Information Assurance Platform: Computational Verification and Privacy

team@iap.network

Abstract

This document explains the computational verification techniques utilised within the Information Assurance Platform (IAP). Computational integrity and privacy (CIP) is predominantely required with regards to off-chain computations, although is not limited to this use case. In such settings a client wants to prove that the result of the computation performed off-chain, in a non-observable manner, is correct. The incentive for performing off-chain computations may be to preserve privacy, but is more commonly for reasons of efficiency. Off-chain computations do not need to be verified by the whole blockchain. In order to verify the correctness of the computation whilst preserving the privacy (i.e. not to reveal information about the computation), techniques such as a zero-knowledge proof can be used. In this document we elaborate on zero-knowledge proofs that are relevant to the IAP. Concretely, we describe a specific type of a zero-knowledge proof i.e. ZK-STARKs --- ZK-STARKs are utilised within the IAP to ensure computational verification, integrity and privacy of computations.

Index

Introduction

Anonymous transactions were popularized by the Bitcoin blockchain and have recently evolved into sophisticated tools layered with unique flexibility. The Ethereum ledger[A1] capabilities have aided the growth of anonymous transaction products like Zcash, JPMorgan's Quorum platform, and as of October 2018 Ernst & Young's EY Ops Chain Public Edition. Zero knowledge proofs are the cryptographic tool enabling each of these products, ensuring the privacy of their respective clients and client transactions. The IAP has partnered with several international organisations (e.g. BDO, EY Global) to collaborate on research and development of next generation applications of such technology.

Zero knowledge proofs (ZKP) are generic constructions that allow one to prove the validity of a statement without leaking any additional information. A ZKP can be used to prove one has a sufficient account balance to support a transaction without revealing the actual balance. ZKPs are also found in digital signatures, secret sharing, electronic voting, and other instances where privacy or anonymity are concerned. A ZKP provides computational integrity while preserving privacy (CIP). The IAP is most concerned with the ability of ZKP to provide transparency while retaining the quality of privacy.

This is of utmost importance in off-chain computations. Off-chain computations improve the efficiency of blockchain computations and save costs associated with a computation. An off-chain computation can be performed on a side or private chain; examples of which are integral to the IAP[A2]. While this can improve efficiency, it leads to the problem of verifiability of the result of said computations. ZKPs are used to ensure the integrity of the result reported while keeping the off-chain computations private.

Other more practical examples of ZKPs comprise the following: the IAP uses ZKPs to drive one of its four main tools in the Information Assurance Toolbelt (IAT); CyberShields[A3]. EY's prototype specifically uses ZKPs to verify token exchange between private clients while maintaining consensus. StarkWare, an emerging company with specific interests in ZKPs on blockchains, raised $30M[1] in funding as of October 2018. ING Bank announced its Zero Knowledge Range Proof[2] solution in Amsterdam at the Enterprise Ethereum Alliance Event. These companies are poised to lead a new wave of technological advancement fueled by ZKPs.

While ZKPs are valuable tools used to ensure statement integrity, the proof sizes can be large in practice. ZK-STARKs are a special category of non-interactive zero knowledge proofs designed so that verification of a proof is succinct. This emerging technology comes as a successor to ZK-SNARKs, another special category of non-interactive ZKPs. The core technology of Zcash is currently based on ZK-SNARKs which provide succinct, non-interactive proofs of knowledge. The ZK-SNARK creators have improved upon the original design principal which includes a trusted set-up phase. The set-up phase could jeopardize the long-term security of ZK-SNARK proofs if secret values used to generate the set-up were ever leaked. To move toward fully trustless applications ZK-STARKs have been developed. ZK-STARKs avoid a trusted set-up phase and are thus considered to be transparent.

In driving its platform for information assurance, cybersecurity and regulatory technology applications, the IAP includes tools that focus on relevant ZKPs; including ZK-STARKs. In this document, we define ZKPs, present a brief history of developments, and illustrate how they will be used in the IAP Network.

Preliminaries

In this section, we present definitions and notations of zero-knowledge proof systems as well as cover the necessary background information of the technologies utilised in zero-knowledge proofs. We state the following definitions for Proof of Knowledge systems (PoK); however, these definitions also hold for Zero-Knowledge Proof systems (ZKP). PoK systems can be seen as a similar type of ZKPs, whereby a ZKP has in some aspects stricter requirements (e.g. the zero-knowledge property).

Definitions and Notations of Zero-Knowledge Proofs

The Involved Parties: Peggy and Victor

There are typically two parties involved in a Proof of Knowledge (PoK) system: a prover (who we will refer to as Peggy) and a verifier (who we further call Victor). Although the reader is welcome to think of Peggy and Victor as people, Peggy and Victor are represented as Turing machines in the context of PoK systems. Note that a Turing machine is solely a mathematical abstraction of a computer. Fortunately, a PoK protocol generally works under both assumptions. Peggy and Victor may also have a specified 'level' of power: they may be all-powerful or may be computationally-bounded. By all-powerful the reader can understand an 'idealistic' case: a theoretical assumption that there exists no computational limitations (for example, with regards to the number of bits of storage). By computationally-bounded the reader can consider the real-world case: every computational device is in some way bounded by the number of bits it can process or store. These are assumptions over which a specific PoK system is defined.

Making a Statement

Peggy's objective in a PoK system is to convince Victor that her statement is correct while Victor wants to verify that Peggy's statement is correct. One statement mentioned previously might be Peggy's claim that her banking account balance exceeds a particular threshold. A more general example of a statement is: Peggy claims she 'has knowledge of a value v'. Victor wants to verify if Peggy indeed has knowledge of this value without requiring Peggy to confess v.

Informally, a statement x is 'something' that Peggy wants to prove to Victor. With 'something' we typically understand a decision problem: if x is 'true' then Peggy is able to convince Victor with high probability (formally, $\Pr(P(x,w) \leftrightarrow V(x) \rightarrow 1) >$

0.99). If a statement is true (we may say a statement is valid, correct or good) then the statement is included in a 'language' *L*. The language *L* contains all 'similar' statements $y \neq x$, which are defined over a well-defined structure.

To foster a better understanding of the above terms, we may phrase a statement in terms with a natural language (for example, English):

Natural language	**Formal**
x is an instance of an alphabet ∑	$a \in \{a,b,c,...,y,z\}$
a word is the collection of some finite elements of ∑ — in sequence	hello
∑* is the set of all words	{'hello',' hola',' hallo', . . . }
a language *L* is defined over ∑* more concretely, *L* is a subset of ∑*	{'hello',' blockchain',' proof', . . . } [English]

In software engineering, we are generally interested in solving algorithms. As such, let us represent these definitions in terms of a function: consider the output of a function b=f(a), where ***a*** is some secret input, *f*(*a*) is some sort of public function and ***b*** is a public output — ***b*** here is equal to the statement ***x*** above. The language ***L*** formally defines rules that statements must fulfill in order to be contained in the language. A formalization of the language ***L*** which corresponds to the statement "I have knowledge of a secret value whose output under public function *f* is ***b***" would be: $x \in L$ if x=f(a); *f*() defines the structure for language ***L***.

Statements are associated with a 'witness' ***w*** (where $w \in W$ and *W* denotes the set of all witnesses for statement *x*). A witness, in this context, is a piece of information that efficiently allows someone to verify the correctness (i.e. truth) of a statement. For example, if a statement claims "*n* is the product of two primes n=p*q" (the integer factorisation problem[3]), then a witness would be one of the prime numbers ***p*** or ***q***[4].

The input pair of statement and witness (*x*,*w*) to a PoK system is used to create a proof string ***π***. Thus, ***π*** is associated to the statement ***x*** where ***π*** acts as a certificate of the statement ***x***. Peggy sends ***π*** to Victor, who can then evaluate it and make a decision to either accept the proof ***π*** or to reject it.

Peggy has her private input ***a*** and wants to compute function ***b***=*f*(*a*). She wants to prove to Victor that ***b*** is indeed the output of *f*(*a*). By utilising a PoK system she makes the statement ***x*** (i.e. $x \in L$ which is equivalent to ***b***=*f*(*a*)) and the PoK system outputs a proof ***π*** for a witness ***w***. Peggy can then send ***π*** to Victor (either by publicly announcing ***π*** or solely distributing ***π*** to Victor for verification).

Notions of Proof Systems

Having defined the actors, formally defined a statement, and provided a PoK system overview, we define two notions which are indispensable to proof systems: the properties of completeness and soundness. Further, in PoK systems, the notion of validity extends the notion of soundness.

We always discuss the definition in natural language and then provide a formal definition. To set-up these notions we say a relation R is defined as follows:

A relation ***R*** is a tuple (*x*,*w*) where the instance ***x*** is an element of the language ***L*** and the witness ***w*** is an element of the set of all witnesses ***W***. Formally, this is stated as: $R = \{(x,w) : x \in L, w \in W\}$.

Definition: Completeness: A PoK system satisfies the completeness property if an honest prover (Peggy) is able to convince an honest verifier (Victor) about the truth of a correct statement ***x***.

Let $(x,w) \in R$. Then Peggy (*P*), with knowledge of witness ***w***, succeeds to convince Victor (*V*) about statement ***x*** of length ***n***. With probability Pr, it holds that $\Pr(P(x,w) \leftrightarrow V(x) \rightarrow 1) = c(n)$. $c(n)$ is typically 1, except otherwise stated.

Definition: Soundness: A PoK system satisfies the soundness property if a malicious prover, Pat, is not able to convince a honest verifier, Victor, about the truth of a false statement x.

Let $(x,w) \in R$. Then Pat (P'), without knowledge of witness w, should not be able to successfully convince Victor about the truth of a statement x' (x' denotes a 'false' statement — a statement where P' doesn't know w or a statement which is not included in the relation $x' \notin R$). As such, it holds that $\Pr(P'(x') \leftrightarrow V(x') \rightarrow 1) \leq s(n)$.

Given the previous defintions, in proof systems it is possible to satisfy the properties of completeness and soundness, yet, Peggy may not 'know' the witness w. As such, a stronger definition than soundness is the definition of validity. The notion of valitity is defined to prove 'knowledge' of a witness.

Definition: Validity: A PoK system satisfies the validity property if the success probability of a 'knowledge extractor' (E), in extracting the witness w, being able to interact with a poentially malicous prover, Pat (P'), must be at least as high as Pat (interacting with Victor), being able to convince V about the truth of the statement x.

Let $(x,w) \in R$, w be a witness, $W(x)$ be the set of all witnesses, κ denote the knowledge error and $E^{P(x)}$ denote oracle access of the knowledge extractor with the prover. Then the PoK system satisfies the validity property, if E given access to any prover P' the following statement holds: $\Pr(E^{P'(x)} \in W(x)) \geq \Pr(P'(x) \leftrightarrow V^*(x') \rightarrow 1) - \kappa(x)$.

Computational Complexity Classes

A PoK is typically associated with a computational complexity class. Complexity classes organize computational problems according to their hardness. The hardness of a problem is measured by the resources it takes to solve the problem. We quantify this in terms of time complexity and space complexity, where the former evaluates the execution time of solutions to the problem and the latter evaluates the memory used to solve the problem.

For PoK systems there exist multiple complexity classes. Two prominent classes are P and NP. P stands for polynomial time and includes all problems that are solvable in polynomial time. One may conceptualize a polynomial time algorithm as something a personal computer can process in less than a few hours. NP stands for non-deterministic polynomial time and includes problems that are not solvable in polynomial time but checkable in polynomial time. This means that given a solution to the problem, it can be verified in polynomial time whether the solution is correct or false. Other complexity classes which are applicable to PoK systems include AM (Arthur Merlin), IP (Interactive Polynomial Time), QIP (Quantum Interactive Polynomial Time), MIP (Multiple-Prover Interactive Polynomial Time) and PCP (Probabilistically Checkable Proofs).

Assessing Scalability

The measures of time and space complexity are used to assess the efficiency and scalability of the prover and the verifier of PoK systems. The time complexity evaluates the amount of time it takes to run an algorithm. Factors that influence the time complexity of a protocol are:

(a) the number of rounds of the protocol, and (b) the size of the inputs.

The space complexity evaluates the amount of memory used by the algorithm. Factors that contribute to the space complexity of a protocol are:

(a) the number of messages sent throughout a protocol, and (b) the length of the messages.

Via these measures, it can be assessed how 'scalable' a systems is. Scalability enables us to understand whether a compuational problem can be solved or checked within given time and/or memory contraints.

Modes of Operations

In a PoK system Peggy exchanges messages with Victor in order to convince Victor about a statement Peggy made. Such a message exchange follows either in an interactive or non-interactive fashion.

Interactive Proofs: In an interactive protocol, Peggy and Victor exchange messages over k rounds. After the k rounds Victor either accepts or rejects the proof.

Non-interactive Proofs: In a non-interactive protocol, Peggy sends one message to Victor. Victor evaluates and either accepts or rejects the proof. We emphasize that Victor does not send messages to Peggy.

In the soundness property we have already indicated that Peggy must know a witness w in order to create a valid statement. If Peggy knows w she should be able to convince Victor that the statement is true and convince Victor to accept the proof. The phrase 'knows' in this context maps to the second part of a PoK. These proofs can be further refined according to the power or strength of a prover into the following two cases:

Proof of Knowledge (PoK): In a 'proof' of knowledge, we assume that Peggy is all-powerful. This means that Peggy is not computationally restricted in any of her executions.

Arguments of Knowledge (ARG): In an 'argument' of knowledge, we assume that Peggy is computationally bounded. As such, there is a limit in either time and/or memory complexity which she can't exceed in order to convince Victor about the truth of her statement x.

Cryptographic Definitions

Zero-knowledge proofs offer computational[5] integrity while preserving privacy. To this end, we haven't defined the terms 'integrity' and 'privacy' so far. The following two definitions clarify the terms.

Definition: Integrity: A system provides integrity if any unauthorised modification can be detected. Consider a transcript being transferred from Peggy to Victor. A system is integrity-preserving if the transcript is the same after being transferred over an untrusted channel. Typically such checking involves 'generate' and 'verify' algorithms. The generate algorithm creates a transcript and a certificate of integrity. The verify algorithm takes the transferred transcript with the certificate and accepts if the transcript is the same or rejects if not. Hash functions are typically used to create such a proof of integrity.

