid}} := g^{v_{\mathsf{mid}}(s)}$
$\widehat{V} := g^{\alpha v_{\mathbf{c}}(s)},\ B_{\mathsf{mid}} := g^{\beta v_{\mathsf{mid}}(s)}$
$\pi := (H, V_{\mathsf{mid}}, \widehat{V}, B_{\mathsf{mid}})$

$\underline{\mathsf{Ver}(\mathsf{crs}, u, \pi)}$

$V := g^{\sum_{i \in [N]} c_i v_i(s)} V_{\mathsf{mid}}$
Extractability check:
$\mathsf{e}(V, g^\alpha) = \mathsf{e}(g, \widehat{V})$
Divisibility check:
$\mathsf{e}(H, g^{t(s)}) = \mathsf{e}(V, \widehat{V}) / \mathsf{e}(g, g)^{-1}$
Linear span check:
$\mathsf{e}(B_{\mathsf{mid}}, g) = \mathsf{e}(V_{\mathsf{mid}}, g^\beta)$

Figure 11: SNARK from SSP. A solution to $S : v_{\mathbf{c}}(x) = \sum_i c_i v_i(x),\ h(x) := \frac{v_{\mathbf{c}}(x)^2 - 1}{t(x)}$.

8 SNARKs: Construction from LIP

The QAP approach was generalized in [BCI+13] under the concept of *Linear Interactive Proof* (LIP), a form of interactive ZK proofs where security holds under the assumption that the prover is restricted to compute only linear combinations of its inputs.

These proofs can then be turned into (designated-verifier) SNARKs by using an *extractable linear-only* encryption scheme.

8.1 Linear-Only Encoding Schemes

An *extractable linear-only* encoding scheme is an encoding scheme where any adversary can output a valid new encoding only if this is a linear combination of some previous encodings that the adversary had as input (intuitively this "limited malleability" of the scheme, will force the prover into the above restriction). At high-level, a linear-only encoding scheme does not allow any other form of homomorphism than linear operations.

Definition 8.1 (Extractable Linear-Only, [BCI+13])**.** *An encoding scheme* Enc *satisfies* extractable linear-only *property if for all* PPT *adversaries* $\mathcal{A}$ *there exists an efficient extractor* $\mathcal{E}_\mathcal{A}$ *such that, for any sufficiently large* $\lambda \in \mathbb{N}$*, any "benign" auxiliarly input* z *and any plaintext generation algorithm* M *the advantage*

$$\mathsf{Adv}^{\mathsf{ext\text{-}lo}}_{\mathsf{Enc},\mathsf{M},\mathcal{A},\mathcal{E}_\mathcal{A}} := \Pr[\mathsf{EXT} - \mathsf{LO}_{\mathsf{Enc},\mathsf{M},\mathcal{A},\mathcal{E}_\mathcal{A}} = \mathbf{true}] = \mathsf{negl}.$$

where $\mathsf{EXT} - \mathsf{LO}_{\mathsf{Enc},\mathsf{M},\mathcal{A},\mathcal{E}_\mathcal{A}}$ *is defined as in Figure 12.*

In order for this definition to be non-trivial, the extractor $\mathcal{E}_\mathcal{A}$ has to be *efficient*. Otherwise a naive way of finding such a linear combination $(a_1, \ldots, a_d)$ could be just to run the adversary $\mathcal{A}$, obtain $\mathcal{A}$'s outputs, decode them, and then output a zero linear function and hard-code the correct values in the constant term.

$$
\begin{array}{l}
\underline{\mathsf{EXT} - \mathsf{LO}_{\mathsf{Enc},\mathsf{M},\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda)} \\
(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{K}(1^\lambda) \\
(m_1, \ldots, m_d) \leftarrow \mathsf{M}(1^\lambda) \\
\sigma \leftarrow (\mathsf{pk}, \mathsf{Enc}(m_1), \ldots, \mathsf{Enc}(m_d)) \\
(\mathsf{ct}; a_1, \ldots, a_d, b) \leftarrow (\mathcal{A} \| \mathcal{E}_{\mathcal{A}})(\sigma; z) \\
\text{where } \mathsf{ct} = (\mathsf{Enc}(m)) \\
\mathbf{return}\ \mathsf{ct} \notin \left\{ \mathsf{Enc}(\sum_{i=1}^{d} a_i m_i + b) \right\}
\end{array}
$$

Figure 12: Game for Extractable Linear-Only.

