SAVER:

Snark-friendly, Additively-homomorphic, and Verifiable Encryption/decryption with Rerandomization

Jiwon Lee[†], Jaekyung Choi*, Jihye Kim*, Hyunok Oh[†]

Hanyang University[†], Kookmin University*

jiwonlee@hanyang.ac.kr

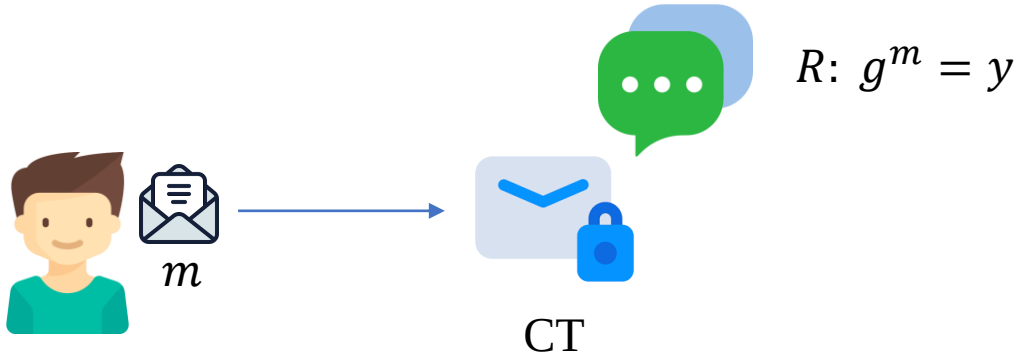
SECURITY & PRIVACY LAB.

2020-05-11 ZKproof 3rd Workshop

Part I: About SAVER

Verifiable Encryption?

- An encryption scheme which proves that a ciphertext encrypts a plaintext satisfying a certain relation R. [CS04]
- Proves that CT is generated correctly, in a certain format
- Relations are fixed, focusing on the validity

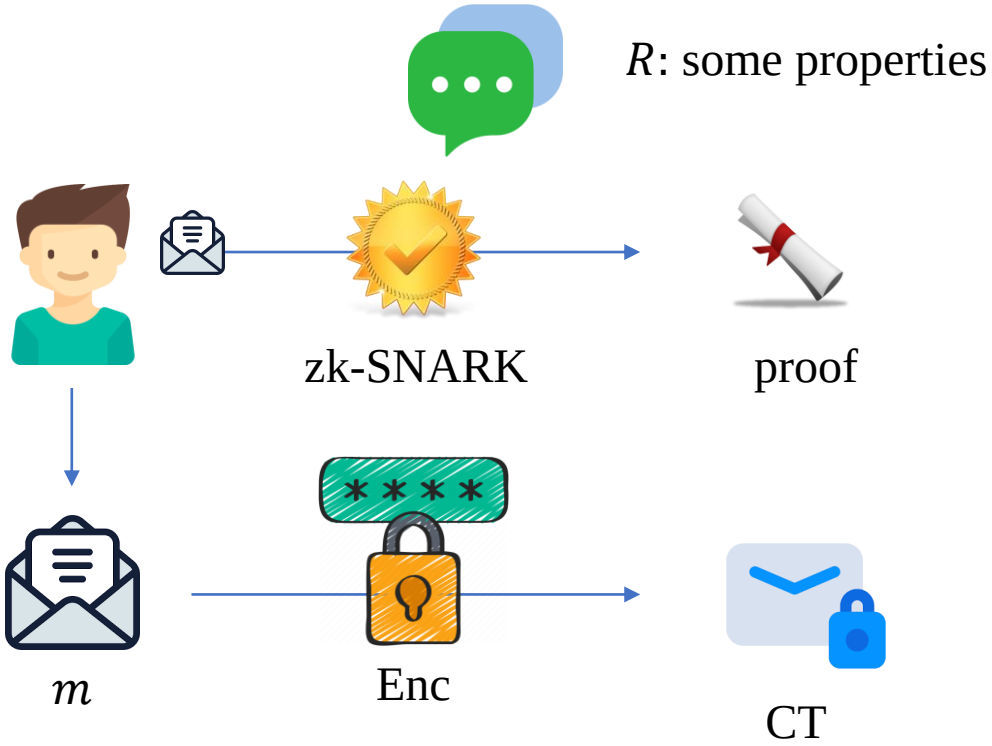


verifiable encryption of a discrete logarithm [CS04]

[CS04] Camenisch, Jan, and Victor Shoup. "Practical verifiable encryption and decryption of discrete logarithms." *Annual International Cryptology Conference*. Springer, Berlin, Heidelberg, 2003.

zk-SNARK + enc = Universal Verifiable Encryption (UVE)

- Encrypt while proving the arbitrary properties of the message
- Useful for many practical applications!



UVE for practical applications

- E-commerce:

raw product transaction

the product satisfies “certain properties”
ex: metadata, copyrights...

UVE for practical applications

- E-commerce:

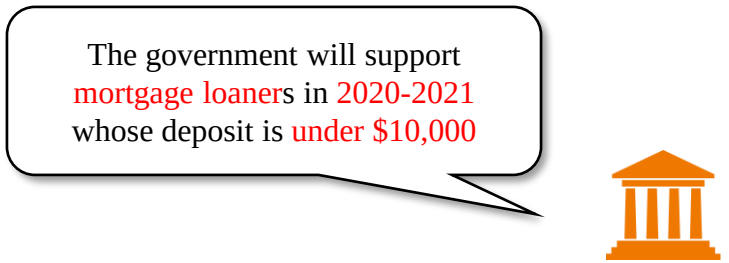
raw product transaction

the product satisfies “certain properties”
ex: metadata, copyrights...

- Contract:

raw contract contract

the contract satisfies “certain properties”
ex: contract type, date, deposit...