Definition: Privacy: A system provides privacy if it does not leak information to any unintended participant. In the context of ZKPs, the proof certificate π should not leak any information about the witness w.

Adversarial Model

In PoK/ZKP systems we consider the following adversaries:

1. a malicious prover
2. a malicious verifier
3. an interested outsider.

In the case an entity behaves honestly we assume that entity follows the given zero-knowledge protocol and sends legitimate messages. A malicious entity may deviate from the protocol to send falsified messages in order to 'break' the protocol and learn any additional information as a result. We assume that both Peggy and Victor have the incentive to complete the protocol; Victor either accepts or rejects the proof. We exclude denial of service attacks from the attack surface[6].

In the case of a malicious prover we consider that Pat wants to convince an honest Victor about a false statement. A ZKP prevents Pat from successfully convincing Victor following the soundness property.

In the case of a malicious verifier we consider that Vanna wants to extract knowledge about the witness w from honest Peggy. Depending on the zero-knowledge protocol, Vanna is able to send challenges to Peggy and learn from the response of her queries. Strictly following a secure zero-knowledge protocol prevents such a case following the zero-knowledge property.

In the case of an interested observer of the communication between Victor and Peggy the observer may learn information from the interaction. The observer may read or alter a message. Reading messages sent between Peggy and Victor should not reveal sensitive information to the observer following a secure zero-knowledge protocol (zero-knowledge property). If the observer alters the message the protocol may fail or Victor may reject the proof. It is therefore important that during the exchange, participants digitally sign their message in order to ensure integrity and authenticity of the messages. Digitally signing a message with an user-specific private key also solves the problem of impersonation attacks (the associated public key can be used to verify the signature of the messages). A special case of alteration attacks is when an observer deletes a message, known as a denial of service attack. If Peggy or Victor detects that a message didn't arrive after a given time, she/he could either try to resend it or both parties change the communication channel (which the observer hopefully can not alter). An external observer may reuse or replay an old message (or parts or a message) from a successful result of the protocol execution. To prevent such cases some sort of freshness guarantee must be added to the messages. This could be either a timestamp, a counter or a cryptographic nonce.

We exclude any type of side-channel attacks from our threat model where the adversary has access to the computational machines of Peggy or Victor. We assume that both Peggy and Victor are running their protocols on secure and trusted machines. We assume that Peggy and Victor are following the zero-knowledge protocol and have no incentive[7] to do otherwise.

Hash functions

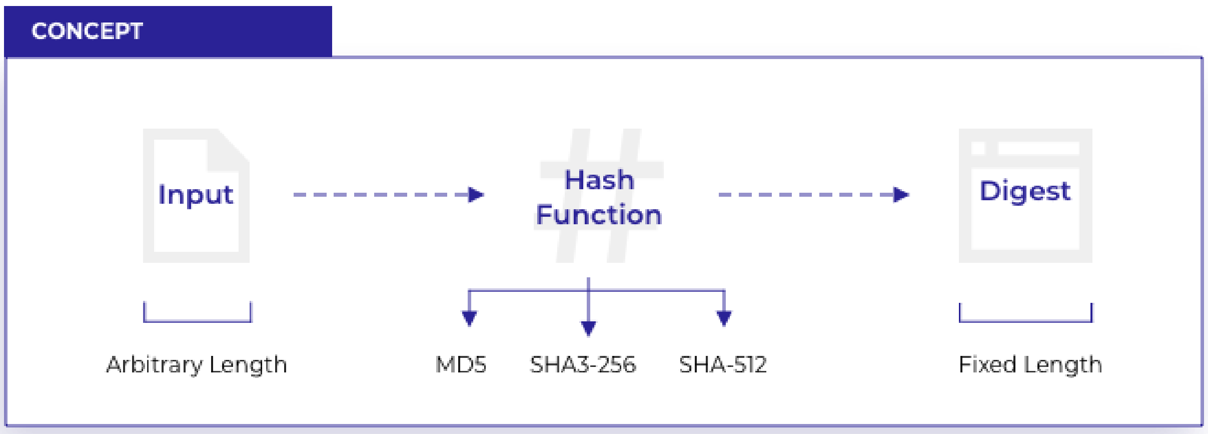
Hash functions are used in several zero knowledge protocols, including ZK-SNARKs and ZK-STARKs.

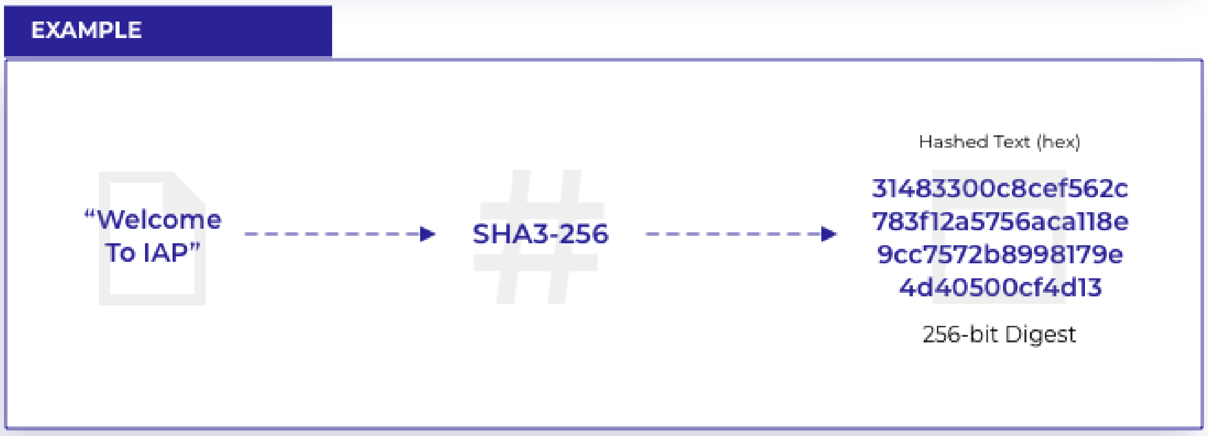
Definition: Hash function: A hash function maps bit strings of arbitrary length to fixed-length outputs. More formally, for a function $H : \{0,1\}^* \rightarrow \{0,1\}^n$ for finite n, we say H is a hash function.

The flexible input size to hash functions makes them useful for constructing digital signatures, ZKPs, and other algorithms. For use in secure protocols, cryptographic hash functions must always be *one-way*.

Definition: One-way function: Given b in the image space of H, it must be infeasible to recover an input string a that satisfies $H(a)=b$.

If the image space of a hash function H is smaller than the size of the domain, then H may also be considered a compression function. In this case it is expected that several such a exist to satisfy $H(a)=b$.





Additional security definitions imposed on hash functions are ***preimage resistance***, ***second-preimage resistance***, and ***collision resistance***. We outline the definitions below.

Definition. Preimage resistance: Given y an element in the image space of H, it is infeasible to find a preimage a such that $H(a)=y$.

Definition. Second-preimage resistance: Given a, it is infeasible to find a second pre-image b such that $H(a)=H(b)$.

Definition. Collision resistance: It is infeasible to find a,b with $a \neq b$, such that $H(a)=H(b)$.

In the context of advanced ZKPs, we are particularly interested in the last property of collision resistance. Collision resistant hash functions like SHA-2 and SHA-3 are believed to be post-quantum secure (meaning that they withstand attacks by a quantum computer). Collision resistance also implies second-preimage resistance[A33].

Elliptic Curves and Pairings

Elliptic Curves and Pairings are essential parts of ZK-SNARKs, a special kind of ZKP discussed below. An elliptic curve is an algebraic structure defined by a curve equation $y^2 = x^3 + Ax + B$ over a field: e.g. the real numbers, the complex numbers, the rational numbers, or a finite field F_p. Elliptic curve cryptography (ECC) defines curves over finite fields. The points (x,y) over the field which satisfy the curve equation, combined with the ***point at infinity*** form the elliptic curve. The point at infinity is added to ensure the set of elliptic curve points form an additive group under a special group law[8,9,A30] .
For an elliptic curve C defined over finite field F_p, let $C(F_p)$ denote the group formed by the points of C[A30].

Suppose for now that the size of $C(F_p)$ is a prime number $r \neq p$. In this case, $C(F_p)$ is almost cyclic since any element besides the point at infinity generates the whole group. The smallest integer k such that $r \mid p^k - 1$ is known as the ***embedding degree*** of C. Let us denote the group $C(F_p)$ by G_1. The multiplicative group of extension field F_{p^k} contains a subgroup of order r which we denote by G_T. The elliptic curve with points from this extension field $C(F_{p^k})$ also forms a group and this group contains G_1. $C(F_{p^k})$ contains another subgroup of order r which we may call G_2.

Definition: Tate pairing: Let g, h be generators of G_1, G_2 respectively. There exists a map called a Tate pairing that maps G_1, G_2 to G_T such that:

- Tate maps the generators g,h to a generator $\mathbf{g}$ of G_T
- For field elements a,b Tate preserves the following equality Tate(ag,bh)=$\mathbf{g}^{ab}$.

Zcash[10], one real-world implementation that uses ZK-SNARKs, digresses to introduce a term called homomorphic hiding in their blog series[11], not found in the scientific literature, but which captures an essential concept. Homomorphic hiding is the simplest form of homomorphic encryption, and is a function $H()$ which satisfies:

- Given $H(x)$, $H(y)$, one can compute an algebraic combination, e.g. $H(x+y)$, $H(xy)$, etc.
- $H(x)$ is easy to compute.
- Given $H(x)$ it is difficult to recover x, for most x.

Let Z_p denote the finite field of p elements: {0,1,...,p-1} and Z_p^* the corresponding multiplicative group $Z_p \setminus \{0\}$. The group Z_p^* is cyclic and thus can be generated by some element $g \in Z_p^*$. Then the HH properties are satisfied by the function $H(x)$ = g^{x mod p-1}, where <g>=Z_p^* and is illustrated as follows:

- $H(x)H(y)$ = g^{x mod p-1} g^{y mod p-1} = g^{x+y mod p-1} = $H(x+y)$
- Modular exponentiation is efficiently computable.
- Given g, g^x it is difficult, given only classical computational power, to recover input x for most x. This is known as the discrete logarithm problem.

The above Tate pairing construction allows us to generate a HH that preserves addition and multiplication, but first we define three functions:

- $E_1(x) = x \cdot g$
- $E_2(x) = x \cdot h$
- $E(x) = x \cdot \mathbf{g}$

where $x \cdot g$ denotes adding the elliptic curve point g to itself x times, or scalar multiplication. Given two values $E_1(x)$, $E_2(y)$, one can compute $E(xy)$ using the Tate pairing.

Quadratic Arithmetic Program

One ingredient of ZK-SNARKs that adds to their efficiency is an Quadratic Arithmetic Program (QAP). A technique emerged in 1992 for translating PoK statements to polynomials for interactive proof systems. In 2013 it was shown how to extend this concept in a non-interactive way by using QAPs.

Arithmetic circuits are a collection of gates and wires used to connect the gates such that for a finite input set the circuit will perform a computation and produce an output. To construct a QAP from a given PoK statement the statement is first converted into an arithmetic circuit. The circuit's multiplication gates must have exactly two input wires: one left and one right, to be converted to a QAP. These help to define a *legal assignment* for the circuit; a set of input wires and output wires that are the result of a multiplication between exactly two wires.

In the QAP each multiplication gate is associated with a field element called a ***target point***. The left and right input wires correspond to left and right polynomials. The output of each multiplication corresponds to an output wire polynomial. The left and right polynomials are defined to be zero on all target points except the ones to which they correspond. The legal assignment elements are coefficients of the left, right, and output polynomials. This construction is carefully designed so that a legal assignment is valid if and only if the polynomial P = LR-O is zero for all the target points[12].

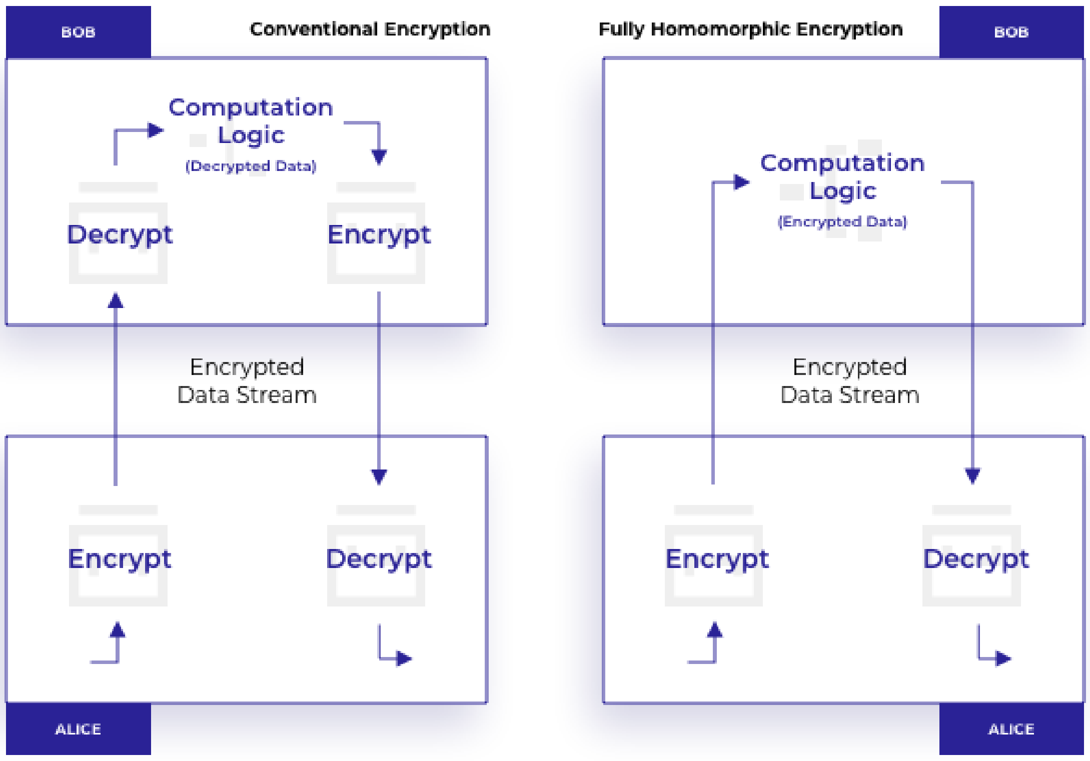
Homomorphic Encryption

Fully homomorphic encryption (FHE), hailed by some as the "Holy Grail" of cryptography[13,A32] has revolutionized digital privacy in some trustless applications. This primitive ensures that data encrypted in a particular way can be processed without access to the secret decryption key. The result of decrypting first then running some computation, ***C***, on plaintext will yield the same result as running ***C*** on ciphertext and then decrypting. FHE allows ciphertext data to be manipulated, queried, and operated on without compromising the privacy of the underlying plaintext; removing the need for trust. An untrusted server or third-party may then compute on user encrypted data and return the encrypted results of the query or operation to the users who then use their own private key to decrypt the results.

