Proof malleability and other issues with zkSNARKs

based on

Non-Malleability of the Fiat--Shamir Transform Revisited for Multi-round SRS-Based Protocols

by Markulf Kohlweiss and Michał Zając

How zkSNARKs are used in blockchain?

Example - ZCash (simplified)
UTXO-based **anonymous** cryptocurrency
–hides who sends and receives money
–hides the amount transferred

set of coins
coin cm[i]
–serial number sn[i]
–value cm[i].value
–secrets related to coin's owner secret key

$\boxed{cm_1}\ \boxed{cm_2}\ \ldots\ \boxed{cm_n}$

-users can add coins, divide and transfer them
- users cannot change "internals" of coins: serial numbers, value
-no two coins have the same serial number

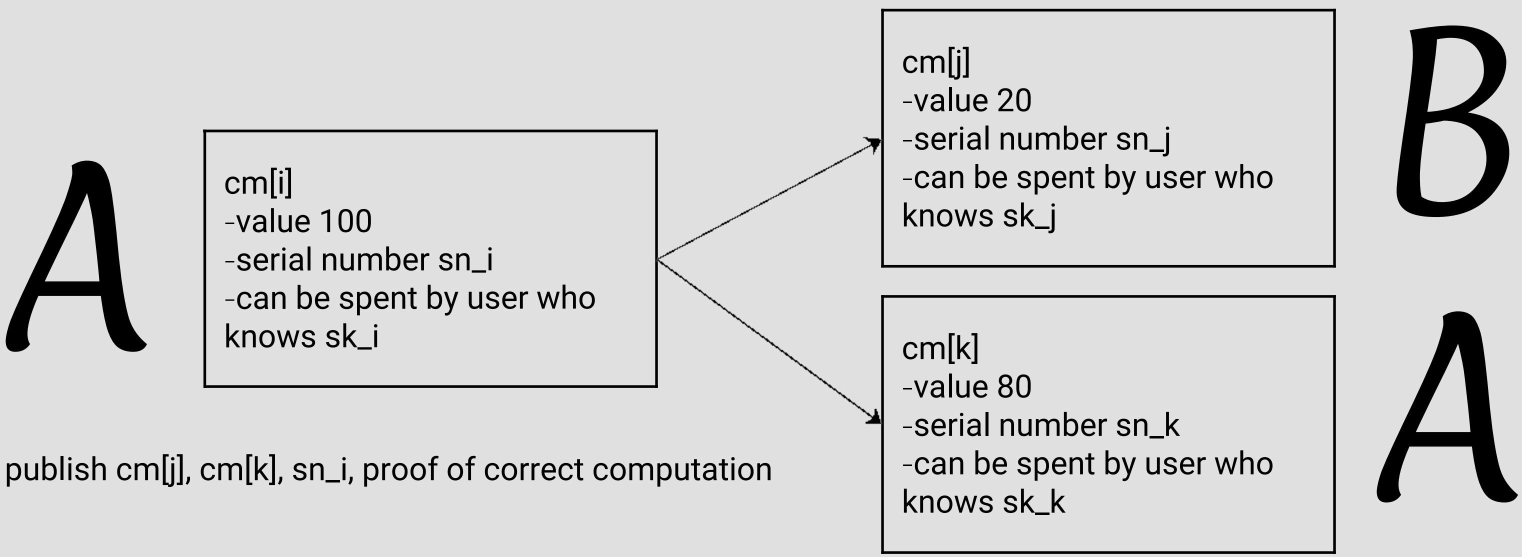
$\boxed{sn_{i_1}}\ \boxed{sn_{i_2}}\ \ldots\ \boxed{sn_{i_k}}$

Payment

To spend a coin cm[i] user:
−divides the coin cm[i] into new cm[j], cm[k]
    −cm[j].value + cm[k].value = cm[i].value
    −new coins = new serial numbers
−reveals cm[i]'s serial number sn := cm[i].sn
−shows that new coins have been generated correctly
−shows that it **knows** a coin-related secret
−shows that one of the coins cm[i] has serial number sn

if cm[j] is created by user A to pay user B
-cm[j] secrets are unknown to A (so A cannot spend cm[j])
-cm[j] contains (one of) public key of B (and depends on B's secret key)

Example: A wants to pay B $20



How to show knowledge?

