1

Introduction

This monograph is about verifiable computing (VC). VC refers to cryptographic protocols called interactive proofs (IPs) and arguments that enable a prover to provide a guarantee to a verifier that the prover performed a requested computation correctly. Introduced in the 1980s, IPs and arguments represented a major conceptual expansion of what constitutes a "proof" that a statement is true. Traditionally, a proof is a static object that can be easily checked step-by-step for correctness, because each individual step of the proof should be trivial to verify. In contrast, IPs allow for interaction between prover and verifier, as well as a tiny but nonzero probability that an invalid proof passes verification. The difference between IPs and arguments is that arguments (but not IPs) permit the existence of "proofs" of incorrect statements, so long as those "proofs" require exorbitant computational power to find.[1]

Celebrated theoretical results from the mid-1980s and early 1990s indicated that VC protocols can, at least in principle, accomplish amazing feats. These include enabling a cell phone to monitor the execution of a powerful but untrusted (even malicious) supercomputer, enabling

[1]For example, an argument, but not an IP, might make use of a cryptosystem, such that it is possible for a cheating prover to find a convincing "proof" of a false statement if (and only if) the prover can break the cryptosystem.

computationally weak peripheral devices (e.g., security card readers) to offload security-critical work to powerful remote servers, or letting a mathematician obtain a high degree of confidence that a theorem is true by looking at only a few symbols of a purported proof.[2]

VC protocols can be especially useful in cryptographic contexts when they possess a property called *zero-knowledge*. This means that the proof or argument reveals nothing but its own validity.

To give a concrete sense of why zero-knowledge protocols are useful, consider the following quintessential example from authentication. Suppose that Alice chooses a random password x and publishes a hash $z = h(x)$, where h is a one-way function. This means that given $z = h(x)$ for a randomly chosen x, enormous computational power should be required to find a preimage of z under h, i.e., an x' such that $h(x') = z$. Later, suppose that Alice wants to convince Bob that she is the same person who published z. She can do this by proving to Bob that she knows an x' such that $h(x') = z$. This will convince Bob that Alice is the same person who published z, since it means that either Alice knew x to begin with, or she inverted h (which is assumed to be beyond the computational capabilities of Alice).

How can Alice convince Bob that she knows a preimage of z under h? A trivial proof is for Alice to send x to Bob, and Bob can easily check that $h(x) = z$. But this reveals much more information than that Alice knows a preimage of z. In particular it reveals the preimage itself. Bob can use this knowledge to impersonate Alice forevermore, since now he too knows the preimage of z.

In order to prevent Bob from learning information that can compromise the password x, it is important that the proof reveals nothing beyond its own validity. This is exactly what the zero-knowledge property guarantees.

A particular goal of this survey is to describe a variety of approaches to constructing so-called zero-knowledge Succinct Non-interactive Arguments of Knowledge, or zk-SNARKs for short. "Succinct" means that the proofs are short. "Non-interactive" means that the proof is

[2]So long as the proof is written in a specific, mildly redundant format. See our treatment of *probabilistically checkable proofs* (PCPs) in Section 9.

static, consisting of a single message from the prover. "Of Knowledge" roughly means that the protocol establishes not only that a statement is true, but also that the prover *knows* a "witness" to the veracity of the statement.[3] Argument systems satisfying all of these properties have a myriad of applications throughout cryptography.

Practical zero-knowledge protocols for highly specialized statements of cryptographic relevance (such as proving knowledge of a discrete logarithm [223]) have been known for decades. However, general-purpose zero-knowledge protocols have only recently become plausibly efficient enough for cryptographic deployment. By general-purpose, we mean protocol design techniques that apply to arbitrary computations. This exciting progress has involved the introduction of beautiful new protocols, and brought a surge of interest in zero-knowledge proofs and arguments. This survey seeks to make accessible, in a unified manner, the main ideas and approaches to the design of these protocols.

Background and Context. In the mid-1980s and 1990s, theoretical computer scientists showed that IPs and arguments can be vastly more efficient (at least, in an asymptotic sense) than traditional **NP** proofs,[4] which are static and information-theoretically secure.[5] The foundational results characterizing the power of these protocols (such as **IP** = **PSPACE** [186], [231], **MIP** = **NEXP** [17], and the PCP theorem [10], [11]) are some of the most influential and celebrated in computational complexity theory.[6]

Despite their remarkable asymptotic efficiency, general-purpose VC protocols were long considered wildly impractical, and with good reason: naive implementations of the theory would have had comically high

[3]For example, the authentication scenario above really requires a zero-knowledge proof *of knowledge* for the statement "there exists a password x such that $h(x) = z$". This is because the application requires that Bob be convinced not just of the fact that there *exists* a preimage x of z under h (which will always be true if h is a surjective function), but also that Alice knows x.