Indistinguishability under Chosen-Plaintext Attacks. A stronger notion of linear-only encoding schemes is *linear-only encryption schemes* that additionally satisfy semantic security in the sense of the game depicted in Figure 13. We say that Enc is IND-CPA secure if, for any PPT adversary $\mathcal{A}$, it holds that

$$
\mathsf{Adv}_{\mathcal{A}}^{\text{ind-cpa}} = \Pr\left[\text{IND-CPA}_{\mathsf{Enc},\mathcal{A}}(1^\lambda)\right] - \frac{1}{2} = \mathsf{negl}.
$$

where $\mathsf{Adv}_{\mathcal{A}}^{\text{ind-cpa}}$ is the advantage of an adversary $\mathcal{A}$ when playing the game $\text{IND-CPA}_{,\mathcal{A}}$.

$$
\begin{array}{l}
\underline{\text{IND-CPA}_{\mathsf{Enc},\mathcal{A}}(1^\lambda)} \\
(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}\left(1^\lambda\right) \\
(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}) \\
b \leftarrow_{\$} \{0, 1\} \\
c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\
b' \leftarrow \mathcal{A}(\mathsf{pk}, c) \\
\mathbf{return}\ b' = b
\end{array}
$$

Figure 13: Game for IND-CPA security notion.

As examples of linear-only encryption schemes, [BCI+13] propose variants of Paillier encryption [Pai99] (as also considered in [GGPR13]) and of ElGamal encryption [ElG85] (in those cases where the plaintext is guaranteed to belong to a polynomial-size set, so that decryption can be done efficiently). These variants are “sparsified” versions of their standard counterparts; concretely, a ciphertext includes not only $\mathsf{Enc}(x)$, but also $\mathsf{Enc}(\alpha x)$, for a secret element α in the message space. (This ”sparsification” follows a pattern found in many constructions conjectured to satisfy ”knowledge-of-exponent” assumptions).

Non-Adaptive SNARK. In [BCI+13], they start from the notion of *linear-targeted malleability*, weaker than linear-only property, that is closer to the definition template of Boneh et al. [BSW12]. In such a notion, the extractor is replaced by an efficient simulator. Relying on this weaker variant, they are only able to prove the security of their preprocessing SNARKs against non-adaptive choices of statements (and still prove soundness, though not knowledge soundness, if the simulator is allowed to be inefficient, i.e., obtain a SNARG instead of a SNARK). Concretely, the linear-only property rules out any encryption scheme where ciphertexts can be sampled obliviously; instead, the weaker notion does not, and thus allows for shorter ciphertexts.

Definition 8.2 (Linear-Targeted Malleability, [BCI+13]). *An encoding scheme* Enc *satisfies* linear-targeted malleability *property if for all* PPT *adversaries* $\mathcal{A}$ *and plaintext generation algorithm* M *there exists a* PPT *simulator* Sim *such that, for any sufficiently large* $\lambda \in \mathbb{N}$, *any "benign" auxiliarly input* z *the following two distributions* $\mathcal{D}_0(\lambda), \mathcal{D}_1(\lambda)$ *in Figure 14 are computationally indistinguishable.*

$\underline{(\mathsf{pk}, \mathsf{st}, \{m_i\}, \{\mathsf{Dec}(\mathsf{ct}_j)\}) \leftarrow \mathcal{D}_0(1^\lambda)}$	$\underline{(\mathsf{pk}, \mathsf{st}, \{m_i\}, \{d_j\}) \leftarrow \mathcal{D}_1(1^\lambda)}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{K}(1^\lambda)$	$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{K}(1^\lambda)$
$(\mathsf{st}, m_1, \ldots, m_d) \leftarrow \mathsf{M}(1^\lambda)$	$(\mathsf{st}, m_1, \ldots, m_d) \leftarrow \mathsf{M}(1^\lambda)$
$\sigma \leftarrow (\mathsf{pk}, \mathsf{Enc}(m_1), \ldots, \mathsf{Enc}(m_d))$	$(\vec{a}_1, \cdots \vec{a}_n, \vec{b}) \leftarrow \mathsf{Sim}(\mathsf{pk}; z)$
$\{\mathsf{ct}_j\}_{j=1}^n \leftarrow \mathcal{A}(\sigma; z)$	$d_j := \sum_{i=1}^d a_{ji} m_i + b_i, \ \forall\, j \in [n]$
where $\mathsf{Dec}(\mathsf{ct}_j) \neq \perp$	

Figure 14: Distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ in Linear Targeted Malleability.