To spend a coin cm[i] user:
– shows that published values were computed correctly
– shows that new coins were generated correctly
– shows knowledge of coin's secrets

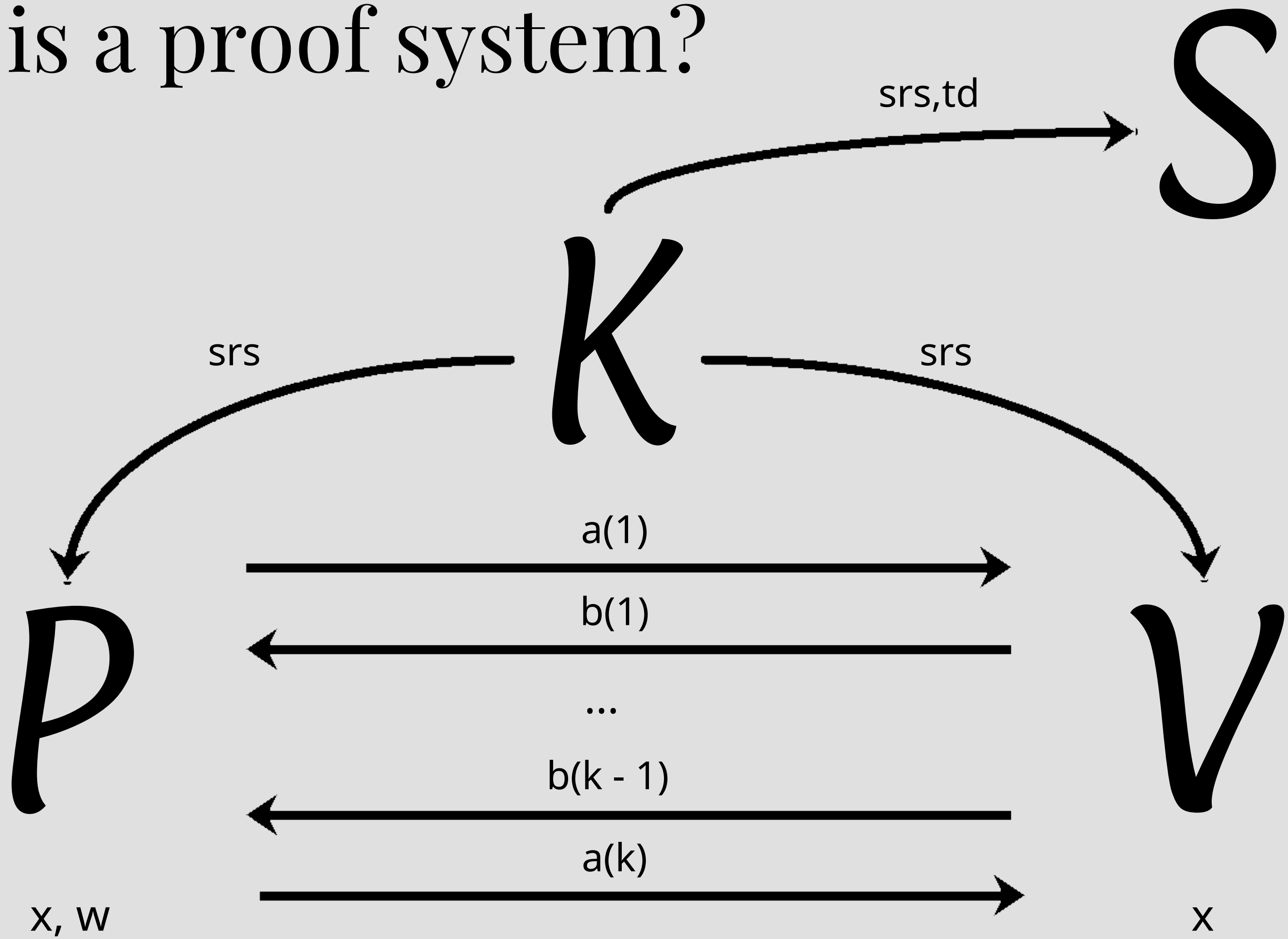
statement x

witness w **for** x: assures that x is correct
w: coin's secrets, computation details
we want witness to remain **private** (otherwise someone could spend the coin on behalf of the owner)

