What are zk-SNARKs?

Note: With Network Upgrade 5 (NU5) in May 2022, Zcash introduced the Orchard shielded payment protocol, which utilizes the Halo 2 zero-knowledge proving system. Halo is a new zk-SNARK that's finally capable of solving two outstanding issues in Zcash: removing the trusted setup while hitting performance targets and supporting a scalable architecture for private digital payments.

Zcash was the first widespread application of zk-SNARKs, a novel form of zero-knowledge cryptography. The strong privacy guarantee of Zcash is derived from the fact that shielded transactions in Zcash can be fully encrypted on the blockchain, yet still be verified as valid under the network's consensus rules by using zk-SNARK proofs.

The acronym zk-SNARK stands for "Zero-Knowledge Succinct Non-Interactive Argument of Knowledge," and refers to a proof construction where one can prove possession of certain information, e.g. a secret key, without revealing that information, and without any interaction between the prover and verifier. In May 2022, Zcash began the process of upgrading its underlying cryptography and moving to a new proof composition called Halo.

"Zero-knowledge" proofs allow one party (the prover) to prove to another (the verifier) that a statement is true, without revealing any information beyond the validity of the statement itself. For example, given the hash of a random number, the prover could convince the verifier that there indeed exists a number with this hash value, without revealing what it is.

In a zero-knowledge "Proof of Knowledge" the prover can convince the verifier not only that the number exists, but that they in fact know such a number – again, without revealing any information about the number. The difference between "Proof" and "Argument" is quite technical and we don't get into it here.

"Succinct" zero-knowledge proofs can be verified within a few milliseconds, with a proof length of only a few hundred bytes even for statements about programs that are very large. In the first zero-knowledge protocols, the prover and verifier had to communicate back and forth for multiple rounds, but in "non-interactive" constructions, the proof consists of a single message sent from prover to verifier. Prior to Halo, the most efficient known way to produce zero-knowledge proofs that are non-interactive and short enough to publish to a block chain was to have an initial setup phase that generates a common reference string shared between prover and verifier. We refer to this common reference string as the public parameters of the system.

If someone had access to the secret randomness used to generate these parameters, they would be able to create false proofs that would look valid to the verifier. For Zcash, this would mean the malicious party could create counterfeit coins. To prevent this from ever happening, Zcash generated the public parameters through two elaborate, multi-party ceremonies for Sprout and Sapling. To learn more about our parameter generation ceremony and see the precautions we took to prevent the secret randomness essential to Zcash from being exposed (e.g. computers being blowtorched), visit our Paramgen page. To learn more about the math behind the parameter generation protocol, read our blog post or whitepapers (1, 2) on the topic.

With the activation of the Orchard shielded pool, we now have the ability to generate zero-knowledge proofs without the need for these "trusted" ceremonies. This is made possible due to Halo, a novel breakthrough in cryptography discovered by researchers at ECC. Halo also sets the stage for dramatic scalability improvements in future upgrades.

How zk-SNARKs are constructed in Zcash

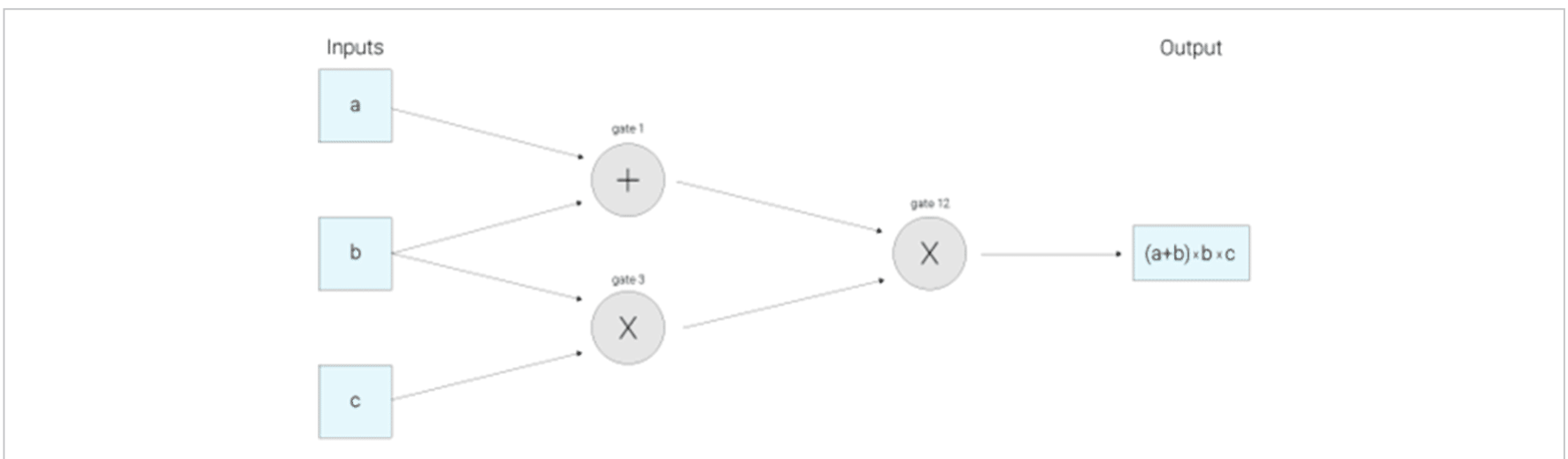
Note: This applies to Sprout and Sapling shielded pools. The Orchard shielded pool relies on zero knowledge proofs that use the Halo 2 proving system.