[4]We formally define notions such as **NP** and **IP** in Section 3.3.

[5]The term information-theoretically secure here refers to the fact that **NP** proofs (like IPs, but unlike arguments) are secure against computationally unbounded provers.

[6]The results **IP** = **PSPACE** and **MIP** = **NEXP** are both covered in this survey (see Sections 4.5.5 and 8.5 respectively).

concrete costs (trillions of years for the prover, even for very short computations). But the last decade has seen major improvements in the costs of VC protocols, with a corresponding jump from theory to practice. Even though implementations of general-purpose VC protocols remain somewhat costly (especially for the prover), paying this cost can often be justified if the VC protocol is zero-knowledge, since zero-knowledge protocols enable applications that may be totally impossible without them. Moreover, emerging applications to public blockchains have elevated the importance of proving relatively simple statements, on which it is feasible to run modern VC protocols despite their costs.

Approaches to Zero-Knowledge Protocol Design, and Philosophy of This Survey. Argument systems are typically developed in a two-step process. First, an information-theoretically secure protocol, such as an IP, *multi-prover interactive proof* (MIP), or *probabilistically checkable proof* (PCP), is developed for a model involving one or more provers that are assumed to behave in some restricted manner (e.g., in an MIP, the provers are assumed not to send information to each other about the challenges they receive from the verifier). Second, the information-theoretically secure protocol is combined with cryptography to "force" a (single) prover to behave in the restricted manner, thereby yielding an argument system. This second step also often endows the resulting argument system with important properties, such as zero-knowledge, succinctness, and non-interactivity. If the resulting argument satisfies all of these properties, then it is in fact a zk-SNARK.

By now, there are a variety promising approaches to developing efficient zk-SNARKs, which can be categorized by the type of information-theoretically secure protocol upon which they are based. These include (1) IPs, (2) MIPs, (3) PCPs, or more precisely a related notion called *interactive oracle proofs* (IOPs), which is a hybrid between an IP and a PCP, and (4) *linear PCPs*. Sections 1.2.1–1.2.3 below give a more detailed overview of these models. This survey explains in a unified manner how to design efficient protocols in all four information-theoretically secure models, emphasizing commonalities between them.

IPs, MIPs, and PCPs/IOPs can all be transformed into succinct interactive arguments by combining them with a cryptographic primitive called a *polynomial commitment scheme*; the interactive arguments can then be rendered non-interactive and publicly verifiable by applying a cryptographic technique called the *Fiat-Shamir transformation* (Section 5.2), yielding a SNARK. Transformations from linear PCPs to arguments are somewhat different, though closely related to certain polynomial commitment schemes. As with the information-theoretically secure protocols themselves, this survey covers these cryptographic transformations in a unified manner.

Because of the two-step nature of zk-SNARK constructions, it is often helpful to first understand proofs and arguments *without* worrying about zero-knowledge, and then at the very end understand how to achieve zero-knowledge as an "add on" property. Accordingly, we do not discuss zero-knowledge until relatively late in this survey (Section 11). Earlier sections are devoted to describing efficient protocols in each of the information-theoretically secure models, and explaining how to transform them into succinct arguments.

By now, zk-SNARKs have been deployed in a number of real-world systems, and there is a large and diverse community of researchers, industry professionals, and open source software developers working to improve and deploy the technology. This survey assumes very little formal mathematical background—mainly comfort with modular arithmetic, some notions from the theory of finite fields and groups, and basic probability theory—and is intended as a resource for anyone interested in verifiable computing and zero-knowledge. However, it does require significant mathematical maturity and considerable comfort with theorems and proofs. Also helpful (but not strictly necessary) is knowledge of standard complexity classes like **P** and **NP**, and complexity-theoretic notions such as **NP**-completeness.