8.2 Linear Interactive Proof

A linear interactive proof (LIP) is defined similarly to a standard interactive proof [GMR85], except that each message sent by a prover (either an honest or a malicious one) must be a linear function of the previous messages sent by the verifier. The SNARK designed by [BCI+13] only makes use of two-message LIPs in which the verifier's message is independent of its input. LIP can be obtained from a Linear PCP, a PCP, for which it is possible to do the verification in such a way that it is sufficient for an honest prover to respond with a certain linear function of the verifier's queries.

Bitansky et al. show first a transformation from any Linear PCP into a two-message LIP with similar parameters. Unlike in the Linear PCP model, if the verifier simply forwards to the LIP prover the queries generated by the Linear PCP verifier, there is no guarantee that the LIP prover will apply the same linear function to each of these queries. Thus, the transformation loses a constant factor in the knowledge error.

Construction of SNARK from LIP. Bitansky et al. [BCI+13] construct a publicly-verifiable preprocessing SNARK from LIPs with low-degree verifiers. Note that, if we aim for public verifiability, we cannot use semantically-secure encryption to encode the message of the LIP verifier, because we need to "publicly test" (without decryption) certain properties of the plaintext underlying the prover's response. The idea, implicit in previous publicly-verifiable preprocessing SNARK constructions, is to use linear-only encodings (rather than encryption) that do allow such public tests, while still providing certain one-wayness properties. When using such encodings with a LIP, however, it must be the case that the public tests support evaluating the decision algorithm of the LIP and, moreover, the LIP remains secure despite some "leakage" on the queries. They show that LIPs with low-degree verifiers (which they call algebraic LIPs), combined with appropriate one-way encodings, suffice for this purpose. More concretely, they consider candidate encodings in bilinear groups (Example 7.6) under similar knowledge-of-exponent and computational Diffie-Hellman assumptions. These LIP constructions imply new constructions of publicly-verifiable preprocessing SNARKs, one of which can be seen as a simplification of the construction of [Gro10] and the other as a reinterpretation (and slight simplification) of the construction of [GGPR13].

9 SNARKs: Construction from PIOP

As we saw previously, SNARKs are commonly build in a modular way. Recent works propose schemes that follow a new framework, enabling more possibilities such as: transparent constructions (without a trusted setup), recursion, aggregation properties, post-quantum security, etc.

In a nutshell, a modular instantiation of recent SNARKS employs *polynomial interactive oracle proofs* (IOP) for the information theoretic step, and *polynomial commitments* for the cryptographic compilation. One advantage of using polynomial commitments is that the setup is only needed for the commitment scheme and is thus independent of the statement being proven. This allows these SNARKs to be universal as opposed to the circuit-based SNARKs presented in the previous section. Another advantage is that the choice of polynomial commitment scheme facilitates finding the right tradeoff between performance and security.

Roadmap. To build such SNARKs, one commonly follows 3 steps:

- *NP Characterization or Arithmetisation for Circuits.* Start with a way to describe the computation to be proven as a system of constraints involving native operations over a finite field. Then, this system of constraints can be changed into a (low-degree) univariate polynomial expression.
- *Polynomial IOP or Algebraic Holographic Proofs.* Then, the proof consists in finding a way to show that such a polynomial equation holds: In a Polynomial Interactive Oracle Proof (PIOP), the prover sends low degree polynomials to the verifier, and rather than reading the entire list of coefficients, the verifier queries evaluations of these polynomials in a random point to an oracle interface.
- *Cryptographic Compiler.* Finally, using polynomial commitment schemes and Fiat-Shamir heuristic, the previous checks can be compiled into an efficient and non-interactive proof system. The resulting zk-SNARK has either universal updatable structured reference string, or transparent setup, depending only on the nature of the polynomial commitment scheme used in this step.

Remark that the only step where we employ cryptographic techniques is the final one. In this cryptographic compilation step the oracles from before are replaced by a suitable cryptographic realization: Polynomial Commitments (PC) [MBKM19, GKKB12, GWC19]. Polynomial commitments allow to check efficiently a series of identities on low-degree polynomials resulting from the arithmetization step. This comes at the expense of introducing computational hardness assumptions for security and sometimes a trusted setup.

Additionally, interaction is removed by employing the Fiat-Shamir heuristic (see Section 3) that allows for publicly computing the verifier's challenges using a hash function modeled as a random oracle.

We now investigate these techniques before giving example instantiations for this approach.

