A²L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs

Erkan Tairi[1], Pedro Moreno-Sanchez[2], Matteo Maffei[1]

[1]TU Wien
[2]IMDEA Software Institute

IEEE Symposium on Security and Privacy 2021

Contributions

- Construction for scalable, secure, interoperable and privacy-preserving payments based on **trustless gateways** (hubs):

Contributions

- Construction for scalable, secure, interoperable and privacy-preserving payments based on **trustless gateways** (hubs):
  - Scalability:
    - Does not require on-chain transactions, works over payment channels

Contributions

- Construction for scalable, secure, interoperable and privacy-preserving payments based on **trustless gateways** (hubs):
  - Scalability:
    - Does not require on-chain transactions, works over payment channels
  - Atomicity:
    - No honest party loses coins

Contributions

- Construction for scalable, secure, interoperable and privacy-preserving payments based on **trustless gateways** (hubs):
  - Scalability:
    - Does not require on-chain transactions, works over payment channels
  - Atomicity:
    - No honest party loses coins
  - Unlinkability:
    - Gateway does not learn who is paying to whom

Contributions

- Construction for scalable, secure, interoperable and privacy-preserving payments based on **trustless gateways** (hubs):
  - Scalability:
    - Does not require on-chain transactions, works over payment channels
  - Atomicity:
    - No honest party loses coins
  - Unlinkability:
    - Gateway does not learn who is paying to whom
  - Interoperability:
    - Exchange coins between different currencies (e.g., ethers for bitcoin)

Scalability

- Payment channels allow parties to perform arbitrarily many payments off-chain, with only two transactions (i.e., open and close channel) going on-chain

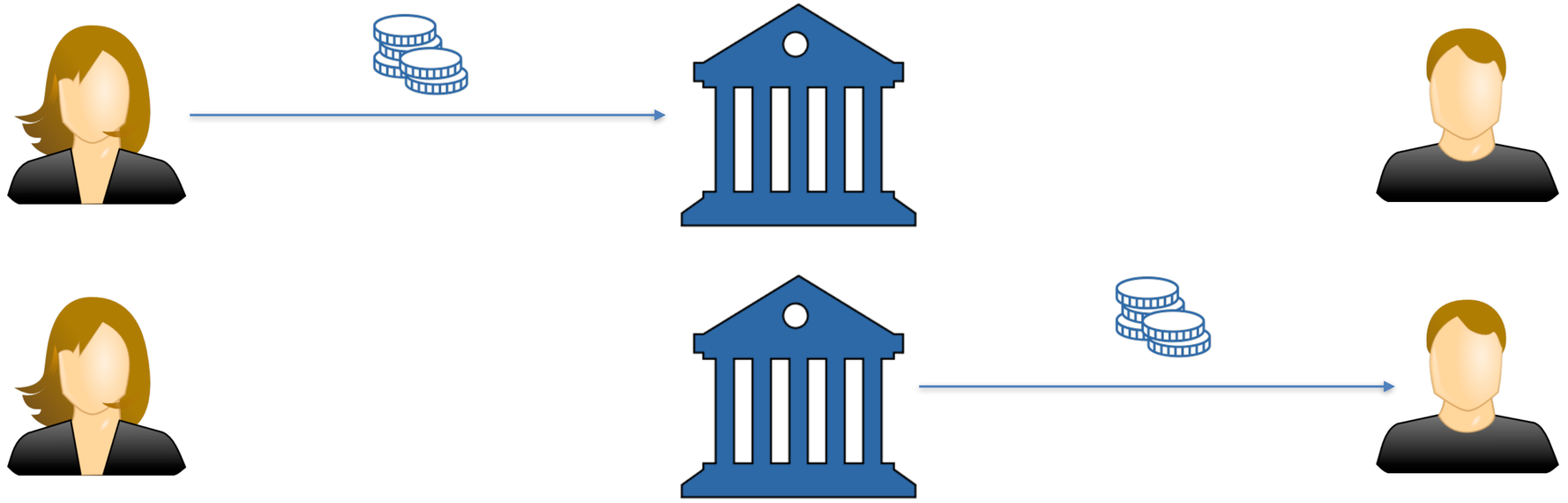
Scalability

- Payment channels allow parties to perform arbitrarily many payments off-chain, with only two transactions (i.e., open and close channel) going on-chain