UVE for practical applications

- E-commerce:

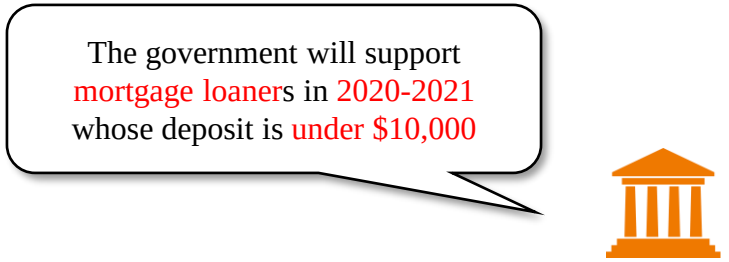
raw product transaction

the product satisfies “certain properties”
ex: metadata, copyrights...

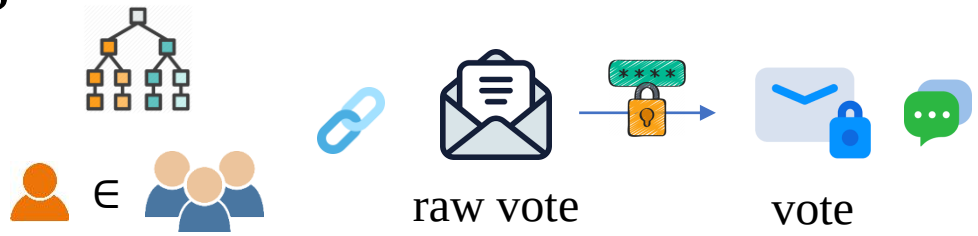
- Contract:

raw contract contract

the contract satisfies “certain properties”
ex: contract type, date, deposit...



- Voting:

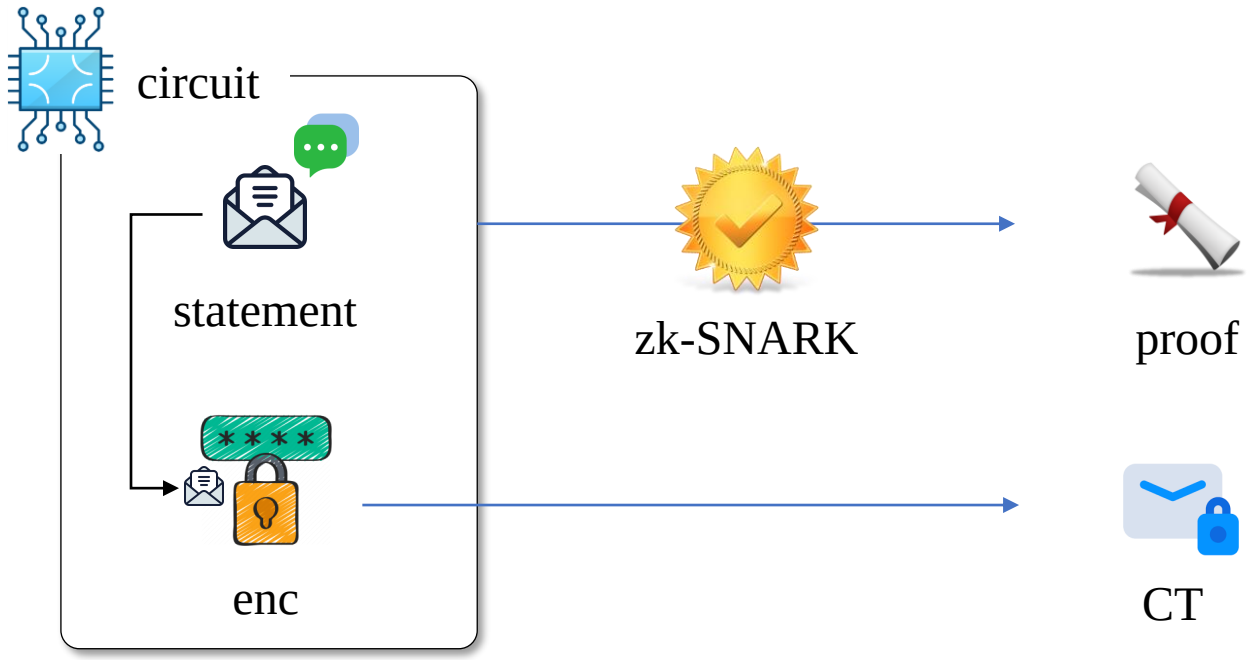


membership test raw vote vote

the voter (linked to m) is “within the membership”
the vote is “valid (ex: vote sum is integer 1)”

Encryption-in-the-circuit (encode directly)

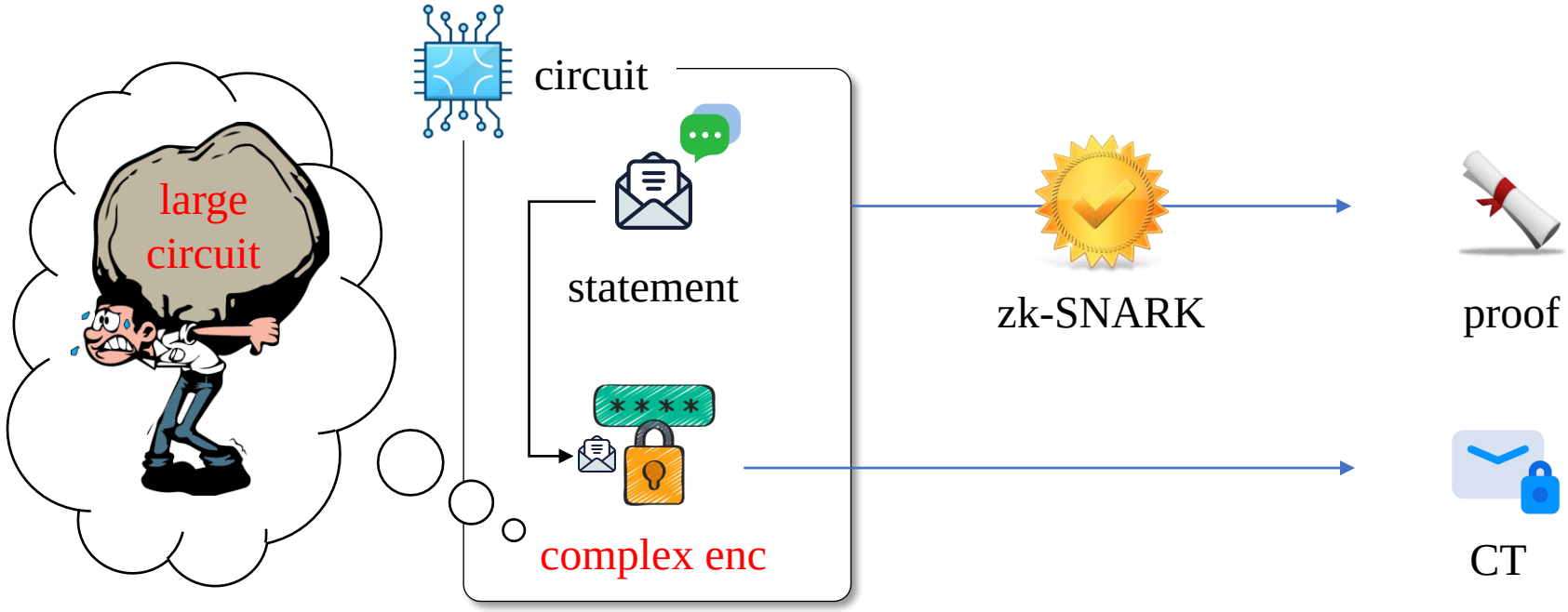
- How to construct UVE (zk-SNARK+enc)?
- Encryption-in-the-circuit: place encryption in the SNARK relation



