Blockchain Privacy and Homomorphic Encryption

Radu Țițiu

Bitdefender
theoretical research

Overview

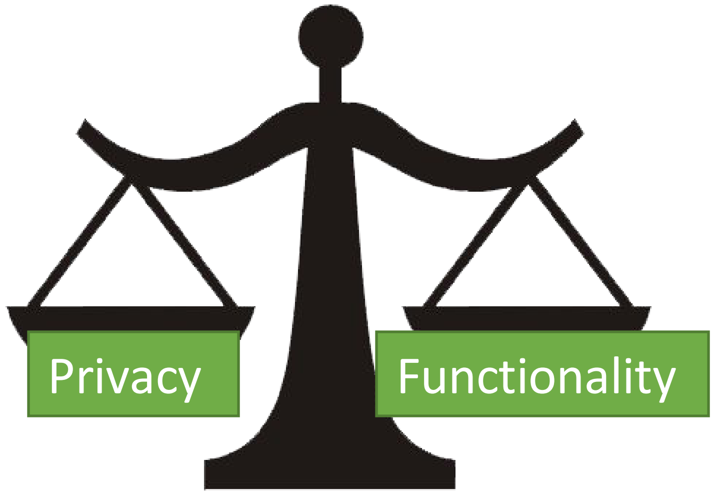
1. What is Homomorphic Encryption?

Homomorphic Encryption (HE)

- Modern encryption that enables **computing directly on encrypted data**

$$\mathrm{Enc}(\ F(x)\ ) = F(\ \mathrm{Enc}(x)\ ),\ \text{for any } F \text{ and } x$$

- Guarantees privacy but still enables functionality



Partially Homomorphic Encryption (PHE) example

RSA is **homomorphic** w.r.t to **only one operation**:

$$N = p \cdot q$$

$$\mathbf{pk} = (N, e)\,, \text{ where } e \text{ is comprime with } \phi = (p-1)\cdot(q-1)$$

$$\mathbf{Enc}(m) := m^e \bmod N$$

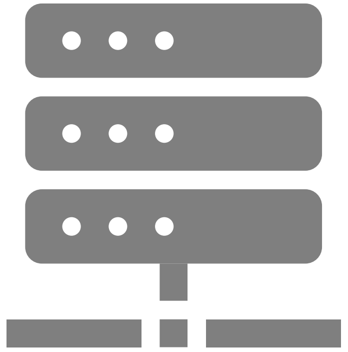
$$\mathbf{Enc}(m_1 \cdot m_2) = \mathbf{Enc}(m_1) \cdot \mathbf{Enc}(m_2)$$

- Other PHE schemes: El Gamal, Paillier.

Homomorphic Encryption (HE)

- (**KeyGen**, **Enc**, **Dec**) + **Eval** ('useful' computations on encrypted data)

private DNA



Predict_risk()

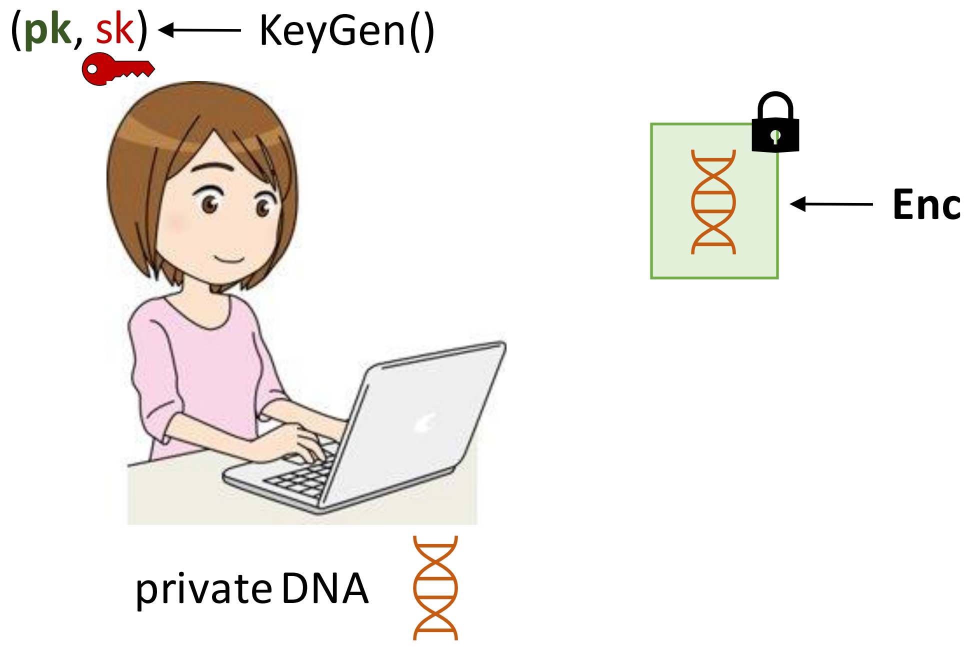
Homomorphic Encryption (HE)

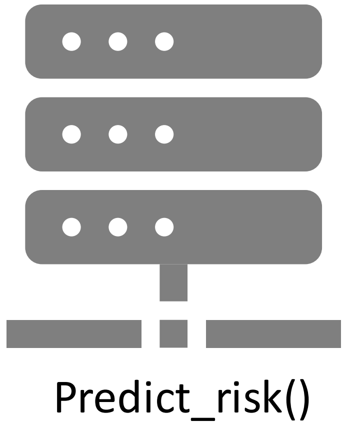
- (**KeyGen**, **Enc**, **Dec**) + **Eval** ('useful' computations on encrypted data)

(pk, sk) ← KeyGen()
private DNA
Predict_risk()

Homomorphic Encryption (HE)

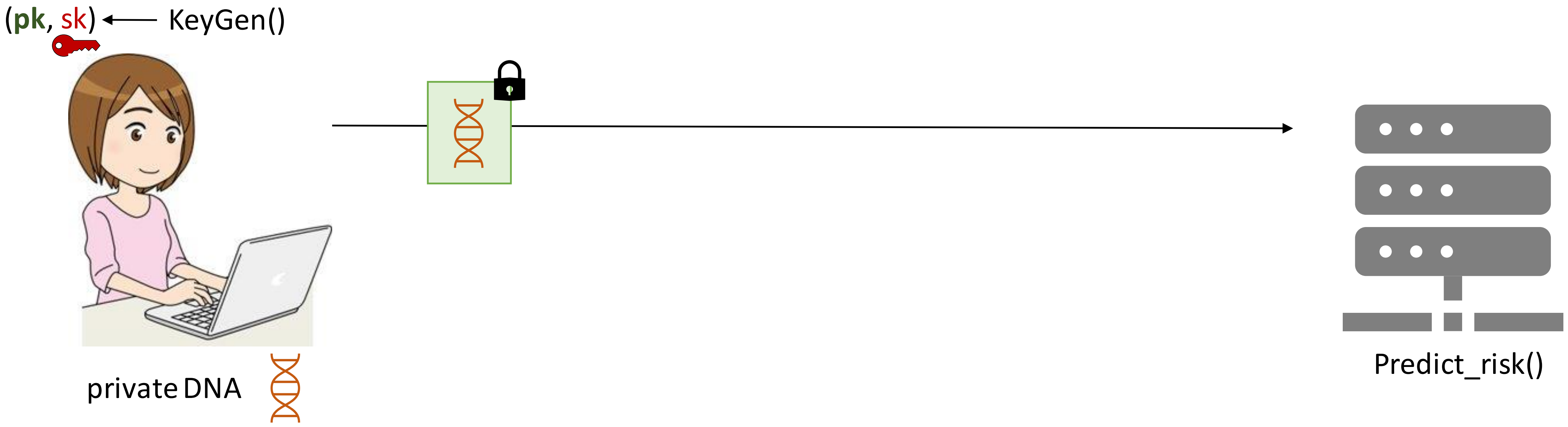
- (**KeyGen**, **Enc**, **Dec**) + **Eval** ('useful' computations on encrypted data)





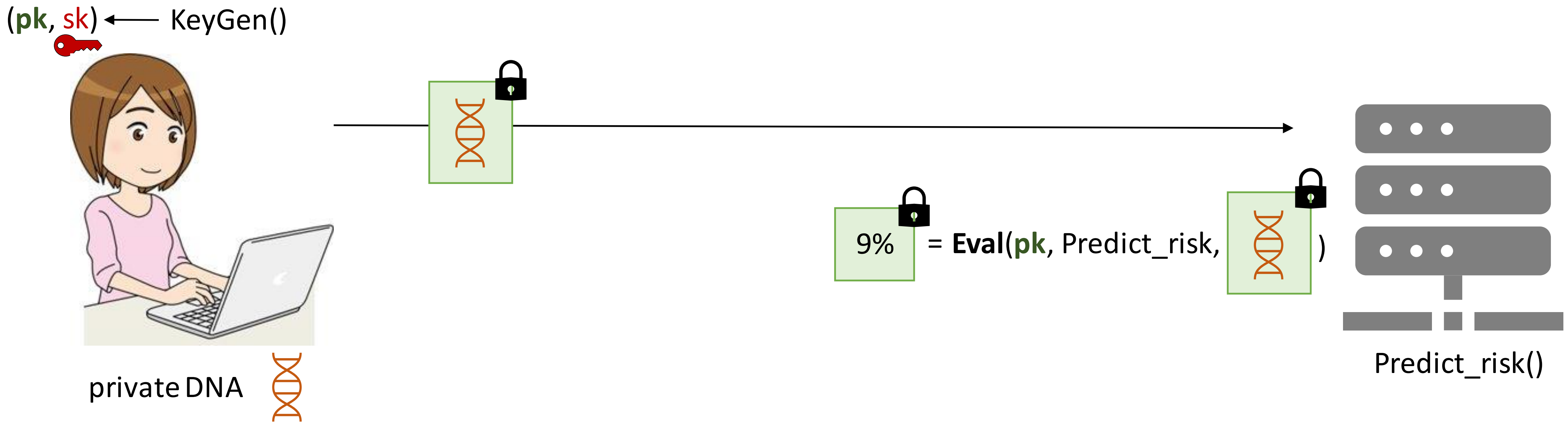
Homomorphic Encryption (HE)

- (**KeyGen**, **Enc**, **Dec**) + **Eval** ('useful' computations on encrypted data)



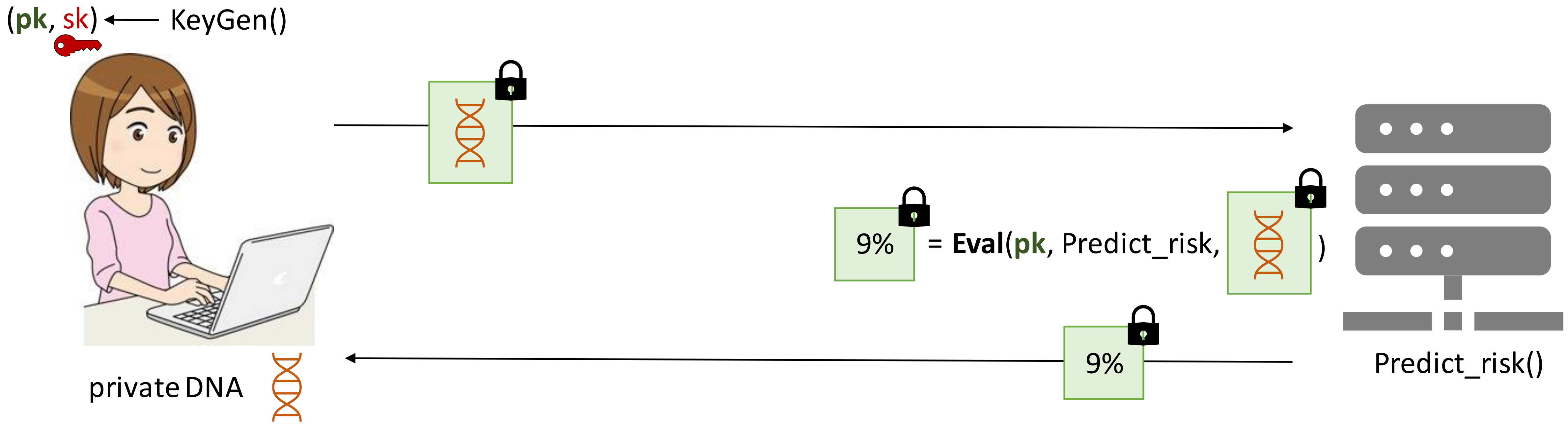
Homomorphic Encryption (HE)