The first FHE scheme was proposed in 2009[A25], and improvements made in 2011 still yielded implementation results with public keys exceeding 2GB[A26]. Several homomorphic encryption libraries exist[14] and although FHE is still under development the significance of this cryptographic primitive will fundamentally change cloud computing and storage. It also holds promise in such advanced areas as unlocking the relationships between DNA sequencing and disease.

IAP Network is currently incorporating homomorphic encryption libraries into the base library of the open source IAP Client, extending the verification capabilities of the IAT[15] CyberShield tool. These privacy preserving abilities of homomorphic encryption allow non-trusted parties to perform computations on sensitive data without the ability to view the plaintext data. This will be extended to FHE once libraries become available.

Database security includes encrypting stored data[16]. The queries to databases can leak information about the corresponding plaintext data and should also be secured. Queries by a lawyer for past cases involving workplace misconduct or other controversial topics might reveal sensitive information about a client. Queries by an employee about job openings might reveal the intention to leave his current employer. The ideal scenario is for a user to encrypt her queries, send the encrypted queries to the server and receive encrypted results from the server. When she is ready she can decrypt the results which are the response to her plaintext query.



Suppose users allow financial holdings to be monitored in real-time by third-party services. These services might include alerts about suspicious activity, predictions based on spending habits, and forecasts for expected future balances. These services are currently provided by trusted parties. Fully homomorphic encryption would allow user financial data to remain encrypted while being analyzed by such business entities.

IAP Network is actively researching additional use cases for FHE with regards to the IAT. FHE has been shown in the literature to provide theoretical solutions for achieving verifiable computation on private data[A24]. This is especially useful for outsourced and cloud computations. CyberTraces and CyberShields are two components of the IAT that offer independently verifiable evidence events. FHE-based verifiable computation would enhance CyberTraces and CyberShields by adding to the variety of security models and assumptions.

One of the most interesting implications of FHE is the opportunity for scientists to study DNA sequencing in a privacy-preserving manner. Genome analysis seeks to study genetic information to identify genetic predispositions and in doing so prevent fatal conditions. Though this line of research offers promising benefits to patients, genetic information is and must remain confidential. Researchers are pursuing FHE as a solution. Since 2014 the annual iDASH Privacy & Security Workshop has sought to address biomedical challenges involving big data.

IAP Network continues to monitor the progress relating to FHE and the analysis tools emerging from the iDASH workshop series. These state-of-the-art techniques may readily extend to identity management and privacy, a major use case of the IAP, by better detecting fraudulent activity based on private user history. Other sensitive data like private votes may be protected; ensuring accurate government polling of electorates. The tallying of private votes can be displayed using CyberChains[A8] to maximize transparency.

Proof of Knowledge Systems

The following section provides an overview of current Proof of Knowledge (PoK) systems. A PoK can be seen as something similar, yet separate to a zero-knowledge proof (ZKP). PoKs have the similar properties as ZKPs in terms of completeness and soundness, but PoKs typically include an extractability property in contrast to the zero-knowledge property of a ZKP. The following presents a theoretical overview of PoK systems, followed by an overview of ZKP systems. Further, in the Practical Zero-Knowledge Proofs section we present an overview of practical implementations of zero-knowledge proofs.

Interactive Proofs

The first class of PoK systems in the IAP are so called interactive proof (IP) systems. IP systems are by far one of the biggest class of proof systems and include multiple specialisations (explained further below in this section). In an IP system the prover and the verifier engage in an interactive protocol; messages are exchanged between both entities, and at a given point in time the verifier accepts or rejects the proof.

Non-deterministic Polynomial Time Proofs (NP)

Interactive proofs pertaining to the class of NP are the simplest form of interactive proofs. These proofs are single-round interactive proofs. We can barely say that such proofs are interactive; they follow the same principles as interactive proofs, yet run in a single round. For simplicity, we say proofs in this category are IP[1] proofs; the '1' denotes a single round. Following the character convention introduced in our recap, there exist an all-powerful prover (Peggy) and a computationally-bounded verifier (Victor). Being all-powerful, we understand that Peggy is not limited by any computational resources such as memory space or computation time. Peggy is considered to be a type of 'idealistic' entity[17]. Victor is represented by a deterministic[18] polynomial-time machine[19].

For any problem in the complexity class NP a proof protocol follows:

1. Peggy looks at the input a, computes $b=f(a)$ and gives Victor a proof π associated with the computation, where π proves that b is indeed the result of $f(a)$.
2. Victor verifies π in polynomial time and if π is valid he accepts, otherwise rejects the proof.

Interactive Polynomial Time Proofs (IP)

Interactive proofs pertaining to the class of IP extend the NP proof class by allowing the protocol to run over multiple rounds. Multiple messages may be exchanged between Peggy and Victor before Victor reviews the proof π and makes a decision to accept or reject the proof. To indicate the number of rounds in the class we denote the class as IP[k], where k is the number of rounds.

Victor has access to a string of random bits whose length is polynomial in the size of the input statement x. The random choices of Victor are kept private. Such a protocol is further referred to as a private coin protocol. Peggy and Victor exchange a polynomial number of messages (i.e. the length of all messages over k rounds are bounded by that polynomial). For any problem in the complexity class IP[k] a proof protocol follows. The protocol runs over two stages; an interaction phase where Peggy and Victor are exchanging messages and a decision phase where Victor makes a decision to either accept or reject the proof.

Interaction Phase:

1 - k: Peggy and Victor exchange messages.

Decision Phase:

1. If the statement is valid, i.e. $x \in L$, then Victor accepts the proof π with probability $\geq ⅔$.
2. If the statement is invalid, i.e. $x \notin L$, then Victor rejects the proof π with probability $\leq ⅓$.

Arthur-Merlin Proofs (AM)

Interactive proofs pertaining to the class of AM extend the IP proof class by allowing Peggy to learn the random choices (i.e. random bits) of Victor. In comparison to IP where all random choices of Victor are private, in AM proofs the random bits of Arthur are public. We denote the all-powerful prover as Merlin and the verifier as Arthur. Due to the public randomness choices, AM proofs are referred to as public coin protocols. Other than the public randomness of Arthur, AM proofs follow the same conventions as IP proofs[20].

Multi-Prover Interactive Proofs (MIP)

Interactive proofs pertaining to the class of MIP extend the IP proof class by allowing two independent provers, Peggy and Pat, to convince Victor about the truth of a statement x. It is important to say that Peggy and Pat are independent; they may communicate with each other beforehand, but cannot communicate with each other once Victor starts to send messages to them. The idea of multi-prover interactive proofs is motivated by minimising the probability of a 'lying' prover. An analogy can be made of the interrogation of a criminal; it may be easier to detect if a criminal is lying if he and his partner are interrogated in separate rooms (i.e. independence). Adding another prover might reveal 'more' or 'different' details (or viewpoints) of a story (i.e. a statement x).

Non-Interactive Proofs (NIP)

The second class of PoK systems we are looking at are the so-called non-interactive proofs. This is a variant of PoK systems in which no interaction between Peggy and Victor is necessary. By 'no-interaction' we mean that Peggy sends a message to Victor who then makes a decision based on the information he obtains from Peggy.

To enable a 'message exchange' between Peggy and Victor both must share a common reference string which enables computational zero-knowledge without the requirement of interaction. Sharing of a common reference string between both parties follows the common reference string model[21]; a specific cryptographic model for non-interactive zero-knowledge proofs.

Another way of achieving non-interactive proofs in the random oracle model[22] is via the Fiat-Shamir heuristic[23]. Fiat-Shamir enables the transformation of an interactive PoK to a non-interactive one. The interactive proof must be public coin (ie. the verifiers randomness needs to be public) in order to be convertible to be non-interactive.

Probabilistic Checkable Proofs (PCP)

The next class of PoK systems we are looking at are the so-called probabilistic checkable proofs. Proofs pertaining to the standard class of PCP are verifiable in polynomial time for a set of single-round, non-interactive decision problems. Specialisations in this class allow interactive decision problems[24] as well as restrict the query complexity to be linearly bounded[25].

One may ask now what is so special about these kind of proofs, since all previous classes discussed before already achieve the same goals. The prominent feature of PCPs is that the verifier Victor has access to only a bounded amount of randomness; we denote such access by $r(n)$ (where n denotes the length of the random bit string) and further refer to it as randomness complexity. Victor only reads a bounded amount of bits from the proof π; we denote this by $q(n)$ and further refer to it as query complexity.

In addition to the existing notations we denote with $c(n)$ the completeness probability: $c(n)$ gives a lower bound to which a proof achieves the completeness property (i.e. Peggy convinces Victor about the truth of a statement). With $s(n)$ we denote the soundness probability: $s(n)$ gives an upper bound to which a proof achieves the soundness property (i.e. a malicious prover, Paul, is able to convince Victor about a false statement).

For any problem in the complexity class $\text{PCP}_{c(n),s(n)}[r(n),q(n)]$ a proof protocol follows:

1. Peggy looks at the input a, computes $b=f(a)$ and gives Victor a proof π associated with the computation. In this context, π proves that b is indeed the result of $f(a)$.

2. Victor verifies π in polynomial time by utilising randomness bounded by $r(n)$ and querying $q(n)$ bits of π:
    1. If the statement is valid, i.e. $x \in L$, then Victor accepts the proof π with probability at least $c(n)$.
    2. If the statement is invalid, i.e. $x \notin L$, then Victor accepts the proof π with probability at most $s(n)$.

Typically in PCP proofs, unless stated otherwise, $c(n)$ is ≥ 1 (probability that Victor accepts a true π) and $s(n)$ is $\leq \frac{1}{2}$ (probability that Victor accepts a false π).

Previous authors have shown several relationships between the PCP complexity class with other complexity classes. One of these is the prominent PCP theorem[26]: PCP[O(log n), O(1)] = NP. This theorem states that every decision problem in the NP complexity class has a probabilistic verifiable proof of logarithmic randomness complexity $r(n)$ = O(log n) and constant query complexity $q(n)$ = O(1).

Probabilistic Checkable Interactive Proofs (PCIP)

Generic PCPs are inherently one-round protocols. A interactive extension was first proposed by Kalai and Raz[A31]. In their extension they allow a verifier to have additional oracle access to a PCP before the verifier starts with an IP protocol. This type of protocol combines the flexibility of interactive proofs (allowing Victor to communicate with Peggy over multiple rounds) with the expressiveness of a PCP (allowing a wider range of problems[27]).

Due to these improvements interactive PCP protocols gained wide attention by the cryptographic community. This resulted in a wide study to improve on the preliminary results of Kalai and Raz; the section below on Interactive Oracle Proofs explains these results in more detail.

Linear Probabilistic Checkable Proofs (LPCP)

Another improvement of generic PCPs is to limit the proof system to be an argument system. As such it only achieves computational soundness, yet improves on the communication and computation time to be linear bounded. This inherently changes the prover Peggy to be computationally bounded (and not all-powerful). As a result of this the proof string π doesn't need to be polynomial in size. Instead the PCP may be viewed as a linear function.

Probabilistic Checkable Proofs of Proximity (PCPP)

Further extending and improving on proofs of the class PCP is the next class of PoK systems; probabilistic checkable proofs of proximity[A29] (PCPP). In a PCPP proof the randomised verifier Victor gains access to a few bits of the proof π, equivalent to a PCP. Victor gets access to only a few bits of input x. Victor is asked to verify if the statement $x \in L$ holds. Victor cannot generally be expected to distinguish if $x \in L$ from the case that $x \notin L$, but should be very 'close' to it; Victor is expected to be able to distinguish between these two cases and reject any x (i.e. $x \notin L$) if x is 'far' from L. In natural language; x is nowhere near to being included in the language L.

To cope with these developments we introduce another parameter to the PCPP PoK system: the proximity parameter δ. δ will indicate how 'far' a statement x is to be allowed to differ to be included in the language $x \in L$. We say Victor is only reading $q' \leq q(n)$ bits from the input statement x.

For any problem in the complexity class $\text{PCPP}_{s(n),\delta}$[$r(n)$,$q(n)$] a proof protocol follows:

1. Peggy looks at the input a, computes $b=f(a)$ and gives Victor a proof π associated with the computation.
2. Victor verifies π in polynomial time by utilising randomness bounded by $r(n)$, querying $q(n)$ bits of π as well as q' bits of x:

If the statement x is δ-close to L, i.e. $|x \in L| \leq \delta$, then Victor accepts the proof π with probability at most $s(n)$; otherwise he rejects the proof π.

A PCPP with proximity parameter δ = 0 is referred to as an exact PCPP, allowing no deviation of x from the language L.

Interactive Oracle Proofs (IOP)

The next class we are looking at was already briefly introduced in the section of PCIPs. Interactive Oracle Proofs[A20] (IOPs) combine the features of IPs and PCPs but in a more subtle way than PCIPs. In a PCIP a verifier has additional oracle access to a PCP before the interactive protocol starts.