9.1 NP Characterization or Arithmetisation for Circuits

A key point in building SNARKs from polynomial commitments is the first step: the compilation of the computation to be proven, usually represented as a circuit into equations on polynomials. More generally, we can consider also program source code. Recent schemes propose different ways to represent this computation as a constraint system involving native operations over a finite field in order to optimise the polynomial equation that needs to be proven. The goal is to have a minimal number of polynomials and thus commitments and a minimal prover overhead due to this transformation.

Recent universal SNARKs use different techniques to achieve the arithmetization. We summarize the approach taken by 3 different influential approaches:

- Sonic [MBKM19] represents the circuit as three vectors of left, right, and output wires. Then, the consistency of both the multiplication gates and the linear constraints are reduced to one large polynomial equation in a variable Y. The equation is then embedded into the X^0 terms of a bivariate polynomial $t(X, Y)$. The verifier will check that the prover computed the bivariate polynomials from witness and circuit specific bivariate polynomials such that the X^0 terms cancel out. Using a PIOP, these bivariate polynomials can be simulated with univariate ones under some conditions.
- Marlin [CHM$^+$19] and Aurora [BSCR$^+$18] represent the arithmetic constraint system as Reed-Solomon codewords which are univariate polynomials. Then what is left to check are polynomial identities, representing the correct computation of a Hadamard product of such codewords.

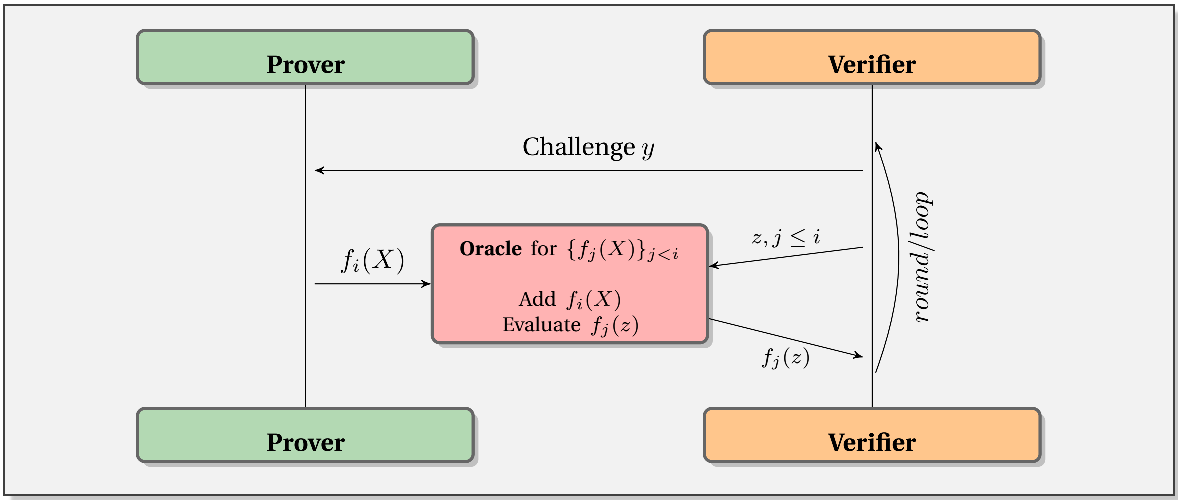


Figure 15: Simplified PIOP Protocol for round i: Prover computes new polynomial $f_i(X)$ from challenge y. Verifier asks any number of queries (z, j), $j \leq i$, to the Polynomial Evaluation Oracle to learn $f_j(z)$.

- PLONK [GWC19] represents the vector of wire values as well as the different gates selectors as polynomials using interpolation. A polynomial division check ensures that the prover knows satisfying inputs and outputs for each gate, but does not ensure a correct wiring, e.g. that the output of one gate is the input to another. A permutation argument establishes the consistency of the assignment of wires to gates.

9.2 Polynomial Interactive Oracle Proofs

Arithmetization translates a property or relation into an equivalent more efficiently provable property or relation. By itself it does however not give us an efficient proof system. A common next step is to consider an information-theoretic proof system that convinces a verifier of the truth of a statement even against a computationally unrestricted prover. Note that this system usually is either inefficient or relies on idealized components called oracles. An additional cryptographic compilation step will take care of that.

