A REVIEW OF ZK-SNARKS AND THEIR USE IN CRYPTOCURRENCIES

JEFFREY QUESNELLE

Abstract. At the heart of cryptosystems that provide authenticity guarantees are arguments given by the prover that they are in possession of some secret quantity. Since the authenticity guarantee requires that the quantity remain secret, the argument should not reveal any clues that would allow an attacker to infer the secret. We review how this concept can be efficiently generalized into zk-SNARKs (*zero-knowledge succinct non-interactive arguments of knowledge*) and summarize their use in cryptocurrencies.

Contents

University of Michigan-Dearborn
E-mail address: jfquesne@umich.edu.

1. Introduction

The advent of public-key cryptosystems [DH76] has issued in the age of ubiquitous authenticated cryptography. Although more efficient systems (in terms of computation time and space) have been developed [RSA78, HPS98], the underlying principles have remained relatively unchanged. To encrypt message M, the sender generates a key pair (e, d) and communicates d to the receiver out-of-band. The sender, equipped with a function f such that $C = f(e, M)$ and $M = f(d, C)$ transmits C to the receiver. The receiver, now in possession of d and C and equipped with f can recover M. In practice, this process should be efficient. For such a system to work, f and the relationship between (e, d) must be carefully constructed and mathematically verified. Crucially, given d and C it must be exceedingly difficult to determine e.

Consider if instead the sender merely wishes to attest that they are in possession of e. Then, they may encrypt a known M (e.g. a digest of a larger message). A verifier can use d to check that the resulting ciphertext does indeed decrypt back to M. Cryptocurrencies use this property in various ways. For example, in Bitcoin a transaction occurs when the owner of coins signs a hash of the previous transaction of the coins along with the public key of the intended recipient [Nak09]. In essence, the sender has given a *witness* or *proof* that they are in possession of e without revealing any information about e itself.

It is reasonable to ask if this concept can be generalized. We review zk-SNARKs, which are non-interactive, zero-knowledge witnesses for arbitrary problems in NP. To put it another way, zk-SNARKs are digital signatures that prove possession of the input to an arbitrary algorithm which shall produce some desired output. This ability has proven useful for creating succinct, auditable transaction ledgers in cryptocurrencies while maintaining privacy.

The organization of this review is as follows: Section 2 provides definitions for the theoretical foundation of zk-SNARKS, Section 3 builds on these definitions to describe zk-SNARKs, Section 4 reviews constructions of zk-SNARKs, and Section 5 reviews applications of zk-SNARKs in cryptocurrencies.

2. Witnesses, knowledge, and proofs

2.1. **Interactive proof systems.** [GMR89] introduced the concept of *interactive proof systems*. In an interactive proof system, two Turing machines share a pair of common tapes over which they can communicate. Each machine shares the same input but is equipped with a unique random tape and a private worker tape. This setup is known as an *interactive protocol*. One machine, the verifier, is limited to a polynomial amount of work on a given input, but may make use of the output of the prover machine in addition to allowing for an arbitrarily small amount of error given a sufficient input length.

Definition 1. Let $L \subseteq \{0,1\}^*$ and (P, V) be an interactive protocol. Then, (P, V) is an *interactive proof system* if:

(1) (Completeness) For each k and sufficiently large $x \in L$ as input to (P, V), V accepts with probability at least $1 - |x|^{-k}$.
(2) (Soundness) For each k, interactive Turning machine P', and sufficiently large $x \notin L$ as input to (P', V), V halts and accepts with probability at most $|x|^{-k}$.

If such machines exist, then the language is said to be in the interactive polynomial time (IP) class. For IP to be interesting, it should contain some languages not in NP. Otherwise, the interactivity would be merely a novelty; the language's membership in NP guarantees the existence of a non-interactive polynomial verifier. Several problems not known to be in NP (such as graph non-isomorphism) were shown to be in IP [GMW86], which tells us that IP is more expressive than NP. In a celebrated result, IP = PSPACE, from which we can conclude that when randomization

and interaction are allowed, the proofs that can be verified in polynomial time are exactly those proofs that can be generated with polynomial space [Sha92].

Recall the presence of the k error in the above definition, wherein the verifier V will accept strings not in L at a probability at most $\frac{1}{k}$ when connected to an arbitrary "prover" machine P'. Thus, the verifier must be resilient even to a malicious prover with unlimited power, maintaining its soundness guarantee even when facing an adversary with a decided computational advantage.

Definition 2. An interactive proof system is an *interactive argument* if it is an interactive proof system with the soundness guarantee relaxed to

(2) (Soundness) For each k, interactive probabilistic polynomial time Turning machine P', and sufficiently large $x \notin L$ as input to (P', V), V halts and accepts with probability at most $|x|^{-k}$.