In order to have zero-knowledge privacy in Zcash, the function determining the validity of a transaction according to the network's consensus rules must return the answer of whether the transaction is valid or not, without revealing any of the information it performed the calculations on. This is done by encoding some of the network's consensus rules in zk-SNARKs. At a high level, zk-SNARKs work by first turning what you want to prove into an equivalent form about knowing a solution to some algebraic equations. In the following section, we give a brief overview of how the rules for determining a valid transaction get transformed into equations that can then be evaluated on a candidate solution without revealing any sensitive information to the parties verifying the equations.

Computation → Arithmetic Circuit → R1CS → QAP → zk-SNARK

The first step in turning our transaction validity function into a mathematical representation is to break down the logical steps into the smallest possible operations, creating an "arithmetic circuit". Similar to a boolean circuit where a program is compiled down to discrete, single steps like AND, OR, NOT, when a program is converted to an arithmetic circuit, it's broken down into single steps consisting of the basic arithmetic operations of addition, subtraction, multiplication, and division (although in our particular case, we will avoid using division).

Here is an example of what an arithmetic circuit looks like for computing the expression (a+b)*(b*c) :



Looking at such a circuit, we can think of the input values a, b, c as "traveling" left-to-right on the wires towards the output wire. Our next step is to build what is called a Rank 1 Constraint System, or R1CS, to check that the values are "traveling correctly". In this example, the R1CS will confirm, for instance, that the value coming out of the multiplication gate where b and c went in is b*c.

In this R1CS representation, the verifier has to check many constraints — one for almost every wire of the circuit. (For technical reasons, it turns out we only have a constraint for wires coming out of multiplication gates.) In a 2012 paper on the topic, Gennaro, Gentry, Parno and Raykova presented a nice way to "bundle all these constraints into one". This method uses a representation of the circuit called a Quadratic Arithmetic Program (QAP). The single constraint that needs to be checked is now between polynomials rather than between numbers. The polynomials can be quite large, but this is alright because when an identity does not hold between polynomials, it will fail to hold at most points. Therefore, you only have to check that the two polynomials match at one randomly chosen point in order to correctly verify the proof with high probability.

If the prover knew in advance which point the verifier would choose to check, they might be able to craft polynomials that are invalid, but still satisfy the identity at that point. With zk-SNARKs, sophisticated mathematical techniques such as homomorphic encryption and pairings of elliptic curves are used to

evaluate polynomials “blindly” – i.e. without knowing which point is being evaluated. The public parameters described above are used to determine which point will be checked, but in encrypted form so that neither the prover nor the verifier know what it is.

The description so far has mainly addressed how to get the S and N in “SNARKs” — how to get a short, non-interactive, single message proof — but hasn’t addressed the “zk” (zero-knowledge) part which allows the prover to maintain the confidentiality of their secret inputs. It turns out that at this stage, the “zk” part can be easily added by having the prover use “random shifts” of the original polynomials that still satisfy the required identity.

For a step-by-step, in-depth explanation of key concepts behind zk-SNARKs in Zcash, see our SNARKs Explainer series with posts on:

1. Homomorphic Hiding
2. Blind Evaluation of Polynomials
3. The Knowledge of Coefficient Test and Assumption
4. How to make Blind Evaluation of Polynomials Verifiable
5. From Computations to Polynomials
6. The Pinocchio Protocol
7. Pairings of Elliptic Curves

Zcash’s Orchard pool uses the Halo 2 library for zero knowledge proofs. Before the NU5 upgrade, Zcash used bellman, a Rust-language library for zk-SNARKs. Before the Sapling upgrade, Zcash used a fork of the C++ library, libsnark. For a deeper dive into the protocols used for Zcash’s zk-SNARKs, refer to the paper on the Pinocchio protocol, which was used until the Sapling upgrade, and Jens Groth’s zk-SNARK which is used currently.

How zk-SNARKs are applied to create a shielded transaction

In Bitcoin, transactions are validated by linking the sender address, receiver address, and input and output values on the public blockchain. Zcash uses zk-SNARKs to prove that the conditions for a valid transaction have been satisfied without revealing any crucial information about the addresses or values involved. The sender of a shielded transaction constructs a proof to show that, with high probability:

- the input values sum to the output values for each shielded transfer.
- the sender proves that they have the private spending keys of the input notes, giving them the authority to spend.
- The private spending keys of the input notes are cryptographically linked to a signature over the whole transaction, in such a way that the transaction cannot be modified by a party who did not know these private keys.

