Lecture 18: Data Privacy via Zero Knowledge Cryptography

Principles of Blockchains, University of Illinois,
Professor: Pramod Viswanath
Scribe: Peiyao Sheng and Viswa Virinchi Muppirala

March 30, 2021

Abstract

The transactions and ledger in the blockchain designs we have seen are in plain text and world-readable. This property was crucially used to *validate* transactions before adding them to blocks and for blocks to be accepted into the ledger. However, there is a need to avoid representing transactions in plain text, e.g., in financial applications such as crytocurrencies. Although banks and credit card companies inherently see the identities of participants in all transactions, the ledger themselves are not openly readable to other parties. How to provide strong identity management between relevant parties (i.e., merchants and customers), while cutting out unnecessary information leakage to middlemen is a pressing problem. In this lecture, we summarize a powerful cryptographic technique for providing transaction privacy at the blockchain layer known as zk-SNARKs. The key idea behind zk-SNARKs is to encrypt transactions in such a way that users can verify their validity without learning anything about the contents of the transaction. zk-SNARKs are the technical foundation of the privacy-preserving cryptocurrency ZCash whose architecture is the focus of this lecture.

Introduction

Bitcoin employs the UTXO format of state/ledger management, where transactions consist of a list of inputs and outputs; outputs specify the recipients' public keys and inputs refers to previous outputs – see Figure 2. One key property of the Bitcoin network is pseudonymity; the public key is the only information associated with each account. A single user can create multiple public keys to protect privacy. However, the relationship *between* transactions leads to information leakage. Based on the public keys of inputs and outputs of a transaction, we can link transactions into a graph; see Figure 1. Now if the identity of one public key is accidentally divulged, the identities of other public keys connected to it in the graph are also potentially traceable.

Trusted third-party mixer. One possible solution to this problem is to employ a third-party mixer (also known as a "laundry service"). Consider a set of outputs from a list of transactions, when creating a new transaction with some of the outputs, UTXO system requires the sender to specify which outputs will be used. The laundry service essentially exchanges the coins (the public keys) of various users so that the public keys can't be traced using a transaction graph. However, this requires peers to trust a centralized third-party who can trace or even steal the coins.

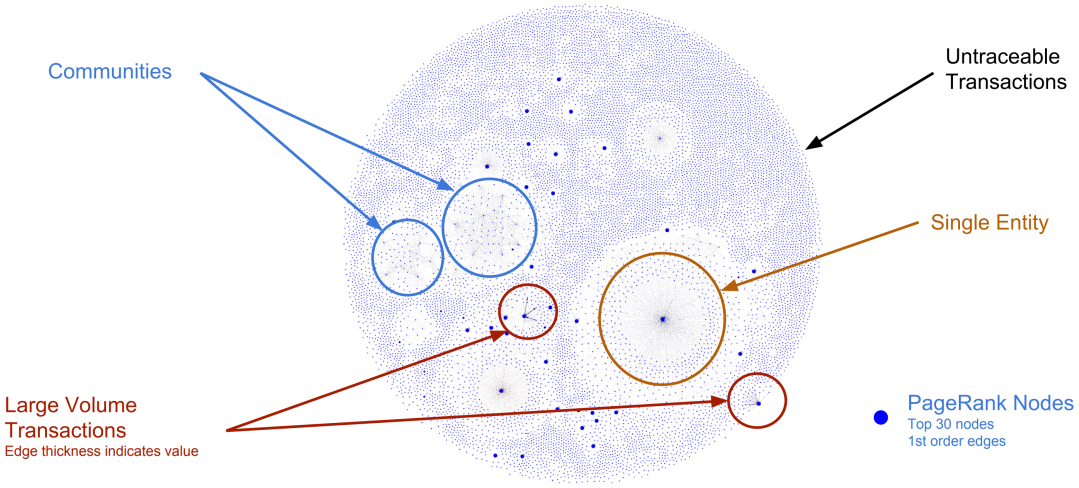


Figure 1: Typical transaction graph for a day [2].