Interactive arguments, also known as computationally-sound proof systems or argument systems, provide interesting gains in the expressiveness of the proof systems within the class. In particular, all languages in NP have interactive arguments [BCC88]. Moreover, this class of languages with argument systems maintains this property when the complexity of the interactivity is bounded to polylogarithmic size, while interactive proof systems do not [GH98].

It is important to note that interactive arguments merely speak to "convincing" the verifier that $x \in L$. For example, a prover in the problem of circuit satisfiability may convince a verifier that a satisfying assignment exists, but it does not necessarily mean that the prover actually "knows" the satisfying assignment.

Definition 3. An interactive argument (P, V) is a *proof of knowledge* if, for accepting instance $x \in L$, there exists a polynomial time machine E that can extract a witness for w from P.

The above definition lacks mathematical rigor; for a full treatment see [BG92]. For now we will simply say that in addition to being able to convince V, P is truly in possession of some extra knowledge about x.

2.2. **Zero knowledge.** Intuitively, to say that something is "proved" is to say that there is a sufficiently convincing argument that it is true. For cryptographic applications it is useful if that proof itself can be verified quickly. Formally, a language L is said to be in the NP class if there exists an algorithm P that accepts or rejects in polynomial time a candidate string x given a witness w_x that is polynomial in length of x. Since w_x need not be computable from x in polynomial time, it can be thought of as encapsulating or serializing some large amount of computation on x. For example, the statement "n is not prime" can be accepted for input 1337 given witness factors $(7, 191)$ since it can be quickly verified that $7 \times 191 = 1337$.

For problems such as the decision version of integer factorization the witness may provide significant information about the underlying search problem (i.e. the factors themselves are used as the witness). Although the IP class is expressive, it is important to ask what information the verifier can extract from the witness string provided by the prover. Can a witness be provided that reveals no information about the underlying solution other than $x \in L$? Intuitively, for an interactive proof system to be "zero-knowledge" a malicious verifier having access to the prover should come away from the exchange with no additional computational ability (in particular any knowledge that would allow the malicious verifier to replicate the prover). [GMR89] formalized this reasoning.

To begin, let $\text{View}_V[P(x) \leftrightarrow V(x)]$ be the record of all interactions between P and V on x, as well as the random tape for V.

Definition 4. Let (P, V) be an interactive proof system for language L. Then, (P, V) is *perfectly zero-knowledge* if, for all $x \in L$, $h \in \{0,1\}^*$, and probabilistic polynomial time Turing machine V', $\text{View}_{V'}[P(x) \leftrightarrow V'(x, h)] = S(x, h)$, where $S(x, h)$ is an expected probabilistic polynomial time algorithm.

The universal quantifiers in the definition capture the power of zero-knowledge; for all members of L, there exists (some) expected polynomial time algorithm that could recreate the communication between any verifier and the prover give some "extra" bit string h. Thus, the interaction between P and all verifiers reveals no new knowledge since there already exists S that can replicate it in polynomial time. This relaxation is analogous to that from interactive proof systems to interactive arguments; in both cases the existence of adversaries with unbounded power is set aside.

Definition 5. An interactive proof system is *computationally* zero-knowledge if no probabilistic polynomial time Turing machine can distinguish $\text{View}_{V'}[P(x) \leftrightarrow V'(x,h)]$ and $S(x,h)$.

Computational indistinguishability [GM84] is a relaxing of the requirement that the quantities in the proof be identical. Rather, we simply say that there is no efficient Turing machine that can tell the two apart. This relaxing aids in the expressiveness of those problems with zero-knowledge proofs. All NP-complete languages having perfect zero-knowledge proof systems would require a collapse of the polynomial time hierarchy to the second level [For87], which is believed to be unlikely[1]. However, all statements in NP have a computational zero-knowledge proof system [GMW91].

2.3. **Probabilistically checkable proofs.** Both NP and IP have interactive verifiers that take polynomial time for some given error $\frac{1}{k}$. Depending on the degree and factors of the polynomial, these verifiers may be prohibitively expensive in practice. [BFL$^+$91] showed that every nondeterministic computational task (including interactive arguments) has a verifier that is exceptionally more efficient.