In an IOP the verifier Victor is not required to read the messages of prover Peggy in their entirety. Victor has oracle access to Peggy's messages and may probabilistically query those messages. In this context we add another parameter: l will denote the length of the proof π (concretely, l sums up the total number of bits of all messages sent from Peggy to Victor). We further remind the thoughtful reader of the parameters k denoted as the round complexity (in natural language: the number of rounds), $c(n)$ the completeness probability (if not specified otherwise, $c(n)$ is typically assumed to be 1), $s(n)$ the soundness probability and $r(n)$ the randomness complexity (i.e. the number of random bits). Specific to IOP systems, in the i-th round of interaction we say Victor sends message m_i to Peggy, who reads the message in full (yes, in full; the mindful reader is reminded that it is Peggy who reads the message in full) and for response messages n_i from Peggy to Victor; Victor has oracle access and queries them probabilistically with $q(n)$ (the query complexity), denoting the total number of bits Victor reads from the messages n_i.

For any problem in the complexity class $\text{IOP}_{c(n),s(n)}$ [$r(n)$,$q(n)$,k,l] a proof protocol follows:

Interaction Phase:

1 - k: Peggy and Victor exchange messages (m_i, n_i).

Decision Phase:

1. Victor verifies π in polynomial time by utilising randomness bounded by $r(n)$ and querying $q(n)$ bits of π:
    1. If the statement is valid, i.e. $x \in L$, then Victor accepts the proof π with probability $\geq c(n)$.
    2. If the statement is invalid, i.e. $x \notin L$, then Victor rejects the proof π with probability $\leq s(n)$.

IOPs generalise IPs and PCPs. A PCP is essentially a IOP with k = 1 and the first message from Victor to Peggy is considered to be empty. An IP is a IOP where Victor is querying the whole proof π.

Interactive Oracle Proofs of Proximity (IOPP)

The last class of PoK systems we are looking at extend the previous class by having additional query access to the input statement x (concretely: querying only a few bits of the input). This is similar to the class of PCPP, and is the so-called interactive oracle proofs of proximity[A17] (IOPP). We can say that an IOPP merges the classes IOP and PCPP. An IOPP is an IOP where Victor has access to a oracle to query input x. A proximity parameter δ decides if a statement x is included in a language $x \in L$, if x is δ-close to elements of L. This means that Victor accepts proofs π where x is δ-close, otherwise Victor rejects the proof.

Similar to the argument of IOPs (that generalise IPs and PCPs) we can say that IOPPs generalise IOPs and PCPPs. A PCPP is an IOPP with k = 1 (single round); further, a IOP is an IOPP where Victor queries the whole input x.

For any problem in the complexity class $\text{IOPP}_{s(n),\delta}$ [$r(n)$,$q(n)$,k,l] a proof protocol follows:

Interaction Phase:

1 - k: Peggy and Victor exchange messages (m_i, n_i).

Decision Phase:

1. Victor verifies π in polynomial time by utilising randomness bounded by $r(n)$, querying $q(n)$ bits of π as well as q' bits of x:

1. If the statement x is δ-close to L, i.e.$|x \in L| \leq \delta$, then Victor accepts the proof π with probability at most $s(n)$.
2. Otherwise he rejects the proof π.

Summarising PoK Systems

There exist a wide variety of PoK systems. Starting with simple proof systems for the class of NP the previously mentioned proof systems continuously add on extra features. Or, in other words, generalise the previous proof systems. In the previous sections we have learned that a PoK system typically consists of two parties: a prover Peggy who wants to convince a verifier Victor about the truth of a statement. In doing so she creates a proof string π which she communicates via a protocol to Victor. Victor then accepts or rejects the proof π.

In Table 1 we present an overview of the previously discussed PoK systems. We use the following notations and parameters:

- # of messages --- indicates how many rounds the protocol runs; possible values are : $k \in N$;
- prover complexity --- indicates the 'power' of Peggy; possible values are: all-powerful or computationally bounded;
- verifier complexity --- indicates the 'power' of Victor; moreover, we indicate the number of bits read from proof string π either as full or as $q(n)$;
- verifier randomness --- indicates the randomness used by Victor; possible values are full or $r(n)$;
- completeness --- indicates the probability that Victor accepts the proof π; for proximity proofs we define the 'proximity' parameter δ; possible values are: δ, $c(n) \in [0,1]$;
- soundness --- indicates the probability that Victor rejects the proof π; in other words, the error rate that Victor accepts a false proof π for a statement $x \notin L$; possible values are: δ, $s(n) \in [0,1]$;

Parameter	**# of messages *k***	**Prover Complexity**	**Verifier Complexity *q(n)***	**Verifier Randomness *r(n)***	**Completeness* *c(n)***	**Soundness* *s(n)***
IP						
NP	single	all-powerful	full	(private) full	1	0
(generic) IP	k	all-powerful	full	(private) full	≥ ⅔	≤ ⅓
MA	single	all-powerful	full	(public)[16] full	≥ ⅔	≤ ⅓
AM	k	all-powerful	full	(public) full	≥ ⅔	≤ ⅓
MIP	k	collection of non-communicating strategies	full	(private) full	≥ ⅔	≤ ⅓
NIP	single	all-powerful	full	(public)[17] full	*c(n)*	*s(n)*
PCP						
(generic) PCP	single	all-powerful	*q(n)*	(private) *r(n)*	1	≤ ½
PCIP	*k*	all-powerful	*q(n)*	(private) *r(n)*	1	≤ ½
LPCP	single	computationally-bounded	*q(n)*	(private) *r(n)*	1	≤ ½
PCPP	single	all-powerful	*q(n)*	(private) *r(n)*	δ	1 - δ
IOP	*k*	all-powerful	*q(n)*	(private) *r(n)*	1	*s(n)*

IOPP	*k*	all-powerful	*q(n)*	(private) *r(n)*	δ	1 - δ

Table 1: Overview of previously discussed proof of knowledge systems. * indicates standard values, unless stated otherwise.

Zero-Knowledge Proofs

Zero knowledge proof systems, originating from a 1985 journal submission[A27], allow one to prove that a statement is true without revealing more than the truth of the statement. A statement such as "893 is a composite number" can easily be proven true by providing one non-trivial factor of 893. This approach reveals more than the truth of the statement and would consequently violate the zero knowledge property. A general zero knowledge proof system is comprised of a statement and some communication exchange between two parties; a prover and a verifier. The prover makes a statement to the verifier and attempts to convince the verifier that the given statement is true.

In graph theory it is known that given two graphs, it is difficult to find a homomorphism between the graphs[28]. Suppose Peggy knows a homomorphism between two graphs G_1,G_2, and wants to prove her knowledge to Victor. Suppose further that Peggy does not wish to reveal her homomorphism but still aims to convince Victor that she does know a mapping. A zero-knowledge proof system can help solve Peggy's problem. Victor challenges Peggy by sending her vertices of $G_1$and requests the image of each vertex under her secret homomorphism. After several rounds of challenges and responses from Peggy, Victor can determine whether the vertex images reported by Peggy respect the properties of a graph homomorphism. Eventually Victor will be convinced that Peggy does know a homomorphism without ever learning what the homomorphism is.

In this scenario Peggy sought to prove the statement "I know a homomorphism between G_1,G_2." Statements in zero knowledge proof systems should be easy to verify via a ***witness***. In the case of the integer 893, there are two witnesses; 19 and 47. In Peggy's case her secret homomorphism is a witness. A witness should not be revealed in a zero knowledge proof, but must be easy to verify. We emphasize that Peggy's interaction with Victor is classified as an interactive zero knowledge proof. The particular proof systems we discuss later, ZK-SNARKs and ZK-STARKs, do not require nor permit Victor to make queries to Peggy. We classify such systems as *non-interactive* proofs.

A statement is formally defined over a language and outputs a proof. The statement "893 is a composite number" can be defined over the language L ={ (n,d) | if d divides n with $1 < d < n$, then n is composite}. A proof π should be necessary and sufficient information to convince a verifier that a statement is true.

While a PoK system fulfills only the properties of completeness and soundness, a zero-knowledge proof additionally has the property of zero-knowledge:

Definition: Zero-Knowledge: A PoK system satisfies the zero-knowledge property if a verifier (Victor) is not able to learn anything else from the proof except that the statement is true or false.

If $(x,w) \in R$ then no verifier should be able to infer anything about the witness w except that x is a valid statement of language L and the prover is in the possession of a witness w which can prove that.

Variants of Zero-Knowledge Proofs

Depending on the 'power' of both Peggy and Victor different variants of the zero-knowledge property can be achieved. For the definitions below we need to state an assumption --- ZK systems assume there exists a simulator accessible to every verifier which is able to 'simulate' the interaction between Peggy and Victor. Then, the following variants apply:

Definition: Perfect Zero-Knowledge: a ZK-PoK system achieves perfect zero-knowledge if the distribution between the simulator S and the real interaction between Peggy and Victor remains completely indistinguishable.

Definition: Statistical Zero-Knowledge: a ZK-PoK system achieves statistical zero-knowledge if the distribution between the *S* and the real interaction between Peggy and Victor is statistically close (i.e. negligible).

Definition: Computational Zero-Knowledge: a ZK-PoK system achieves computational zero-knowledge if the distribution between the *S* and the real interaction between Peggy and Victor remains indistinguishable by a computationally-bounded algorithm (i.e. polynomial-time Turing machine).

Advanced Zero-Knowledge Proofs

The above overview of a PoK and a natural specialisation as a zero-knowledge proof allow us to now dig deeper into desirable properties of such proof systems. Achieving the basic properties of such systems (i.e. completeness and soundness) can be done fairly efficiently with current state-of-the-art technology. Achieving the zero-knowledge property can be achieved in a similar manner. If more properties are added then the zero-knowledge systems become increasingly complex. To utilise such systems in real-world settings, an application designer must keep certain limitations or trade-offs in mind.

The following lists desirable properties of advanced PoK systems. We will then cover two advanced proof systems: ZK-SNARKs and ZK-STARKs.

(Desirable) Properties of Proof Systems

Definition: Succinctness: A PoK system is succinct if the time to verify the validity of the statement is much smaller than simply recomputing the statement (i.e. the function). The naïve approach to verifying correctness of a statement is to simply recompute the function with the given inputs. This may not always be the optimal solution. For example, if we consider resource intensive operations then it might be too expensive to recompute the original function during the verification step. The property of succinctness is very useful despite coming with the price that while verification time is much smaller, the time to create a proof is larger than simply computing the function.

Definition: Transparency: A PoK system is transparent if any randomness used by the prover or verifier is public. PoKs following such a convention are typically referred to as Arthur-Merlin or Merlin-Arthur proofs. PoK systems which don't follow these conventions typically rely on a trusted set-up phase. Private randomness is used during this phase to generate certain parameters used for later operations. It is essential that the randomness in this phase doesn't leak to any malicious entity (i.e. prover, Pat) otherwise Pat may be able to convince Victor about a false statement *x*' without Victor being able to identify that there was something amiss within the protocol.

Definition: Universality: A PoK system is universal if it can be applied to any computational problem in its relevant computational complexity class. Goldreich et al.[A28] proved that for every problem in the class of NP, there exists a zero-knowledge proof. Being compliant to solve any computational problem typically comes with an efficiency trade-off.

Definition: Scalability: A PoK system is scalable if the prover runs in quasilinear time and verifier validates the proof in polylogarithmic time. Verifier scalability is very important in order to achieve succinct proofs. To be useful in any real-world scenario the prover must also scale when the function complexity (i.e. circuit size[29]) and the input size (i.e. number of variables) increase.

Definition: Post-Quantum Security: A PoK system is post-quantum secure if it withstands the attacks of a quantum computer. Shor's algorithm[30] presents a quantum polynomial time attack on some public-key cryptography based on the integer factorisation problem and the discrete logarithm problem. Because quantum attacks against collision-resistant hash functions have not been found to run in quantum polynomial time, it is believed that if a PoK system is only based on collision-resistant hash functions then the PoK system remains post-quantum secure.

Practical Zero-Knowledge Proofs

The recent introduction of zero knowledge proofs to a more mainstream audience has initiated efforts to standardize these protocols. The first ZK Proof Standards Workshop[31], held in May 2018, was organized to further the development and use of zero knowledge proofs by gathering together field experts from both industry and academia. This first workshop aimed to develop metrics for assessing the security of ZK Proofs and to find the balance between performance and security tradeoffs for practical implementations and applications, among other goals. Whitepapers[32] from the workshop proceedings reveal the many factors involved in establishing a secure and practical zero knowledge proof.

The efficiency of a zero knowledge proof is based on the running time of the prover, the running time of the verifier, the memory required to store proofs and witnesses, and the cost of any initialization or set-up phase. We quantify these factors in terms of time complexity and space complexity and further classify zero knowledge proofs into complexity classes.

In practice the ***statement*** produced by the prover is converted to a circuit. The size of the proof should be sublinear in the size of the statement circuit to be considered an efficient zero knowledge proof. ZK-STARK constructions convert a statement to a set of polynomials over binary fields that satisfy certain constraints. This process is known as arithmetization.

Different zero knowledge proof techniques arithmetize statements differently. In the Ligero protocol[A12] statements are converted to circuits and then to matrix over a finite field, not necessarily binary. In range checking proofs the statement is converted to a set of multivariate polynomials over finite fields. The range checking proofs are not considered quantum-secure as they depend on the intractability of the discrete logarithm problem. These different approaches to statement conversions are being addressed by the ZKProof.org Workshop37 series as well as ideal file formats and compilers.

Existing Libraries and Implementations

ZK-SNARKs have been adopted by the tech community in the form of libraries and implementations. The most prominent library is Libsnark, developed by the SCIPR lab, which offers C++ implementations of all aspects of ZK-SNARKs. IAP Network has implemented the Zcash libsnark library into the IAT CyberShield tool for its proof of concept. Ethsnarks, Zsl-q, and Jsnark offer Ethereum, Quorum, and Java variants, respectively. A general toolbox for ethereum and off-chain zero knowledge is offered in ZoKrates, and an end-to-end toolchain in Pequin. As previously mentioned, ZK-SNARKs require a trusted set-up phase and library Mpc offers a set-up alternative using secure multiparty computation protocols. Distributed ZK-SNARKs are available with Dizk. Additional open-source resources are Bellman, Groth16, and Snarky.