Here we consider information theretic proof systems that involve allow the verifier to query polynomials on inputs of its choice. The polynomials are chosen by the prover based on prior verifier messages. This approach for building SNARKs was discovered by at least two groups in parallel which referred to them as Polynomial Interactive Oracle Proofs (PIOP) [BFS19, **?**] and Algebraic Holographic Proofs (AHP) [CHM$^+$19].

We refer to them as Polynomial IOP and introduce them with some simplifications:

Definition 9.1 (Polynomial IOP)**.** *Let $\mathcal{R}$ be a relation, $\mathbb{F}$ a field, and $d \in \mathbb{N}$ a degree bound.*
A polynomial IOP is a pair of interactive prover and verifier algorithms (P, V).

- *(P, V) are a sound interactive proof system.*
- *in each round i the verifier sends a challenge y to the prover.*
- *in each round the prover P outputs a univariate polynomials $f_i(X)$ of degree at most d which is added to a list of polynomials maintained by an oracle.*
- *in each round the verifier can interact with polynomials $f_1, \ldots, f_i$ of this and previous rounds by repeatedly querying some $f_j(X)$, $j \leq i$ on values z of its choice.*
- *the verifier is public coin.*

9.3 Cryptographic Compiler employing Polynomial Commitments

As already mentioned, the key ingredient for building universal SNARKs in the PIOP framework are polynomials commitments.

9.3.1 Polynomial Commitments

Polynomial commitments (PCs) first introduced by [KZG10] are commitments for the message space $\mathbb{F}^{\leq d}[X]$, the ring of polynomials in X with maximum degree $d \in \mathbb{N}$ and coefficients in the field $\mathbb{F} = \mathbb{Z}_p$, that support an interactive argument of knowledge $(\mathsf{ComGen}, \mathsf{Open}, \mathsf{Check})$ for proving the correct evaluation of a committed polynomial at a given point without revealing any other information about the committed polynomial.

A polynomial commitment scheme over a field consists in 4 algorithms $\mathsf{PC} = (\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Check})$ defined as follows:

$\mathsf{ComGen}(1{\cdot}d) \to (\mathsf{ck}, \mathsf{vk})$**:** given a security parameter fixing a field $\mathbb{F}$ and a maximal degree d samples a group description gk, and commitment and verification keys $(\mathsf{ck}, \mathsf{vk})$. We implicitly assume ck and vk each contain gk.

$\mathsf{Com}(\mathsf{ck}; f(X)) \to C$**:** given ck and a polynomial $f(X) \in \mathbb{F}^{\leq d}[X]$ outputs a commitment C.

$\mathsf{Open}(\mathsf{ck}; C, x, y; f(X)) \to \pi$**:** given a commitment C, an evaluation point x, a value y, the polynomial $f(X) \in \mathbb{F}[X]$, it output a prove π for the relation:

$$\mathcal{R}_{\mathsf{kzg}} := \left\{ (\mathsf{ck}, C, x, y; f(X)) \ : \ \begin{array}{r} C = \mathsf{Com}\,(\mathsf{ck}; f(X)) \\ \wedge \ \deg(f(X)) \leq d \\ \wedge \ y = f(x) \end{array} \right\}$$

$\mathsf{Check}(\mathsf{vk}, C, x, y, \pi) \to 1/0$: Outputs 1 if the proof π verifies and 0 if π is not a valid proof for the opening (C, x, y).

A polynomial commitment satisfy computational knowledge binding property:

Definition 9.2 (Computational Knowledge Binding)**.** *For every* PPT *adversary $\mathcal{A}$ that produces a valid commitment-proof C, π that verifies, i.e. such that* $\mathsf{Check}(\mathsf{vk}, C, x, y, \pi) = 1$*, there is an extractor $\mathcal{E}_{\mathcal{A}}$ that is able to output a pre-image polynomial $f(X)$ with overwhelming probability:*

$$\Pr\left[C = \mathsf{Com}(\mathsf{ck}; f(X); r) \ \middle| \ \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{ComGen}(1^\lambda, d) \\ (C, \pi; f(X)) \leftarrow (\mathcal{A} \| \mathcal{E}_{\mathcal{A}})(\mathsf{ck}) \\ \mathsf{Check}(\mathsf{vk}, C, x, y, \pi) = 1 \end{array} \right] = 1 - \mathsf{negl}.$$

9.3.2 Generic SNARKs Compiler from PIOP and Polynomial Commitments

Sonic [MBKM19], Marlin [CHM+19] and Plonk [GWC19] are using variants of KZG [] described below, therefore they are pairing-based SNARKs and rely on trusted setup.