Theorem 6. *Let S be a nondeterministic computational task described in error-correcting code on instance x with witness y. Then, there exists a task S' such that*

1. *S' accepts the same instances as S,*
2. *each instance/witness pair is verifiable in polylogarithmic time, and*
3. *a witness for S' can be computed from a satisfying witness for S in polynomial time.*

Such constructions are known as *probabilistically checkable proofs*, which were formalized by [BFL$^+$91]. By paying a fixed polynomial cost once[2], a witness for P can be constructed that takes polylogarithmic (or "near-linear") time, given a specific encoding of the problem. [FGL$^+$91] considered the modified case of problems in NP with no encoding requirements, specifically giving an approximation algorithm for determining the size of the largest clique in a graph that used polynomial time to verify a logarithmically sized witness. This lead to a new characterization of NP as given in [AS98].

Definition 7. The class NP are those languages whose proofs can be verified in probabilistically polynomial time using a logarthmic number of random bits and a sublogarithmic number of bits from the proof.

2.4. **Succinctness.** Using probabilistically checkable proofs and collision-resistant hashes, [Kil92] detailed a zero-knowledge proof (interactive argument) for circuit satisfiability running in polynomial time giving 2^{-k} error. The construction uses four messages between the prover and verifier. Given sufficiently large problems the system was shown to be more efficient than naive verification.

Definition 8. Let x be a string in an NP language L which takes t time to verify membership on the machine M_L. Then, the interactive argument system (P, V) is *succinct* if communication and verification time are $O(\text{poly}(k+|M_L|+|x|+\log t))$ and the proving time is $O(\text{poly}(k+|M_L|+|x|+t))$ where $\text{poly}(n)$ is some fixed polynomial independent of the other parameters.

[1] While this is not the same as P = NP, it is a step closer

[2] $|P|^{1+\epsilon}$ for proof P and arbitrary error $\epsilon > 0$

Since circuit satisfiability is NP-complete, this corresponds to a zero-knowledge proof for all of NP. This performance can be used as a baseline for measuring other zero-knowledge (interactive or otherwise) systems.

3. From proofs to zk-SNARKS

3.1. **SNARGs.** Up until now we have reviewed only interactive systems; non-interactivity has eluded us. [Kil92, BG08] demonstrated a four message interactive succinct proof of knowledge for recognizing all languages in NP. This construction, which commits to a probabilistically checkable proof and then shows consistency with a Merkle hash tree, was made non-interactive with one message in the random oracle model [Mic00] by applying the Fiat-Shamir heuristic [FS87]. In the standard model such arguments exist only for a strict subset of NP [BCC$^+$16]. However, this difficulty can be effectively sidestepped by using a two message argument where one of the messages is generated independently from the problem itself [GW11].

Definition 9. Let R be the product of all members of an NP language L and their corresponding witnesses (i.e. $R = \{(x, w) \mid x \in L \text{ with witness } w\}$). Let $\Pi = (G, P, V)$ be efficient algorithms and λ be an arbitrary security parameter. Then, Π is a *succinct, non-interactive argument (SNARG)* if

(1) (Completeness) For all $(x, w) \in R$, when $G(\lambda) \to (\sigma, \tau)$ and $P(\sigma, x, w) \to \pi$, the probability that $V(\tau, x, \pi) = 0$ is negligible in terms of $|\lambda|$.
(2) (Soundness) For all efficient P', when $G(\lambda) \to (\sigma, \tau)$ and $P'(\sigma, \tau) \to (x, \pi)$, the probability that $V(\tau, x, \pi) = 1$ and $x \notin L$ is negligible in terms of $|\lambda|$.
(3) (Succinctness) $|\pi| = \text{poly}(|\lambda|)(|x| + |w|)^{o(1)}$.

The definition for SNARGs bears a close resemblance to that of interactive arguments. The primary difference is the third machine G, and the bounds for the size of the output of P. G is known as the "generator", and given a security parameter λ outputs σ, a common reference string, and τ, a "verification state". Since G relies only on λ (and not x or even L) the generator can be run offline to create parameters (σ, τ). Armed with a witness w for x, the prover takes the common reference string σ and produces a proof π for the statement $x \in L$. Finally, the verifier utilizes the verification state τ to verify the proof π, ultimately being convinced that $x \in L$.

There exist several variants of SNARGs based on the specific soundness condition used. The above definition is an *adaptive, publicly-verifiable* SNARG. A SNARG is adaptive if its soundness guarantee holds even if the adversarial prover P' can choose x; a non-adaptive SNARG would have $P'(\sigma, \tau, x) \to \pi$. Furthermore, a SNARG is publicly-verifiable if P' has access to τ in addition to σ. Otherwise, the SNARG is said to be *designated-verifier*.

A final variant of SNARGs are *preprocessing* SNARGs. Informally, preprocessing SNARGs are those where the generator is permitted to be "expensive". Since generation is generally counted towards the verification time, a *fully-succinct* SNARG is bound by $\text{poly}(\log t)$ for traditional verification time t, while a preprocessing SNARG is bound by $\text{poly}(t)$.