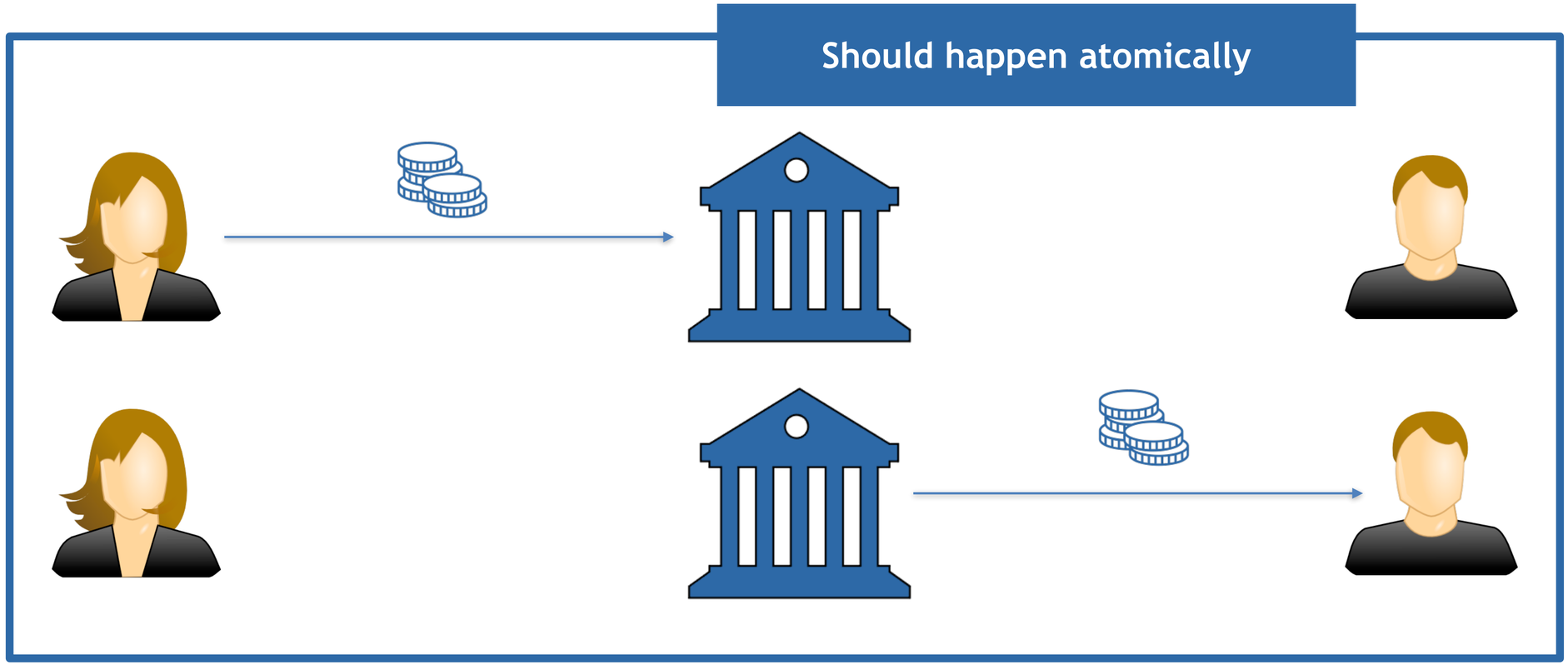
- One cannot open payment channel with everyone, hence, in practice parties use gateways (payment channel hubs (PCHs))