Encryption-in-the-circuit

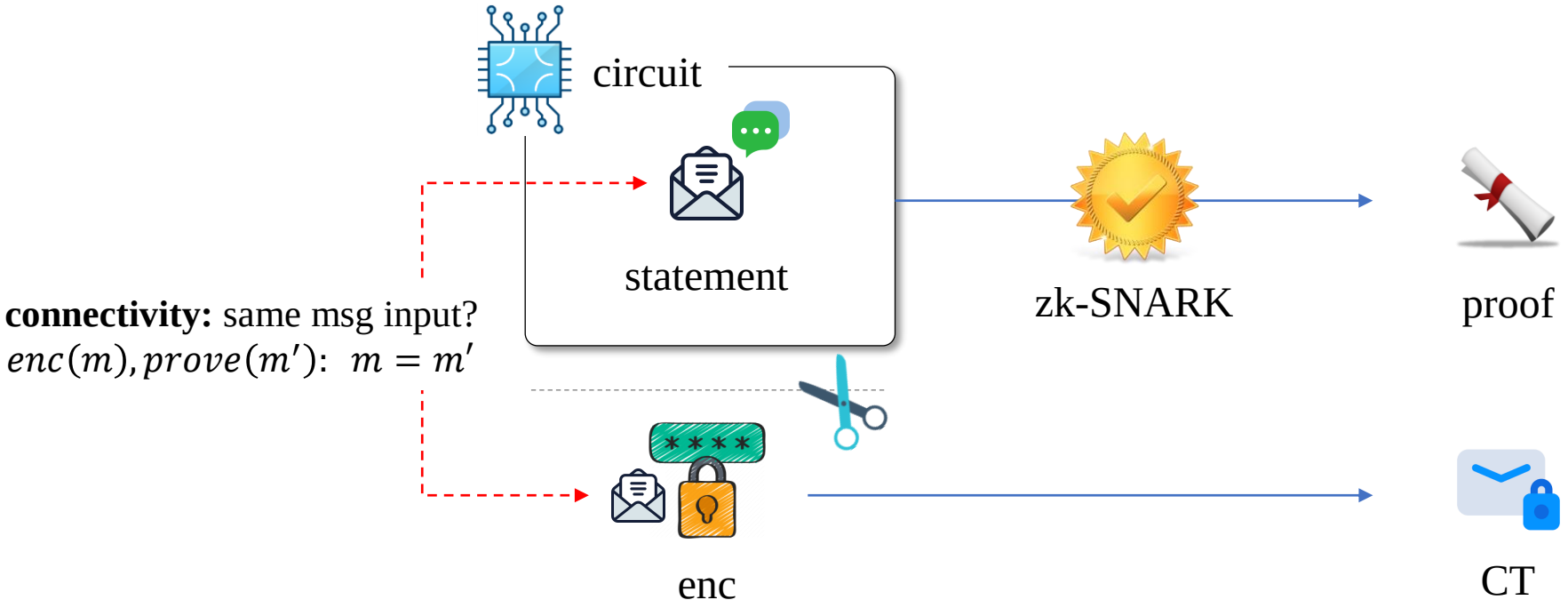
- For simple encryption, acceptable... (but not efficient)
  RSA-OAEP-2048 enc: 8.9s proving time, 216MB-sized CRS in [Gro16]

- Unrealistically heavy, if encryption needs more functionalities
  ex: identity-based encryption, attribute-based encryption...