Ordering of Information-Theoretically Secure Models in This Survey. We first cover IPs, then MIPs, then PCPs and IOPs, then linear PCPs. This ordering roughly follows the chronology of the models' introduction to the research literature. Perhaps ironically, the models have been applied to practical SNARK design in something resembling *reverse*

chronological order. For example, the first practical SNARKs were based on linear PCPs. In fact, this is not a coincidence: a primary motivation for introducing linear PCPs in the first place was the goal of obtaining simpler and more practical succinct arguments, and specifically the *impracticality* of arguments derived from PCPs.

Section-by-section Outline. Section 2 familiarizes the reader with randomness and the power of probabilistic proof systems, through two easy but important case studies. Section 3 introduces technical notions that will be useful throughout the survey. Sections 4 describes state-of-the-art interactive proofs. Section 5 describes the Fiat-Shamir transformation, a key technique that is used to remove interaction from cryptographic protocols. Section 7 introduces the notion of a polynomial commitment scheme, and combines it with the IPs of Section 4 and the Fiat-Shamir transformation of Section 5 to obtain the first SNARK covered in the survey. Section 8 describes state-of-the-art MIPs and SNARKs derived thereof. Sections 9–10 describe PCPs and IOPs, and SNARKs derived thereof.

Section 6 is a standalone section describing techniques for representing computer programs in formats amenable to application of such SNARKs.

Section 11 introduces the notion of zero-knowledge. Section 12 describes a particularly simple type of zero-knowledge argument called Σ-protocols, and uses them to derive commitment schemes. These commitment schemes serve as important building blocks for more complicated protocols covered in subsequent sections. Section 13 describes efficient techniques for transforming non-zero-knowledge protocols into zero-knowledge ones. Sections 14–16 cover practical polynomial commitment schemes, which can be used to turn any IP, MIP, or IOP into a succinct zero-knowledge argument of knowledge (zkSNARK). Section 17 covers our final approach to designing zkSNARKs, namely through linear PCPs. Section 18 describes how to recursively compose SNARKs to improve their costs and achieve important primitives such as so-called *incrementally verifiable computation*. Finally, Section 19 provides a taxonomy of design paradigms for practical zkSNARKs, and delineates the pros and cons of each approach.

Suggestions for Reading the Monograph. The monograph may happily be read from start to finish, but non-linear paths may offer a faster route to a big-picture understanding of SNARK design techniques. Suggestions to this effect are as follows.

Sections 2 and 3 introduce basic technical notions used throughout all subsequent sections (finite fields, IPs, arguments, low-degree extensions, the Schwartz-Zippel lemma, etc.), and should not be skipped by readers unfamiliar with these concepts.

Readers may next wish to read the *final* section, Section 19, which provides a birds-eye view of all SNARK design approaches and how they relate to each other. Section 19 uses some terminology that may be unfamiliar to the reader at this point, but it should nonetheless be understandable and it provides context that is helpful to have in mind when working through more technical sections.

After that, there are many possible paths through the monograph. Readers specifically interested in the SNARKs that were the first to be deployed in commercial settings can turn to Section 17 on linear PCPs. This section is essentially self-contained but for its use of pairing-based cryptography that is introduced in Section 15.1 (and, at the very end, its treatment of zero-knowledge, a concept introduced formally in Section 11).

Otherwise, readers should turn to understanding the alternative approach to SNARK design, namely to combine a *polynomial IOP* (of which IPs, MIPs, and PCPs are special cases) with a *polynomial commitment scheme*.

To quickly understand polynomial IOPs, we suggest a careful reading of Section 4.1 on the sum-check protocol, followed by Section 4.6 on the GKR interactive proof protocol for circuit evaluation, or Section 8.2 giving a 2-prover MIP for circuit satisfiability. Next, the reader can turn to Section 7, which explains how to combine such protocols with polynomial commitments to obtain succinct arguments.

To understand polynomial commitment schemes, the reader can either tackle Sections 10.4 and 10.5 to understand IOP-based polynomial commitments, or instead turn to Sections 12 and 14–16 (in that order) to understand polynomial commitments based on the discrete logarithm problem and pairings.

A compressed overview of polynomial IOPs and polynomial commitments is provided in a sequence of three talk videos posted on this monograph's webpage.[7] Readers may find it useful to watch these videos prior to a detailed reading of Sections 4–10.

