Zcash was the first zk-SNARKs application. Zero-knowledge statement is true without revealing any information. Succinct the proofs are short and quick. Non-interactive: One message from prover to verifier. Trusted setup (how we generate proofs) can be compromised. To avoid that multiparty setup ceremony (a lot of people who creates a part of parameters and after delete secret data) Sprout and Sapling

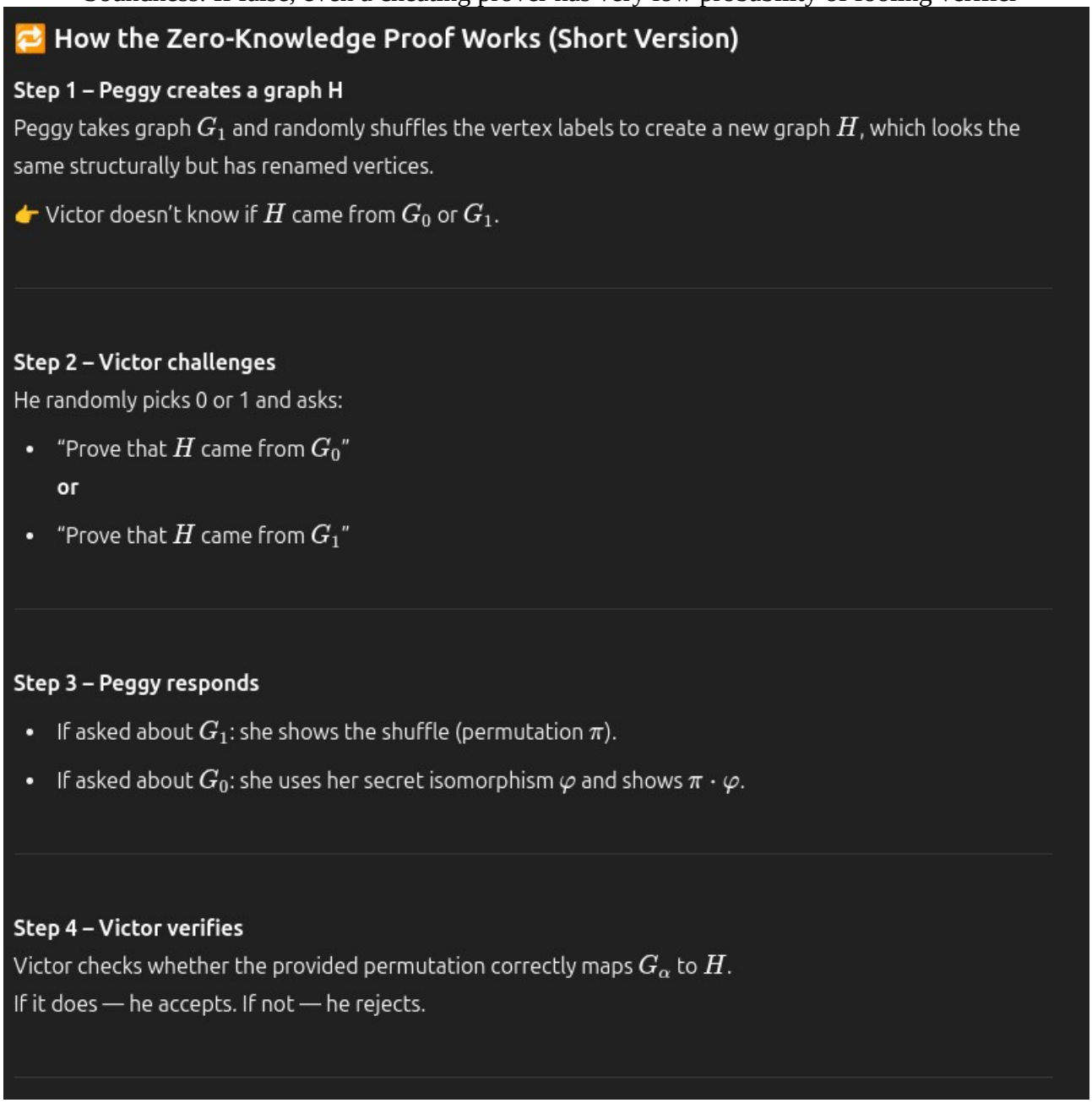
A Survey of Zero-Knowledge Proofs with Applications to Cryptography

If Peggy doesn't know the secret: She can only succeed if Victor happens to pick the same path she entered, which is a 50% chance. Repeating this test many times reduces the chance of her bluffing to near zero.