Zero-knowledge proof of knowledge

Soundness:
-No user should be able to convince other users on a false statement
Knowledge soundness:
-No user should be able to convince other users on a statement that it doesn't know a witness for
Zero-knowledge:
-The witness remains hidden
Succinctness:
-Proofs are short (zkSNARKs)

What is a proof system?



x: New coins cm have been generated correctly
w: coin cm[i], secret sk

Formally: Represent the problem as a circuit C
show that C(x, w) = 0

What can go wrong? SRS generation

Computational cost
-SRS size -- a few GB
-some proof systems require
one SRS per circuit

Solution
universal SRS—one for all circuits (up to given size)

Zero knowledge
-what if ***K*** **malicious** and tries to break **privacy** of *P* (attack zero knowledge)

Solution
show subversion-resistant ZK
-***P*** can verify whether the SRS gives ZK
-SRS can be generated fully maliciously

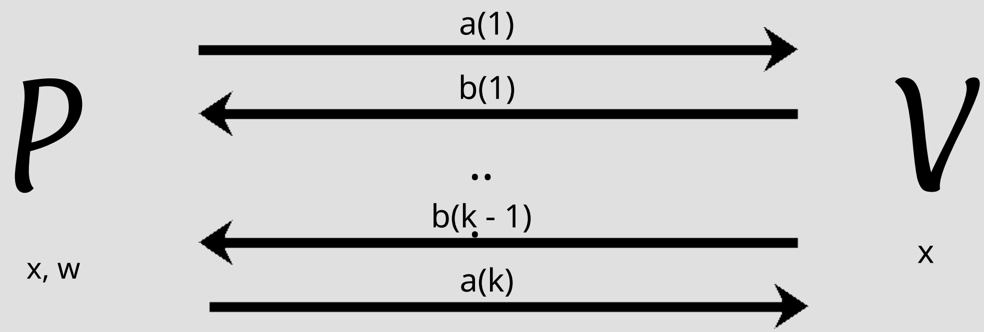
Soundness
-what if ***K*** **malicious** and tries to break **soundness** of the system (fool *V*)
-(note: each SRS comes with a trapdoor that allows to break soundness - required by zero knowledge)

Solution
MPC ceremonies for SRS generation
Show updatbility of SRS
-each prover can make a new SRS' based on the old one
-updates are verifiable
-SRS assures soundness if at least one of the updaters is honest

Interactive to non-interactive

Proof system designed as **interactive** protocol

Fiat-Shamir transformation to get **non-interactive** protocol

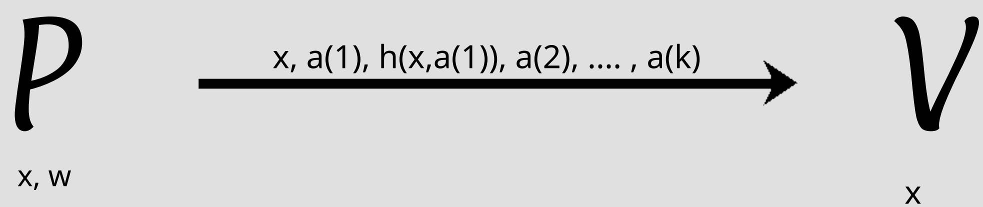


Random oracle
H: $\{0,1\}^* \rightarrow \{0,1\}^k$
Input x return random
$y \leftarrow \{0,1\}^k$