Library	Description	Reference
libsnark	de facto library	https://github.com/scipr-lab/libsnark
ethsnarks	ethereum variant	https://github.com/HarryR/ethsnarks
zsl-q	JPMorgan Quorum zero knowledge security layer	https://github.com/jpmorganchase/zsl-q
dizk	distributed zk-snarks	https://github.com/scipr-lab/dizk
jsnark	java implementation of zk-snarks	https://github.com/akosba/jsnark
bellman	rust implementation of zk-snarks	https://github.com/zkcrypto/bellman
groth16	ZK-SNARK variant by Groth16	https://github.com/zkcrypto/groth16
mpc	Secure multi-party protocol for trusted setup phase	https://github.com/QED-it/mpc
snarky	Library for R1CS SNARKs	https://github.com/o1-labs/snarky
pequin	End-to-end toolchain for SNARKs	https://github.com/pepper-project/pequin
ZoKrates	ZK-SNARK toolbox for Ethereum; off-chain	https://github.com/JacobEberhardt/ZoKrates

Table 2: ZK-SNARK libraries and resources

Beyond these open-source resources ZK-SNARKs have also made tangible contributions to existing financial platforms. The IAP, Zcash, Ethereum, and the J.P. Morgan Quorum[33] blockchain's zero knowledge security layer (ZSL) are currently running ZK-SNARKs to achieve zero knowledge proofs in a succinct, non-interactive way[34]. Although the proof sizes are small enough for anonymity in crypto transactions these protocols are not considered quantum-secure. ZK-SNARKs base security on certain hardness assumptions which allows for smaller memory requirements. Their quantum-secure counterparts abandon assumptions with known quantum attacks and instead use merkle trees to achieve an asymmetric construction. The merkle trees are thus included in the PoK. Circumventing this security tradeoff would result in significantly larger proof sizes for quantum-secure zero knowledge proof systems.

Another recent zero knowledge proof in a quantum-secure setting is ZKBoo[35] and ZKB++[36]. ZKBoo is interactive, while ZKB++ is non-interactive, boasts smaller proofs sizes, and has a highly detailed proof of security. The latter scheme is used as core ingredient of the signature scheme Picnic[37].

Bulletproofs are short non-interactive zero knowledge proofs that do not require a trusted set-up like ZK-SNARKs. They do however depend on the difficulty of solving the discrete logarithm problem; they are not quantum-secure. They improve on one feature of ZK-SNARKs but are not classified as a ZK-STARK.

In many financial scenarios it suffices to show that a value x lies within a finite interval $[a,b]$. Consider a potential home-buyer who wishes to prove himself financially prepared for a mortgage but does not wish to disclose his exact account balance. A range checking proof would allow him to prove that his account balance x is within the acceptable range $[a,b]$ for a mortgage. The specificity of this zero knowledge problem yields simpler solutions than ZK-SNARKs and ZK-STARKs and are known as ZKRPs, or zero knowledge range proofs. ING is currently incorporating efficient ZKRPs for Ethereum into their blockchain product. The security of the ZKRP depends on the intractability of the discrete logarithm problem which is not quantum-secure.

Library	Description	Reference
ZK-STARK	ZK-STARK library	https://github.com/elibensasson/libSTARK
ING ZKRP	Zero knowledge range proof	https://github.com/ing-bank/zkrangeproof
ZKBoo	Quantum-secure zero knowledge proof	https://github.com/Sobuno/ZKBoo
Picnic	Post quantum signature	https://github.com/Microsoft/Picnic
Picnic	Post quantum signature	https://github.com/IAIK/Picnic
Bulletproofs	Library for bulletproofs	https://github.com/bbuenz/BulletProofLib
Bulletproofs	Library for bulletproofs	https://github.com/dalek-cryptography/bulletproofs
Bulletproofs	Library for bulletproofs	https://github.com/ElementsProject/secp256k1-zkp

Table 2: Special categories of ZK Proofs

The greater tech community has been active in deploying libraries and implementations of ZKP tools. There are several other resources and ZKP implementations[38].

Use Cases

ZKPs are instrumental in creating anonymous transactions, proving the computational integrity of a given result, confirming private identification information, and underpinning data assurance for compliance and information assurance needs. As discussed, Zcash is an anonymous payment system secured by ZK-SNARKs. Below we expand on some technical components of this system. JPMorgan's Quorom platform and EY Ops Chain Public Edition contain zero knowledge layers to enable the privacy of client activity. ZK-STARKs are publicly verifiable, but this is not a general requirement of ZKPs, and only non-interactive ZKPs can achieve publicly verifiable property. IAP Network takes things a step further by embedding ZKPs directly into the IAT; enabling any industry application built on the IAP to provide proof that such application has computed its output data correctly.

Computational integrity is essential in cases where a third party is tasked with making some specialized computations on data that an individual cannot efficiently run for herself. These can include audits, compliance checks, and cloud computations. In many cases the computation is known, but the process is tedious and too expensive for personal computers and memory. Blockliance, one of the first round of applications under development to use the IAP, makes it easier for auditors to efficiently and cheaply verify the applied attestations of their clients against their own infrastructure in innovative and new ways[A9].

We summarize some of the commonly used zero-knowledge proofs used to establish computational integrity. Private information like age, credit score, or debt are numeric values and can be proven to lie within a finite interval while preserving the zero knowledge property by using a range-checking proof. Proofs of membership take private input strings, not necessarily numeric, to determine whether the string belongs to a given set or not. In both cases the proof can be used in an identity framework. Additional ZKPs may be used for private input, public function pairs like proofs of membership or proofs that private electronic votes have been properly tallied.

The recent ZKProof workshop at Zcon0[39] identified three prominent use cases of ZKPs: identity framework, asset transfer and regulation compliance. The following table summarises several use cases[40] established by academics and industry leaders during the workshop:

Use Case	**Statement**	**Private input**	**Function**
Range-checking proof[A23] Verifying sufficient credit score	"Credit score is within the desired range" $x \in [a,b]$	*Yes* Credit score	*Public* Greater than, less than
Set membership[A14] Is a passport number found on a no-fly list	"Passport number is an element of the no-fly list" $x \in S$	*Yes* Passport number	*Public* Set intersection
Verify computation[A11] Verify hash value for blockchain without revealing the input	"$f(x)=y$"	*Yes* Input x	*Public* $f()$
Knowledge of committed values, equality of committed values, relationships between committed values	"I know x such that $h(x)=y$" "I know $x_1 < x_2$ such that $h(x_1)=y_1$, $h(x_2)=y_2$"	*Yes* Inputs x,x_1,x_2	*Public* h
Knowledge of a signature on a message/committed value	"This is a valid signature on a private committed value"	*Yes* Committed value	*Public* Signature algorithm

Table 3: Several use cases of ZK Proofs

IAP Network Use Cases

While the main use case of zero-knowledge proofs integrated in the IAP is to provide proofs of computational integrity whenever a computation is executed off the public chain, there are many other use cases (as outlined in the IAP non-technical whitepaper[41]).

Use of ZK Proofs in IAP Stack

Computations executed off-chain (i.e. on IAP Chains[A4] or the non-IAA node IAP Client[A5]) are less trustworthy than on-chain (i.e. on IAA nodes[A6] or IAP Core[A7]) computations as they are not accessible for recomputation by the nodes of IAP Mainnet (Plato). To be able to prove that the computations on data on the side chain are executed as expected (ie. the 'claimed' result of the computation is truly the result of the computation), a CyberShield of the result can be computed. The result of the computation and the associated CyberShield certificate can then be pushed back to the public chain. In this way it is possible for the IAA Node to prove the correctness of the off-chain computation while keeping the computation private. The CyberShield certificate proves the correctness and in doing so, does not reveal anything else than the correctness of this assertion.

The IAP offers solutions to improve transparency of financial processes when properly paired with CyberShields. Government spending and assurance of philanthropic expenditures are among the many cases where financial accountability can benefit from CyberChains to track account changes. CyberShields should be incorporated when protecting sensitive or private data by setting the witness to the be the private data, and associating the statement to be proved to some arithmetic representation.

Electronic polling of electorate and boards requires the utmost transparency of vote tallying while preserving the privacy of individual votes. IAP CyberChains can be used to maintain the history of vote tallies while CyberShields must be incorporated to show that each counted vote was properly cast without revealing the underlying vote. Verifiable tally computations, secured by CyberTraces, can improve the integrity of results.

The following table briefly outlines the use of ZK Proofs for the IAP use cases identified in the IAP non-technical whitepaper. We colour-coded the categories of the IAP use cases as follows:

Use Case	Statement	Private Input	Examples / Applications
🟦 **Cybersecurity**			
Continuous Identity and Authentication	"I am authorised to use this service"	Identity	M-Pin
Supply Chain Security	"I am participant x" "Box x contains content y"	Identity Content of Cargo	OriginTrail
Internet of Things (IoT) device monitoring and control	"The device state of x is as expected"	Device state	Research Paper
Digital identity, authentication and signing of video/audio	"I am x" "I want to prove my signature"	Identity Signature	Set membership f(x)
Secure code scanning	"The source code scanner completed scan X of code Y"	Source code	Verification of code scan
Vulnerability management	"Software x complies with security controls y"	Software code & configuration	Research Paper
Intrusion detection and analysis machine	"A baseline X has been interrupted by type Y"	Historical activity of systems in	Research Paper

learning and AI		aggregate	
Source code malware analysis with machine learning	"Source code x is free of known malware indicators"	Source code	BitAV
Disaster Recovery and Data Redundancy	"We have computed backups at date X and successfully transmitted them to site Y" "Operational procedure X was successfully tested at site Y with expected outcome Z"	Data	IBM Cloud
Digital forensics	"Hash X was computed using library Y"	Confidential data	Research Paper
Change management & auditing, FIM	"Approved Change X was deployed using procedure Y with integrity"	Yes Historical valid file changes	zkledger
Development, Operations & Security			
Smart contract security	"smart contract x follows security requirement y"	Smart Contract	aztec protocol
Secure continuous delivery assurance	"Item x has been processed at location y"	Location y	Chronicled
Asset discovery and scanning	"Network discovery scan Y completed successfully"	Private IP ranges	Computation verification
Society, Charity & Humanitarian			
Transparency in donations and funding	"Agency accepted donation amount x from entity y, x is within the approved donation range, y is on the list of approved donors"	Amount x, Entity y	Research Paper
Non-profit agency spending	"Agency spent money on x, and x is within the set of approved expenses"	Expense x	Research paper
Assurance of expenditures	"I've spend amount x on expense y"	Amount x	Nonprofit Assurance
Asset tracking	"The location of asset x is y"	Location	Stratumn, Platin
Identity Management & Privacy			
Account takeovers mitigation	I am the authorised person to access account x	identity	Sovrin
Personal document assurance	"My passport number is y"	Passport information	IBM Blockchain Platform
Qualifications and education assurance	"I have qualification x" "I have a MSc degree in y"	Qualification certificate	TiiQu
Licensing assurance	"I have license x"	Licence certificate	Chronicled

Medical Health			
Medical record integrity and trust	"Patient x has disease y"	Patient record	Iryo
Assurance of compliance and best practices	"Process x has been computed following y steps"	Y steps	Healthcare compliance
Business			
KYC, AML, anti-fraud	"I am person x"	identity	ING, Sphere
Government			
Polling of electorate (referendums, elections)	"Votes x1...xn sum to y"	votes x1...xn	Stratumn
Direct politics	"Person X does not exist within voting set Y"	Person x	Set membership
Transparency of spending and allocation	"We've spend amount x on y"	amount x	Government transparency
Proof of process	"Y is the result of executing process x"	process x	Government transparency
Tax	"For item x I've paid y tax"	Item x, tax amount y	QED-it
Law Enforcement			
Equality of application of law	"Case x and y result in charge z"	Cases x and y	Book
Verification of data with privacy	"Y is the result of f(x)"	data input x	QED-it
Chain of Custody	"Party X transferred item y to Party Z"	Item y	Chronicled
Integrity of evidence	"Evidence x has not been altered"	Evidence	Stratumn
Presence of suspect in database	"Suspect X has not had a record in Database Y before"	Evidence	ING

Table 4: IAP use cases of ZK Proofs

ZK-SNARKs

The first advanced zero-knowledge protocol type that is utilised by IAP Network are ZK-SNARKs. ZK-SNARKs revolutionized zero knowledge proofs by offering non-interactive proofs that were succinct enough to deploy in digital cash applications. ZK-SNARKs require a somewhat costly set-up phase but the protocol runs efficiently with proof sizes under 200 bytes. There are number-theoretic assumptions ensuring the preservation of the zero knowledge property. After giving a brief formal definition we give a light introduction to these mathematical tools.

Formal Definition of ZK-SNARKs

ZK-SNARK stands for *zero-knowledge succinct non-interactive argument of knowledge*. We follow the formal definition of a ZK-SNARK by Bitanky et al.[A16]. We first give the definition of a ZK-SNARG, which is a weaker notion of a ZK-SNARK (ie. a ZK-SNARK adds a stronger argument for adaptive soundess).

Definition: ZK-SNARG: a *zero-knowledge succinct non-interactive argument* is a triple of algorithms (P,G_V,V). For a security parameter k the verifier runs $G_V$to create a tuple (vgrs, priv), a verifier-generated reference string vgrs and some corresponding verification coins priv. An honest prover P with access to a Turing machine M, the statement x, which is limited by $|x| < t$ steps, and a witness w creates a proof certificate π, which he sends to the verifier V. V then either accepts or rejects π by evaluating x, π utilising his private coins priv. A ZK-SNARG must fulfill the following properties:

1. **Completeness**: for any tuple $(x,w) \in R$,

   $$\Pr[V(x, \pi, \text{priv}) = 1 : (\text{vgrs}, \text{priv}) = G_V, P(x,w,\text{vgrs})= \pi] = 1$$

2. **Succinctness**: the length of the proof certificate π as well as the running time of the verifier, V, is bound by:

   $$p(k + |\pi|) = p(k + |M| + |x| + \log t)$$

where p is a universal polynomial independent of the relation R. Furthermore, $G_V$runs in poly(k) and the length of the parameters of tuple (vgrs, priv) is bound by poly(k).