In addition, shielded transactions must satisfy some other conditions that are described below.

Bitcoin tracks unspent transaction outputs (UTXOs) to determine what transactions are spendable. In Zcash, the shielded equivalent of a UTXO is called a “commitment”, and spending a commitment involves revealing a “nullifier”. Zcash nodes keep lists of all the commitments that have been created, and all the nullifiers that have been revealed. Commitments and nullifiers are stored as hashes, to avoid disclosing any information about the commitments, or which nullifiers relate to which commitments.

For each new note created by a shielded payment, a commitment is published which consists of a hash of: the address to which the note was sent, the amount being sent, a number “rho” which is unique to this note (later used to derive the nullifier), and a random nonce.

Commitment = HASH(recipient address, amount, rho, r)

When a shielded transaction is spent, the sender uses their spending key to publish a nullifier which is the hash of the secret unique number (“rho”) from an existing commitment that has not been spent, and provides a zero-knowledge proof demonstrating that they are authorized to spend it. This hash must not already be in the set of nullifiers tracking spent transactions kept by every node in the blockchain.

Nullifier = HASH(spending key, rho)

The zero-knowledge proof for a shielded transaction verifies that, in addition to the conditions listed above, the following assertions are also true:

- For each input note, a revealed commitment exists.
- The nullifiers and note commitments are computed correctly.
- It is infeasible for the nullifier of an output note to collide with the nullifier of any other note.

In addition to the spending keys used to control addresses, Zcash uses a set of proving and verifying keys to create and check proofs. These keys are generated in the public parameter ceremony discussed above, and shared among all participants in the Zcash network. For each shielded transaction, the sender uses their proving key to generate a proof that their inputs are valid. Miners check that the shielded transaction follows consensus rules by checking the prover’s computation with the verifying key. The way that Zcash’s proof generation is designed requires the prover to do more work up-front, but it simplifies verifying, so that the major computational work is offloaded to the creator of the transaction (this is why creating a shielded Zcash transaction can take several seconds, while verifying that a transaction is valid only takes milliseconds).

The privacy of Zcash’s shielded transactions relies upon standard, tried-and-tested cryptography (hash functions and stream ciphers), but it’s the addition of zk-SNARKs, applied with the system of commitments and nullifiers, that allows senders and receivers of shielded transactions to prove that encrypted transactions are valid. Other methods of providing privacy for cryptocurrencies rely upon obscuring the linkage between transactions, but the fact that Zcash transactions can be stored on the blockchain fully encrypted opens up new possibilities for cryptocurrency applications. Encrypted transactions allow parties to enjoy the benefits of public blockchains, while still protecting their privacy. Planned future upgrades will allow users to selectively disclose information about shielded transactions at their discretion. See our Near Future of Zcash blog post on future plans for Zcash.

For a more in-depth explanation of how shielded transactions are constructed in Zcash, see our blog post on How Transactions Between Shielded Addresses Work. For full details on the current Zcash protocol, refer to our protocol specification.

Future applications of zk-SNARKs

Creating shielded transactions in Zcash is only one example out of many possible applications of zk-SNARKs. Theoretically, you can use a zk-SNARK to verify any relation without disclosing inputs or leaking information. Generating proofs for complex functions is still too computationally intensive to be practical for many applications, but the Zcash team is pushing the boundaries for optimizing zk-SNARKs, and is already breaking new ground with more efficient implementations.

As it currently stands, Zcash's implementation of zk-SNARKs can be added to any existing distributed ledger solution as a Zero-knowledge Security Layer for enterprise use cases. The scientists on the Zcash team are among the most knowledgeable researchers of zk-SNARKs in the world, and are constantly working on coming up with new applications and improving the efficiency of zero-knowledge protocols. If you have a business need that could benefit from the application of zero-knowledge proofs or blockchain solutions with robust privacy, get in touch with our business development team.

Additional Information

Reference

The Basics
What are zk-SNARKS?
Technical Explainer: Halo on Zcash
Protocol Specification

Papers

Original Zerocash paper
Succinct Non-Interactive Zero-Knowledge Proofs
Multi-Party Protocol Parameter Generation

Sapling

What is JubJub?
Sapling blog series

Zcash Ceremony

Parameter Generation Info
Paramgen Video Explainer

Science Projects

HAWK: Private Smart Contracts
BOLT: Private Payment Channels
History of Attacks on Secure Hash Functions
Cryptocurrency UX research

Technical Blog Posts

Matt Green: Zero-Knowledge Proofs
Anatomy of a Zcash Transaction
How Private Transactions Work
How Zcash Parameters Are Generated
The Encrypted Memo Field
Pairing Cryptography in Rust