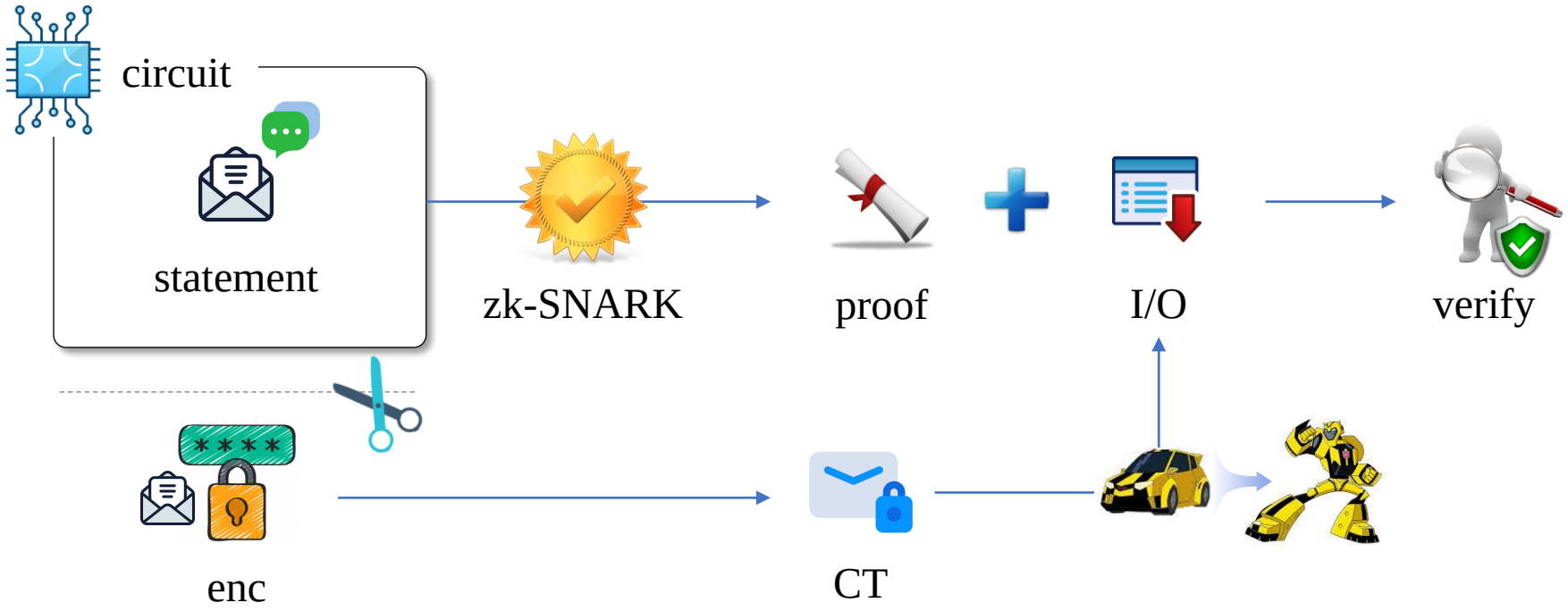
How to make it better?

- Let us detach encryption from the SNARK circuit!
- Encrypt outside the circuit, and let it be compatible with the SNARK
- In this case, connectivity between enc and zk-SNARK is important



Check the connectivity in SNARK verification

- Message is the input (I/O) of SNARK
- Linear encodings (g^m) look like exp-ElGamal enc ($g^m \cdot h^y$)
- Possible to design a CT compatible with the I/O of SNARK verification
- Can be understood as an extension of cc-SNARK (from LegoSNARK) to the encryption



- A universal verifiable encryption, for arbitrary properties
- Can be connected to the SNARK verification
- Avoid encryption-in-the-circuit

Verification plug-in of SAVER (simplified)

- Used [Gro16] for the zk-SNARK
- After the zk-SNARK setup, use CRS as an ingredient for encryption PK
- Split the message into n small blocks (for exp-ElGamal: finding DL)
- Encrypt by mixing a random to the I/O shaped message
- Plug-in the CT instead of I/O in equality check

[Gro16] setup

$$CRS = \{G, H, G^{\delta}, G^{\gamma}, G_i \dots\}$$

$$G_i = G^{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}}$$

Verification plug-in of SAVER (simplified)

- Used [Gro16] for the zk-SNARK
- After the zk-SNARK setup, use CRS as an ingredient for encryption PK
- Split the message into n small blocks (for exp-ElGamal: finding DL)
- Encrypt by mixing a random to the I/O shaped message
- Plug-in the CT instead of I/O in equality check

[Gro16] setup

$$CRS = \{G, H, G^{\delta}, G^{\gamma}, G_i \dots\}$$

$$G_i = G^{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}}$$

SAVER keygen

$$s_i, t_i \leftarrow Z_p^*$$

$$PK = G^{\delta s_i}, G_i^{t_i}, G^{-\gamma \cdot (\Sigma_{j=1}^{n} s_j)}, \dots$$

Verification plug-in of SAVER (simplified)

- Used [Gro16] for the zk-SNARK
- After the zk-SNARK setup, use CRS as an ingredient for encryption PK
- Split the message into n small blocks (for exp-ElGamal: finding DL)
- Encrypt by mixing a random to the I/O shaped message
- Plug-in the CT instead of I/O in equality check

[Gro16] setup

$$CRS = \{G, H, G^{\delta}, G^{\gamma}, G_i \dots\}$$

$$G_i = G^{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}}$$

SAVER keygen

$$s_i, t_i \leftarrow Z_p^*$$

$$PK = G^{\delta s_i}, G_i^{t_i}, G^{-\gamma \cdot (\Sigma_{j=1}^{n} s_j)}, \dots$$

parse message

$$M = (m_1 || \cdots || m_n)$$

$|m_i| = short\ bits$ (e.g. 4)

in decrypt

$obtain\ T^{m_i}$ for $T \in G_T$

$find$ discrete log of T^{m_i}

Verification plug-in of SAVER (simplified)

- Used [Gro16] for the zk-SNARK
- After the zk-SNARK setup, use CRS as an ingredient for encryption PK
- Split the message into n small blocks (for exp-ElGamal: finding DL)
- Encrypt by mixing a random to the I/O shaped message
- Plug-in the CT instead of I/O in equality check

encrypt

$$PK = G^{\delta s_i}, G_i^{t_i}, G^{-\gamma \cdot (\Sigma_{j=1}^{n} s_j)}, \dots$$

$$CT_i = \{G^{\delta s_i \cdot r} \cdot G_i^{m_i}\}$$

$$\pi = (A, B, C)$$

$$C_{new} \leftarrow C \cdot G^{-\gamma \cdot \left(\Sigma_{j=1}^{n} s_j\right) \cdot r}$$

Verification plug-in of SAVER (simplified)