1. **Soundness**: for non-adaptive soundness: any prover, P^*, $k \in N$ and $x \notin L$:

   $$\Pr[V(x,\pi, \text{priv}) = 1 : (\text{vgrs}, \text{priv}) = G_V, P^*(x, \text{vgrs}) = \pi] \leq \text{negl}(k)$$

For adaptive soundness: any prover, P^*, $k \in N$:

$$\Pr[V(x,\pi, \text{priv}) = 1 : (\text{vgrs}, \text{priv}) = G_V, P^*(\text{vgrs}) = (x, \pi), x \notin L] \leq \text{negl}(k)$$

Definition: ZK-SNARK: a ZK-SNARK is a triple of algorithms (P, G_V, V), where adaptive soundness is replaced by the following stronger argument:

1. **Adaptive proof of knowledge**: for any prover, P^*, there exists an extractor E_P such that for all large $k \in N$ and all auxiliary inputs $z \in \{0,1\}^{\text{poly}(k)}$: $\Pr[(\text{vgrs}, \text{priv}) = G_V, P^* (z, \text{vgrs}) = (x, \pi), V(x, \pi, \text{priv}) = 1 \text{ and } E_P(z, \text{vgrs}) = (x,w), w \notin R(x)] \leq \text{negl}(k)$

Theoretical Realisation of ZK-SNARKs

Vitalik Buterin provides an excellent overview of the theory of ZK-SNARKs in QAPs[42], elliptic curve pairings[43] and ZK-SNARKs[44]. Zcash provides a detailed introduction in a series of blog posts[45]. We present a brief summary in the following paragraphs. An interested reader is referred to previously mentioned references.

We start by using homomorphic hiding, defined in Preliminaries --- Elliptic Curves and Pairings, to compute blind evaluation of polynomials; an integral feature of ZK-SNARKs. Recall that a polynomial $p(x)$ of degree n over a finite field F_p is of the form $p(x) = a_0 + a_1x + ... + a_nx^n$ where $a_i \in F_p$. Note that for $H(x) = g$^{x mod p-1},

$H(ax+by) = g$^{$ax+by$ mod p-1} = g^{ax mod p-1} g^{by mod p-1} = $H(x)^a H(y)^b$.

Suppose now that Alice has a polynomial $p(x)$ of degree n and Bob wishes to learn $H(p(s))$ for some secret s. Without knowing the coefficients of p, Bob can compute the HH of 1, s, s^2, ..., s^n and send these values: $H(1)$, $H(s)$, $H(s^2)$, ..., $H(s^n)$ to Alice. Alice can then compute:

$H(1)^{a_0} H(s)^{a_1} H(s^2)^{a_2} \dots H(s^n)^{a_n} = H(a_0 + a_1x + a_2x^2 + \dots + a_nx^n) = H(p(s))$.

In the Zcash protocol the polynomial $p(x)$ is simply too large to send to Bob and thus blind evaluation of polynomials is employed.

During the blind evaluation of polynomial, Alice knew the coefficients $\{a_i\}_{i=0}{}^n$ and Bob trusted that she sent the correct computation $H(p(s))$ to him. One way for Alice to prove to Bob that she does know the coefficients is with the knowledge of coefficient test (KC) as described in the blog series[45].

Two security goals of this scenario are:

- **Blindness**: Alice does not learn s, Bob does not learn $p(x)$
- **Verifiability**: the probability that Alice can send a value not of the form $H(p(s))$ for some polynomial $p(x)$ of degree n is negligible.

To properly achieve the second property the Knowledge of Coefficient Assumption (KCA) is formally introduced. Suppose now that the HH is $H(x) = xg$ for the same generator g as before. Bob will test Alice to ensure that the final value $H(p(s))$ is of the correct form as follows.

He selects a random field element $\alpha \in F_p$. He computes the HH of $1, s, s^2, \dots, s^n$ as before and sends these values: $H(1) = g$, $H(s) = sg$, $H(s^2) = s^2g$, ..., $H(s^n) = s^ng$ to Alice. In addition, Bob also sends the HH of α, αs, αs^2, ..., αs^n to Alice.

When Alice combines the first set of homomorphic hidings, she recovers $a = P(s)g$. Combining the second set recovers $b = \alpha P(s)g$. Alice sends Bob a,b and he checks to ensure that $a = \alpha b$. Bob accepts only if equality holds.

The problem statement one generally wishes to prove using a ZKP is in a generic form. Statements to run a ZK-SNARK must be in a special polynomial form; the first step being to convert the statement to an arithmetic circuit and the second being translation from an arithmetic circuit to a Quadratic Arithmetic Program (QAP). Once the assignment has been converted to a QAP Alice might wish to prove she has a valid QAP. The Pinocchio Protocol allows Alice to prove to Bob she has a QAP which satisfy some particular constraints. This method requires some interaction between Alice and Bob. Recall that ZK-SNARKS are non-interactive. The final step is to convert the proof to a non-interactive system; this is achieved using elliptic curve pairings, as described in section "Elliptic Curves and Pairings".

Practical Implementations of ZK-SNARKs

ZK-SNARKs have gained both theoretical as well as practical research attention over the last few years. This has resulted in a well-represented set of libraries implementing ZK-SNARKs that have been utilised in applications as itemised above. ZK-SNARKs are widely recognised and used in the blockchain community in Zcash, Ethereum, J.P. Morgan's Quorum blockchain and the IAP among others. Due to the security concerns surrounding the trusted set-up phase and known vulnerabilities to quantum attacks, ZK-STARKs are the natural next step in ZKP technology.

ZK-STARKs

ZK-STARKs are an emerging technology recently proposed by Ben-Sasson et al.[A14]. As detailed earlier, ZK-STARKs are advanced zero-knowledge proof systems which evolved from a long line of research and are superior to their predecessors (eg. ZK-SNARKs, Bulletproofs, Ligero, ...) in terms of efficiency, scalability and cryptographic assumptions. The name ZK-STARK stands for ***zero-knowledge scalable transparent arguments of knowledge***. ZK-STARKs fulfill a variety of properties which promote the use of zero-knowledge proofs for long-term use in real-world applications to ensure computational integrity and verifiability while retaining privacy.

Advanced Properties and Improvements to Related Work

ZK STARK technology improves upon related zero-knowledge proofs using a number of advanced properties:

Transparency: in comparison with ZK-SNARKs, a ZK-STARK does not depend on a trusted setup phase. The security assumptions of ZK-STARKs are entirely based on primitives falling into the category of symmetric key cryptography (ie. Merke trees with collision-resistant hash functions; these hash-functions are typically based on AES[46] constructions). While ZK-SNARKs use elliptic curve pairings and therefore require a secure setup phase for certain parameters, ZK-STARK proofs can be setup entirely transparently. The prover may learn the randomness of the verifier at a certain stage (although not in advance, as otherwise the prover may be able to create a non-sound proof).

Post-Quantum Security: the use of collision-resistant hash functions is currently believed to be post-quantum secure, as there is currently no known efficient algorithm which enables quantum computers to significantly better break symmetric key primitives (than classical computers).

Improved Performance: ZK-STARKs improve upon the communication and computation complexity in regards to other existing (see above) zero-knowledge proof systems. Concretely, in a ZK-STARK the prover achieves quasi-linear time complexity while a verifier achieves polylogarithmic time complexity. ZK-STARKs utilise novel methods to attain improved efficiency and scalability, including innovative components such as Algebraic Intermediate Representation (AIR), Algebraic Linking Interactive Oracle Proof (ALI), Algebraic Placement and Routing (APR), Reed-Solomon Proximity Testing (RPT) and the Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI) protocol; further details to these methods are explained below.

Definitions of ZK-STIKs and ZK-STARKs

Informally, ZK-STIK stands for ***zero-knowledge scalable transparent interactive oracle proof of knowledge***. The following definitions follow the original definitions of ZK-STIKs and ZK-STARKs by Ben-Sasson et al.[A14]:

Consider a non-deterministic machine M, an instance x of a language L and a witness w. M takes a relation R as input, where R consists of all pairs (x,w) such that $x \in L$ and $w \in W$. Furthermore, M runs in time $T(n)$ and uses memory $S(n)$. M either accepts or rejects R depending if a pair $(x,w) \in R$.

Definition: Computational Integrity (CI): the relation R_{CI} is a set of pairs (x,w) where:

- $x = (M, a, b, T, S)$, with M a non-deterministic Turing machine, a the input, b the output thereof and T the time and S the memory bounds.
- w is a description of non-deterministic choices of M from *input* a such that it leads to outcome b using $\leq T$ steps, and using a memory tape of size of most S.

A computational integrity language L_{CI} is defined as:

$L_{CI} = \{x = (M, a, b, T, S) \mid \exists\, w, (x, w) \in R_{CI}\}$.

Definition: Interactive Oracle Proof (IOP): Let R be a relation as defined above and similarly L be an induced language of such a relation. $\varepsilon \in [0,1]$ denotes the soundness error. An interactive oracle proof (IOP) system for relation R, language L and soundness error ε is a pair of randomised interactive algorithms (P,V) which satisfy the following properties:

- *operation*: the verifier (V) has input x, while the prover (P) has the input pair (x,w). The prover and verifier exchange messages in several rounds ($r(x)$ denotes the round complexity). The prover starts to send a message to which the verifier is given oracle access; the verifier responds to these messages. We denote with $\langle \mathrm{P}(x,w) \leftrightarrow V(x) \rangle$ the output of the verifier; this output is either *accept* or *reject*
- **completeness**: if $(x,w) \in R$ then $\Pr[\langle P(x,w) \leftrightarrow V(x) \rangle = accept] = 1$
- **soundness**: if $x \notin L$ then $\Pr[\langle P(x,w) \leftrightarrow V(x) \rangle = accept] \leq \varepsilon$

Definition: Zero-Knowledge Scalable Transparent Interactive Oracle Proof of Knowledge (ZK-STIK): Let R, L, P, V be as defined in the above definition of IOP, then we say a ZK-STIK is a system which satisfies the following properties:

- *transparency*: all messages and queries by the verifier are public random coins.

- *fully scalable*: for every instance ***x***, both conditions must hold:
    - *scalable verifier*: $T_V = \text{poly}(\ n, \log T(n), \log 1/\varepsilon(n)\ )$
    - *scalable prover*: $T_P = T(n)\ \text{poly}(\ n, \log T(n), \log 1/\varepsilon(n)\ )$
- *proof of knowledge*: any prover P^* that is given oracle access to a random extractor E and by existence of a knowledge error function $\varepsilon'(n) \in [0,1]$, causes the verifier to accept an instance ***x*** with probability $p(n) > \varepsilon'(n)$, outputs in time $\text{poly}(\ T(n)/p(n) - \varepsilon'(n)\ )$ a witness ***w*** such that $(x,w) \in R$
- *privacy preservation*: there exists a randomised simulator ***Sim*** which runs in time poly($T(n)$) and samples (perfectly) the distribution on transcripts of interactions between prover ***P*** and verifier ***V***.

The definition of a ZK-STIK holds in a theoretical setting where the prover is not limited by his computational power. In any real-world setting, proofs of knowledge are transformed into arguments of knowledge. In such a system a prover is computationally bounded. A ZK-STARK is the natural transformation of a ZK-STIK in such an environment. As a reminder, ZK-STARK stands for ***zero-knowledge scalable transparent arguments of knowledge***. ZK-STARKs are defined in two settings:

Definition: Interactive ZK-STARK (iZK-STARK): Let ***S*** = (***P***,***V***) be a system satisfying the definition of a ZK-STIK. An interactive ZK-STARK is a ZK-STIK system ***S*** which is converted into a ZK-STARK by utilising a family of collision-resistant hash functions.

As such, a ZK-STIK system with perfect ZK achieves computationally ZK under an iZK-STARK.

Definition: Non-interactive ZK-STARK(niZK-STARK): Let ***S*** = (***P***,***V***) be a system satisfying the definition of a ZK-STIK. A non-interactive ZK-STARK is a ZK-STIK system ***S*** which is converted into a ZK-STARK by utilising the method described Ben-Sasson et al.[A14].

Transformation from a High-level Problem Statement to a ZK-STARK

Given a high level statement we outline the necessary steps to transform such a statement into a ZK-STARK. A prover and verifier algorithm can then take this protocol and execute it.

Let ***C*** denote a function and ***a*** be the input to such a function. This function can represent any computational decision problem included in the computational complexity class of NP (following the universality property of ZK proofs). Concretely, ***C*** and ***a*** follow any of the three listed conventions:

1. Public function ***C*** and private input ***a***.
2. Private function ***C*** and public input ***a***.
3. Private function ***C*** and private input ***a***.

Ben-Sasson et al.[A14] states the following definition of a high level statement that is to be proven by a prover to a verifier:

(1) α is the result of executing C for T steps on (public) input x

Following our notations, α is the output ***b*** (see the above definition of CI); ***x*** is the input ***a***. Such a computational decision problem is then transformed into a mathematical problem via the steps outlined below. The mathematical 'language' which is used to iterate through ZK-STARKs are polynomials. The computational problem defined as function ***C*** is converted to a polynomial representation $P(x)$. Following this convention we aim to satisfy the following equation:

(2) $C(P(x)) = Z(x) * D(x)$

Where $C(x)$ denotes a constraint checking polynomial and is (typically) defined as $C(x) = (x-1) * (x-2) * \ldots * (x-t)$ for any integer ***t***. The goal of such a definition is to ensure that the constraint checking polynomial is zero (ie. $C(P(x)) = 0$). We define a domain range polynomial denoted as $Z(x)$. Consequently, $Z(x)$ is defined over the range where we want to prove that the polynomial $P(x)$ holds. In other words $Z(x)$ is defined as $Z(x) = (x-1) * (x-2) * \ldots * (x-r)$, with ***r*** an integer bounding the maximum range of the polynomial $Z(x)$. The polynomial $D(x)$ is a natural multiple of $Z(x)$. ($D(x)$can be computed via division from *C*(*P*(*x*)) divided by *Z*(*x*)). This 'mathematical trick' holds if we can prove that the polynomial *P*(*x*) is of low degree; in this context, "low degree" means the degree is significantly smaller than the field size in which the polynomials are defined[A14].