- (**KeyGen**, **Enc**, **Dec**) + **Eval** ('useful' computations on encrypted data)



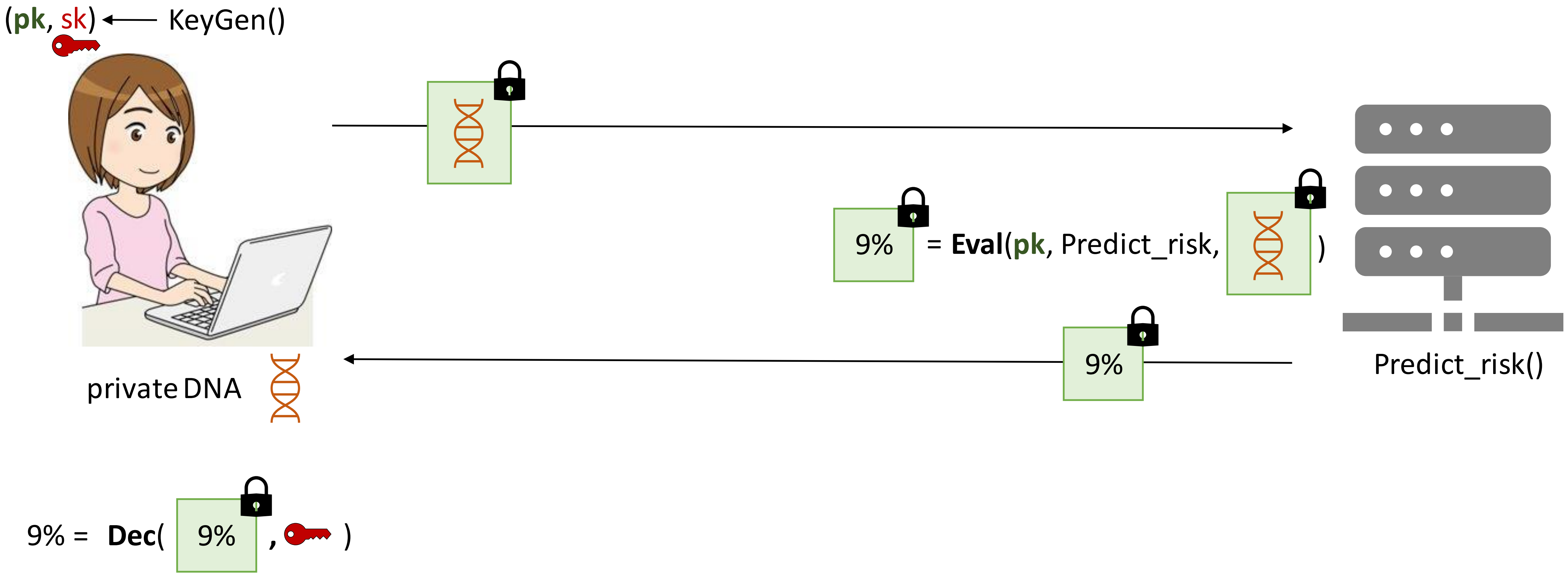
Homomorphic Encryption (HE)

- (**KeyGen**, **Enc**, **Dec**) + **Eval** ('useful' computations on encrypted data)



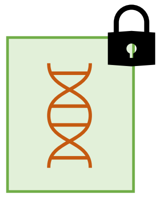
Homomorphic Encryption (HE)

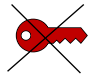
- (**KeyGen**, **Enc**, **Dec**) + **Eval** ('useful' computations on encrypted data)



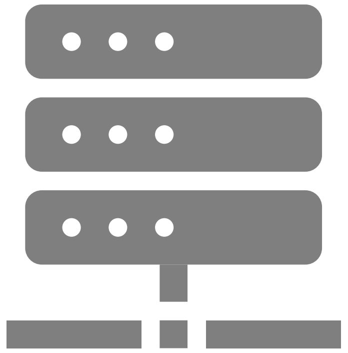
Homomorphic Encryption (HE)

private DNA

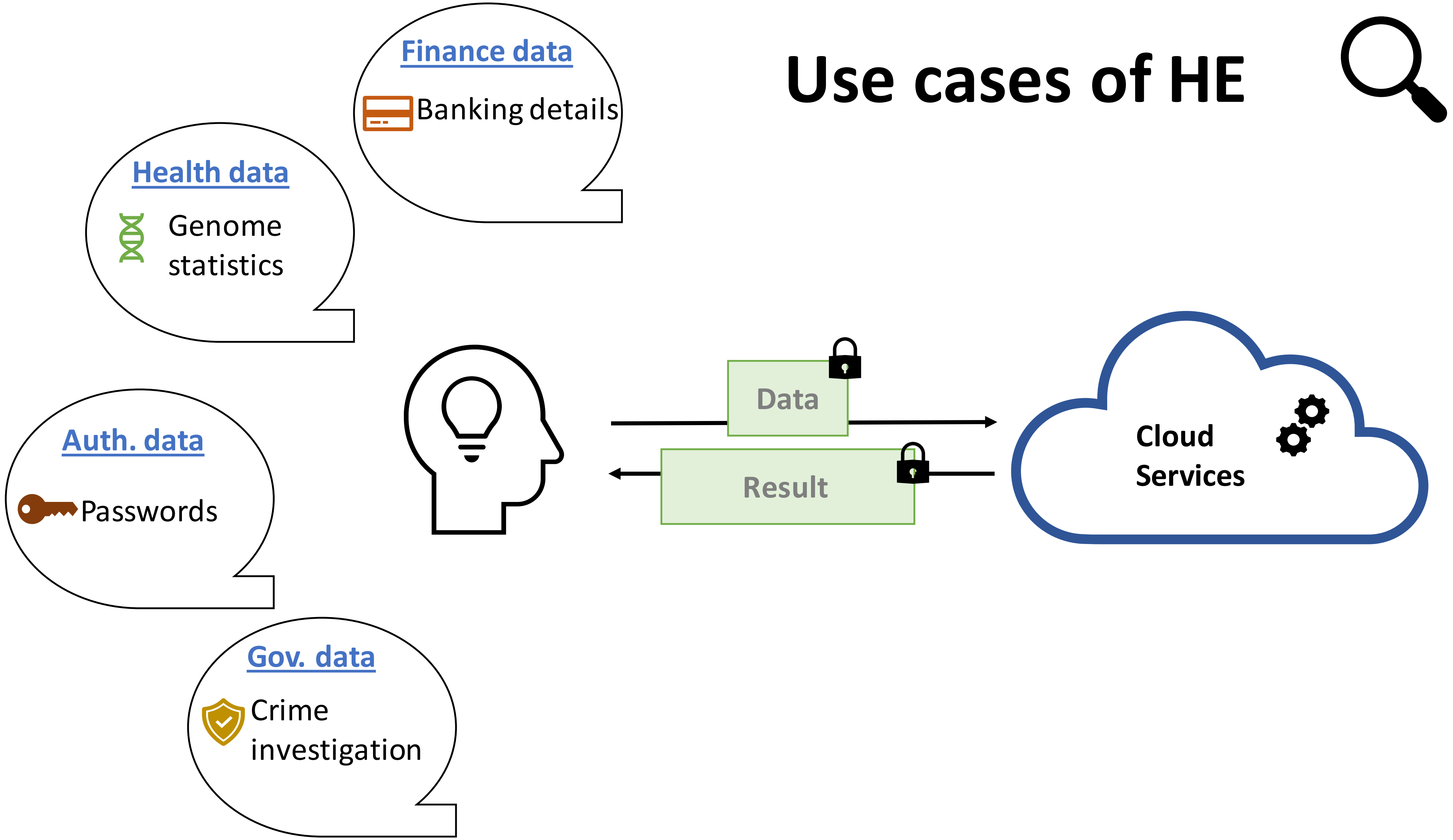




- Sensitive data is **always encrypted**
- **Computation** on encrypted data is done **without the secret key**
- Only Alice learns the result of the computation



Use cases of HE



Does HE solve all our privacy problems?

- First theoretical solution in 2009 [Gen'09]
- Much progress towards a practical scheme: [BFV13, GSW13, CKKS16, CGGI16] etc..
- HE libraries: Helib, TFHE, Microsoft SEAL, Concrete etc.

👍

➢ Many applications where privacy is important
- Medical
- Financial
- Cloud computing
- Etc.

➢ Post-quantum secure

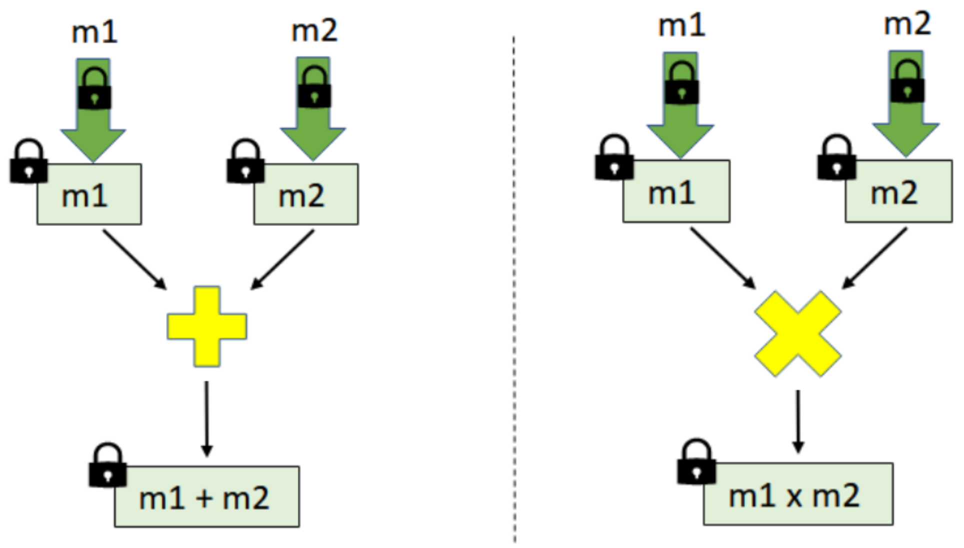
👎

➢ Computationally expensive

➢ **Limited practicality** (for now)**:** depends on the specific application (functionality)

A closer look at the Eval algorithm

- All existing HE schemes simultaneously support **two basic homomorphic evaluations**:



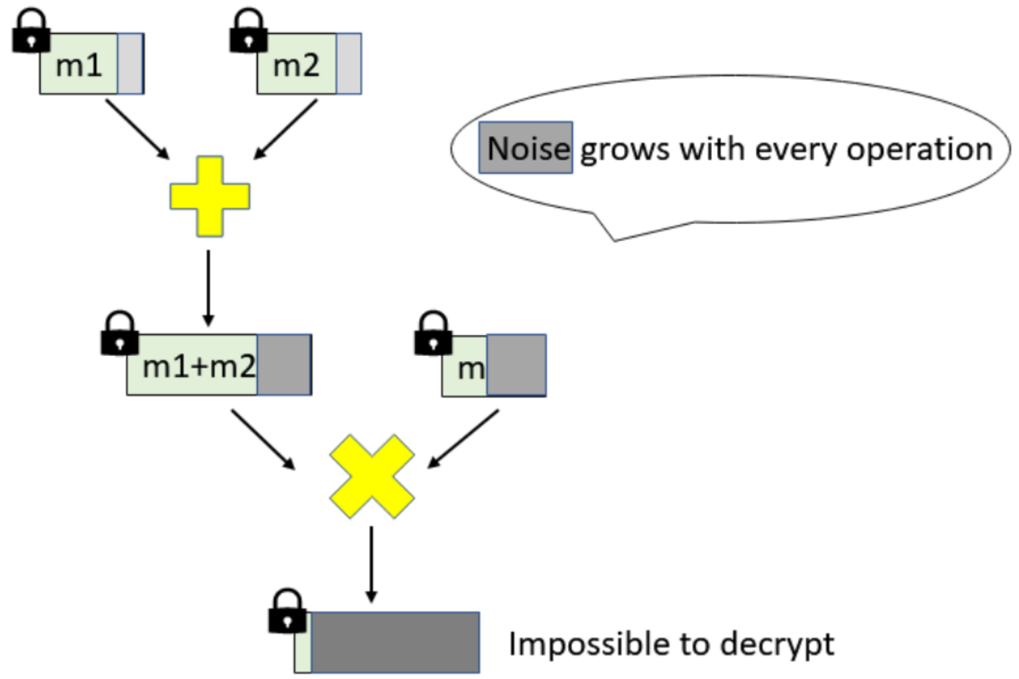
- Enough to homomorphically evaluate complex functionalities!

- For ex. any boolean circuit can be expressed using NAND gates exclusively

$\text{NAND}(a, b) = a \times b + 1 \bmod 2$, for any bits $a, b \in \{0, 1\}$.

Noisy ciphertexts in HE

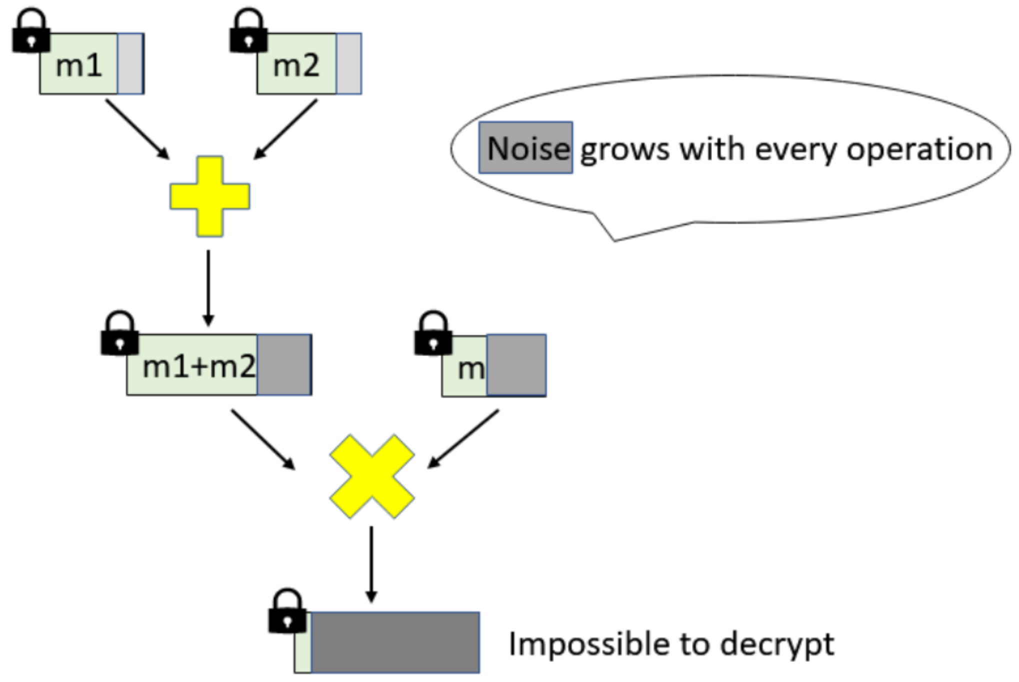
- The most efficient HE schemes inherit the **'noisy'** nature of **lattice-based** cryptography



No. of operations is **limited**!

Noisy ciphertexts in HE

- The most efficient HE schemes inherit the **'noisy'** nature of **lattice-based** cryptography

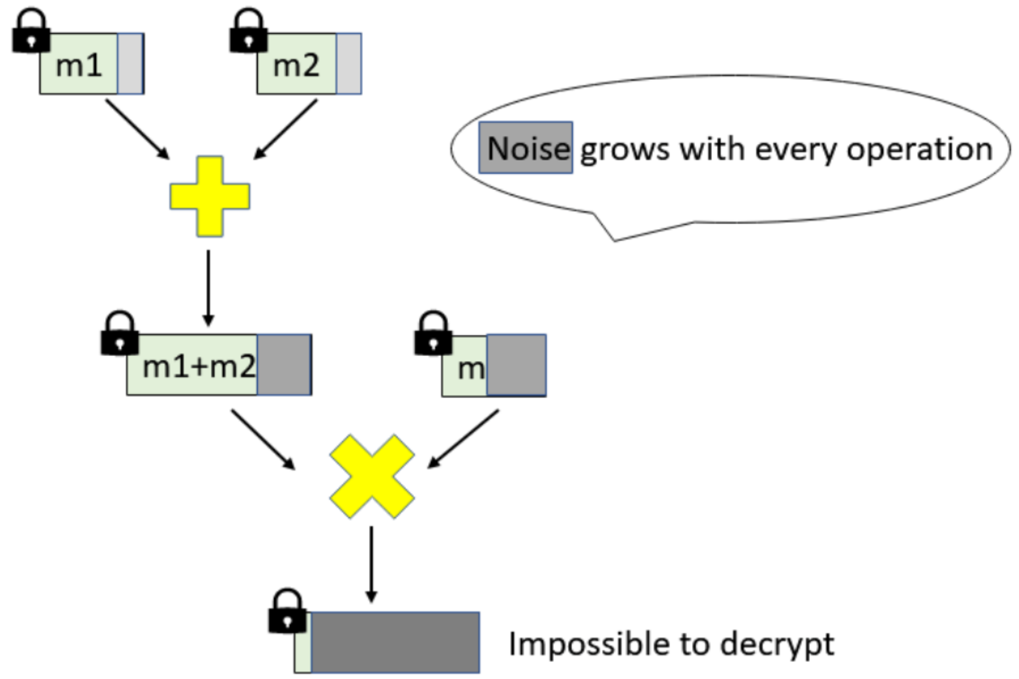


Somewhat HE (SHE):
- Supports predetermined no of operations
- Params can get huge

No. of operations is **limited**!

Noisy ciphertexts in HE

- The most efficient HE schemes inherit the **'noisy'** nature of **lattice-based** cryptography



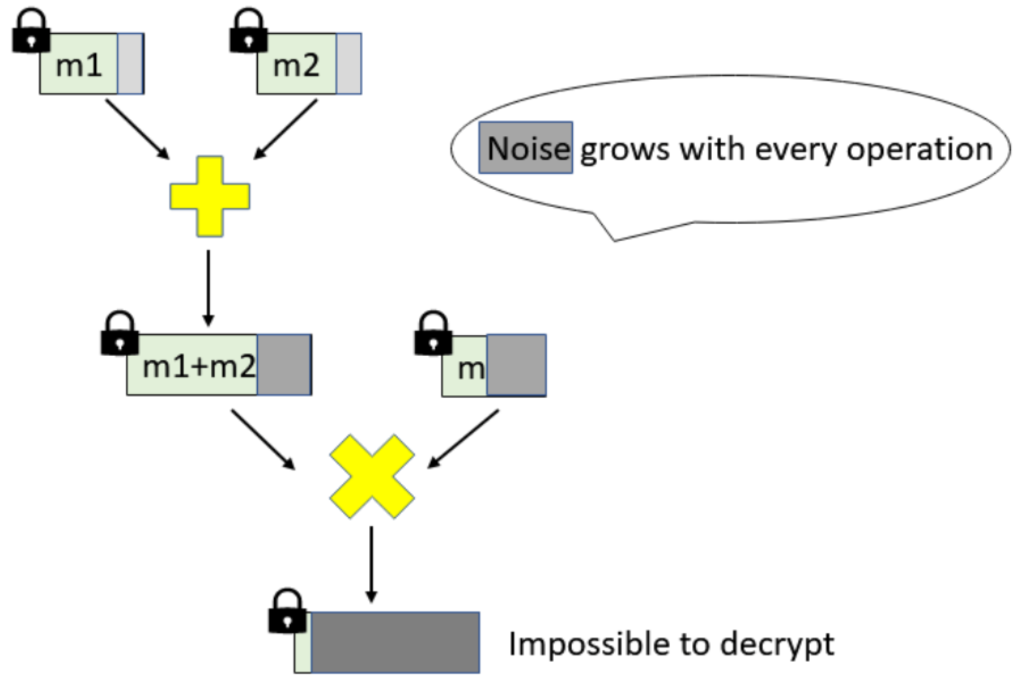
No. of operations is **limited**!

Somewhat HE (SHE):
- Supports predetermined no of operations
- Params can get huge

Fully HE (FHE):
- Supports unlimited no of operations
- Use of bootstrapping is expensive

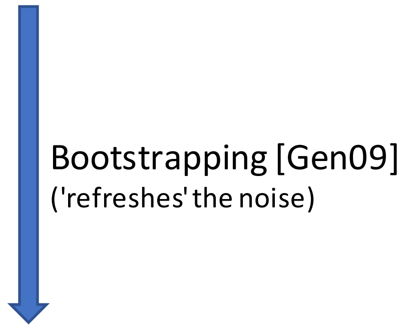
Noisy ciphertexts in HE

- The most efficient HE schemes inherit the **'noisy'** nature of **lattice-based** cryptography



No. of operations is **limited**!

Somewhat HE (SHE):
- Supports predetermined no of operations
- Params can get huge



Fully HE (FHE):
- Supports unlimited no of operations
- Use of bootstrapping is expensive

2. HE in Blockchain

Why is HE used in the context of Blockchain?

PRIVACY!

- By design, the blockchain is **public:** anyone can verify correctness, without trusting a CA
- **Privacy** (confidentiality, anonimity) is also highly desirable
- Proposed solutions in the UTXO model; (ex. Bitcoin)
  (Monero, Zerocoin, Zerocash)
- Proposed solutions in the Account-based model (Ethereum)
  (Hawk, Ekiden: not fully decentralized and too expensive simple operations)

(P)HE-based proposals:

Zether (2019): private transactions

SmartFHE (2021): private smart contracts

Zama (2022): private smart contracts

ZeeStar (2022): private smart contracts

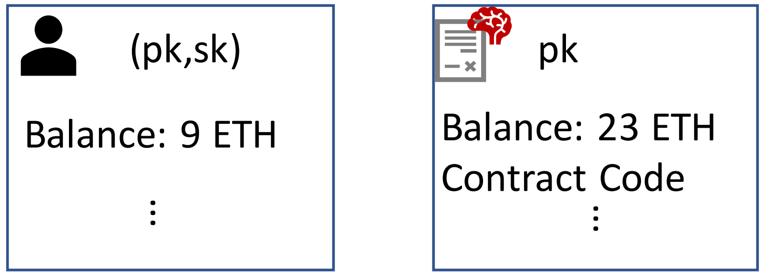
How is HE used in the context of Blockchain?