Decentralized laundry system. As a second step, we can consider removing the trusted party for the laundry system to get a decentralized privacy service that is directly *integrated* into the UTXO format. Imagine there is a special UTXO transaction with an input and an output. Instead of directly pointing to some previous output for input, we attach a proof to the input. The proof is valid when it can convince everyone that the input coin is owned by the sender and has not been spent. Besides, the proof *will not* reveal which output the input relates to.

The property of not revealing connections between inputs and outputs can be achieved by *zero-knowledge proofs* (discussed in detail below). One of the more efficient cryptographic methods to generate such proof is zk-SNARK[1], which stands for "zero-knowledge, succinct and non-interactive", and it generates proofs that are short and easy to verify. Combining the zk-SNARK cryptographic tool within the UTXO framework, Zcash[5] extends Bitcoin's protocol by adding new types of transactions that provide a separate privacy-preserving currency, in which transactions reveal neither the payment's origin, destination, or amount. The new transactions also support both split and aggregation of the coins. Zcash uses zk-SNARK to generate efficient proofs which replace the traditional links between inputs and outputs. The proof is used to show that (1) the sender holds the secret keys corresponding to *some* of the public keys of previous outputs (the list of outputs is called *hiding set*); (2) and the amount spent is no more than the total amount of these outputs it holds. This helps in hiding both the user identity and the value of coins transferred. In this lecture, we study the Zcash architecture in detail and its integration atop the Bitcoin stack (see Figure 2).

Zero-knowledge Model

The zk-SNARK cryptographic tool is best introduced in the context of a basic model of a "zero-knowledge proof system" that will be used to formalize the setting and guarantees of zero knowledge proofs.

Language and NP. A language L is a set of statements such that $L(x) = 1$ if $x \in L$. Example: if x is a number, L can be an indicator of whether the number is composite. And NP is defined

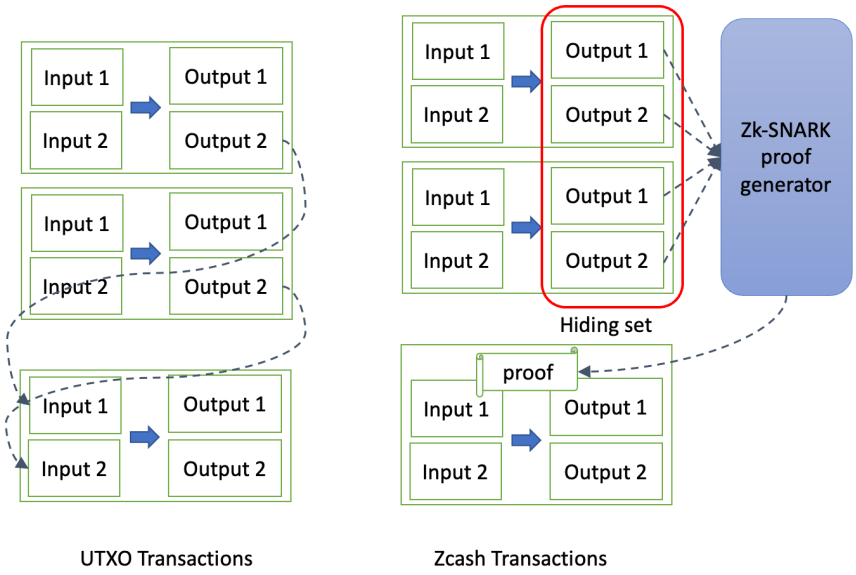


Figure 2: Left: UTXO structure where the unspent outputs from previous transactions are referred by inputs of a later transaction. Right: Zcash transaction structure with zero knowledge proof.

to be the class of languages L that have a polynomial time verifier V such that

$$L(x) = 1 \iff \exists w,\ s.t.\ V(x, w) = 1$$

where w is a polynomial sized witness, who can be used by a verifier given input x to determine whether $L(x) = 1$. Example: x is an integer, L is testing for composite. We can find a witness here to be the prime factorization of x, and the verifier can testify whether the product of w equals to x in polynomial time.