Atomicity in PCHs

Atomicity in PCHs



Atomicity in PCHs

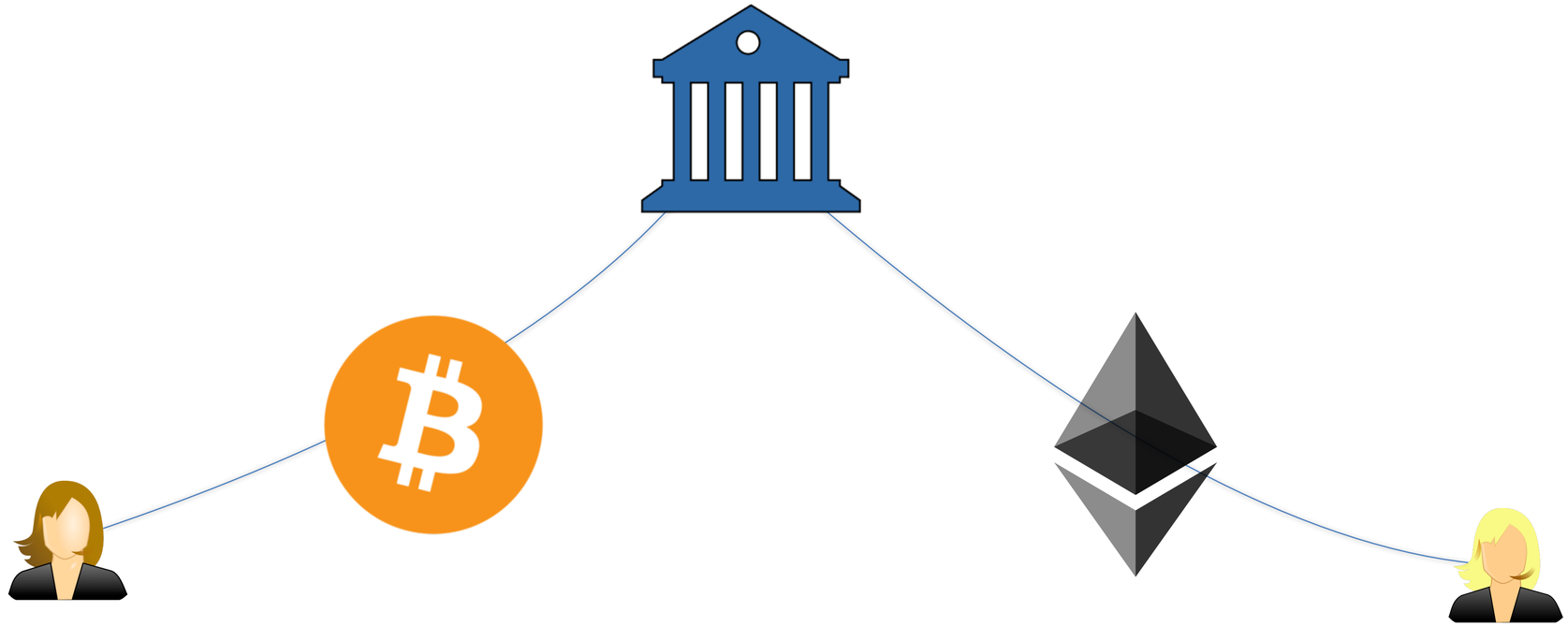


Unlinkability in PCHs

- The gateway should not learn who is paying to whom (i.e., link sender/receiver pairs)

Interoperability in PCHs

- Create a PCH payment protocol backwards compatible with Bitcoin (and as many currencies as possible)

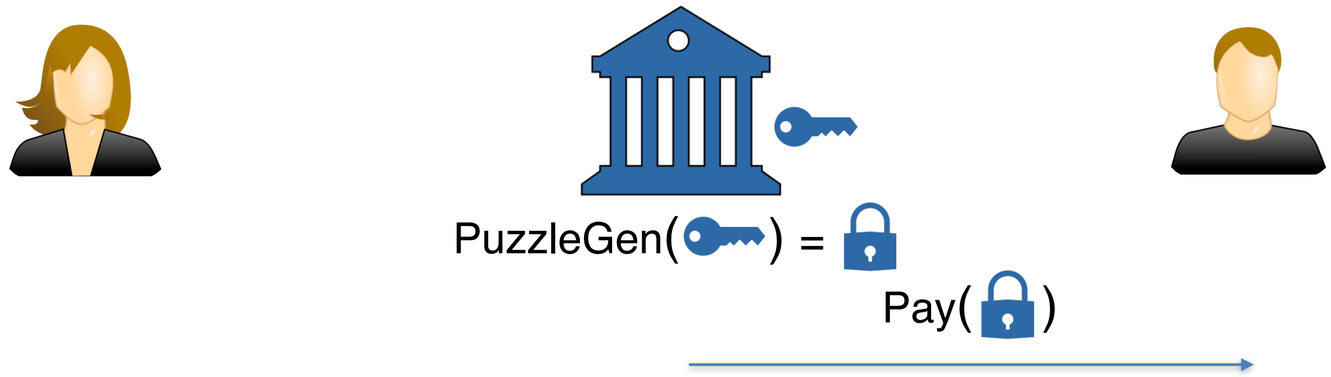


State-of-the-art

- TumbleBit is the state-of-the-art PCH introduced by Heilman et al. (NDSS'17), composed of *puzzle promise* and *puzzle solver* protocols