For SNARGs to be secure we must make certain knowledge extractability assumptions. [GW11] showed that SNARGs cannot be proven secure under any falsifiable assumption via block-box reduction. However, [BCC$^+$16] proved that existence of extractable collision-resistant hash functions are necessary and sufficient conditions for secure SNARGs.

3.2. **SNARKs.** Given secure SNARGs, it makes sense to search for an analogue of proofs of knowledge for interactive arguments. Recall that a proof of knowledge codifies the idea that the prover really "knows" w, defined in terms of an extractor that can determine the witness given the prover. We will use a similar idea, incorporated into a modified soundness requirement.

Definition 10. A *succinct non-interactive argument of knowledge (SNARK)* is a SNARG with the soundness requirement changed to:

(2) (Proof of knowledge and soundness) For all efficient P' there is an efficient E such that when $G(\lambda) \to (\sigma, \tau)$, $P'(\sigma, \tau) \to (x, \pi)$, and $E(\sigma, \tau) \to w$, the probability that $V(\tau, x, \pi) = 1$ and $x \in L$ is negligible in terms of $|\lambda|$.

The variants of adaptive/non-adaptive, publicly-verifiable/designed-verifier, and fully-succinct/preprocessing are comparable for SNARKs and SNARGs.

3.3. **zk-SNARKs.** The last piece of the puzzle is to apply zero-knowledge to SNARKs.

Definition 11. Let (G, P, V) be a SNARK. Then, (G, P, V) is a *perfect zk-SNARK* if there is an efficient simulator S such that for all stateful distinguishers D, whenever $D(\pi) = 1$ and $x \in L$, the probability of $G(\lambda) \to (\sigma, \tau)$, $D(\sigma, \tau) \to (x, w)$, and $P(\sigma, x, w) \to \pi$ is the same as the probability of $G(\lambda) \to (\sigma, \tau, \text{trap})$, $D(\sigma, \tau) \to (x, w)$, and $S(\text{trap}, x) \to \pi$.

To summarize, a zk-SNARK is a SNARG where the prover "knows" the witness w and where the proof π does not give any information that would help an adversary determine what w is. Thus, we have fully realized the vision of an efficient "digital signature" for proving membership in arbitrary languages in NP.

4. zk-SNARK constructions

The definition of a zk-SNARK tells us what a zk-SNARK *is*, but not how one might be constructed. Several such constructions have been developed, ranging from the theoretical to the practical.

4.1. **Theoretical constructions.**

4.1.1. *Pairing based.* [Mic00]'s computationally sound proofs were the first appearance of what we now call zk-SNARKs, although the explicit formulation used here had not been defined yet. [Gro10] first improved upon [Mic00] by detailing a preprocessing zk-SNARK for circuit satisfiability in sublinear amount of communication without relying on the random oracle model. The construction uses pairing in bilinear groups to commit to the witness of satisfiability while revealing no information about the witness, assuming the hardness of Diffie-Hellman and knowledge of exponent. Without loss of generality, the circuit C is assumed to consist of NAND gates. From a high level, the scheme works by committing to tuples $(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n)$, $(b_1, b_2, \ldots, b_n, 0, \ldots, 0)$ and the corresponding outputs $(-u_1, -u_2, \ldots, u_n, 0, \ldots, 0)$ where a_i and b_i are inputs to gate i and u_i is the gate's output[3] for $n = |C|$. Then, commitments are made that show the internal consistency with the of the circuit (e.g. $u_i = -a_i b_i$ for all $i \in n$). The commitments themselves use the Pedersen commitment scheme [Ped92], which provides the zero-knowledge guarantees while maintaining consistency.

The [Gro10] construction provides for trade-offs between time and space complexities ranging from a $\Theta(1)$ argument and $\Theta(|C|^2)$ reference string paired with $\Theta(|C|^2)$ proving to a $\Theta(|C|^{\frac{2}{3}})$ argument and reference string and $\Theta(|C|^{\frac{4}{3}})$ proving. [Lip12] improved the construction's argument and reference string complexity to $\Theta(|C|^{\frac{1}{2}+o(1)})$.

4.1.2. *Quadratic span based.* In an attempt to generalize and improve on the prover complexity of [Gro10] and [Lip12], [GGP$^+$13] introduced the quadratic span programs, with the goal of quasi-linear prover times. Quadratic span programs accept an input whenever a target polynomial can be expressed as a product of two linear combinations of vectors of polynomials.