Prover and Zero-knowledge. Now consider a language L in NP, e.g. the composite testing problem. Assume there is a *prover* who has found a witness w (the prime factorization of x) and wants to prove to a verifier. But the verifier only has access to x, so the prover's task is to convince the verifier that x is composite ($L(x) = 1$). The prover can send w over and now the verifier can directly compute $V(x, w)$. However, this way the verifier has direct access to the witness.In the zero knowledge system, the prover wants to generate a *zero-knowledge proof*, using which the verifier will learn nothing about the specific witness w, while still completing the verification.

To understand how zero knowledge proofs are even possible, consider the following discrete logarithm example. Given x, a prover wants to prove she knows a witness w such that $g^w = x$, where g is a generator of a cyclic group with prime order q. With w and x, a verifier is easy to verify the fact, however, to achieve zero-knowledge, the verifier only has access to x and should learn nothing about w during the verification process. The proving process is described as below:

1. The prover picks a random $v \in \mathbb{Z}_q^*$, and sends the verifier $t = g^v$.
2. The verifier picks a random $c \in \mathbb{Z}_q^*$ and sends it back to the prover.

3. The prover computes $r = v - cx$ and returns r to the verifier.
4. Finally the verifier checks whether the condition $t = g^r x^c$ holds.

When prover really knows w, it's easy to verify the correctness (Completeness). And it can be proved that another party who does not know w can construct such proofs with negligible probability (Soundeness). And as we can see, during the proof and verification process, witness w is wrapped with random numbers so the verifier still learns *nothing* about the witness, hence the term "zero knowledge". Besides this example, we can construct zero knowledge proofs for many other problems. The remarkable fact is that all languages in NP have zero knowledge proofs. [3]. In this lecture we treat zero knowledge proofs as a black box interface (represented by the zk-SNARK cryptographic library) and learn how to invoke it appropriately.

Efficiency. The performance of the cryptographic tools (e.g., how long it takes to generate the proof or verify the proof, how large is the proof size) are critical to practical usage. We denote the execution time required to run $V(x, w)$ by T; this is the baseline complexity of verification. We consider four metrics of complexity of zero knowledge proof systems.

- *Prover complexity.* Efficient provers can generate proofs in expected time $O(T \log T)$. The complexity is only marginally more compared to the baseline.
- *Verification complexity and proof size.* A desirable characteristic of such proof systems is succinctness, informally meaning that the proof size is small and thus can be verified efficiently. Succinct proof sizes of constant or logarithmic compared to the statement size are possible and thus can be validated in expected time $O(1)$ or $O(\log T)$. Although the complexity is sublinear, the constants are large and verification is not that efficient for practice usage (e.g., in Zcash).
- *Interactivity of verification.* Most zero knowledge protocols are interactive, including the example we discussed above. However, noninteractive proofs are most attractive for blockchain applications. Although a generic technique to convert interactive protocols into noninteractive protocols while retaining security properties exists (the Fiat-Shamir heuristic), it works under ideal conditions (including needing a random oracle model) and a more tailored approach is of interest in practice.
- *Setup assumptions.* Many zero knowledge protocols depend on a "trusted setup", e.g. zk-SNARKs. Specifically, the parameters necessary to generate and verify the proofs must be computed by a trusted party. Otherwise, the protocol could be reverted and the money can be generated in the air. To improve on this, s protocols like zk-STARKs (Zero-Knowledge Scalable Transparent ARguments of Knowledge) utilize publicly verifiable randomness instead of trusted setup to create trustlessly verifiable computation systems. The engineering of these theoretical cyrptographic concepts into practical libraries is presently being actively pursued.

From Zero Knowledge Proof Systems to Zcash

We connect anonymity in Bitcoin to zero knowledge proof systems by first defining new data structures and addressing mechanisms on top of Bitcoin architecture.

1. **Address.** Same as Bitcoin, there are two types of addresses, public key address $\mathsf{addr}_{\mathsf{pk}}$ and secret key address $\mathsf{addr}_{\mathsf{sk}}$.