Other recent works are looking for different such polynomial commitment schemes that do not require setup for the keys in order to obtain transparent SNARK schemes.

Some of the SNARKs avoiding trusted setup:

- DARK [BFS19] building polynomial commitments schemes from groups of unknown order (such as RSA –with setup, or class groups –without setup)
- Halo [BGH19] which uses ideas from [**?**] and applying them to polynomial commitments to enable verification without using pairings, but at the cost of some overhead that can be amortised in batch verification.

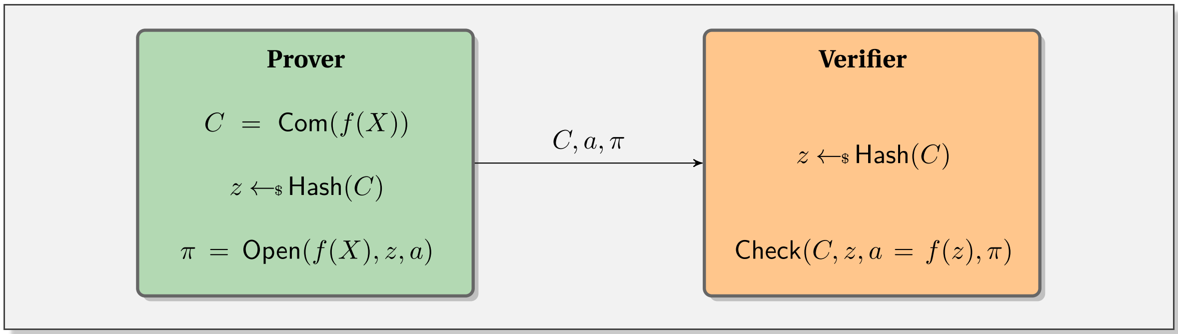


Figure 16: Compilation from PIOP to universal SNARK using Polyomial Commitments.

The following compiler is adapted from [BFS19]. We consider only the uni-variate case. Generalizations to multivariate PIOP exist.

- Compute $(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{ComGen}(1^{,}d)$.
- In any round in which the PIOP prover adds a degree d-degree polynomial f_i to $\mathcal{O}$, send a commitment $C_i \leftarrow \mathsf{Com}(\mathsf{ck}; f_i(X))$ to the verifier instead.
- In any round in which the PIOP verifier requests an evaluation of polynomial f_j on value z, the prover computes and sends $a \leftarrow f_j(z)$, $\pi \leftarrow \mathsf{Open}(\mathsf{ck}; C_j, z, a; f_j(X))$ and the verifier checks that

$$\mathsf{Check}(\mathsf{vk}, C_j, z, a, \pi) = 1 \ .$$

In a further step the above public-coin protocol is made non-interactive by computing the verifiers challenges using a hash function Hash as depicted in Figure 16.

9.3.3 KZG Polynomial Commitment

An example of a polynomial commitment scheme that is used to build universal SNARKs is the KZG scheme in [KZG10]. This is a pairing-based construction that allows for openings of evaluations on a point with a single group element-sized proof of correct opening and a constant time verifier for this check:

More precisely, $\mathsf{KZG.PC} = (\mathsf{KZG.ComGen}, \mathsf{KZG.Com}, \mathsf{KZG.Open}, \mathsf{KZG.Check})$ is defined over bilinear groups $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$ as follows:

$\mathsf{KZG.ComGen}(1^\lambda, n) \to (\mathsf{ck}_{\mathsf{kzg}}, \mathsf{vk}_{\mathsf{kzg}})$: Set keys $\mathsf{ck}_{\mathsf{kzg}} = \{g^{\alpha^i}\}_{i=0}^{n-1}$, $\mathsf{vk}_{\mathsf{kzg}} = h^\alpha$.

$\mathsf{KZG.Com}(\mathsf{ck}_{\mathsf{kzg}}; f(X)) \to C_f$: For $f(X) = \sum_{i=0}^{n-1} f_i X^i$, computes $C_f = \prod_{i=0}^{n-1} g^{f_i \alpha^i} = g^{f(\alpha)}$.

$\mathsf{KZG.Open}(\mathsf{ck}_{\mathsf{kzg}}; C_f, x, y; f(X)) \to \pi$: For an evaluation point x, a value y, compute the quotient polynomial $q(X) = \dfrac{f(X) - y}{X - x}$ and output a proof $\pi = C_q = \mathsf{KZG.Com}(\mathsf{ck}_{\mathsf{kzg}}; q(X))$.