2.1 Ethereum transactions

2.2 Basic idea in Zether

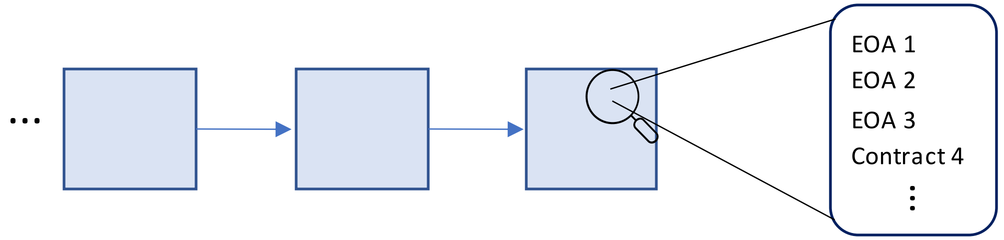
The Ethereum Network

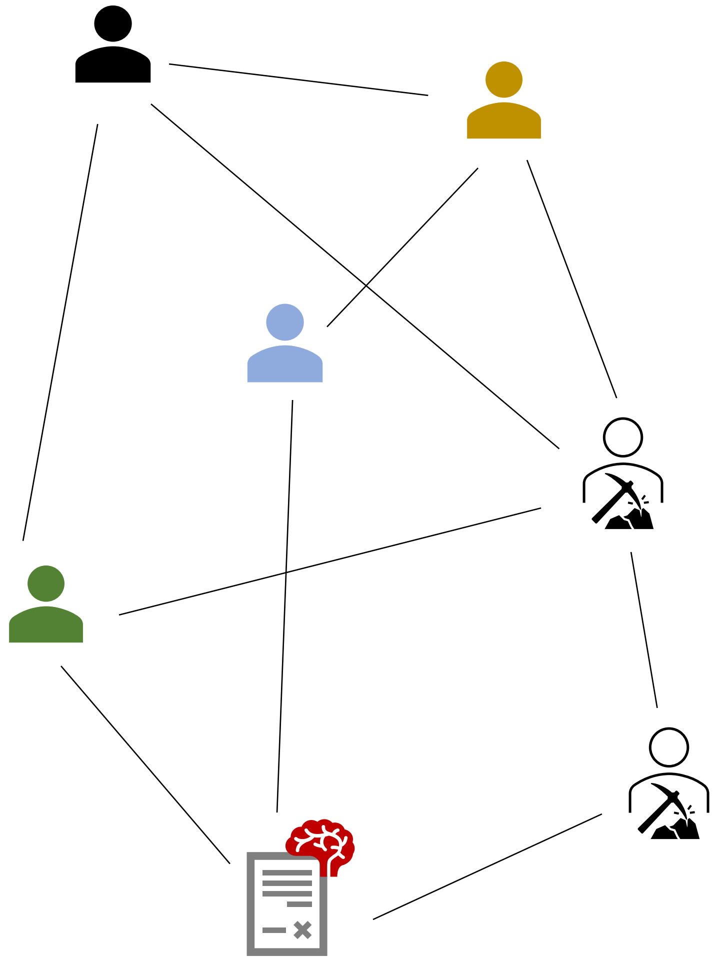
Accounts: Externally Owned Accounts (EOA) and Contracts



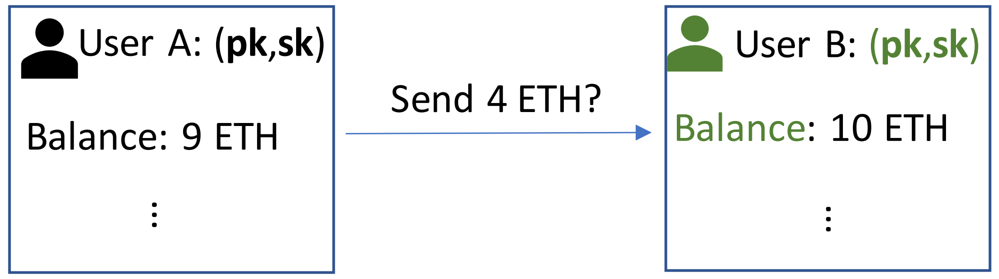
The miners:
- Process transactions and execute contract instructions
- Collect rewards / fees

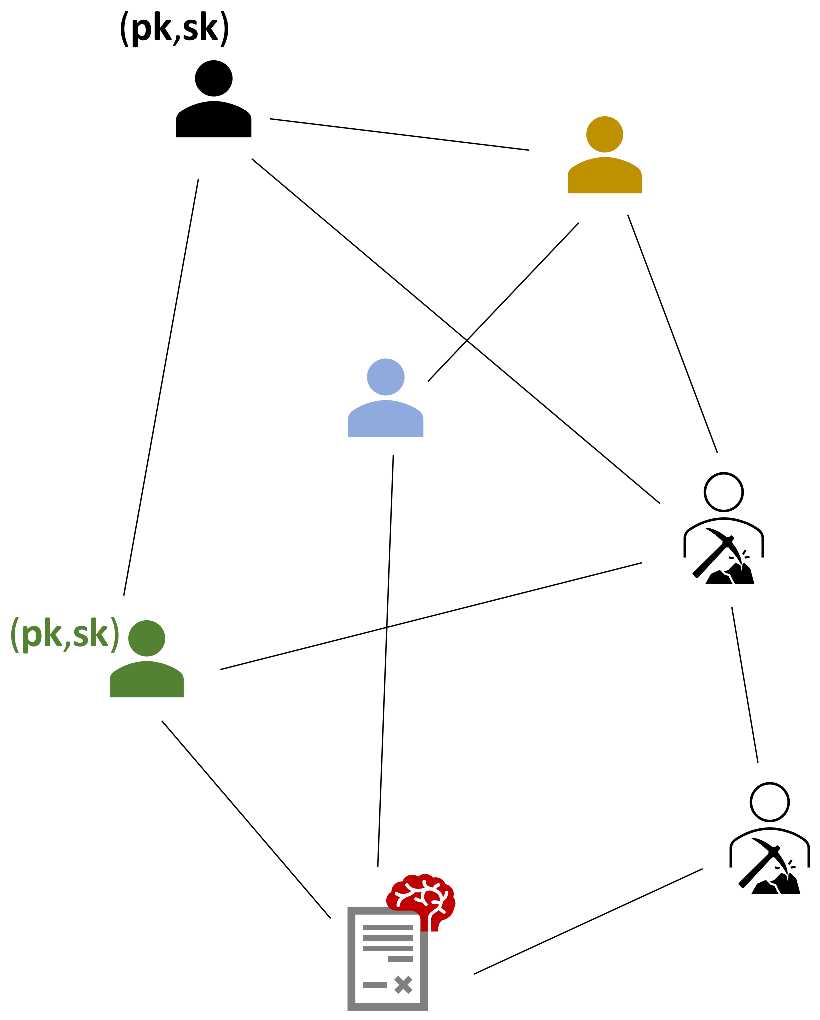
The state of all the accounts are recorded on the blockchain





Transactions on the Ethereum Network





Transactions on the Ethereum Network

User A: (**pk**,**sk**)

Balance: 9 ETH

⋮

User B: (**pk**,**sk**)

Balance: 10 ETH

⋮

Transaction A->B

From: **pk**

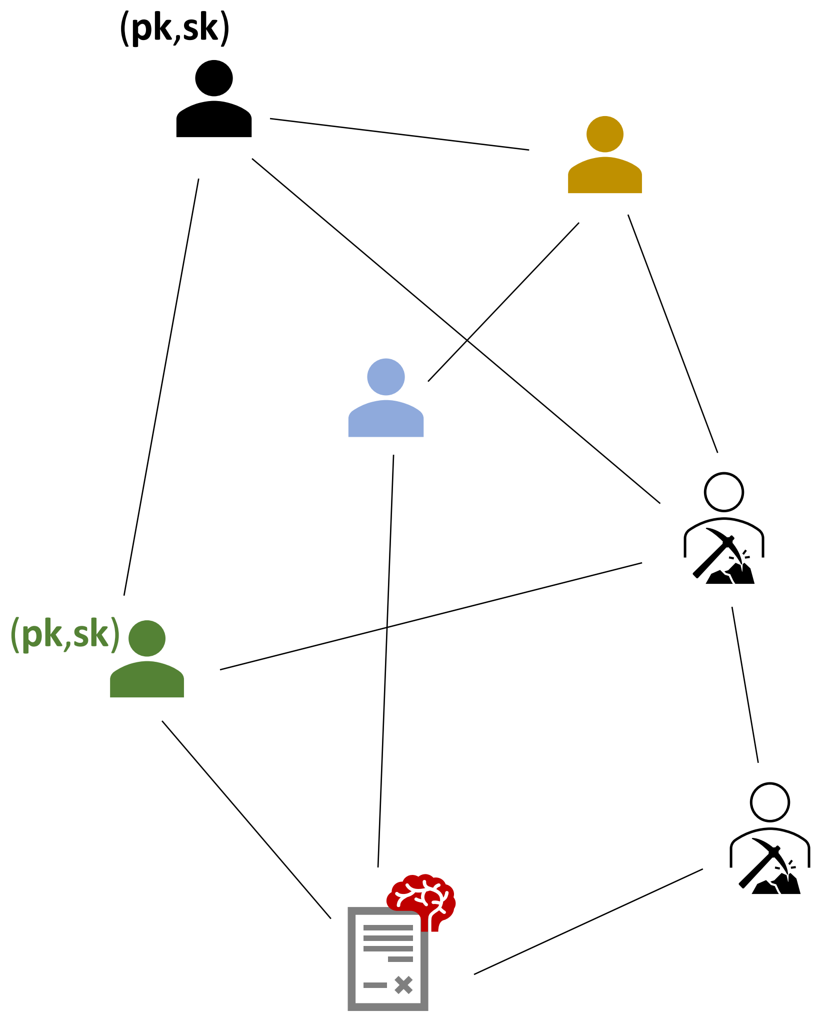
To: **pk**

Ammount: 4 ETH

Messages

Signature(**sk**)

⋮



Transactions on the Ethereum Network

User A: (**pk**,**sk**)

Balance: 9 ETH

⋮

User B: (**pk**,**sk**)