Replace V’s challenges by random oracle values

Real life - use **hash function**

Conjecture: the resulting protocol is secure



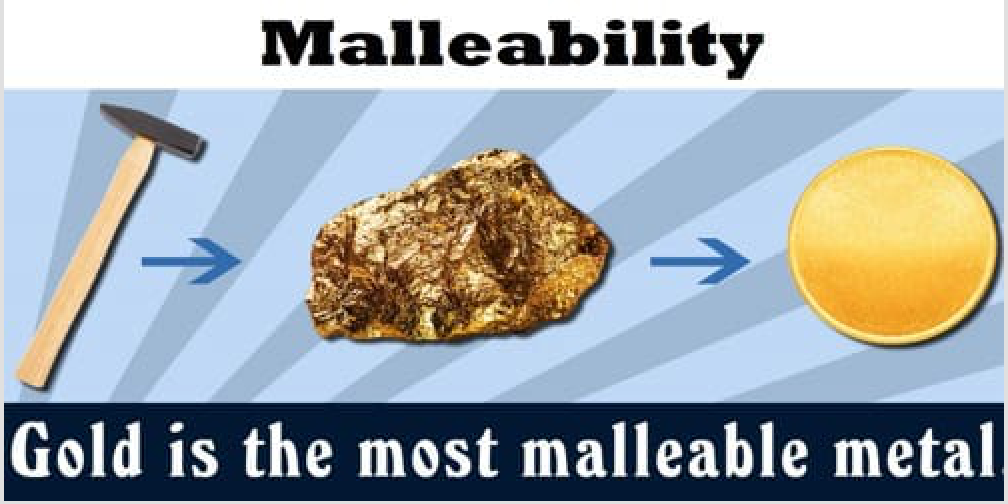
Problem:
Fiat-Shamir transformation proven **secure** only for a **specific class of protocols**
Most of interesting (for us) protocols are **outside** that class

Malleability

malleability the ability to be hit or pressed into different shapes easily without breaking or cracking (= starting to split)

The softness and malleability of gold makes it perfect for making jewellery.

After Oxford Learner's Dictionary



Malleability. Toy example.

Payment system
-A has a secret key sk that allows it to sign messages
-Everybody knows A’s public key, so they can verify A’s signatures
-To transfer funds, A signs a message m ***“Transfer X USD to B”*** - results in **signature** σ
-B shows the (σ, m) at the bank
-Bank verifies (σ, m) and checks whether (σ, m) is not **already stored**
-Bank transfers X USD from A’s account to B’

Malleability of signature
-given (σ, m) it may be possible to create a valid (σ’, m) **without knowing sk**

Malleable signature, malicious recipient B
-A signs a message m ***“Transfer X USD to B”*** - results in **signature** σ
-B shows the (σ, m) at the bank
-Bank verifies (σ, m) and checks whether (σ, m) is not **already recorded**
-Bank transfers X USD from A’s account to B’s
-B mauls (σ, m) into (σ’, m) and sends it to Bank **again**
-If (σ’, m) not recorded, Bank sends the funds **again**

Does malleability happens in real life?