[3]For convenience, $+1$ is used for true and -1 is used for false

Definition 12 ([GGP+13])**.** A *quadratic span program (QSP)* Q over field F contains two sets of polynomials $V = \{v_k(x)\}$, $W = \{w_k(x)\}$ for $k \in \{0, \ldots, m\}$, and a target polynomial $t(x)$, all from $F[x]$. Q also contains a partition of the indices $I = \{1, \ldots, m\}$ into two sets $I_{labeled}$ and I_{free}, and a further partition of $I_{labeled}$ as $\cup_{i\in[n],j\in 0,1} I_{ij}$. For input $u \in 0, 1^n$, let $I_u = I_{free} \cup_i I_{i,u_i}$ be the set of indices that "belong" to input u. Q accepts an input $u \in \{0,1\}^n$ iff there exist tuples $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ from F^m, with $a_k = 0 = b_k$ for all $k \notin I_u$, such that $t(x)$ divides $\big(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\big) \cdot \big(w_0(x) + \sum_{k=1}^{m} b_k \cdot w_k(x)\big)$.

Crucially, it was shown that QSPs and boolean circuits are interchangeable with only a constant overhead. Once converted, the properties of polynomials allow for easy construction of a preprocessing zk-SNARK. Using QSPs, a zk-SNARK for circuit satisfiability was shown using only seven group elements, with the reference string linear in the size of the circuit and prover time quasi-linear. In addition, [GGP+13] also presented a variation of QSPs that work on arithmetic circuits, known as quadratic arithmetic programs, and again showed a zk-SNARK construction.

Definition 13 ([GGP+13])**.** A *quadratic arithmetic program (QAP)* Q over field F contains three sets of polynomials $V = \{v_k(x)\}$, $W = \{w_k(x)\}$, $Y = \{y_k(x)\}$ for $k \in \{0, \ldots, m\}$, and a target polynomial $t(x)$, all from $F[x]$. Let f be a function having input variables with labels $1, \ldots, n$ and output variables with labels $m - n' + 1, \ldots, m$. We say that Q is a QAP that computes f if the following is true: $a_1, \ldots, a_n, a_{m-n'+1}, a_m \in F^{n+n'}$ is a valid assignment to the input/output variables of$f$ iff there exist $(a_{n+1}, \ldots, a_{m-n'}) \in F^{m-n-n'}$ such that $t(x)$ divides $\big(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\big) \cdot \big(w_0(x) + \sum_{k=1}^{m} a_k \cdot w_k(x)\big) - \big(y_0(x) + \sum_{k=1}^{m} a_k \cdot y_k(x)\big)$.

4.1.3. *Fully succinct.* The previously discussed constructions are all preprocessing zk-SNARKs. The more general type, fully succinct zk-SNARKs (whose provers are linear only in the security parameter λ), have also been explored. [BCC+13] gave a general construction for converting preprocessing SNARKs into fully succinct SNARKs. Using *incrementally-verifiable computation*, several SNARKs are composed that effectively "prove" the steps taken during the extra preprocessing phase. By doing this, asymptoticly fully succinct SNARKs are given.

4.2. **Practical implementations.**

4.2.1. *Pinocchio.* Although constructions of zk-SNARKs were known in the literature, a fully implemented end-to-end system for describing a problem and generating a zk-SNARK for it remained open until [PGHR13]. Towards this end, [PGHR13] created a system capable of compiling a subset of C programs into a structure that a specific zk-SNARK could verify. Although the subset was fairly strict[4], the ability to ingest regular C code made SNARKs generally accessible to cryptographic implementors.

The [PGHR13] system consists of two parts. The first part is a compiler, `qcc`, which converts the supported C subset to an arithmetic circuit. Recall that an arithmetic circuit is akin to boolean circuits with the gates of logical AND, OR, etc. replaced with mathematical operations such as addition and multiplication. Through various techniques all of the fundamental operations in C are modeled using arithmetic gates[5]. Once converted, a quadratic arithmetic program is constructed for the arithmetic circuit. The second portion of the system is a zk-SNARK for QAPs, the primary operation of which is exponentiation in an elliptic curve group.

From a practical standpoint the [PGHR13] system was the first practical end-to-end system for using zk-SNARKs in software. The performance was shown to be orders of magnitude faster than

[4]Non-self-modifying with fixed memory access and compile-time constants for loops and array access

[5]Surisingly, is was more efficient to still use QAPs for C's boolean operations

previous constructions: a sample problem (multivariate polynomial evaluation) is quoted as having 41 second generation time, 246 second proof creation time, and 12 millisecond verification time.