- Used [Gro16] for the zk-SNARK
- After the zk-SNARK setup, use CRS as an ingredient for encryption PK
- Split the message into n small blocks (for exp-ElGamal: finding DL)
- Encrypt by mixing a random to the I/O shaped message
- Plug-in the CT instead of I/O in equality check

encrypt

$$PK = G^{\delta s_i}, G_i^{t_i}, G^{-\gamma\cdot(\Sigma_{j=1}^{n} s_j)}, \dots$$

$$CT_i = \{G^{\delta s_i \cdot r} \cdot G_i^{m_i}\}$$

$$\pi = (A, B, C)$$

$$C_{new} \leftarrow C \cdot G^{-\gamma\cdot\left(\Sigma_{j=1}^{n} s_j\right)\cdot r}$$

verification

$$e(A,B) = e\left(G^{\alpha}, H^{\beta}\right) \cdot e(\Pi_{i=1}^{n} CT_i, H^{\gamma}) \cdot e(C_{new}, H^{\delta})$$

$$\Pi_{i=1}^{n} e\left(G^{\delta s_i \cdot r} \cdot G_i^{m_i}, H^{\gamma}\right) \cdot e(C_{new}, H^{\delta})$$

$$e(G,H)^{\gamma\cdot\Sigma_{i=1}^{n}\delta s_i \cdot r} \cdot e\left(\Pi_{i=1}^{n} G_i^{m_i}, H^{\gamma}\right) \cdot e\left(C \cdot G^{-\gamma\cdot\left(\Sigma_{j=1}^{n} s_j\right)\cdot r}, H^{\delta}\right)$$

$$e(A,B) = e\left(G^{\alpha}, H^{\beta}\right) \cdot e\left(\Pi_{i=1}^{n} G_i^{m_i}, H^{\gamma}\right) \cdot e(C, H^{\delta})$$

same as [Gro16] vfy

Other functionalities in SAVER

- Additively-homomorphic : $CT_a \cdot CT_b = CT_{(a+b)}$

  obviously, from linearity of exponential ElGamal encryption

- Verifiable Decryption: $M, \nu \leftarrow Dec(CT), Verify\_dec(CT, M, \nu)$

  the uniqueness of decrypted plaintext is verifiable without the SK

- Rerandomization: $Rerand(CT) \approx Enc(Dec(Enc(M)))$

  the ciphertext and proof is rerandomizable (unlinkable to the original)

Application: Vote-SAVER

- Voting system built from the SAVER
- Encrypt the vote, prove the voter's right (membership test)
- Properties

  non-malleability: the result is tamper-proof — blockchain

  receipt-freeness: the voter cannot reproduce the vote (for vote-buying) — rerandomization

  individual verifiability: the voter can ensure his vote exists in the result — verifiable encryption

  universal verifiability: anyone can check the validity of the tally result — additive-homomorphism verifiable decryption

  voter anonymity: the voter's identity is hidden, even from any authority — zk-SNARK

  non-repudiation: the vote can be only generated from the voter's SK — zk-SNARK

Vote-SAVER framework

Admin

Blockchain

CRS_R

membership test validity of M **R**

Alice

My (Alice) public key is pk_{Alice}

$pk_{ID} = H(sk_{ID})$

Bob

My (Bob) public key is pk_{Bob}

$pk_{ID} = H(sk_{ID})$

Vote-SAVER framework

Voter(s)