Transformation of *C* to *P*(*x*)

The following steps outline the procedure to convert a high level program *C* into a polynomial *P*(*x*):

1. The starting point is a program *C* written in, for example, a high level language such as C[47] (the programming language).
2. Utilise a compiler (see Ben-Sasson et al.[A18] and Ben-Sasson et al.[A21]) to transform program *C* into TinyRAM[A19] code.

Utilising the libstark library[48] the existing TinyRAM assembly code is transformed via the following steps:

3. Utilise libstark library to convert TinyRAM code to a constraint system.
4. Utilise libstark library to convert constraint system into a binary algebraic intermediate representation (BAIR).

Following the steps outlined by Ben-Sasson et al.[A14] (and implemented in the libstark library) a ZK-STARK is then generated and evaluated via the following steps:

5. Represent a problem in the algebraic intermediate representation (AIR)[49]. The representation of the computational problem *C* into polynomials *P*(*x*) can be done manually or via the steps outlined above. Utilising the libstark library this transformation is a continuous process of no concern to the user.
6. Transform AIR format into algebraic placement and routing (APR) format.
7. Reformulate APR to reed-solomon proximity testing (RPT) problem
8. Utilise fast reed-solomon interactive oracle proof of proximity (FRI) protocol to evaluate the resulting ZK-STARK[50].

Underlying Protocols and Steps of ZK-STARKs

This section outlines the steps discussed previously in a more detailed manner. As a starting point we assume a computational problem of form (1) which is stated in AIR representation. Via the steps outlined below the computational decision problem (statement *x* and witness *w*) is reduced to a Reed-Solomon Proximity Testing problem which can then be solved by the FRI protocol.

Algebraic Intermediate Representation (AIR): the starting point is a natural AIR representation of an instance of a computational integrity problem *x* and a witness *w*, denoted as x_{AIR} and w_{AIR}.

x_{AIR} is specified by (i) a transition relation over a space of machines states and (ii) a set of boundary constraints (like inputs and outputs).

w_{AIR} is specified as a valid execution of the computation, given by an execution trace. w_{AIR} is a sequence of machine states that ensure boundary and transition constraints.

Each AIR instance (x_{AIR} and w_{AIR}) is a polynomial representation over variables (inputs and outputs). An 'algebraic' transition relation is specified over the set of those polynomials representing the 'current' and 'next' state of an execution trace. An execution trace is defined as a sequence of elements of a field *F*.

Algebraic Placing and Routing (APR): the process of APR is the reduction from an AIR instance to an APR instance/witness pair (x_{AIR}, w_{AIR}). During this process, states of the execution trace are 'placed' on nodes of an affine graph in such a way that two consecutive states are connected by an edge. This process is similar to placing and routing in computer and circuit design, however; the APR process is bound by algebra rather than physical entities.

Algebraic Linking Interactive Oracle Proofs (ALI): the protocol that describes a further reduction of a APR instance pair (x_{AIR}, w_{AIR}) to an instance of a Reed-Solomon Proximity Testing pair (x_{RPT}, w_{RPT}) is referred to as ALI. The reduction is done via a 1-round IOP. The randomness of the verifier is used to 'link' the constraints of the transition relation into a single (random) one.

Reed-Solomon Proximity Testing (RPT): the instances (x_{RPT}, w_{RPT}) are now represented in the form of a Reed Solomon (RS) code. Via the ALI protocol the numerous constraints of a RS code are linked together into a single code. Such code can then be evaluated (for its low degree testing) by the following FRI protocol.

Fast-Reed Solomon Interactive Oracle Proof of Proximity (FRI): the FRI protocol represents a novel interactive oracle proof of proximity (IOPP) for RS codes. The protocol was proposed by Ben-Sasson et al.[A14] and is a major part of ZK-STARKs. The FRI protocol resembles the Fast Fourier transform and is the first RS-IOPP protocol which achieves strictly linear arithmetic complexity for the prover and strictly logarithmic arithmetic complexity for the verifier.

Usage of a ZK-STARK

ZK-STARKs are useful in verifying computation; especially in cases where there is incentive to report incorrect computational results. A ZK-STARK provides computational integrity while ensuring privacy of the computation.

Zero-knowledge proofs in general can be applied to any computational decision problem in the complexity class NP. The result of the computation is an ***accept*** or ***reject*** state. We classify high level computational problems into the following groups:

1. Public function and private input.
2. Private function and public input.
3. Private function and private input.

Usage for each of these classes are motivated as follows:

1. Outsourced computations --- the typical cloud computing scenario, where the evaluated function is public yet, the prover may want to keep their inputs private;
2. Machine learning operations, where the function is private and the inputs are typically public - the prover's competitiveness may come from having a private algorithm to evaluate a function, whilst the inputs may be publicly available;
3. A combination of both of the above scenarios.

Efforts to standardise zero-knowledge proofs aim to create a standardised interface for zero-knowledge proofs. ZKProof.org lists a range of documents[51] that outline potential applications[52] for ZK-STARKs. Examples include: commitment schemes, signature schemes, set membership proofs or proofs of identity.

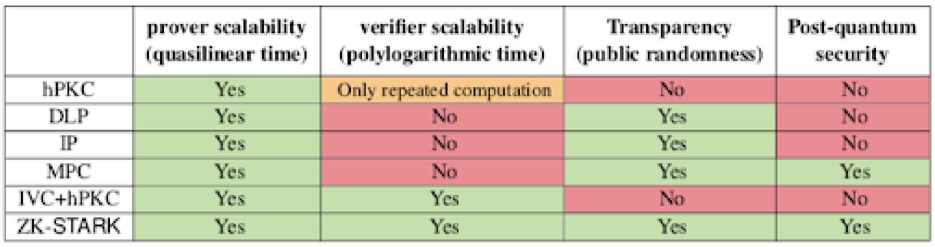
The IAP sets out mechanisms to use ZK-STARKs to provide computational verification predominantly in the context of privately executed functions. In this context the proofs pertaining to private functions or private input data can be publicly verified to be true. For further applications of ZK-STARKs within the IAP we refer to the sections IAP Network Use Cases and Use Cases.

Performance Evaluation of ZK-STARKs

ZK-STARKs have been evaluated with regards to their performance in a theoretical and practical 'apples-to-apples' fashion by Ben-Sasson et al.[A14]. Adam Luciano provides an additional analysis of the benchmarking results by Ben-Sasson in this blog post[53].

ZK-STARKs dominate the current state-of-the-art transparent zero-knowledge proof systems in terms of ***prover complexity*** (10x faster) and with regards to large-scale computations a ZK-STARK dominates in terms of ***verification time*** and *communication complexity*.

The following table presents the results of the theoretical evaluation of ZK-STARKs in comparison to homomorphic public-key cryptography (hPKC) schemes by Ishai et al.[54], discrete logarithm problem (DLP) scheme by Groth[55], interactive proofs, also known as proofs for muggles (IP) by Goldwasser et al.[56], MPC-in-head (MPCh) schemes by Ishai et al.[57], and incrementally verifiable computation (IVC) schemes by Valiant[58].



	prover scalability (quasilinear time)	verifier scalability (polylogarithmic time)	Transparency (public randomness)	Post-quantum security
hPKC	Yes	Only repeated computation	No	No
DLP	Yes	No	Yes	No
IP	Yes	No	Yes	No
MPC	Yes	No	Yes	Yes
IVC+hPKC	Yes	Yes	No	No
ZK-STARK	Yes	Yes	Yes	Yes

Table from Ben-Sasson et al.[A14]

The following graphs present concrete numbers of an evaluation of specific zero-knowledge system implementations of the classes discussed before. Concretely, the ZK-STARKs as implemented in the libSTARK library[59] are evaluated against the libSNARK library, the SCI implementation[A13], the BCCGP implementation[A15] and the Ligero implementation[A12]. ZK-STARKs are evaluated in terms of prover time complexity (left), verifier time complexity (middle) and communication complexity (right). As mentioned before, ZK-STARKs clearly dominate in terms of prover time complexity. For large-scale problem statements ZK-STARKs also dominate against the compared zero-knowledge proof systems.

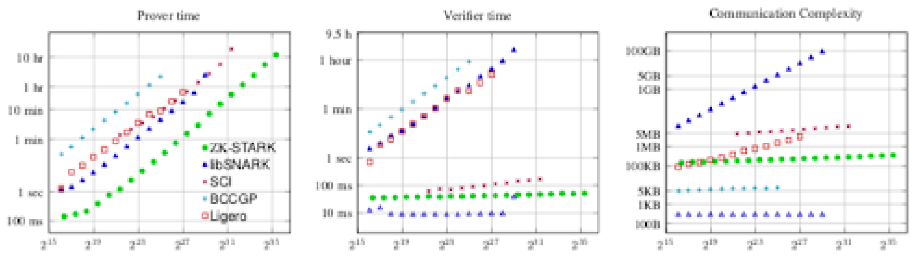


Figure from Ben-Sasson et al.[A14].

For a more detailed evaluation we refer to the original whitepaper about ZK-STARKs by Ben-Sasson et al.[A14].

ZK-SHARKs

ZK-SHARKs, short for zero-knowledge succinct hybrid arguments of knowledge, were recently proposed by Virza el al.[A34]. and are a hybrid between ZK-SNARKs and ZK-STARKs. Typically, zero-knowledge proofs achieve two out of three desired properties for large-scale circuits: (a) fast (milliseconds) verifier; (b) short proofs (kilo-bytes); and (c) a trusted setup phase for soundness or zero-knowledge. ZK-SHARKs combine the fast verification of zk-SNARKs with the no-trusted-setup of some non-succinct Non-interactive Zero-Knowledge Proofs. Since this type of proof system was just recently proposed, IAP is actively monitoring its progress and will deploy ZK-SHARKs in Plato, IAPs Mainnet, when these proofs reach a certain level of maturity.

Active Research and Development

At the time of writing IAP Network is using state-of-the-art zero-knowledge protocols proposed by academic research such as ZK-SNARKs and ZK-STARKs, continuing the active area of research by the IAP team in zero-knowledge proof systems. IAP Network is establishing a benchmarking framework to compare its own implementation of ZK-SNARKs and ZK-STARKs to other ZK libraries and frameworks as mentioned in section "Existing Libraries and Implementations", the results of which will be shared under the IAP Open Standard. This framework comprises a common functionality (use cases) to be implemented while performing an 'apples-to-apples' comparison between the frameworks and various graphs to compare the data points gained from the benchmarking. The IAP Network team is comparing the ZK frameworks over a range of computational devices and settings in order to gain insights of the performance of ZK proof systems in these environments.

From these insights the team will propose improvements of current state-of-the-art ZK frameworks. These improvements shall include optimised software implementations, support of a multitude of programming languages[60] as well as improvements to the underlying cryptography of the zero-knowledge proofs.

IAP Network strongly believes these contributions can enhance user satisfaction and user privacy. The team is actively working towards a trustworthy, privacy-preserving, verifiable and transparent future.

Conclusion

In this document we set out to illustrate the computational verification techniques utilised within the IAP. IAP Network utilises zero-knowledge proofs to verify off-chain computations which otherwise can not be proven to be correct. The incentive of off-chain computations in the first place is to preserve the privacy of said computation, which traditionally causes the issue that third parties cannot verify the computation; having instead to confront the problem of trust. This document has introduced and surveyed existing PoK systems --- these systems increasingly evolve and provide additional features. The academic research community has made some breakthroughs, adding advanced properties such as transparency, post-quantum security and improved efficiency and scalability to improve on legacy systems. Research progress has evolved to a point where practical solutions are available. Multiple libraries have been implemented and published to drive the real-world adaption of zero-knowledge proofs in various applications. The IAP Network is utilising zero-knowledge proofs in a variety of applications. ZK-SNARKs and ZK-STARKs are examples of current state-of-the-art advanced zero-knowledge proofs. ZK-SNARKs are very efficient zero-knowledge proof systems, yet these systems rely on a trusted set-up phase and elliptic curve cryptography, which is not secure against post-quantum attacks. IAP Network is actively researching additional options, including implementing ZK-STARKs, a successor of ZK-SNARKs which improves on previous work by removing the set-up phase, being transparent, in some scenarios is more efficient, includes scalability, and more desirable cryptographic assumptions that are post-quantum secure.

Copyright

Disclaimer

References

[1] "StarkWare closes $30M Series A led by Paradigm - Medium." 28 Oct. 2018, https://medium.com/starkware/series-a-665d94c855f4.

[2] "ING launches Zero-Knowledge Range Proof solution, a major addition ...." 16 Nov. 2017, https://www.ingwb.com/themes/blockchain-articles/ing-launches-major-addition-to-blockchain-technology.

[3] "Factorization of a 768-bit RSA modulus - Cryptology ePrint Archive." 12 Dec. 2009, https://eprint.iacr.org/2010/006.pdf.

[4] From just looking at n finding the two primes p and q is considered to be computationally hard, especially if p and q are large. Knowledge of p or q allows someone to easily find the other by a simple division operation.

[5] With the prefix 'computational' the security level of the statement is given. We further define the different levels in Section “Variants of Zero-Knowledge Proofs” of this paper, page 24; briefly, we consider the following levels: perfect, statistical and computational

[6] Peggy and Victor may easily halt a proof by stopping to respond to the protocol.

[7] In an secure zero-knowledge protocol, if any party deviates from the protocol, this leads to denial of service attacks. But such behaviour doesn’t give Peggy or Victor the advantage to learn sensitive information or convince the other party of statements that are not true.

[8] "Elliptic Curve Cryptosystems - Semantic Scholar." https://pdfs.semanticscholar.org/dcc3/24cc5b90ef070dda6d06d9e360b2fba95894.pdf.