4.2.2. *TinyRAM.* The construction [PGHR13] supported a large subset of C from a semantic standpoint, but the programs themselves could not have any data dependencies, e.g. a conditional on some calculated value derived from the input. To formalize an environment for such computations to take place, [BSCG+13b] created the TinyRAM architecture, an idealized random-access machine equipped with a fixed word size and number of registers, a program counter and conditional flag, and addressable memory. The instructions in TinyRAM are those one would generally expect in a RISC architecture: loads, stores, compares, jumps, and so on. In addition, a special instruction signifies the machine has reached an accepting state and should terminate.

The [BSCG+13b] system uses a modified version of `gcc` to generate TinyRAM instructions from C. Then, as with [PGHR13], a conversion to an arithmetic circuit is performed, although the conversion in [BSCG+13b] is much more complicated to account for the consistency of the memory. From the TinyRAM instructions a routing network with constraints is created, and the generated arithmetic circuit verifies that the constraints are met for a particular input.

Once the arithmetic circuit is constructed, all that remains is to pass it into a zk-SNARK for satisfiability. The SNARK presented is a modified version of that found in [BCI+13], which itself generalizes the quadratic arithmetic programs found in [GGP+13].

4.2.3. *vnTinyRAM.* Both [PGHR13] and [BSCG+13b] require that the generation step be rerun for each different program. This generation step is not trivial, and accounts for a significant portion of the overall time the system takes to execute. While the reference string and verification state can be reused for different inputs to the same program, expensive work is required whenever a new program is used. [BSCT+14] created a universal circuit generator which is bound only by the maximum number of instructions in a program l, the maximum input size n, and some specific time bound t. This "once-and-for-all key generation" can be used to verify all programs up a given size, and the size of the generated circuit is $O((l + n + t) \cdot \log(l + n + t))$, which means that the circuit grows essentially linearly in all three inputs.

In addition to a universal circuit generator, [BSCT+14] operates on a more powerful machine than [BSCG+13b]: vnTinyRAM. vnTinyRAM is an extension of TinyRAM that allows for self-modifying code, and can be thought of as an idealized von Neumann machine. As was done previously, a modified version of `gcc` takes C programs and outputs vnTinyRAM instructions. vnTinyRAM also switches to byte-addressable (as opposed to word-addressable) memory.

Following the familiar construction, the [BSCG+13b] system converts the intermediate machine instructions into an arithmetic circuit. Then, a zk-SNARK for arithmetic circuits provides the proof and verification mechanisms for the program execution. Rather than construct a new zk-SNARK in its entirety, [BSCT+14] provided several tailored optimizations to the SNARK found in [PGHR13].

Detailed complexities in time and space are provided, and apples-to-apples comparison with previous implementations showed modest performance gains. For example, a one-million gate circuit with a one-thousand bit input with 128 bit security had a generation time of 117 seconds, a proving time of 147 seconds, and a verification time of 5 milliseconds. In addition, all proofs are 288 bytes regardless of the program or input.

The correctness of the optimizations made in [BSCT+14] relies on an unproven lemma presented in the paper. [Par15] showed that this lemma is incorrect, and the efficiency gains were (at least theoretically) incorrect. More seriously, it was shown how to create invalid proofs that would be accepted by the verification algorithm. Since the publication of [Par15], modifications have

been made to open source implementations of vnTinyRAM that correct the flaw, with negligible performance implications[6].

The libsnark project is a free, open source software library for constructing zk-SNARKS (and regular SNARKs) via a variety of the methods presented in literature. The zk-SNARK system used by libsnark is the that presented in [BSCT+14] (with the correction from [Par15]).

5. Applications in cryptocurrencies

For a cryptocurrency to be trustworthy, a user must be able independently verify the correctness of transactions. In Bitcoin [Nak09] this property is achieved via a publicly auditable transaction ledger. Every transaction is fully transparent; the source and destination addresses as well as the amount of the transaction are "in the clear". If a real world identity is associated with a Bitcoin address then a serious loss of privacy can occur. For example, suppose Alice is paid by her employer in Bitcoin. Should her address ever be known by coworkers then the amount of her salary would be become public knowledge. Solutions for solving Bitcoin's privacy problems range from cycling through many addresses, to "mixers", where several users send coins to one address which are subsequently pooled and combined before being returned.

Since Bitcoin's inception several new cryptocurrencies have been developed that attempt to include privacy features directly into the protocol of the coin, the most popualar of which are Dash[7], zcash[8], and Monero[9]. The central challenge for these coins is maintaining correctness (e.g. that coins are not double spent) while ensuring the privacy of the users engaging in the transactions. Of these three, zcash uses zk-SNARKs to enforce this property.

5.1. Zerocash.

The essential operation of a cryptocurrency is the transaction, where some amount of "coins" are sent from Alice to Bob. Afterwards, Bob is free to send these coins to someone else, but Alice must not be able to "respend" the coins. Zerocash [BSCG+14] provides privacy and fungibility by encoding these constraints as an arithmetic circuit with a zk-SNARK proof for the circuit appended to the blockchain as a record of the transaction, which can be efficiently verified by anyone.