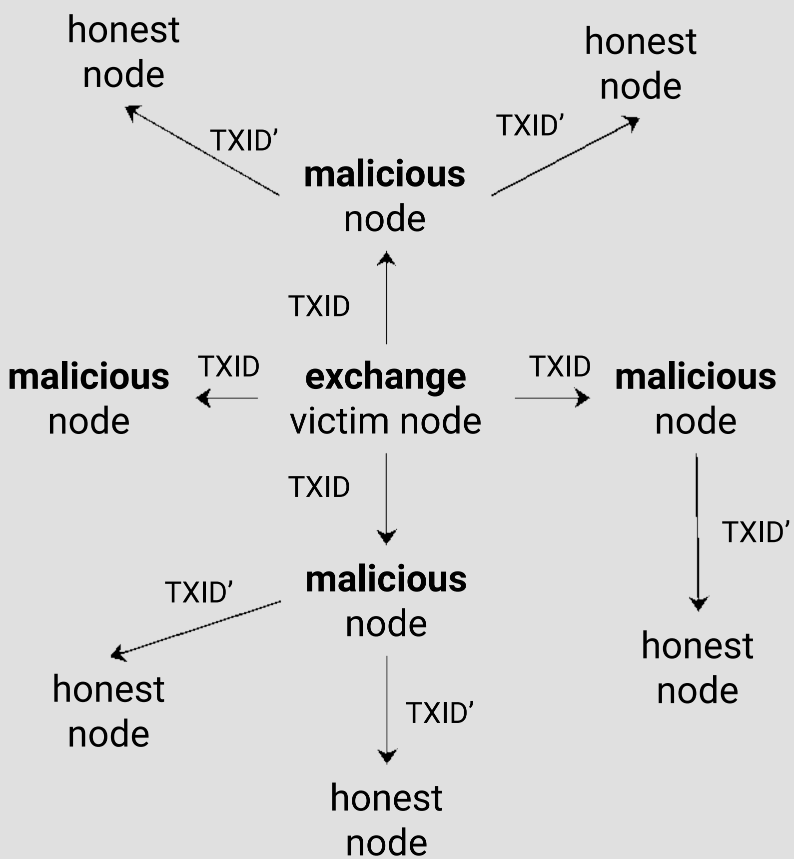
Bitcoin transaction malleability

transaction TX,
TXID = hash(TX)
part of TX: signature σ

signature σ is **malleable**

Change TXID without invalidating TX:
-malleate σ to σ'
-get a new TXID'

Attack scenario
-victim - exchange E
-adversary A makes a withdrawal from the exchange
-A gets payment with identifier TXID
-A mauls TXID into TXID' and propagates it to the network (TXID' still gives A money)
-TXID' gets included into a blockchain
-A claims to E that it did not get paid
-E checks the blockchain and does not see TXID
-E makes the payment again



Malleability of proof systems

To spend a coin cm[i] user:
– shows that published values were computed correctly
– shows that new coins were generated correctly
– shows knowledge of coin's secrets

statement x
witness w
proof π

if cm[j] is created by user A to pay user B
-cm[j] secrets are unknown to A but are known to B (so A cannot spend cm[j, but B can)
-cm[j] contains (one of) public key of B (and depends on B's secret key)

append tx, x, π

C

append tx, x', π'

x' - as x, but states C as the recipient, not B
π' - valid proof for x'

Note: C may not be able to maul tx to get funds meant for B, but can make the tx invalid

Assume we have a method that prevents that attack vector
Do we still need to care about malleability?
Reversed question: Do we **need** malleability?
If we don't - use non-malleable zkSNARK

Security model

Soundness:
-No user should be able to convince other users on a false statement
Knowledge soundness:
-No user should be able to convince other users on a statement that it doesn't know a witness for
Zero-knowledge:
-The witness remains hidden
Succinctness:
-Proofs are short (zkSNARKs)

Assumption Adversary doesn't see other proofs

Non malleable NIZK - adversary sees other proofs

Fighting malleability

Groth16 zkSNARK
-shortest proofs from all zkSNARKs
-efficient verification
-well studied - based on well-known assumptions, propert security analysis
-non-interactive out-of-the-box (no FS transformation)
-non-universal - one SRS per circuit
-randomizable - given proof π for statement x one can compute proof π' for the same statement

Making Groth16 non-malleable (simplified)
-Let Sig be a non-malleable signature scheme
-pkSig, skSig - one-time user's signing key and verification key
-hSig = PRF(pkSig)
-**add hSig to the statement**
-**prove that hSig was computed correctly**
-publish verification key pkSig
-sign the proof using skSig

What if we use a proof system that is universal and non-malleable out-of-the-box?

Plonk, Sonic
-most efficient* zkSNARKs with **universal updatable** SRS
-Plonk - created by researchers from Aztec
-Sonic - created by researchers from Zcash, IOHK, UCL
-less efficient than Groth16
-secure? security shown only for interactive versions
-non-malleable?

Our result

Plonk and Sonic		
universal SRS	✓	
updatable SRS	✓	
subversion zero-knowledge	✓	
secure (non-interactive protocol)	✗	✓
non-malleable	✗	✓

+

-Defined non-malleability of multi-round protocols using FS transformation
-Named properties that are required to have non-malleable proofs
-Shown security and non-malleability for a wide class of protocols
-Shown new properties of polynomial commitment schemes (important building block for zkSNARKs)
-On a way to give a framework that allows to create non-malleable, updatable zkSNARKs