Blockchain

$\boldsymbol{CRS_R}$

membership test
validity of M $\boldsymbol{R}$

election key

Admin

Voters *pklist*
Alice: pk_{Alice}
Bob: pk_{Bob}
⋮

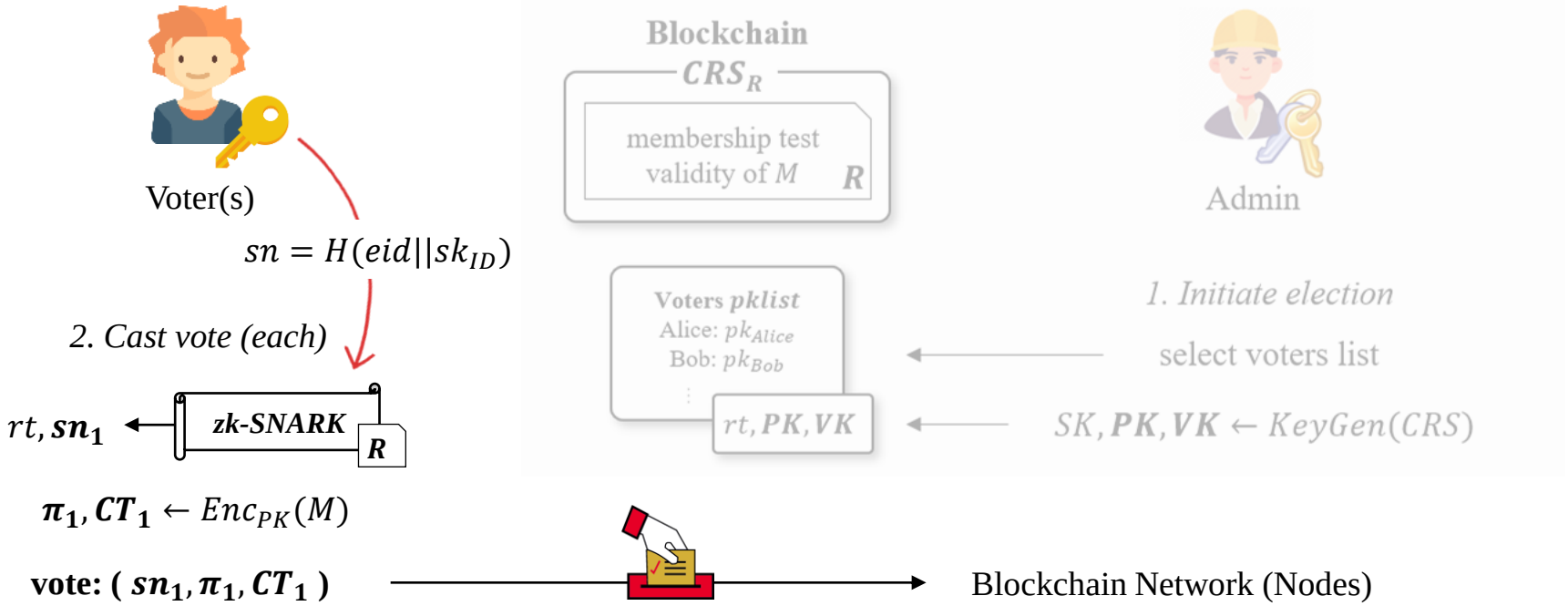
$rt, \boldsymbol{PK}, \boldsymbol{VK}$

1. Initiate election

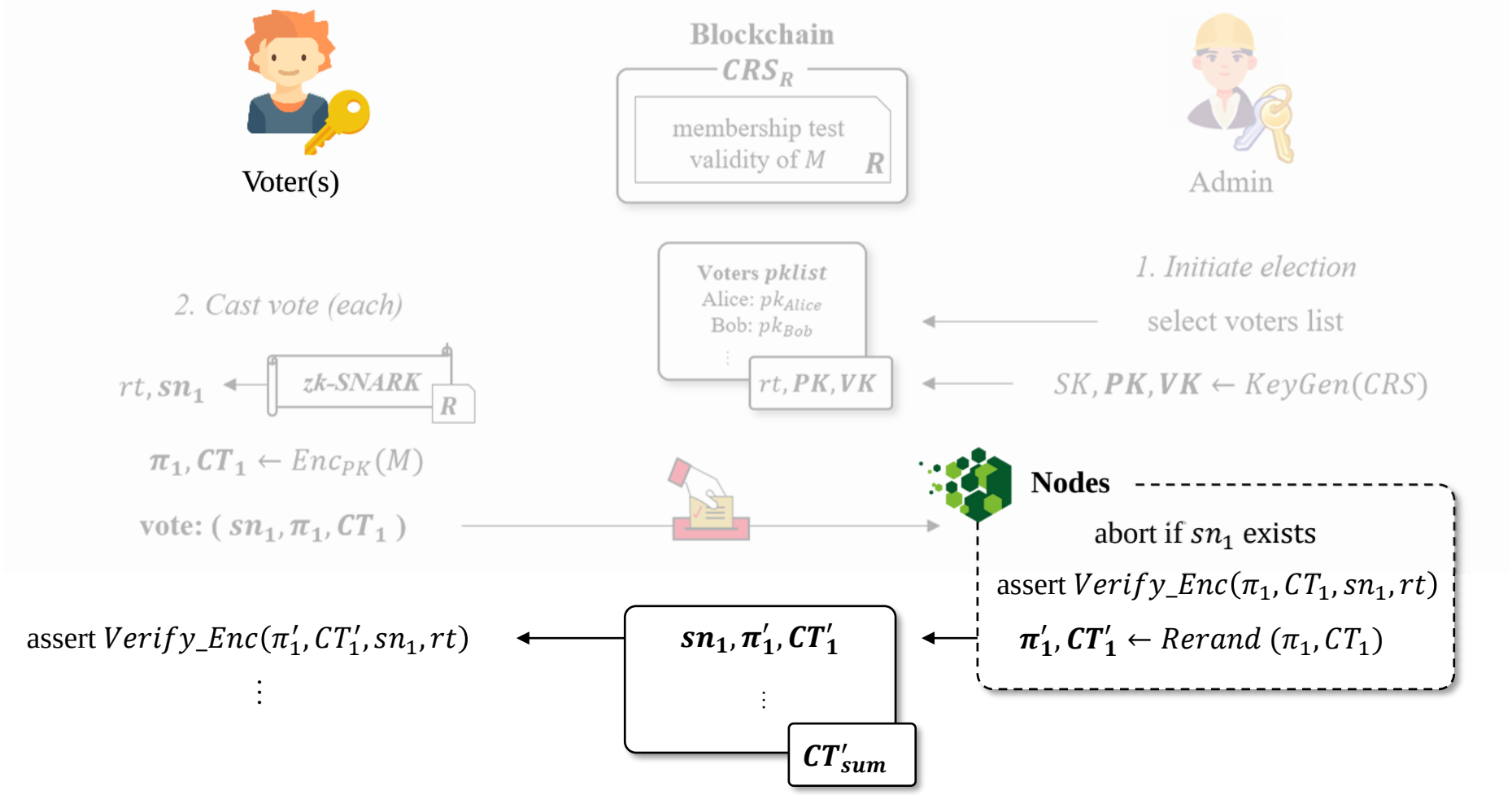
select voters list

$SK, \boldsymbol{PK}, \boldsymbol{VK} \leftarrow KeyGen(CRS)$

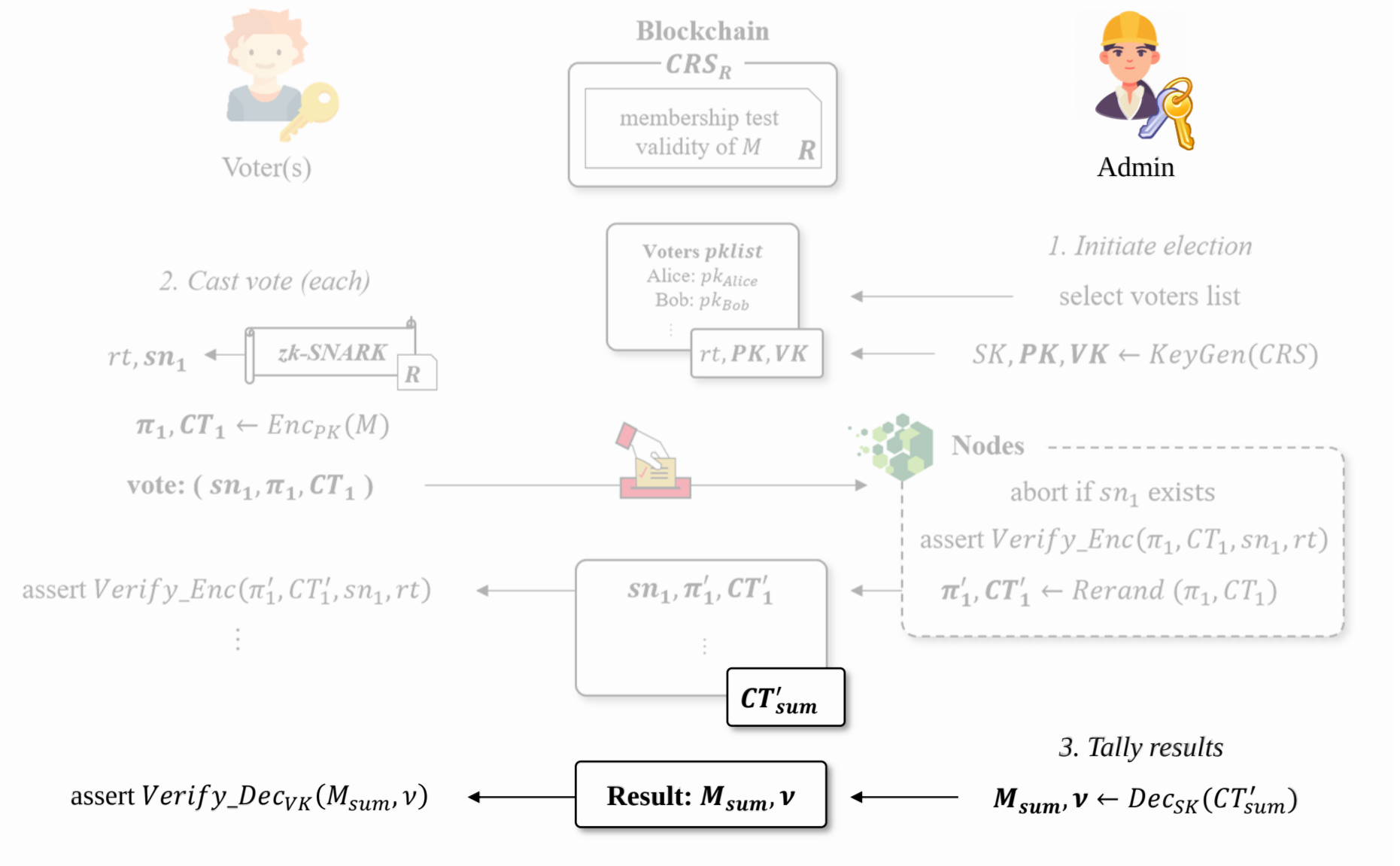
Vote-SAVER framework



Vote-SAVER framework



Vote-SAVER framework



What's different?

- Receipt-freeness & verifiability
  - ✓ Vote is receipt-free, due to the rerandomization
  - ✓ Individual, universal, eligibility verifiability altogether
- Voter is the SK holder
  - ✓ Other systems: authority distributes the key (compromises privacy)
  - ✓ Vote-SAVER: even authority cannot identify the voter

Implementation

- SAVER implemented on Ubuntu 3.4Ghz machine
- Relation: Vote-SAVER (membership test + vote validity) from Ajtai hash tree of height 16