Zerocash defines the concept of a "decentralized payment scheme", and then shows how zk-SNARKs can be used to build such a system.

Definition 14. A *decentralized payment scheme* (DAP) is a set of polynomial algorithms with the following properties

(1) `Setup`. Generates a set of public parameters used by the remaining algorithms. `Setup` must be run by a trusted party.
(2) `CreateAddress`. From the public parameters, generate a public address to which coins can be sent, and a corresponding private key which will allow coins sent to the public address to be spent.
(3) `Mint`. Creates new coins and logs their creation on the blockchain.
(4) `Pour`. Transfers the value from input coins to new outputs coins and logging the transaction on the blockchain, optionally revealing their amounts. Allows subdiving, merging, and transferring coins.
(5) `Verify`. Verify that a transaction log is correct.
(6) `Receive`. Determines the balance of unspent coins for a particular address based on all transactions saved on the blockchain.

[6] See `https://github.com/scipr-lab/libsnark/commit/af725eeb`
[7] `https://dash.org/`
[8] `https://z.cash/`
[9] `https://getmonero.org/`

In addition, the DAP must be complete, which means that any unspent coin is able to spent, and secure, which means that it maintains ledger indistinguishability, transaction non-malleability, and balance correctness.

The DAP operations (`Mint`, `Pour`, etc.) can be generalized to nearly any cryptocurrency. The security guarantees of ledger indistinguishability (no information is provided by transactions other than that which is strictly public), and transaction non-malleability (transactions cannot be modified and still be valid) provide the core differentiator as an anonymous payment system. The balance correctness guarantee (cannot spend more coins that one's unspent balance) is a necessity for any viable cryptocurrency. Zerocash is an implementation of a DAP which makes extensive use of zk-SNARKs in its operations.

The `Pour` operation, where coins are combined, subdivided, and moved is the essential core of Zerocash, as it facilitates the transfer of value between users. An instance of a `Pour` operation involves the certain public commitments generated from the coins to be poured. A witness for `Pour` are the private keys for the coins, as well as the specific amounts, sources, and destinations. To maintain privacy protections, the witness must remain secret. To facilitate this a zk-SNARK for `Pour` was created.

The program or algorithm for the `Pour` zk-SNARK accepts if the witness is valid for the given instance under the rules of the DAP. The actual zk-SNARK used is the QAP zk-SNARK from [BSCT$^+$14]. However, rather than generate the circuit for the zk-SNARK via a high-level language (e.g. the vnTinyRAM construction in [BSCT$^+$14]), Zerocash uses a much more efficient "hand-designed" circuit. The dominating operation in the circuit is the SHA-256 hash function. An arithmetic circuit for SHA-256 was created "from scratch" which consisted of around 28,000 gates. For comparison, implementing the same logic in C and "compiling" to TinyRAM via the method detailed in [BSCG$^+$13b] resulted in a circuit with $5.7 \cdot 10^6$ gates.

The circuit for the `Pour` zk-SNARK is constant. Thus, the same reference string and verification state will be used in every `Pour`. The `Setup` procedure generates these public parameters (the prepocessing phase for the zk-SNARK). Then, to create a transaction the zk-SNARK's proof generation is performed using the witness which requires information only the valid coin owner should know. The output of the proof generation is the proof π (only 288 bytes), and is appended to the blockchain to record the transaction. The correctness of π can be efficiently verified by any user of the system by performing the zk-SNARK's verification operation. While the circuit itself ensures balance correctness, the inherent properties of zk-SNARKs confer ledger indistinguishability and transaction non-malleability.

5.1.1. *zcash.* The Zerocash system described in [BSCG$^+$14] has been implemented as a real-world cryptocurrency known as zcash. The protocol itself is an extension or "fork" of Bitcoin. zcash maintains the transparent transactions of Bitcoin and adds optional private transactions known as "shielded transactions" using the zk-SNARK method described above. While verification of shielded transactions is fast (about the same as a verification of a transparent transaction), the proof generation operation, which must be performed every time a user wishes to send coins, is much more expensive than the counterpart in transparent transactions, totaling several minutes on a powerful modern desktop. Perhaps because of this, at the time of writing only around 20% of all zcash transactions are shielded transactions.

Rather than trust a single party to honestly generate the public parameters, the zcash creators designed a multi-party parameter generation scheme. This method was later formalized in [BSCG$^+$15]. Six people took part in the parameter generation, and the system is secure if at least one of the parties was honest.