Balance: 10 ETH

⋮

Transaction A->B

From: **pk**

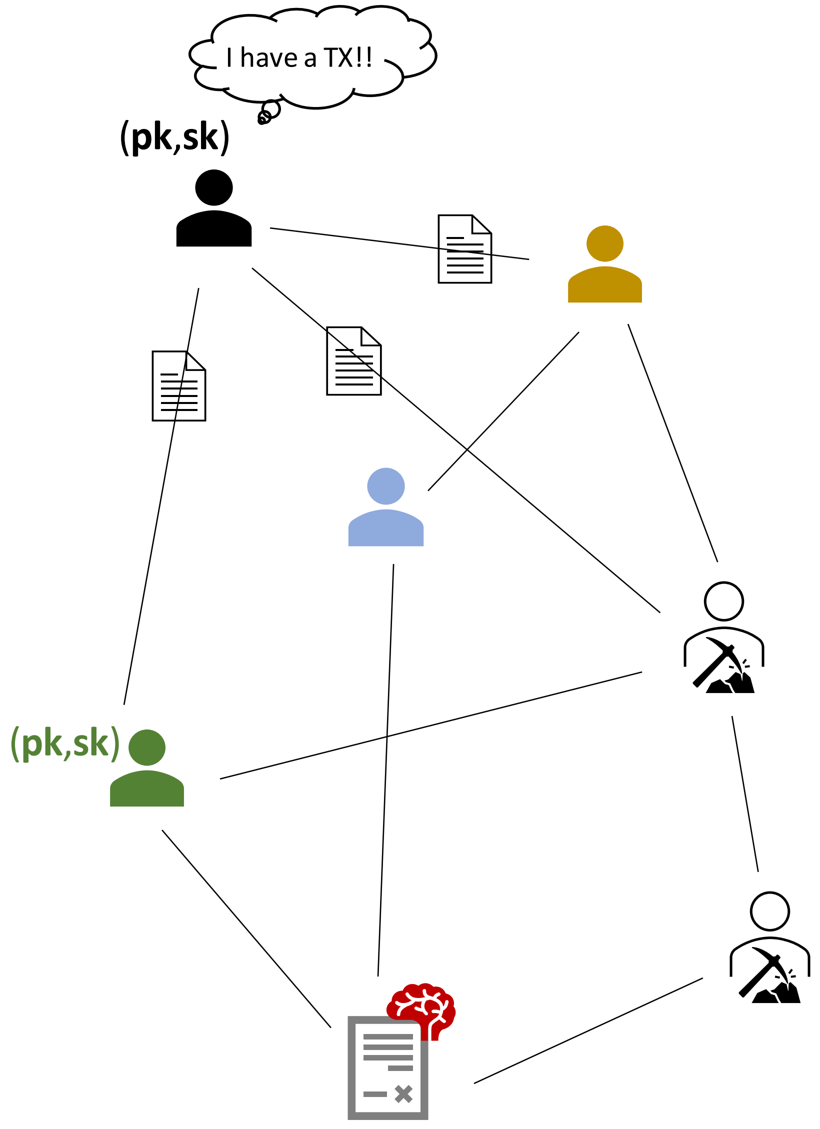
To: **pk**

Ammount: 4 ETH

Messages

Signature(**sk**)

⋮



Transactions on the Ethereum Network

User A: (**pk**,**sk**)

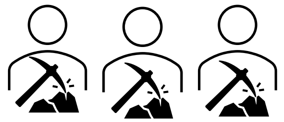
Balance: 9 ETH

⋮

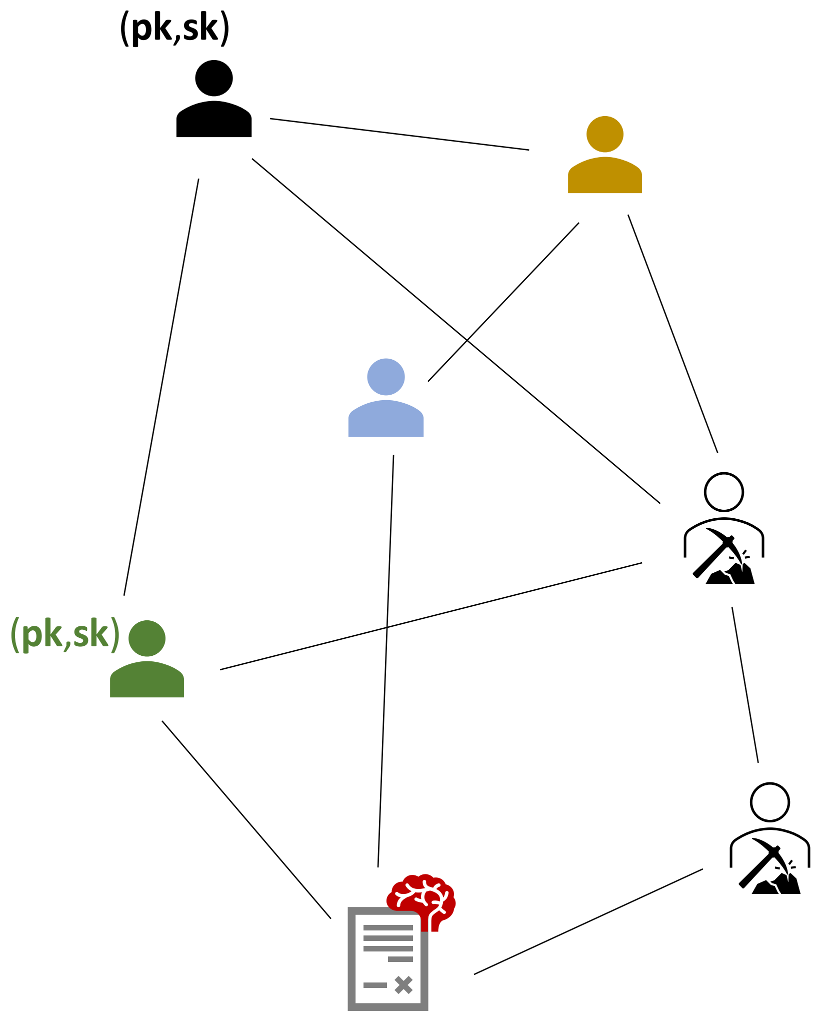
User B: (**pk**,**sk**)

Balance: 10 ETH

⋮



- Check if TX is valid



Transactions on the Ethereum Network

User A: **(pk,sk)**

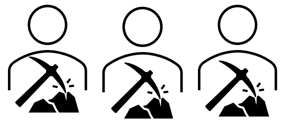
Balance: 5 ETH

⋮

User B: **(pk,sk)**

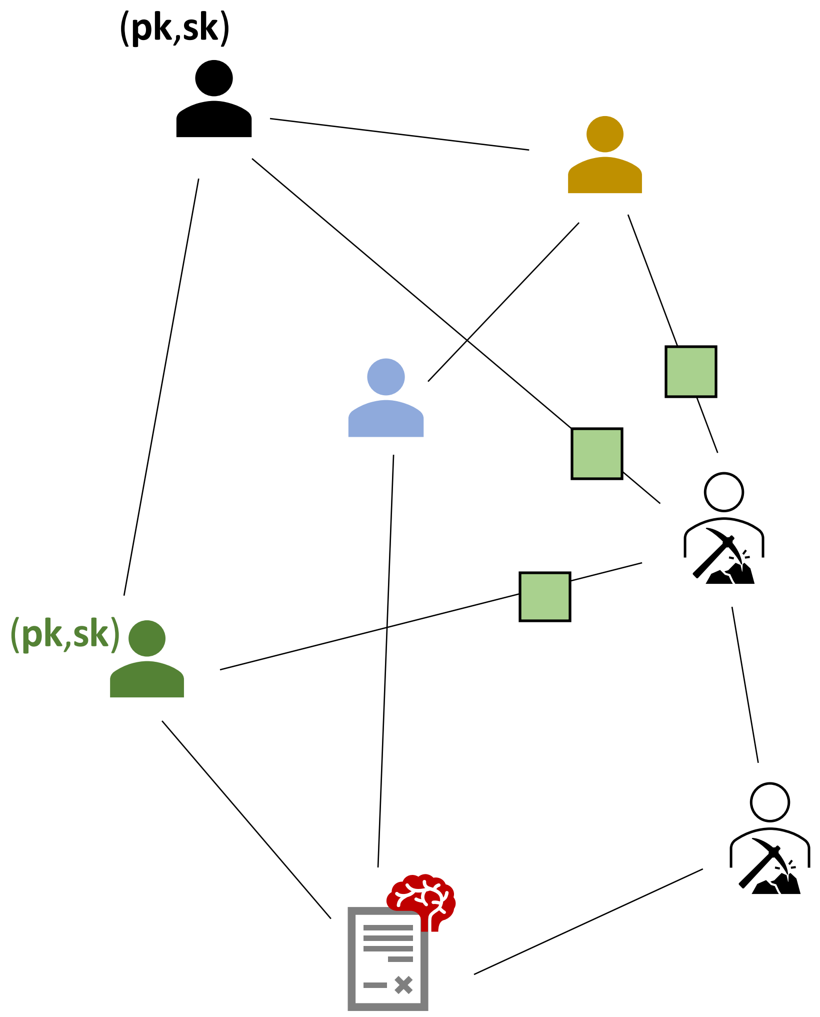
Balance: 14 ETH

⋮



- Check if TX is valid
- Update the state of the accounts:

  Balance:= Balance – 4 & Balance:= Balance + 4



Transactions on the Ethereum Network

User A: (**pk**,**sk**)

Balance: 5 ETH

⋮

User B: (**pk**,**sk**)

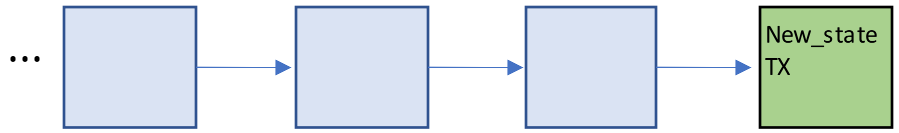
Balance: 14 ETH

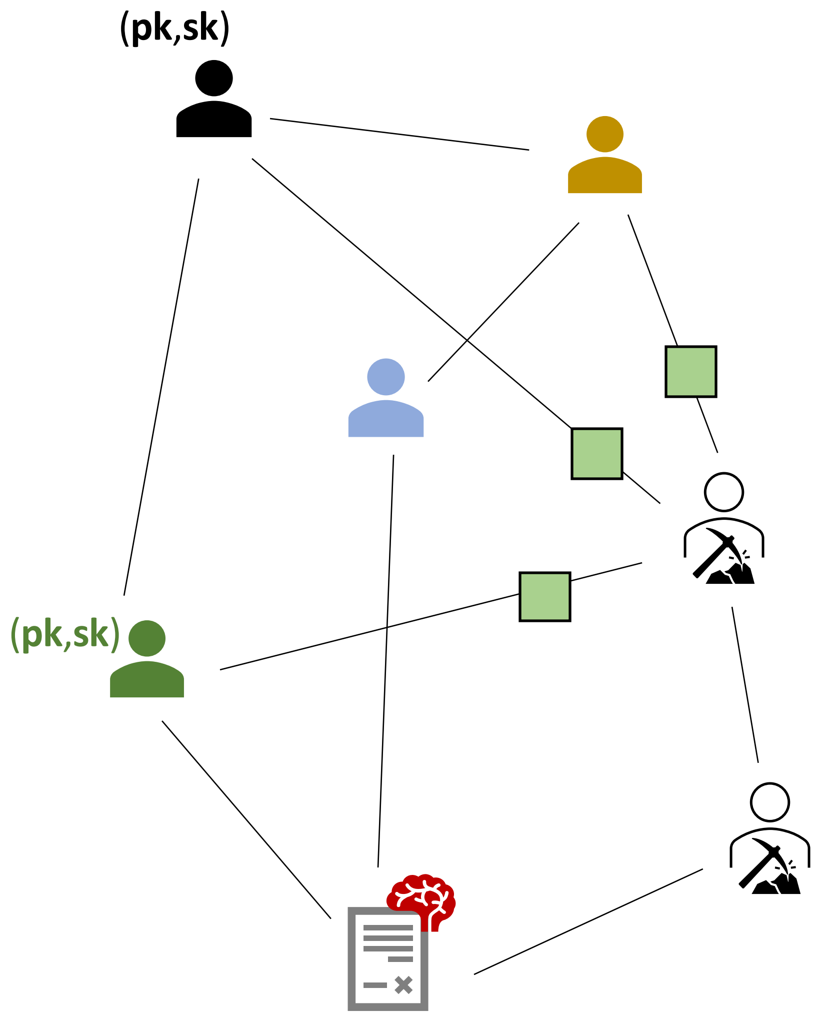
⋮

- Check if TX is valid
- Update the state of the accounts:

  Balance:= Balance – 4 & Balance:= Balance + 4

- The winner of the proof-of-work 'lottery' includes the new state and the TX into the next block and broadcasts it to the network





Transactions on the Ethereum Network

User A: (**pk**,**sk**)

Balance: 5 ETH

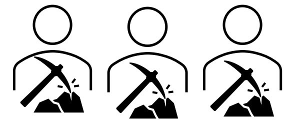
⋮

pk_C

Balance: 14 ETH

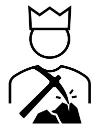
Contract code

⋮

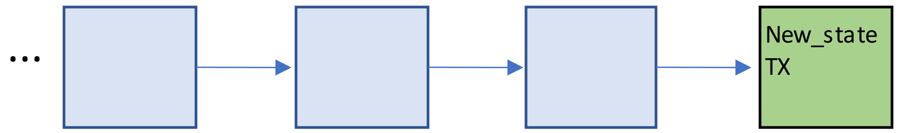


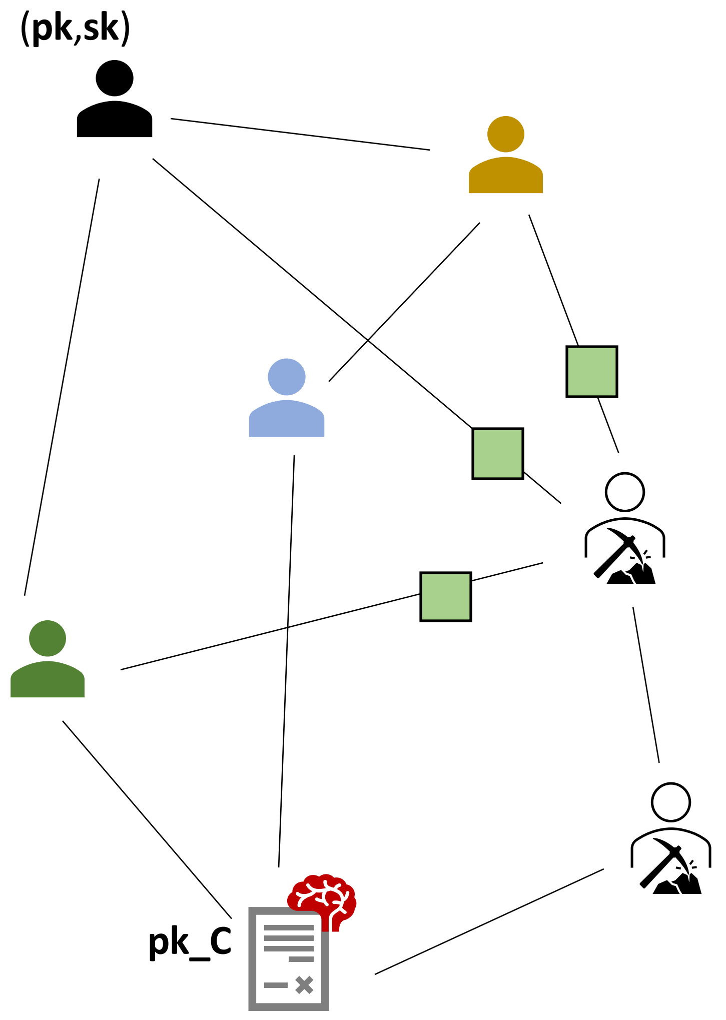
- Check if TX is valid
- Update the state of the accounts:

Balance:= Balance – 4 & Balance:= Balance + 4 & execute contract instructions



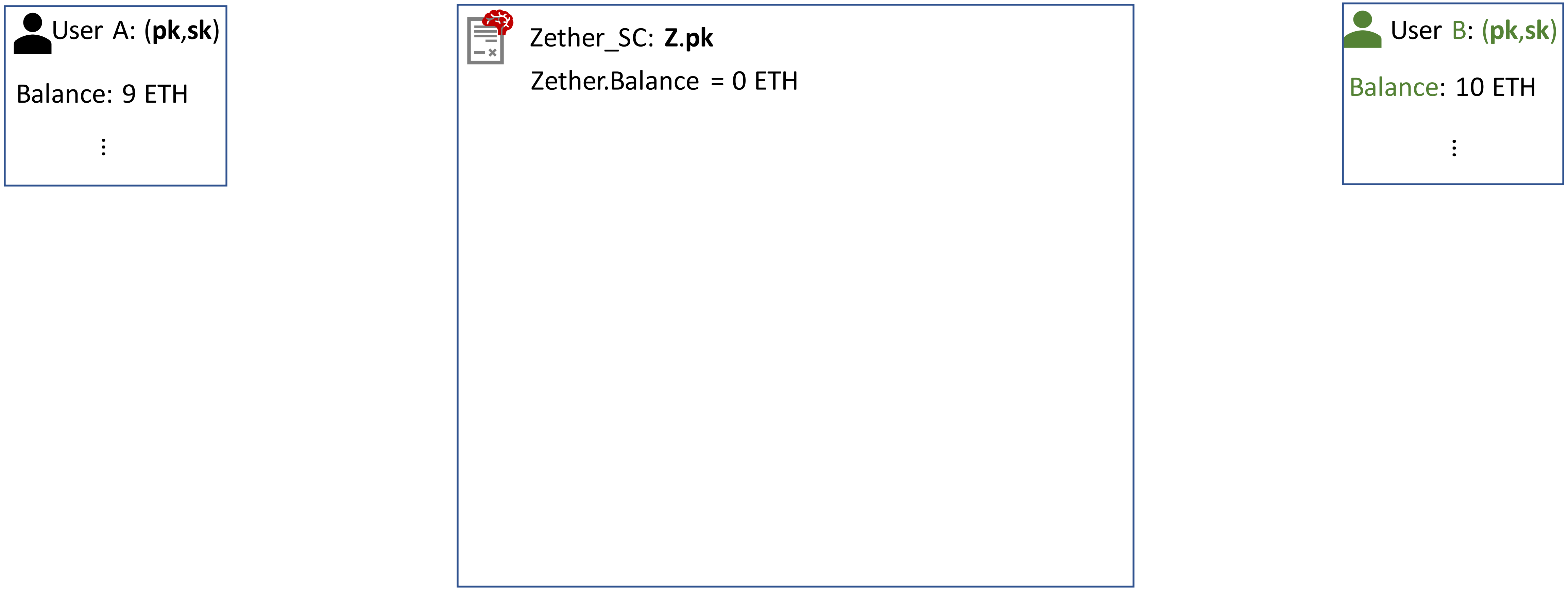
- The winner of the proof-of-work 'lottery' includes the new state and the TX into the next block and broadcasts it to the network





The basic idea in Zether

Zether Smart-Contract (ZSC)



Zether Smart-Contract (ZSC)

Zether Account Setup

User A: (**pk**,**sk**)

Balance: 9 ETH

⋮

(HE.pk, HE.sk) $\leftarrow$ HE.**KG**()

Enc_Bal $\leftarrow$ HE.**Enc**(Balance, HE.pk)

Zether_SC: **Z.pk**

Zether.Balance = 0 ETH

User B: (**pk**,**sk**)

Balance: 10 ETH

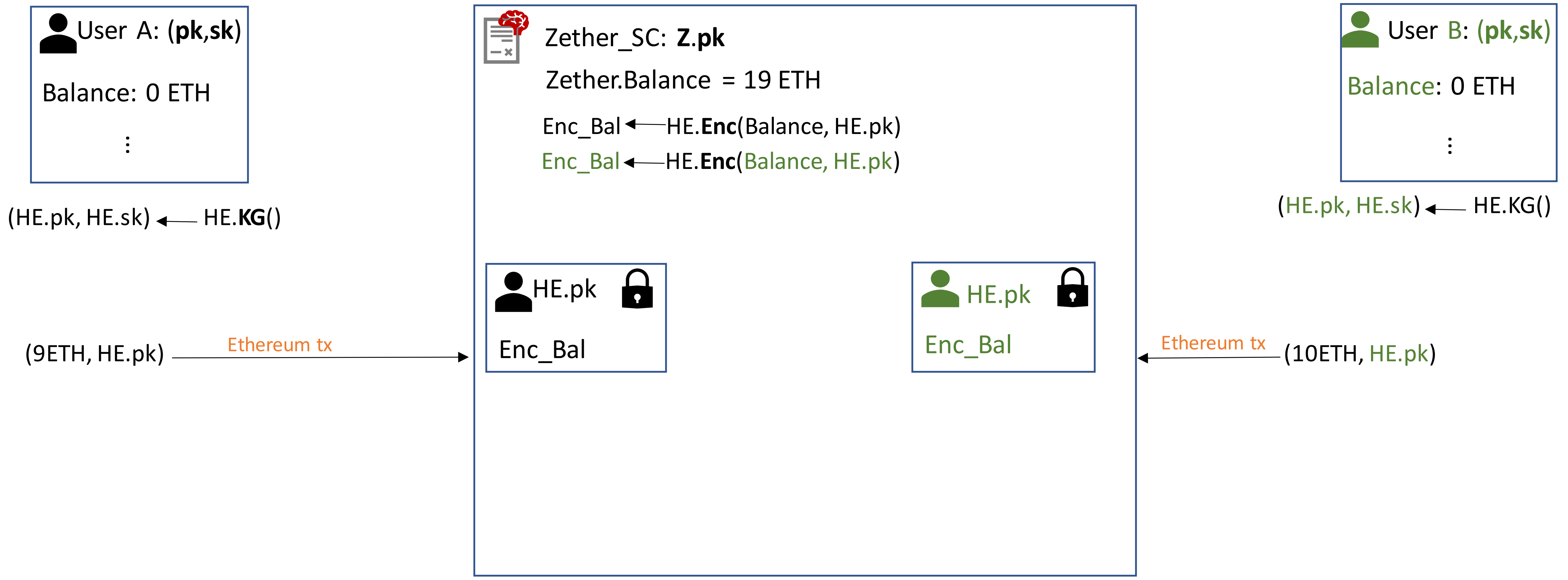
⋮

(HE.pk, HE.sk) $\leftarrow$ HE.KG()

Enc_Bal $\leftarrow$ HE.**Enc**(Balance, HE.pk)