Two principles of zk is

Completness: If the statement is true, honest will always convince the verifier

Soundness: If false, even a cheating prover has very low probability of fooling verifier



building a ZKP for the **Graph 3-Colorability (G3C)** problem, which is **NP-complete**.

Since any NP problem can be reduced to G3C, ZKPs for all NP problems follow.

🔐 Zero-Knowledge Protocol for G3C (Protocol 4.2):

Step-by-step:

1. **Peggy (prover)** picks a valid 3-coloring and applies a random permutation to the colors.
2. She locks each vertex's color in a numbered box and sends all boxes to **Victor (verifier)**.
3. Victor randomly picks an edge and asks to see the colors of its endpoints.
4. Peggy gives keys to open only the two relevant boxes.
5. Victor checks that the two colors are different.

This is repeated **m^2 times** to reduce cheating probability.

The Fiat-Shamir protocol is a zk authentication method.

Zerocash: Decentralized Anonymous Payments from Bitcoin

Bitcoin is not anonymous. Use mixers (tumblers) gives limitations: delays, rely on trust, vulnerable to theft.
Zerocoin extends Bitcoin with zkproofs to anon tx. Zerocoin works as a decentralized mixing service but has performance and functionality limitations. You make zerocoin burning BTC and getting a secret, that it is u the one who created a coin. But noone in blockchain can see that it is yours. You are sending this coin to another person using zk knowledge without revealing your information. In blockchain «anonymously someone sent coin to Alice». Limitations: slow proofs, just fixed amount of money, high storage proofs and no direct payments in Zerocoins.
Zerocash fixes that.

Zk-Snark is Succinct, Non-interactive, ZK, proof of knowledge
pk — proving key — generate proof
vk- verification key — everyone verify it
Can proof every statement that can be transformed to NP language

Construction finctions:
Each coin has a secret number, in blockchain is hidden commitment(hash). All coins in Merkle Tree. Each coin is connected to a public key of recipients. One time digital signatures and MACs to prevent tampering with transaction. Public outputs to Bitcoin addresses.

Zerocash overview:
practical implementation of decentralized anonymous payment (DAP), bult with 128- bit security. This is zk-SNARK proof system.

Zerocash POUR — Private Anounymous Statemend. SHA256, POUR circuit (optimizing of inner schemes)

zk-SNARK vs NIZK. NIZK verification time and proof size depend on the complexity of the statement.

The most efficient zk-SNARKs for arithmetic circuits use Quadratic Arithmetic Programs (QAPs). Security in QAPS : 1. Knowledge-of-Exponent (KoE) Assumption: If someone creates a valid zk-SNARK proof, they must actually know the secret (called a "witness"). 2. Diffie–Hellman Assumptions in Bilinear Groups: These are standard assumptions in cryptography that say:
It's hard to reverse or forge values like g**(ab) unless you know a or b. Ensures the cryptography (like commitments and encryption) are secure.

Three main zk-SNARK implementations: Parno et al.en-Sasson et al. [BCG+13], Ben-Sasson et al. [BCTV14]

Key data structures used in DAP. On top of blockchain.
Key Components:
1. Ledger
2. Public params — global constants created during setup
3. Addresses -addrpk — public to recieve, addrsk- secret to spend (gift in box and you have a key from box)
4. Coins (c) — cm(c)- commitment(hidden representation), v(c)- coin value.
New transaction types:
1. Mint Transaction (txMINT) — create, add commitment to the ledher
2. Pour Tx (txPOUR) — spend to old coins and create two new ones.
All coin commitments (CMLIST) are stored in a mercel tree. Each txPour refers to the root of this tree. You scan the blockchains, find boxes with encrypted coins. TxPOUR to output coins in top level blockchain