- Main results

	$\|M\| = 256bits$	$\|M\| = 512bits$	$\|M\| = 1024bits$	$\|M\| = 2048bits$
SAVER encrypt — **encrypt**	1.6 ms	2.4 ms	7.4 ms	8.8 ms
SAVER encrypt — **[Gro16] prove**	0.73 s	0.73 s	0.73 s	0.74 s
PK size	1.22 KB	2.27 KB	4.36 KB	8.55 KB
CRS size	16 MB	16 MB	16 MB	16 MB

❖ Source codes available @ https://github.com/snp-lab/SAVER

❖ Real voting system demo available @ https://www.okvoting.com

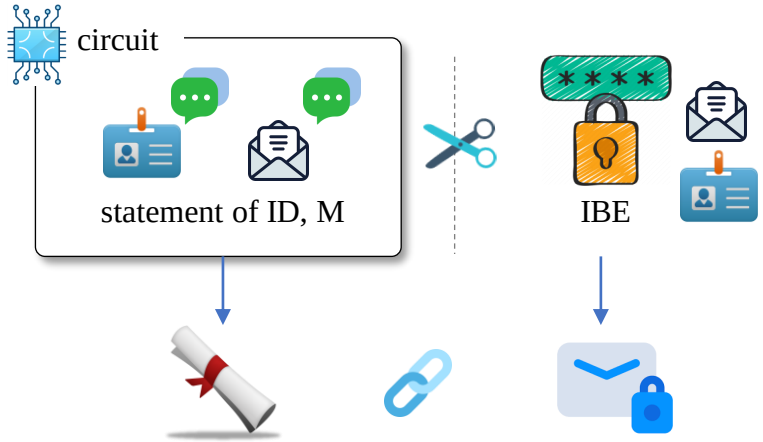
Summary of contributions

- Universal verifiable encryption: prove any properties of the message
- Snark-friendly: encrypt outside the circuit, connect to the zk-SNARK
- Functionalities: such as rerandomization, verifiable decryption
- Security: formal proof for knowledge soundness, IND-CPA, etc.
- Vote-SAVER: specific novel application, where voter holds the SK
- Implementation: experimental results and real demo system

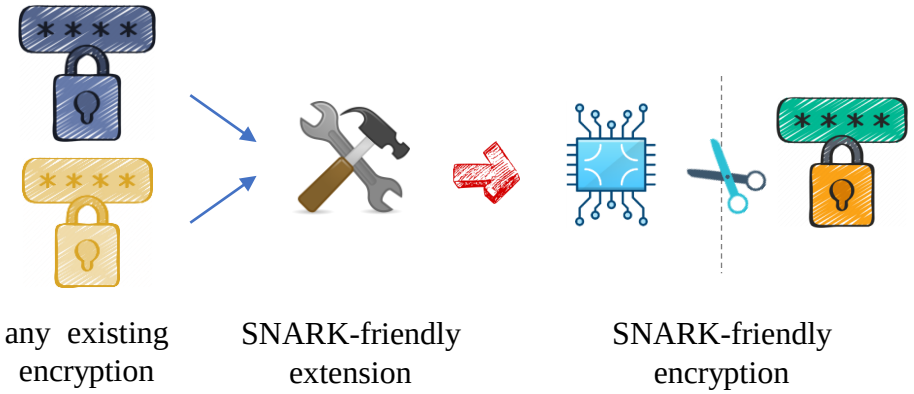
Part II: Discussions

SNARK-friendly extension (lifting transformation)?