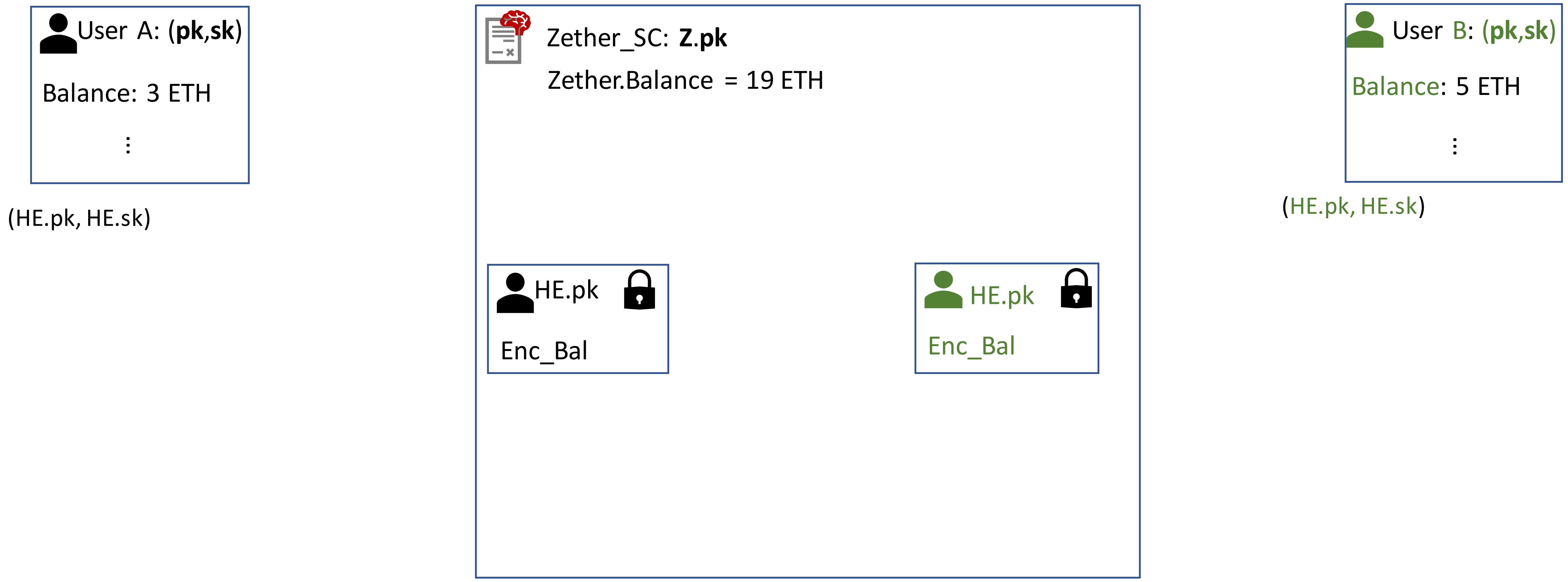
Zether Smart-Contract (ZSC)

Zether Account Setup



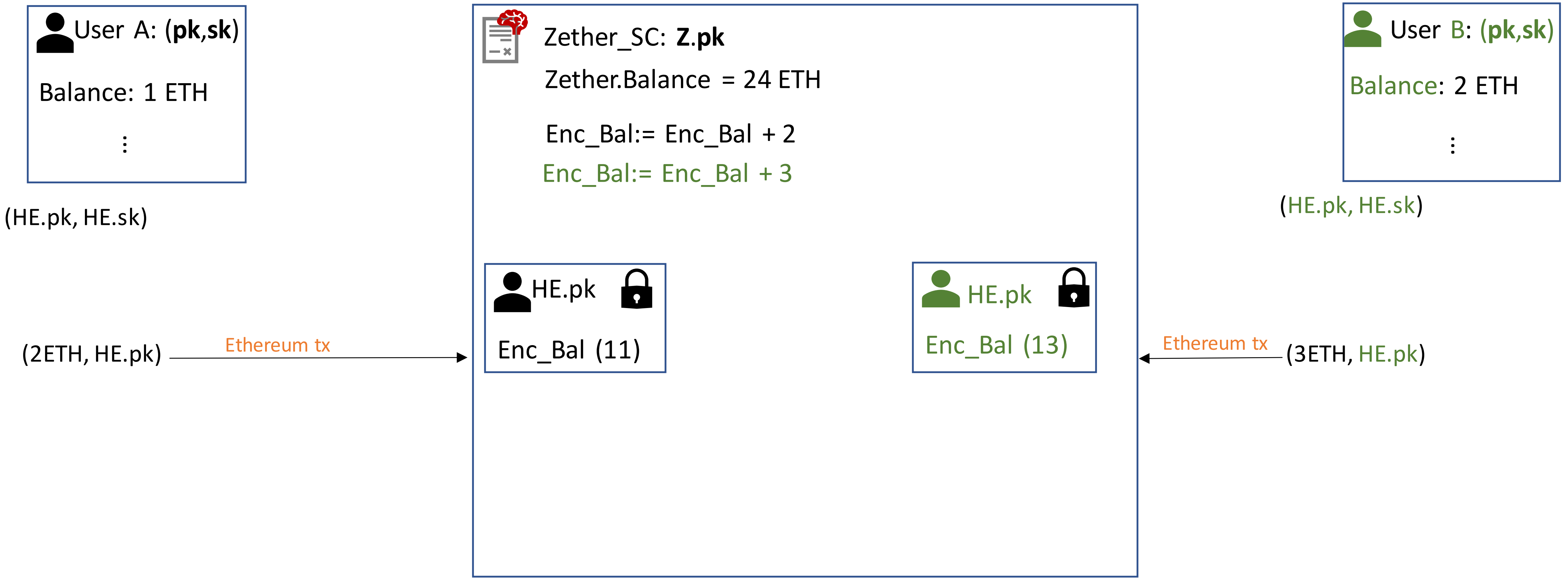
Zether Smart-Contract (ZSC)

Zether Top-Up



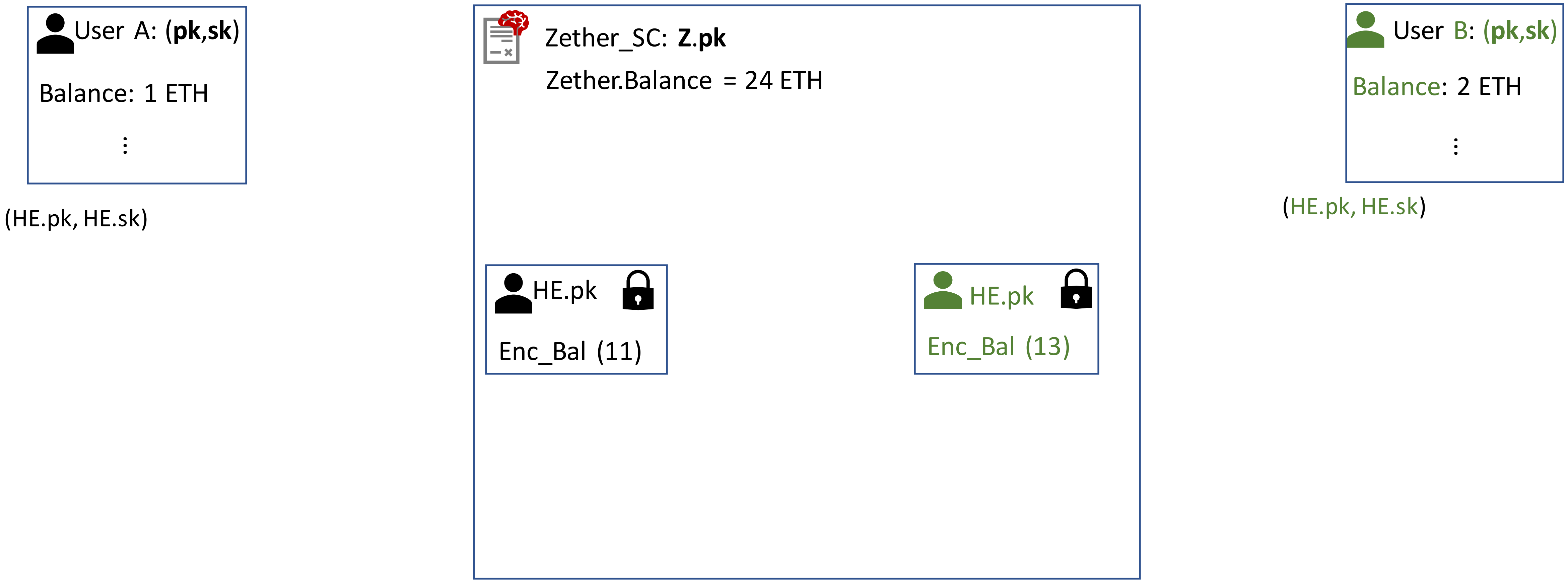
Zether Smart-Contract (ZSC)

Zether Top-Up



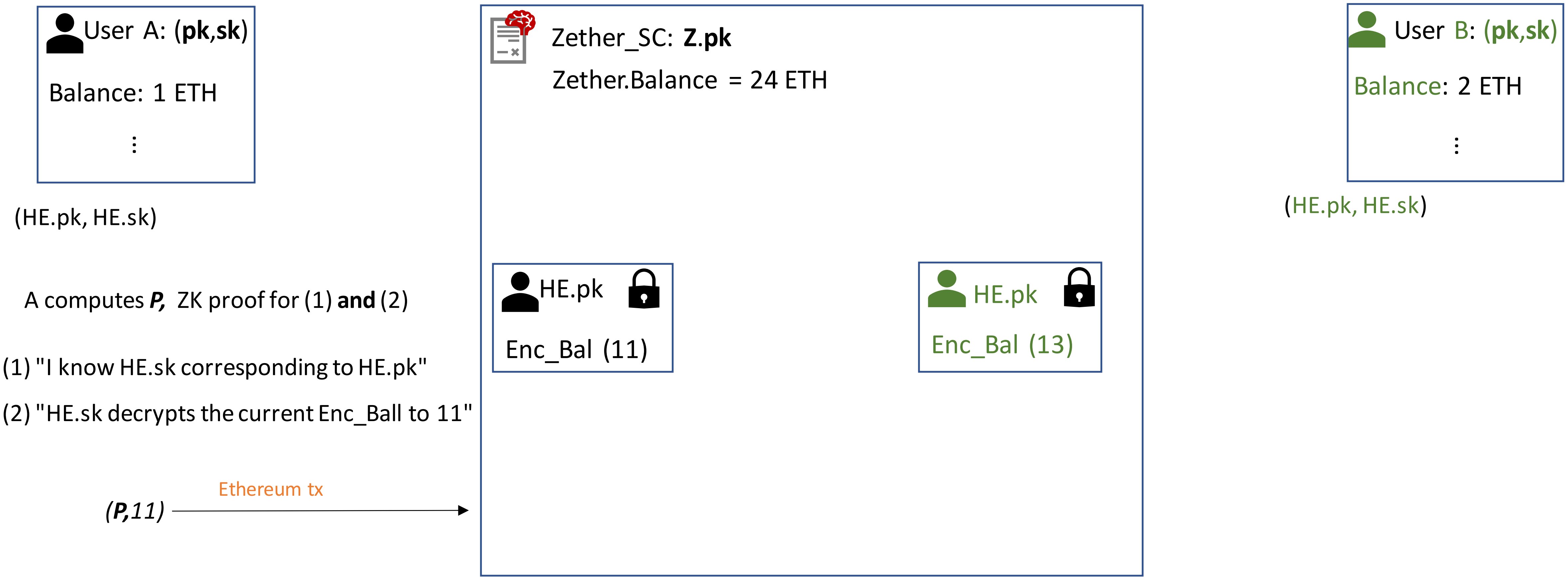
Zether Smart-Contract (ZSC)

Zether Convert back to ETH:



Zether Smart-Contract (ZSC)

Zether Convert back to ETH:



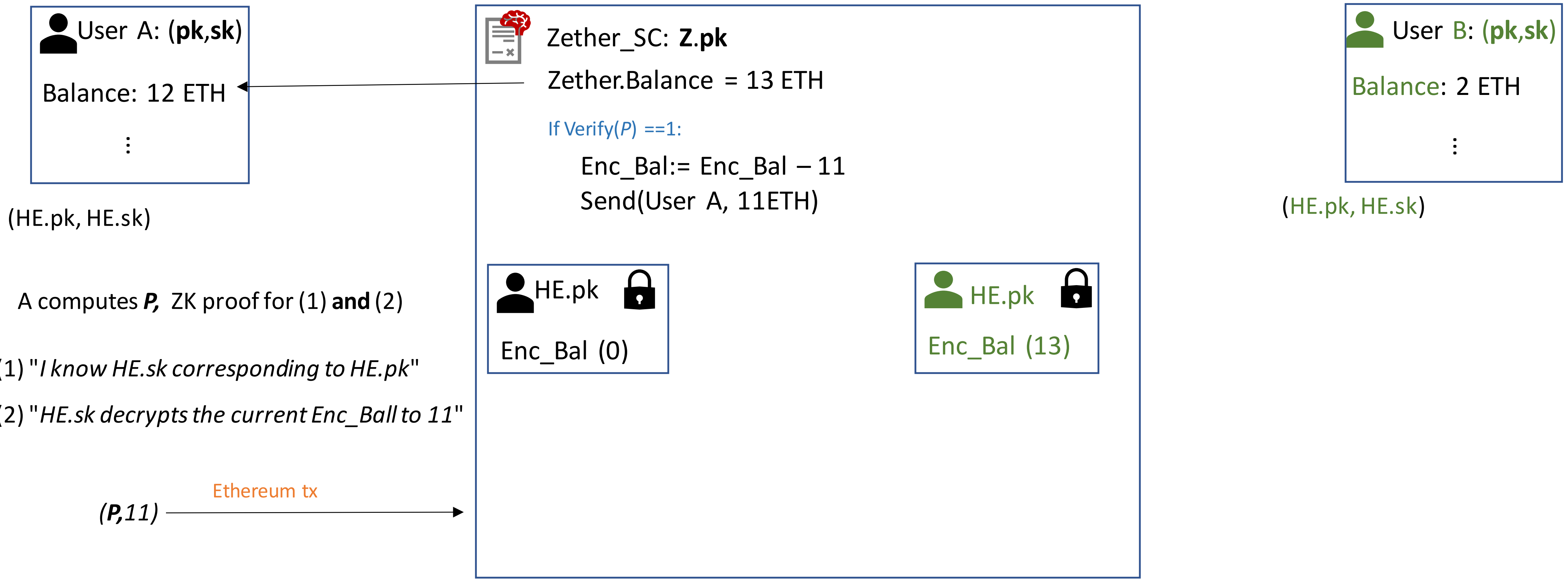
A computes ***P***, ZK proof for (1) **and** (2)

(1) "I know HE.sk corresponding to HE.pk"

(2) "HE.sk decrypts the current Enc_Ball to 11"

Zether Smart-Contract (ZSC)

Zether Convert back to ETH:



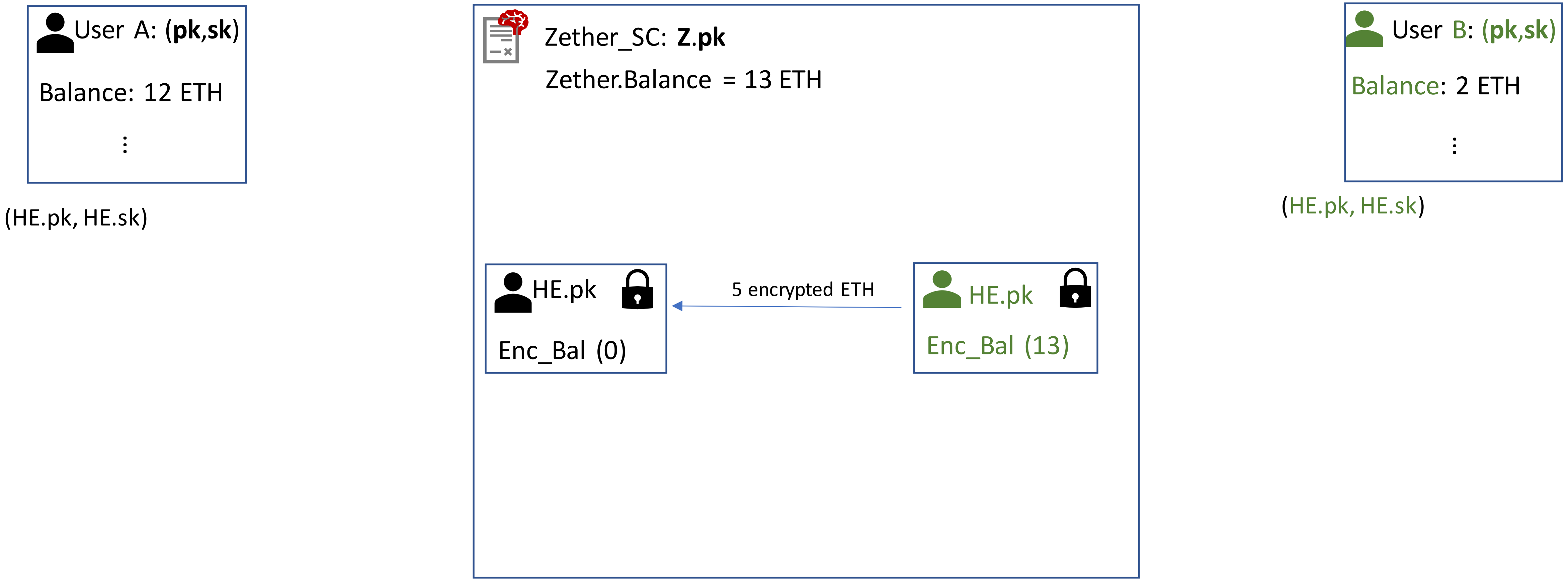
A computes ***P***, ZK proof for (1) **and** (2)

(1) "*I know HE.sk corresponding to HE.pk*"

(2) "*HE.sk decrypts the current Enc_Ball to 11*"

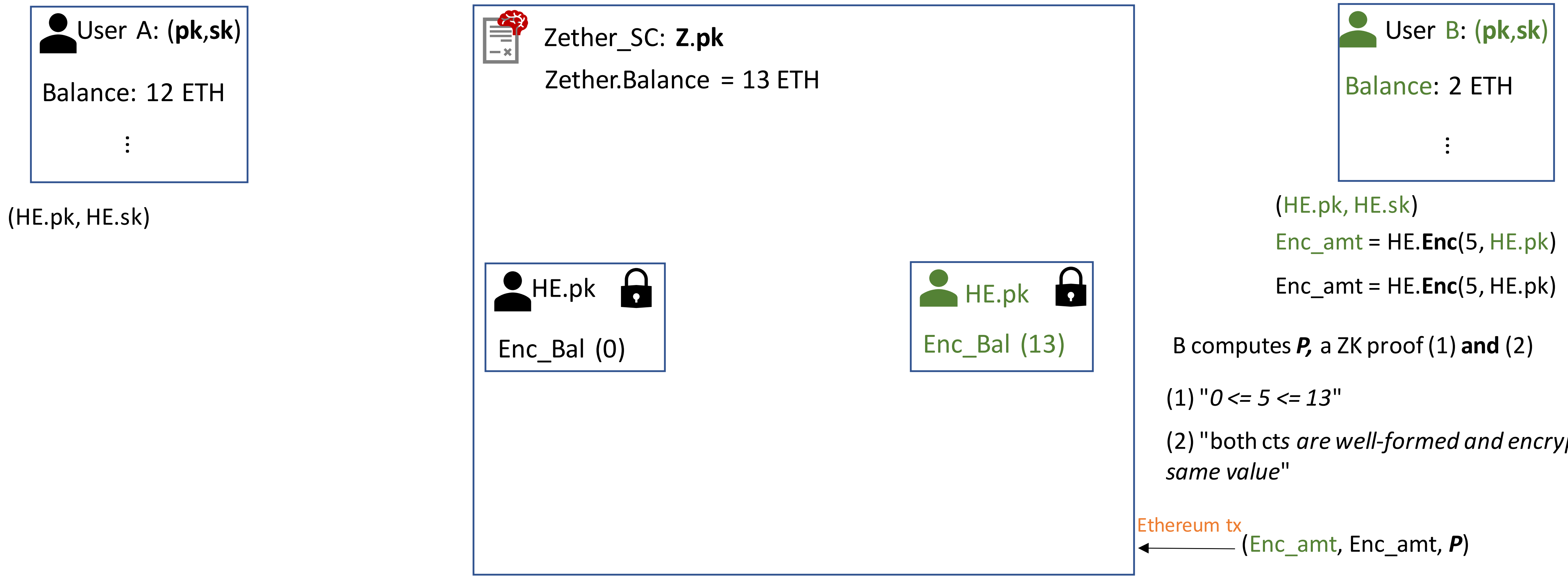
Zether Smart-Contract (ZSC)

Zether Transfer:



Zether Smart-Contract (ZSC)

Zether Transfer:



Enc_amt = HE.**Enc**(5, HE.pk)

Enc_amt = HE.**Enc**(5, HE.pk)

B computes ***P,*** a ZK proof (1) **and** (2)

(1) "*0 <= 5 <= 13*"

(2) "both cts *are well-formed and encrypt the same value*"

Zether Smart-Contract (ZSC)

Zether Transfer:

User A: (**pk**,**sk**)

Balance: 12 ETH

⋮

(HE.pk, HE.sk)

Zether_SC: **Z.pk**

Zether.Balance = 13 ETH

```
If Verify(P) == 1:
    Enc_Bal := Enc_Bal – Enc_amt
    Enc_Bal := Enc_Bal + Enc_amt
```

HE.pk

Enc_Bal (5)

HE.pk

Enc_Bal (8)

User B: (**pk**,**sk**)

Balance: 2 ETH

⋮

(HE.pk, HE.sk)

Enc_amt = HE.**Enc**(5, HE.pk)

Enc_amt = HE.**Enc**(5, HE.pk)

B computes ***P***, a ZK proof (1) **and** (2)

(1) "*0 <= 5 <= 13*"

(2) "both cts *are well-formed and encrypt the same value*"

Ethereum tx

(Enc_amt, Enc_amt, ***P***)

Conclusions

- Zether uses only Partial Homomorphism (additive El Gamal)
- The superior expressiveness of HE (add + mult) may give privacy for more complex private SC
  (Zama, smartFHE)
- Privacy for any SC is an active research topic

Thanks!

- [Gen09]: https://crypto.stanford.edu/craig/craig-thesis.pdf
- [BFV13]: https://eprint.iacr.org/2012/144.pdf
- [GSW13]: https://eprint.iacr.org/2013/340.pdf
- [CKKS16]: https://eprint.iacr.org/2016/421.pdf
- [CGGI16]: https://eprint.iacr.org/2016/870.pdf
- Helib: https://github.com/homenc/HElib
- TFHE: https://tfhe.github.io/tfhe/
- Microsoft SEAL: https://github.com/microsoft/SEAL
- Concrete: https://github.com/zama-ai/concrete
- Hawk: https://eprint.iacr.org/2015/675.pdf
- Ekiden: https://arxiv.org/abs/1804.05141
- Zether: https://crypto.stanford.edu/~buenz/papers/zether.pdf
- SmartFHE: https://eprint.iacr.org/2021/133.pdf
- Zama on SC: https://www.zama.ai/post/private-smart-contract-using-homomorphic-encryption-ethcc-2022
- ZeeStar: https://files.sri.inf.ethz.ch/website/papers/sp22-zeestar.pdf