$\mathsf{KZG.Check}(\mathsf{vk}_{\mathsf{kzg}} = h^\alpha, C_f, x, y, \pi) \to 1/0$: Check if $e(C_f \cdot g^{-y}, h) = e(C_q, h^\alpha \cdot h^{-x})$.

References

[AF07] Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 118–136. Springer, Heidelberg, February 2007.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM CCS 17*, pages 2087–2104. ACM Press, 2017.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In Gary L. Miller, editor, *STOC*, pages 99–108. ACM, 1996.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[Bab85] László Babai. Trading group theory for randomness. In *17th ACM STOC*, pages 421–429. ACM Press, May 1985.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd FOCS*, pages 106–115. IEEE Computer Society Press, October 2001.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, 2018.

[BBC+18] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO (2)*, volume 10992 of *Lecture Notes in Computer Science*, pages 669–699. Springer, 2018.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, October 1988.

[BCC+14] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. Cryptology ePrint Archive, Report 2014/580, 2014. http://eprint.iacr.org/2014/580.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

[BCG+13a] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013.

[BCG+13b] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for c: Verifying program executions succinctly and in zero knowledge. Cryptology ePrint Archive, Report 2013/507, 2013. http://eprint.iacr.org/2013/507.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th ACM STOC*, pages 505–514. ACM Press, May / June 2014.

[BDG+13] Nir Bitansky, Dana Dachman-Soled, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, Adriana López-Alt, and Daniel Wichs. Why "Fiat-Shamir for proofs" lacks a proof. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 182–201. Springer, Heidelberg, March 2013.

[BDP00] Joan Boyar, Ivan Damgård, and René Peralta. Short non-interactive cryptographic proofs. *Journal of Cryptology*, 13(4):449–472, September 2000.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

[BFS19] Benedikt Bünz, B. Fisch, and Alan Szepieniec. Transparent snarks from dark compilers, 2019.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. https://eprint.iacr.org/2019/1021.

[BHZ87] Ravi B. Boppana, Johan Hastad, and Stathis Zachos. Does co-np have short interactive proofs? *Information Processing Letters*, 25(2):127 – 132, 1987.

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based SNARGs and their application to more efficient obfuscation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 247–277. Springer, Heidelberg, May 2017.

[BISW18] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Quasi-optimal snargs via linear multi-prover interactive proofs. Cryptology ePrint Archive, Report 2018/133, 2018. https://eprint.iacr.org/2018/133.

[BLV03] Boaz Barak, Yehuda Lindell, and Salil P. Vadhan. Lower bounds for non-black-box zero knowledge. In *44th FOCS*, pages 384–393. IEEE Computer Society Press, October 2003.

[BP15] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 236–261. Springer, Heidelberg, November / December 2015.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[BSCR+18] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for r1cs. Cryptology ePrint Archive, Report 2018/828, 2018. https://eprint.iacr.org/2018/828.

[BSW12] Dan Boneh, Gil Segev, and Brent Waters. Targeted malleability: homomorphic encryption for restricted computations. In Shafi Goldwasser, editor, *ITCS 2012*, pages 350–366. ACM, January 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[CCH+18] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum. Fiat-shamir from simpler assumptions. *IACR Cryptology ePrint Archive*, 2018:1004, 2018.

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. Cryptology ePrint Archive, Report 2018/131, 2018. https://eprint.iacr.org/2018/131.

[CD09] Ran Canetti and Ronny Ramzi Dakdouk. Towards a theory of extractable functions. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 595–613. Springer, Heidelberg, March 2009.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[CHM+19] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. Cryptology ePrint Archive, Report 2019/1047, 2019. https://eprint.iacr.org/2019/1047.

[CL08] Giovanni Di Crescenzo and Helger Lipmaa. Succinct np proofs from an extractability assumption. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *CiE*, volume 5028 of *Lecture Notes in Computer Science*, pages 175–185. Springer, 2008.

[CLW18] Ran Canetti, Alex Lombardi, and Daniel Wichs. Non-interactive zero knowledge and correlation intractability from circular-secure fhe. *IACR Cryptology ePrint Archive*, 2018:1248, 2018.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999.

[COS19] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. Cryptology ePrint Archive, Report 2019/1076, 2019. https://eprint.iacr.org/2019/1076.

[Dam92] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456. Springer, Heidelberg, August 1992.