2. **A coin in Zcash** We define a coin c, which has the same role as a transaction output of Bitcoin, with the following attributes,

   - Coin commitment $\mathsf{cmt}(c)$
   - Coin value $v(c)$
   - Coin serial number $\mathsf{sn}(c)$
   - Coin address $\mathsf{addr}_{\mathsf{pk}}(c)$

   The commitment of a coin can be thought of a hash of all the data contained in the coin and serial number can be thought of as an output of pseudo-random generator. A collision-resistant hash function like the SHA256 compression function is used to generate all the addresses and the coin attributes.

3. **Pour transaction structure.** Pour transaction is a transaction that contains two inputs and two outputs. Compared to Bitcoin transaction, the pour transaction consumes the input coins by revealing their serial numbers, but does not reveal any other information such as the values of the input or output coins, or the addresses of their owners. All the publicly visible information of the transaction can be denoted as

$$\mathsf{tx} := (\mathsf{rt}, \mathsf{sn}_1^{\text{old}}, \mathsf{sn}_2^{\text{old}}, \mathsf{cmt}_1^{\text{new}}, \mathsf{cmt}_2^{\text{new}}, v_{\text{pub}}, \mathsf{info}, \mathsf{proof})$$

   where rt is the Merkle root, the inputs are serial numbers of two old coins, the outputs are commitments of two new coins. v_{pub} is the fraction of the input value that may be publicly revealed (optional) , info a transaction string (optional) and proof is used to prove the ownership of the old coins and the validity of the transaction. The pour transaction takes two coins as inputs and outputs so that it can implement both split and aggregation of the coins. If two coins can be split or aggregated in a pour transaction, multiple coins can be split or aggregated in multiple pour transactions. The details of these properties will be discussed below.

Zcash Framework

We begin by formalizing the transaction linkage problem that the UTXO state management system of Bitcoin faces. We would like to design a *pour transaction* that creates two new coins from two old coins without revealing the information of the coins (especially the public keys).

First Attempt: use commitment . The first attempt is to create the transaction only with the commitments of coins, i.e. a pour transaction contains $(\mathsf{cmt}_1^{\text{old}}, \mathsf{cmt}_2^{\text{old}}, \mathsf{cmt}_1^{\text{new}}, \mathsf{cmt}_2^{\text{new}}, \mathsf{proof})$. And the proof should imply

(1) the one who provides the proof has access to the old and new coins

(2) the coins satisfy $v(c_1^{\text{old}}) + v(c_2^{\text{old}}) \geq v(c_1^{\text{new}}) + v(c_2^{\text{new}})$

(3) it has access to $\mathsf{addr}_{\mathsf{sk}}(c_1^{\text{old}})$ and $\mathsf{addr}_{\mathsf{sk}}(c_2^{\text{old}})$.

We can state this problem formulation as an NP statement, black-boxing the zero knowledge proof generation process. The statement contains:

- $x = (\mathsf{cmt}_1^{\text{old}}, \mathsf{cmt}_2^{\text{old}}, \mathsf{cmt}_1^{\text{new}}, \mathsf{cmt}_2^{\text{new}})$;

- $L(x)$ is an indicator of whether x is a valid pour transaction. $L(x) = 1$ if x is a valid pour transaction;
- we define the witness $w = (c_1^{\text{old}}, c_2^{\text{old}}, c_1^{\text{new}}, c_2^{\text{new}}, \mathsf{addr}_{\mathsf{sk}}(c_1^{\text{old}}), \mathsf{addr}_{\mathsf{sk}}(c_2^{\text{old}}))$.

Given x and w, a verifier $V(x, w) = 1$ if the following conditions are true:

- $(\mathsf{cmt}(c_1^{\text{old}}), \mathsf{cmt}(c_2^{\text{old}}), \mathsf{cmt}(c_1^{\text{new}}), \mathsf{cmt}(c_2^{\text{new}})) = x$
- $v(c_1^{\text{old}}) + v(c_2^{\text{old}}) \geq v(c_1^{\text{new}}) + v(c_2^{\text{new}})$
- $\mathsf{addr}_{\mathsf{pk}}(c_1^{\text{old}})$ matches $\mathsf{addr}_{\mathsf{sk}}(c_1^{\text{old}})$ and $\mathsf{addr}_{\mathsf{pk}}(c_2^{\text{old}})$ matches $\mathsf{addr}_{\mathsf{sk}}(c_2^{\text{old}})$

It is easy to see that $L(x) = 1 \iff V(x, w) = 1$.