Material That can be Skipped on a First Reading. Sections 4.2–4.5 are devoted to detailed example applications of the sum-check protocol and explaining how to efficiently implement the prover within it. While these sections contain interesting results and are useful for familiarizing oneself with the sum-check protocol, subsequent sections do not depend on them. Similarly, Section 5 on the Fiat-Shamir transformation and Section 6 on front-ends are optional on a first reading. Sections 9.3 and 9.4 provide PCPs that are mainly of historical interest and can be skipped.

Sections 11 and 13 offer treatments of zero-knowledge that largely stand on their own. Similarly, Section 18 discusses SNARK composition and stands on its own.

1.1 Mathematical Proofs

This survey covers different notions of *mathematical proofs* and their applications in computer science and cryptography. Informally, what we mean by a proof is anything that convinces someone that a statement is true, and a "proof system" is any procedure that decides what is and is not a convincing proof. That is, a proof system is specified by a verification procedure that takes as input any statement and a claimed "proof" that the statement is true, and decides whether or not the proof is valid.

What properties do we want in a proof system? Here are four obvious ones.

- Any true statement should have a convincing proof of its validity. This property is typically referred to as *completeness*.
- No false statement should have a convincing proof. This property is referred to as *soundness*.

[7] https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html.

- Ideally, the verification procedure will be "efficient". Roughly, this means that simple statements should have short (convincing) proofs that can be *checked* quickly.
- Ideally, proving should be efficient too. Roughly, this means that simple statements should have short (convincing) proofs that can be *found* quickly.

Traditionally, a mathematical proof is something that can be written and checked line-by-line for correctness. This traditional notion of proof is precisely the one captured by the complexity class **NP**.[8] However, over the last 30+ years, computer scientists have studied much more general and exotic notions of proofs. This has transformed computer scientists' notions of what it means to prove something, and has led to major advances in complexity theory and cryptography.

1.2 What Kinds of Non-Traditional Proofs Will We Study?

All of the notions of proofs that we study in this survey will be probabilistic in nature. This means that the verification procedure will make random choices, and the soundness guarantee will hold with (very) high probability over those random choices. That is, there will be a (very) small probability that the verification procedure will declare a false statement to be true.

1.2.1 Interactive Proofs (IPs)

To understand what an interactive proof is, it is helpful to think of the following application. Imagine a business (verifier) that is using a commercial cloud computing provider to store and process its data. The business sends all of its data up to the cloud (prover), which stores it, while the business stores only a very small "secret" summary of the data (meaning that the cloud does not know the user's secret summary). Later, the business asks the cloud a question about its data, typically

[8] Roughly speaking, the complexity class **NP** contains all problems for which the correct answer on any input is either YES or NO, and for all YES instances, there is an efficiently-checkable (traditional) proof that the correct answer is YES. See Section 3.3 for details.

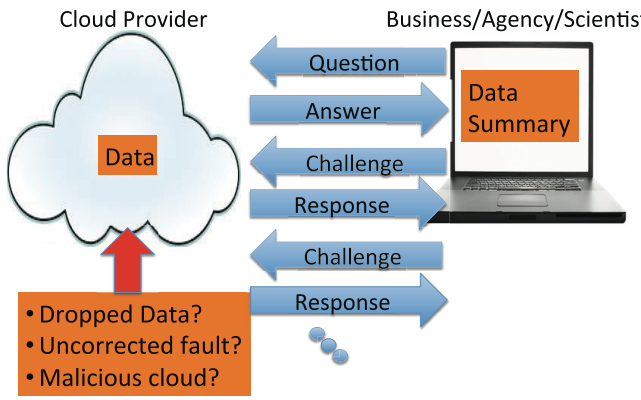

Figure 1.1: Depiction of an interactive proof or argument used to check that a cloud computing provider is storing and processing a user's data correctly.

in the form of a computer program f that the business wants the cloud to run on its data using the cloud's vast computing infrastructure. The cloud does so, and sends the user the claimed output of the program, $f(\mathsf{data})$. Rather than blindly trust that the cloud executed the program on the data correctly, the business can use an interactive proof system (IP) to obtain a formal *guarantee* that the claimed output is correct.

In the IP, the business interrogates the cloud, sending a sequence of challenges and receiving a sequence of responses. At the end of the interrogation, the business must decide whether to accept the answer as valid or reject it as invalid. See Figure 1.1 for a diagram of this interaction.