[9] "Elliptic Curve Cryptography: a gentle introduction - Andrea Corbellini." 17 May. 2015, https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/.

[10] "Zcash." https://z.cash/.

[11] "Explaining SNARKs Part I: Homomorphic Hidings - Zcash." 28 Feb. 2017, https://z.cash/blog/snark-explain/.

[12] For more details, see Explaining SNARKs Part V.

[13] "A first glimpse of cryptography's Holy Grail - ACM Digital Library." 1 Mar. 2010, https://dl.acm.org/citation.cfm?id=1666445.

[14] See http://sealcrypto.org, https://github.com/shaih/HElib, https://github.com/CryptoExperts/FV-NFLlib, https://git.njit.edu/groups/palisade, https://github.com/vernamlab/cuHE, https://github.com/vernamlab/cuFHE, https://github.com/kimandrik/HEAAN, https://tfhe.github.io/tfhe/.

[15] Information Assurance Toolbelt - see “IAP Techincal White Paper” https://github.com/iapnetwork/docs/blob/master/whitepapers/technical-whitepaper.md.

[16] "Database Hardening Best Practices - UC Berkeley Security." https://security.berkeley.edu/resources/best-practices-how-articles/system-application-security/database-hardening-best-practices.

[17] This does not exist in the real-world, since all computational devices have realistic restrictions such as memory and time restrictions; yet, for the sake of our purpose, we consider ‘idealistic’ entities and prove that systems are secure under these assumptions.

[18] Meaning for multiple executions it yields always the same result.

[19] Typically a Turing machine; all entities are, in general, represented as an algorithm rather than a physical entity, yet, a physical entity can interact with the machine to provide inputs and receive outputs

[20] A special case of AM are MA (Merlin-Arthur) proofs, in which there is only one message transferred from the prover Merlin to the verifier Arthur. This leads to a non-interactive protocol, which is further discussed below. MA proofs always start with Merlin sending his message to Arthur, and since Arthur only executes following Merlin, Merlin does not learn the randomness of Arthur in his turn.

[21] "Common reference string model - Wikipedia." https://en.wikipedia.org/wiki/Common_reference_string_model.

[22] "Random oracle - Wikipedia." https://en.wikipedia.org/wiki/Random_oracle.

[23] "Fiat–Shamir heuristic - Wikipedia." https://en.wikipedia.org/wiki/Fiat–Shamir_heuristic.

[24] See the section "Probabilistic Checkable Interactive Proofs (PCIP)" in this paper, page 20.

[25] See the section "Linear Probabilistic Checkable Proofs (LPCP)" in this paper, page 20.

[26] "PCP theorem - Wikipedia." https://en.wikipedia.org/wiki/PCP_theorem.

[27] For the interested reader, PCPs allow problems pertaining to the class of NEXP to be verified. https://complexityzoo.uwaterloo.ca/Complexity_Zoo:N#nexp.

[28] "Graph homomorphism - Wikipedia." https://en.wikipedia.org/wiki/Graph_homomorphism.

[29] More sophisticated PoK systems such as ZK-SNARKs and ZK-STARKs reformulate a high-level problem into polynomials via a process of arithmetization in which the high level code is reduced by addition and multiplication circuits.

[30] "Shor's algorithm - Wikipedia." https://en.wikipedia.org/wiki/Shor's_algorithm.

[31] "ZKProof Standards." https://zkproof.org/.

[32] "Documents - ZKProof Standards." https://zkproof.org/documents.html.

[33] "ZSL · jpmorganchase/quorum Wiki · GitHub." https://github.com/jpmorganchase/quorum/wiki/ZSL.

[34] "GitHub - ConsenSys/zero-knowledge-proofs: Zero Knowledge Proofs ...." https://github.com/ConsenSys/zero-knowledge-proofs.

[35] "GitHub - Sobuno/ZKBoo." https://github.com/Sobuno/ZKBoo.

[36] "GitHub - IAIK/gzkbpp: Implementation of the ZKB++ proof system." https://github.com/IAIK/gzkbpp.

[37] "GitHub - Microsoft/Picnic: Reference implementation of the Picnic post ...." https://github.com/Microsoft/Picnic.

[38] See https://www.pepper-project.org/summary-systems.htm --- pepper, zebra, buffet, pantry, zaatar, allspice and ginger, https://github.com/pepper-project/zebra, https://github.com/xlab-si/emmy, https://github.com/IAIK/gzkbpp, https://github.com/cryptobiu/libscapi, https://github.com/dalek-cryptography/zkp, https://github.com/Lamden/ez-zk, https://github.com/amiller/python-zk-proofs, https://archive.codeplex.com/?p=vc.

[39] "first workshop - ZKProof Standards." https://zkproof.org/first_workshop.html.

[40] "ZKProof Standards Applications Track Proceedings." 1 Aug. 2018, https://zkproof.org/proceedings-snapshots/zkproof-applications-20180801.pdf.

[41] "IAP Whitepaper - Information Assurance Platform (IAP)." https://cdn.iap.network/iap-whitepaper.pdf.

[42] "Quadratic Arithmetic Programs: from Zero to Hero – Vitalik Buterin ...." 11 Dec. 2016, https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649.

[43] "Exploring Elliptic Curve Pairings – Vitalik Buterin – Medium." 15 Jan. 2017, https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627.

[44] "Zk-SNARKs: Under the Hood – Vitalik Buterin – Medium." 2 Feb. 2017, https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6.

[45] "Explaining SNARKs Part I: Homomorphic Hidings - Zcash." 28 Feb. 2017, https://z.cash/blog/snark-explain/.

[46] "Advanced Encryption Standard - Wikipedia." https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

[47] "C (programming language) - Wikipedia." https://en.wikipedia.org/wiki/C_(programming_language).

[48] "GitHub - elibensasson/libSTARK: a library for zero knowledge (ZK ...." https://github.com/elibensasson/libSTARK.

[49] Ben-Sasson starts with the representation of a computational problem in the algebraic intermediate representation format.

[50] ie. prove that the resulting polynomial P(x) is of low degree.

[51] "Documents - ZKProof Standards." https://zkproof.org/documents.html.

[52] "ZKProof Standards Applications Track Proceedings." 1 Aug. 2018, https://zkproof.org/proceedings-snapshots/zkproof-applications-20180801.pdf.

[53] "ZK-STARKs — Create Verifiable Trust, even against Quantum ...." 25 Jun. 2018, https://medium.com/coinmonks/zk-starks-create-verifiable-trust-even-against-quantum-computers-dd9c6a2bb13d.

[54] "Efficient Arguments without Short PCPs∗ - UCLA CS." http://web.cs.ucla.edu/~rafail/PUBLIC/79.pdf.

[55] "Efficient zero-knowledge arguments from two ... - ACM Digital Library." 4 Dec. 2011, https://dl.acm.org/citation.cfm?id=2184052.

[56] "Delegating Computation: Interactive Proofs for Muggles - Electronic ...." https://eccc.weizmann.ac.il/report/2017/108/download/.

[57] "Zero-knowledge from secure multiparty ... - ACM Digital Library." 11 Jun. 2007, https://dl.acm.org/citation.cfm?id=1250794.

[58] "Incrementally verifiable computation or proofs of ... - ACM Digital Library." 19 Mar. 2008, https://dl.acm.org/citation.cfm?id=1802616.

[59] "GitHub - elibensasson/libSTARK: a library for zero knowledge (ZK ...." https://github.com/elibensasson/libSTARK.

[60] For details, see the supported SDKs in the IAP technical whitepaper (#Architectural Concept)

[A1] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger byzantium. Tech. rep. (2018)

[A2] The IAP, “IAP Techincal White Paper” https://github.com/iapnetwork/docs/blob/master/whitepapers/technical-whitepaper.md#the-iap.

[A3] CyberShield, proof of computation - see “IAP Techincal White Paper” https://github.com/iapnetwork/docs/blob/master/whitepapers/technical-whitepaper.md#cybershield.

[A4] IAP Chains - see “IAP Techincal White Paper” https://github.com/iapnetwork/docs/blob/master/whitepapers/technical-whitepaper.md#information-assurance-toolbelt.

[A5] IAP Client - see “IAP Techincal White Paper” https://github.com/iapnetwork/docs/blob/master/whitepapers/technical-whitepaper.md#information-assurance-toolbelt.

[A6] IAA Node - see “IAP Techincal White Paper” https://github.com/iapnetwork/docs/blob/master/whitepapers/technical-whitepaper.md#information-assurance-toolbelt.

[A7] IAP Core - see “IAP Techincal White Paper” https://github.com/iapnetwork/docs/blob/master/whitepapers/technical-whitepaper.md#information-assurance-toolbelt.

[A8] CyberChain, proof of procedure - see “IAP Techincal White Paper” https://github.com/iapnetwork/docs/blob/master/whitepapers/technical-whitepaper.md#cyberchain.

[A9] "Blockliance" https://blockliance.io/.

[A11] Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, Report 2016/492 (2016), https://eprint.iacr.org/2016/492.

[A12] Ames, S., Hazay, C., Ishai, Y., and Venkitasubramaniam, M. 2017. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). ACM, New York, NY, USA, 2087-2104. DOI: https://doi.org/10.1145/3133956.3134104, https://acmccs.github.io/papers/p2087-amesA.pdf.

[A13] Ben-Sasson, E., Ben-Tov, I., Chiesa, A., Gabizon, A., Genkin, D., Hamilis, M., Pergament, E., Riabzev, M., Silberstein, M., Tromer, E., Virza, M.: Computational integrity with a public random string from quasi-linear pcps. Cryptology ePrint Archive, Report 2016/646 (2016), https://eprint.iacr.org/2016/646.

[A14] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018), https://eprint.iacr.org/2018/046.

[A15] Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. Cryptology ePrint Archive, Report 2016/263 (2016), https://eprint.iacr.org/2016/263.

[A16] Bitansky, N., Canetti, R., Chiesa, A., and Tromer, E.: 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12). ACM, New York, NY, USA, 326-349. DOI=<http://dx.doi.org/10.1145/2090236.2090263 https://dl.acm.org/citation.cfm?id=2090263>.

[A17] Ben-Sasson,E.,Chiesa,A.,Gabizon,A.,Riabzev,M.,Spooner,N.:Interactive oracle proofs with constant rate and query complexity. Cryptology ePrint Archive, Report 2016/324 (2016), https://eprint.iacr.org/2016/324.

[A18] Ben-Sasson,E.,Chiesa,A.,Genkin,D.,Tromer,E.,Virza,M.:Snarksforc:Verifying program executions succinctly and in zero knowledge. Cryptology ePrint Archive, Report 2013/507 (2013), https://eprint.iacr.org/2013/507.

[A19] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Tinyram architecture specification. Tech. rep. (2013) https://www.scipr-lab.org/doc/TinyRAM-spec-0.991.pdf.

[A20] Ben-Sasson E., Chiesa A., Spooner N. (2016) Interactive Oracle Proofs. In: Hirt M., Smith A. (eds) Theory of Cryptography. TCC 2016. Lecture Notes in Computer Science, vol 9986. Springer, Berlin, Heidelberg. https://link.springer.com/chapter/10.1007/978-3-662-53644-5_2.

[A21] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879 (2013), <https://eprint.iacr.org/2013/879 https://eprint.iacr.org/2013/879.pdf>.

[A22] Blum, M., De Santis, A., Micali, S., & Persiano, G. (1991). Noninteractive zero-knowledge. SIAM Journal on Computing, 20(6), 1084-1118.

[A23] Camenisch, J., & Chaabouni, R. (2008, December). Efficient protocols for set membership and range proofs. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 234-252). Springer, Berlin, Heidelberg. https://github.com/ing-bank/zkrangeproof.

[A24] Fiore, D., Gennaro, R., & Pastro, V. (2014, November). Efficiently verifiable computation on encrypted data. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (pp. 844-855). ACM.

[A25] Gentry, C., & Boneh, D. (2009). A fully homomorphic encryption scheme (Vol. 20, No. 09). Stanford: Stanford University.

[A26] Gentry, C., & Halevi, S. (2011, May). Implementing gentry's fully-homomorphic encryption scheme. In Annual international conference on the theory and applications of cryptographic techniques (pp. 129-148). Springer, Berlin, Heidelberg.

[A27] Goldwasser, S., Micali, S., & Rackoff, C. (1989). The knowledge complexity of interactive proof systems. SIAM Journal on computing, 18(1), 186-208. https://epubs.siam.org/doi/pdf/10.1137/0218012.

[A28] Goldreich, O., Micali, S., and Wigderson, A.: 1991. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. J. ACM 38, 3 (July 1991), 690-728. DOI: https://doi.org/10.1145/116825.116852.

[A29] Ishai Y., Weiss M. (2014) Probabilistically Checkable Proofs of Proximity with Zero-Knowledge. In: Lindell Y. (eds) Theory of Cryptography. TCC 2014. Lecture Notes in Computer Science, vol 8349. Springer, Berlin, Heidelberg. https://link.springer.com/chapter/10.1007/978-3-642-54242-8_6.

[A30] Koblitz, N. (1987). Elliptic curve cryptosystems. Mathematics of computation, 48(177), 203-209.

[A31] Kalai Y.T., Raz R. (2008) Interactive PCP. In: Aceto L., Damgård I., Goldberg L.A., Halldórsson M.M., Ingólfsdóttir A., Walukiewicz I. (eds) Automata, Languages and Programming. ICALP 2008. Lecture Notes in Computer Science, vol 5126. Springer, Berlin, Heidelberg. https://link.springer.com/chapter/10.1007/978-3-540-70583-3_44.

[A32] Micciancio, D.: 2010. A first glimpse of cryptography's Holy Grail. Commun. ACM 53, 3 (March 2010), 96-96. DOI: https://doi.org/10.1145/1666420.1666445.

[A33] Stinson, D. R. (2005). Cryptography: theory and practice. CRC press.

[A34] Virza M., Tromer E., Raykova M., "zk-SHARKs - Combining succinct verification and public-coin setup." 10 Apr. 2019, https://dci.mit.edu/zksharks.