- SAVER is a specific scheme with specific functionalities... can it be applied to other encryptions?
  ex: Identity-Based Encryption (IBE), Attribute-Based Encryption (ABE)...
- Split the message and connect them as in SAVER
- Also connect the additional features (e.g. identity, attribute)
- Possible to devise a general SNARK-friendly extension technique



SNARK-friendly IBE



SNARK-friendly extension technique

General SNARK-friendly encryption?

- Is SNARK-friendly technique compatible with general ZKPs?
  not just pre-processing SNARKs... say for example, Sonic, Plonk, etc.

- If based on the pairing group, then yes.
  i.e. Sonic, Plonk – compatible, possible to adjust the I/O and verification

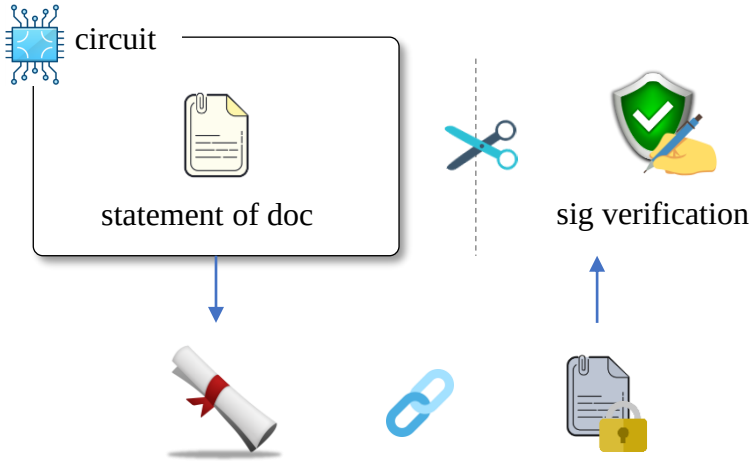
- For other groups, it's an open problem... requires a connection to the pairing group
  i.e. DARK (class group), STARK (FRI) - not compatible

SNARK-friendly System?

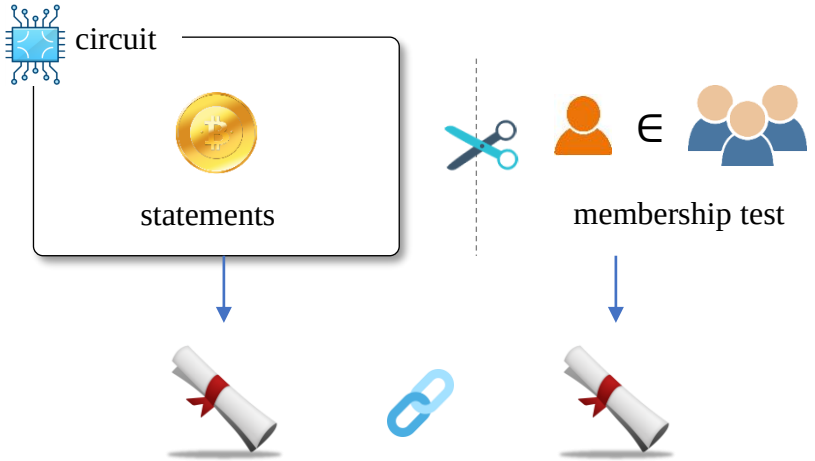
- SNARK-friendly X
  ex: encryption, verification, membership...

- Focusing on “connectivity” between SNARK + X
  ex: SNARK-friendly verification – detaches “signature verification” from the SNARK
  SNARK-friendly membership – detaches “membership proof” from the SNARK

- Good point to categorize/standardize...



SNARK-friendly verification



SNARK-friendly membership

From the viewpoint of commit-and-prove...

- The SNARK-friendly technique is an extension of cc-SNARK from LegoSNARK
- Commit-and-prove, X-and-prove, X-with-prove: from generality to specificity

	commit-and-prove (CP-SNARK) Commit ahead of time: don't know what to do or prove (no X, no CRS)	**X-and-prove** X ahead of time: don't know what to prove (no CRS)	**X-with-prove (cc-SNARK)** X + prove at the same time: only needs connectivity (aware of the X and CRS)
init:	$C_M = commit(M)$	not required	not required
X (enc):	$CT = enc(M)$	$CT = enc(M)$	$CT = enc(M), \pi = prove(M)$
prove:	$\pi = prove(M)$	$\pi = prove(M)$	
check:	$CT \leftrightarrow C_M \leftrightarrow \pi$	$CT \leftrightarrow \pi$	not required

More General **More Practical**

Summary of discussions

- SNARK-friendly encryption for other encryptions?
  - SNARK-friendly IBE, HIBE, ABE...
  - Connecting the “message” and also “additional feature (e.g. identity)” outside the circuit
  - General extension to any pairing-based encryptions

- SNARK-friendly encryption from other SNARKs?
  - If based on pairing groups, other SNARKs can also be tuned into SNARK-friendly encryption
  - If not, we need another connection between groups

- SNARK-friendly technique for other systems?
  - SNARK-friendly X: detach X from the circuit, then connect X and SNARK
  - For X: verifications, memberships, machine learnings...

- SNARK-friendly technique from the viewpoint of commit-and-prove?
  - SNARK-friendly technique is an extension from the cc-SNARK
  - Generality vs. Practicality: commit-and-prove, X-and-prove, X-with-prove