Completeness of the IP means that if the cloud correctly runs the program on the data and follows the prescribed protocol, then the user will be convinced to accept the answer as valid. Soundness of the IP means that if the cloud returns the wrong output, then the user will reject the answer as invalid with high probability *no matter how hard the cloud works to trick the user* into accepting the answer as valid. Intuitively, the interactive nature of the IP lets the business exploit the element of surprise (i.e., the fact that the cloud cannot predict the business's next challenge) to catch a lying cloud in a lie.

It is worth remarking on an interesting difference between IPs and traditional static proofs. Static proofs are *transferrable*, meaning that if

Peggy (prover) hands Victor (verifier) a proof that a statement is true, Victor can turn around and convince Tammy (a third party) that the same statement is true, simply by copying the proof. In contrast, an interactive proof may not be transferrable. Victor can try to convince Tammy that the statement is true by sending Tammy a transcript of his interaction with Peggy, but Tammy will not be convinced unless Tammy trusts that Victor correctly represented the interaction. This is because soundness of the IP only holds if, every time Peggy sends a response to Victor, Peggy does not know what challenge Victor will respond with next. The transcript alone does not give Tammy a guarantee that this holds.

1.2.2 Argument Systems

Argument systems are IPs, but where the soundness guarantee need only hold against cheating provers that run in polynomial time.[9] Argument systems make use of cryptography. Roughly speaking, in an argument system a cheating prover cannot trick the verifier into accepting a false statement unless it breaks some cryptosystem, and breaking the cryptosystem is assumed to require superpolynomial time.

1.2.3 Multi-Prover Interactive Proofs, Probabilistically Checkable Proofs, etc.

An MIP is like an IP, except that there are multiple provers, and these provers are assumed not to share information with each other regarding what challenges they receive from the verifier. A common analogy for MIPs is placing two or more criminal suspects in separate rooms before interrogating them, to see if they can keep their story straight. Law enforcement officers may be unsurprised to learn that the study of MIPs has lent theoretical justification to this practice. Specifically, the study of MIPs has revealed that if one locks the provers in separate rooms and then interrogates them separately, they can convince their

[9] Roughly speaking, this means that if the input has size n, then the prover's runtime (for sufficiently large values of n) should be bounded above by some constant power of n, e.g., n^{10}.

interrogators of much more complicated statements than if they are questioned together.

In a PCP, the proof is static as in a traditional mathematical proof, but the verifier is only allowed to read a small number of (possibly randomly chosen) characters from the proof.[10] This is in analogy to a lazy referee for a mathematical journal, who does not feel like painstakingly checking the proofs in a submitted paper for correctness. The PCP theorem [10], [11] essentially states that *any* traditional mathematical proof can be written in a format that enables this lazy reviewer to obtain a high degree of confidence in the validity of the proof by inspecting just a few words of it.

Philosophically, MIPs and PCPs are extremely interesting objects to study, but they are not directly applicable in most cryptographic settings, because they make unrealistic or onerous assumptions about the prover(s). For example, soundness of any MIP only holds if the provers do not share information with each other regarding what challenges they receive from the verifier. This is not directly useful in most cryptographic settings, because typically in these settings there is only a single prover, and even if there is more than one, there is no way to force the provers not to communicate. Similarly, although the verifier only reads a few characters of a PCP, a direct implementation of a PCP would require the prover to transmit the whole proof to the verifier, and this would be the dominant cost in most real-world scenarios (the example of a lazy journal referee notwithstanding). That is, once the prover transmits the whole proof to the verifier, there is little real-world benefit to having the verifier avoid reading the whole proof.

However, by combining MIPs and PCPs with cryptography, we will see how to turn them into argument systems, and these *are* directly applicable in cryptographic settings. For example, we will see in Section 9.2 how to turn a PCP into an argument system in which the prover does *not* have to send the whole PCP to the verifier.

[10] More precisely, a PCP verifier is allowed to read as much of the proof as it wants. However, for the PCP to be considered efficient, it must be the case that the verifier only needs to read a tiny fraction of the proof to ascertain with high confidence whether or not the proof is valid.

Section 10.2 of this survey in fact provides a unifying abstraction, called *polynomial IOPs*, of which all of the IPs, MIPs, and PCPs that we cover are a special case. It turns out that any polynomial IOP can be transformed into an argument system with short proofs, via a cryptographic primitive called a polynomial commitment scheme.