[Dam93] Ivan Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with proprocessing. In Rainer A. Rueppel, editor, *EUROCRYPT'92*, volume 658 of *LNCS*, pages 341–355. Springer, Heidelberg, May 1993.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430. Springer, Heidelberg, May 2000.

[DFGK14] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 532–550. Springer, Heidelberg, December 2014.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 54–74. Springer, Heidelberg, March 2012.

[DMP90] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge with preprocessing. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 269–282. Springer, Heidelberg, August 1990.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990.

[FN16] Dario Fiore and Anca Nitulescu. On the (in)security of SNARKs in the presence of oracles. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 108–138. Springer, Heidelberg, October / November 2016.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Information Processing Letters*, 67(4):205 – 214, 1998.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003.

[GKKB12] Aditi Gupta, Sam Kerr, Michael S. Kirkpatrick, and Elisa Bertino. Marlin: making it harder to fish for gadgets. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 1016–1018. ACM Press, October 2012.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier CS-Proofs. Cryptology ePrint Archive, Report 2011/456, 2011. `http://eprint.iacr.org/2011/456`.

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. LNCS, pages 581–612. Springer, Heidelberg, 2017.

[GMNO18] Rosario Gennaro, Michele Minelli, Anca Nitulescu, and Michele Orrù. Lattice-based zk-snarks from square span programs. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM Conference on Computer and Communications Security*, pages 556–573. ACM, 2018.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. pages 1069–1083, 2016.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*, pages 174–187. IEEE Computer Society Press, October 1986.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994.

[GOS06a] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Heidelberg, August 2006.

[GOS06b] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 803–815. Springer, Heidelberg, July 2005.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

[GVW02] Oded Goldreich, Salil Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *computational complexity*, 11(1-2):1–53, 2002.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptology ePrint Archive*, 2019:953, 2019.

[HL18] Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions. Cryptology ePrint Archive, Report 2018/385, 2018. https://eprint.iacr.org/2018/385.

[HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 408–423. Springer, Heidelberg, August 1998.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

[KP98] Joe Kilian and Erez Petrank. An efficient noninteractive zero-knowledge proof system for NP with general assumptions. *Journal of Cryptology*, 11(1):1–27, January 1998.

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. LNCS, pages 224–251. Springer, Heidelberg, 2017.

[KW93] M. Karchmer and A. Wigderson. On span programs. In IEEE Computer Society Press, editor, *In Proc. of the 8th IEEE Structure in Complexity Theory*, pages 102–111, Gaithersburg, MD, USA, 1993. IEEE Computer Society Press.

[KW18] Sam Kim and David J. Wu. Multi-theorem preprocessing nizks from lattices. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO (2)*, volume 10992 of *Lecture Notes in Computer Science*, pages 733–765. Springer, 2018.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[KZM+15] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. How to use SNARKs in universally composable protocols. Cryptology ePrint Archive, Report 2015/1093, 2015. http://eprint.iacr.org/2015/1093.

[Lip05] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC 2005*, volume 3650 of *LNCS*, pages 314–328. Springer, Heidelberg, September 2005.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Heidelberg, December 2013.

[LLNW18] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Lattice-based zero-knowledge arguments for integer relations. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO (2)*, volume 10992 of *Lecture Notes in Computer Science*, pages 700–732. Springer, 2018.

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *ACM CCS 19*, pages 2111–2128. ACM Press, 2019.

[Mer79] Ralph Charles Merkle. *Secrecy, authentication and public key systems.* PhD thesis, Stanford University, June 1979.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004.

[Nao03a] Moni Naor. On cryptographic assumptions and challenges. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

[Nao03b] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Heidelberg, August 2003.

[Nit19] Anca Nitulescu. *A tale of SNARKs: quantum resilience, knowledge extractability and data privacy.* Theses, Ecole Normale Supérieure de Paris - ENS Paris, April 2019.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

[PPY19] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Leakage cell probe model: Lower bounds for key-equality mitigation in encrypted multi-maps. Cryptology ePrint Archive, Report 2019/1132, 2019. https://eprint.iacr.org/2019/1132.

[SBV+12] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. Cryptology ePrint Archive, Report 2012/622, 2012. http://eprint.iacr.org/2012/622.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

[Set19] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. Cryptology ePrint Archive, Report 2019/550, 2019. https://eprint.iacr.org/2019/550.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

[Sho99] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.

[Wee05] Hoeteck Wee. On round-efficient argument systems. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 140–152. Springer, Heidelberg, July 2005.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Xiaodong Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *IACR Cryptology ePrint Archive*, 2019.