Algorithms in DAP scheme:
1. Setup — input security parameter, output public parameters
2. CreateAddress — create a payment address using pp get an adress key pair
3. Mint — inputs: pp, vakue, addrpk; output: coin c, txMint with coin commitment cm. Deposit funds into the private sytem
4. Pour — inputs: old coins and their secrets, root, new values and addresses, public value, info; output: newcoins, txpour
5. VerifyTransaction — input: pp, tx, ledger; output 1 or 0
6. Receive- input key pair, ledger; output list of unspent coins

Any valid, unspent coins (not used before) can always be spent using the Pour algorithm.

Three key points of security:
1. L-IND. Ledger Indistinguishability (ledger does not leak secret information)
2. TR-NM. Transaction Non Malleability (attacker cannot modify a valid pour txs)
3.BAL. Balance. Total spent ≤ Total minted + received

To build a DAP scheme system relies of following standards:
1. Collision-Resistant Hash Function (CRH)
2. Pseudorandom Functions (PRFs)
3. Statistically-Hiding Commitments (COMM). Hide data until box opening COMM(randomness, value). Statistically — no computational power.
4. One-Time Strongly Unforgeable Digital Signatures (Sig)
5. Key-Private Public-Key Encryption (Enc)

zk-SNARK for POUR Transactions.
Pour statement validate that a txPOUR is rightly generated.
Proves statement in this zero knowledges:

Ownership of old coins, well-formedness of new coins(correct values + addresses), balance check and non-malleability enfrocements.

Completeness and Security: Complete — valid coins can always be spent, secure — satisfies three key points of security.
ONLY setup should be trusted

Signature Sceme : ECDSA for compatibility with Bitcoin. One-time strong unforgeability
Encryption Scheme: ECEIS.

CH : SHA-256 circuit optimizing, that was written according specification of SHA-256. You divide 32bit number and make a SHA256 check one bit by one and you do not need to check all 32 together.
Example:
C = A XOR B
Ci = Ai XOR Bi
A: 1 0 1 0 1 0 1 0 ...
B: 0 1 1 0 0 1 0 1 ...
XOR: 1 1 0 0 1 1 1 1 ... → C

The arithmetic circuit CPOUR verifies: Merkle Tree Membership (coin commitment), Integer Addition, Integer Comparision. Gate is operation OR AND XOR NOT

Can be built on top of every ledger currency. Is not directly compatible with the existing Bitcoin network.
Two integration strategies:
1. Replace Bitcoin, use only zerocash DAP interface. Every tx requires zk-SNARK proof, which is expensive, bitcoins scripting features are lost, you should confirm transaction before spending it (not normal bitcoin behaviour)
2. Dual — Currency system, bitcoin = zerocoin, coexistance. Nodes validates tx using verifytransaction and update merkle tree
You can also extend Bitcoin to make it compatable for Zerocash. Hardfork or additional scripts.

Performance:
1. zkSNARK Performance (POUR Statement)
keygen: 5-8 min
proofgen — 3 min but can be lowered to 46s using multiple threads
proofver — 5-8 ms
proofsize 288bytes
2. DAP performance
Setup 5 min
CreateAddress 326 ms
Mint 23 microsec
Pout 2min
VerifyTransaction : 6 ms
Receive 1.6 ms

Transaction confirmation about 180 seconds
Block Verification 80ms (PoW, verify tx, 5-8 ms more per each tx)

Extension:

1. Everlasting Anonymity
 Use each address only once
 Not include sensitive data in ciphertext.
Each pour transaction creates two new ciphertexts, C1 and C2. Each of them encrypts the following information about the new coin: Coin value (v_new), Random parameters (ρ, r, s), Everything the recipient needs to spend the coin in the future
2. Fast Block Propogation. VerifyTX only after propogation
3. Improved Storage. Reduce CMList and SNList