The methodology is to convert these statements into *algebraic circuits* and the proof is generated by "circuit satisfiability". The proof is an encrypted tuple $\pi = [g^H, g^Z]$, where H and Z are polynomials computed during the proving phase, and there is a verifying key $vk = g^T$. We recall in the example of the discrete logarithm above, the proving process did not reveal the information of the witness since it is hard to revert the logarithm. An example construction can be found in this post for further reading. Now anyone in the blockchain who has access to w can generate the proof without revealing any information related to w, and anyone who has access to x can verify the validity of the transaction. But this method has a vulnerability that the commitment is still traceable.

Second Attempt: use two types of commitments An idea to improve on the previous formulation is to use two types of commitments. The first is normal commitment cmt and the second is a unique serial number sn (generated by a pseudo-random generator). In a transaction, we use serial numbers to represent old coins and commitments to represent new coins, i.e., the transaction includes $(\mathsf{rt}, \mathsf{sn}_1^{\text{old}}, \mathsf{sn}_2^{\text{old}}, \mathsf{cmt}_1^{\text{new}}, \mathsf{cmt}_2^{\text{new}}, \mathsf{proof})$, where rt specifies the Merkle-tree root of the commitments of outputs in the ledger. The witness is still the same, $w = (\mathsf{rt}, c_1^{\text{old}}, c_2^{\text{old}}, c_1^{\text{new}}, c_2^{\text{new}}, \mathsf{addr}_{\mathsf{sk}}(c_1^{\text{old}}), \mathsf{addr}_{\mathsf{sk}}(c_2^{\text{old}}))$. And again $L(x) = 1 \iff V(x, w) = 1$, $V(x, w) = 1$ if the following conditions are true:

For $i \in \{1, 2\}$

- The commitments cmt_i of c_i^{new} appear on the ledger, verified using the Merkle-tree root rt, i.e., $\mathsf{cmt}(c_1^{\text{new}}) \in \mathsf{rt} \wedge \mathsf{cmt}(c_2^{\text{new}}) \in \mathsf{rt}$.
- The address secret key $\mathsf{addr}_{\mathsf{sk},i}^{\text{old}}$ matches the address public key of c_i^{old}
- $(\mathsf{sn}(c_1^{\text{old}}), \mathsf{sn}(c_2^{\text{old}}), \mathsf{cmt}(c_1^{\text{new}}), \mathsf{cmt}(c_2^{\text{new}})) = (\mathsf{sn}_1^{\text{old}}, \mathsf{sn}_2^{\text{old}}, \mathsf{cmt}_1^{\text{new}}, \mathsf{cmt}_2^{\text{new}})$
- $v(c_1^{\text{old}}) + v(c_2^{\text{old}}) \geq v(c_1^{\text{new}}) + v(c_2^{\text{new}})$.

Similar to the previous formulation, those who have w can generate the proof and who have x can verify the validity. However, compared to the previous method, there is no direct connection between old coins and new coins since they use different types of commitments. The only concern is that without the connection, how can we know which previous output is being spent? To solve this problem, we record all serial numbers appearing in previous transactions as a *nullifier set* and conduct an additional check to see whether the input serial number is already in the nullifier set.

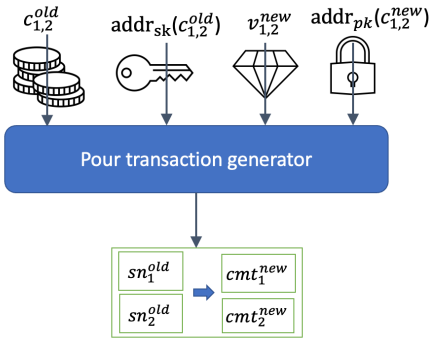


Figure 3: Generate a pour transaction.