Appendix A. STARKs

Although the multi-party parameter generation described in [BSCG+15] amortizes the required trust in a zk-SNARK over several parties, it does not explicitly eliminate the need for trust. Eli Ben-Sasson (author of cited works [BSCT+14], [BSCG+14], among others) has recently informally revealed STARKs (succinct transparent arguments of knowledge) [Ben17] which do not require a trusted party to perform the parameter generation. In addition, STARKs provide post-quantum cryptography resistance. Proofs for STARKs do appear to be considerably larger, although [Ben17] suggests some mitigations a cryptocurrency could use. As of the time of writing STARKs have yet to formally appear in the literature.

References

[AS98] Hello Sanjeev Arora and Shmuel Safra. Probalistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70-112, 1998.

[BBF+15] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. *Proceedings of 36th IEEE Symposium on Security and Privacy (S&P '15)*, pages 271-286, 2015.

[BCC+13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and boostrapping for SNARKs and proof-carrying data. *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 111-120, 2013.

[BCC+16] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The Hunting of the SNARK. *Journal of Cryptology*, pages 1-78, 2016.

[BCC88] Gilles Brassard, David Chaum, and Claude Crpeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156-189, 1988.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. *Proceedings of the 10th Theory of Cryptography Conference (TCC '13)*, pages 315333, 2013.

[Ben17] Eli Ben-Session. Zerocash, Bitcoin, and transparent computational integrity. `https://cyber.stanford.edu/sites/default/files/elibensasson.pdf`. Retrieved May 31, 2017.

[BFL+91] Lszl Babai, Lance Fortnow, Leonid A. Levi, and Mario Szegedy. Checking computations in polylogarithmic time. *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC '91)*, pages 21-31, 1991.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. *Advances in Cryptology (Crypto 92) Proceedings*, 1992.

[BG08] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM Journal on Computing*, 38(5):1661-1694, 2008.

[BSCG+13a] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 585-594, 2013.

[BSCG+13b] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: verifying program executions succinctly and in zero knowledge. *Proceedings of the 33rd Annual International Cryptology Conference (CRYPTO '13)*, pages 90-108, 2013.

[BSCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 459-474, 2014.

[BSCG+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 287-304, 2015.

[BSCT+14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. *Proceedings of the 23rd USENIX security symposium (Security '14)*, pages 781796, 2014.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644-654, 1976.

[FGL+91] Uriel Feige, Shafi Goldwasser, Lszl Lovsz, Shmuel Safra, and Mario Szegedy. Approximating clique is almost NP-complete *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 2-12, 1991.

[For87] Lance Fortnow. The complexity of perfect zero-knowledge. *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC '87)*, pages 204-209, 1987.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. *Proceedings of the 6th Annual International Cryptology Conference (CRYPTO '87)*, pages 186-194, 1987.

[GGP+13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT '13)*, pages 626645, 2013.

[GH98] Oded Goldreich and Johan Hstad. On the complexity of interactive proofs with bounded communication. *Information Processing Letters*, 67(4):169-192, 1996.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer Science and Systems*, vol. 28, 1984.

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

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 174-187, 1986.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690-728, 1991.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. *Proceedings of the 16th International Conference of Theory and Application of Cryptology and Information Security (ASIACRYPT '10)*, pages 321-340, 2010.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. *Proceedings of the 43rd Annual Symposium on Theory of Computing (STOC '11)*, pages 99-108, 2011.

[HPS98] Jeffrey Hoffstein, Jill Pipher, Joseph Silverman. NTRU: A ring-based public key cryptosystem. *Algorithmic Number Theory*, Lecture Notes in Computer Science, vol 1423, 1998.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36-63, 2001.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC '92)*, pages 723-732, 1992.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. *Proceedings of the 9th Theory of Cryptography Conference (TCC '12)*, pages 169-189, 2012.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253-1298, 2000.

[Par15] Bryan Parno. A note on the unsoundness of vnTinyRAM's SNARK. *Cryptology ePrint Archive*, 2015/437, 2015.

[Ped92] Torben Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. *Advances in Cryptology (CRYPTO '91) Proceedings*, 1992.

[PGHR13] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: nearly practical verifiable computation. *Proceedings of the 34th IEEE Symposium on Security and Privacy (Oakland '13)*, pages 238252, 2013.

[RSA78] Ron Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120-126, 1978.

[Nak09] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2009. URL: `http://www.bitcoin.org/bitcoin.pdf`

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869-877, 1992.

[Val08] Paul Valiant. Incrementally verifiable computation or proof of knowledge imply time/space efficiency. *Proceedings of the 5th Theory of Cryptography Conference (TCC '08)*, pages 1-18, 2008.