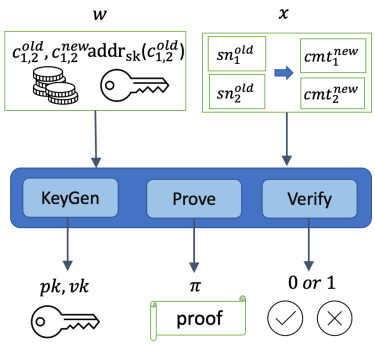


Figure 4: Generate a zk-SNARK proof.

Zcash Protocol: Putting it all together

The key modification of Bitcoin made by Zcash is the introduction of pour transactions. Different from a normal UTXO transaction, the inputs and outputs of a pour transaction are replaced by the commitments and serial numbers to break the link between old and new coins.

Create a pour transaction. Based on the second solution, the full structure of a pour transaction (as defined earlier) contains $(\mathsf{rt}, \mathsf{sn}_1^{\text{old}}, \mathsf{sn}_2^{\text{old}}, \mathsf{cmt}_1^{\text{new}}, \mathsf{cmt}_2^{\text{new}}, v_{\text{pub}}, \mathsf{info}, \mathsf{proof})$, to create a pour transaction, the sender needs to have the following information (see Figure 3):

- Old coins $c_1^{\text{old}}, c_2^{\text{old}}$;
- Secret keys of old coins $\mathsf{addr}_{\mathsf{sk}}(c_1^{\text{old}}), \mathsf{addr}_{\mathsf{sk}}(c_2^{\text{old}})$;
- New values $v_1^{\text{new}}, v_2^{\text{new}}$;
- Public value v_{pub} s.t. $v_1^{\text{old}} + v_2^{\text{old}} \geq v_1^{\text{new}} + v_2^{\text{new}} + v_{pub}$;
- New addresses $\mathsf{addr}_{\mathsf{pk}}(c_1^{\text{new}}), \mathsf{addr}_{\mathsf{pk}}(c_2^{\text{new}})$.

In a real system, a transaction generator can be called by a sender given required information to generate a pour transaction and the new coins. Then the new coins will be sent to the recipients off chain and the transaction will be post on chain.

Generate a zk-SNARK proof. zk-SNARK in a cryptographic zero knowledge, succinct and a non-interactive verification method. When a prover knows the witness for a NP-statement, they can produce a short proof that can be verified by anyone without revealing the witness. The NP-statements we will encounter in Zcash are going to be satisfyability statements such as "the hash function matches this particular value for this particular input". As a library, there are three polynomial time algorithms during the proving process, KeyGen, Prove and Verify. KeyGen is the trusted setup which generates proving and verifying keys, pk and vk respectively once and for all. The function Prove takes pk, the witness w and the public input x to output a short proof $\pi = \mathsf{Prove}(pk, w, x)$. The function Verify takes vk, the public input x and the proof π and outputs a Boolean value $\mathsf{Verify}(vk, x, \pi)$.

Incentives in Zcash. Zcash is a fork of Bitcoin main chain, and still follows the same basic protocol with an added privacy-preserving service. So the incentives in Zcash are the identical to that in the Bitcoin protocol, including both mining rewards and transaction fees. Since the zk proof verification process is efficient, the increased verification time is not significant (especially given the slow mining rate in Bitcoin).

References

Zerocoin [4] extends Bitcoin to provide a decentralized laundry system using zero knowledge proofs. Zero-knowledge proofs allow users to periodically convert their bitcoins into zerocoins of fixed denominations and later provide a proof that they own one of the zerocoins to recover their bitcoins. However, the proof used by Zerocoin is not efficient, which creates a large overhead on the blockchain and makes it less usable. Moreover it neither gives a mechanism to split or aggregate zerocoins nor lets the users transact in zerocoin, routine day-to-day transactions are still conducted in Bitcoin.

The Zcash architecture discussed in this lecture was originally proposed in this manuscript. A more informal description is presented here. The zk-SNARK library is described informally here.

References

[1] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 781–796, 2014.

[2] Michael Fleder, Michael S Kester, and Sudeep Pillai. Bitcoin transaction graph analysis. *arXiv preprint arXiv:1502.01657*, 2015.

[3] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np statements in zero-knowledge and a methodology of cryptographic protocol design. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 171–185. Springer, 1986.

[4] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411, 2013.

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