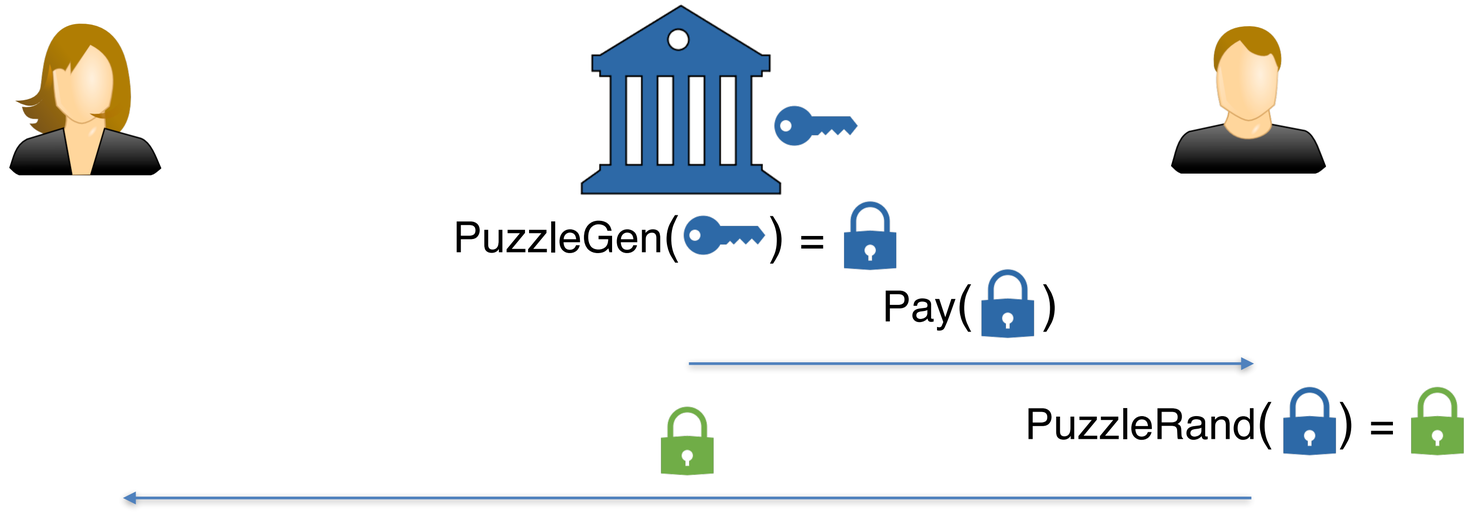
State-of-the-art

- TumbleBit is the state-of-the-art PCH introduced by Heilman et al. (NDSS'17), composed of *puzzle promise* and *puzzle solver* protocols



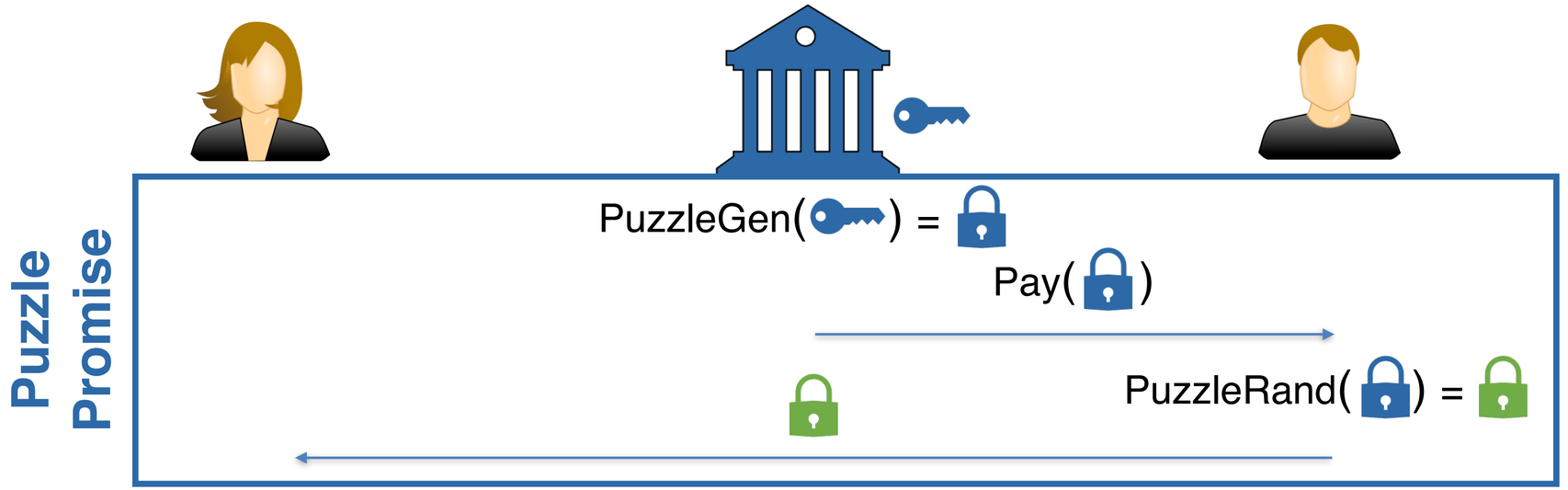
State-of-the-art

- TumbleBit is the state-of-the-art PCH introduced by Heilman et al. (NDSS'17), composed of *puzzle promise* and *puzzle solver* protocols



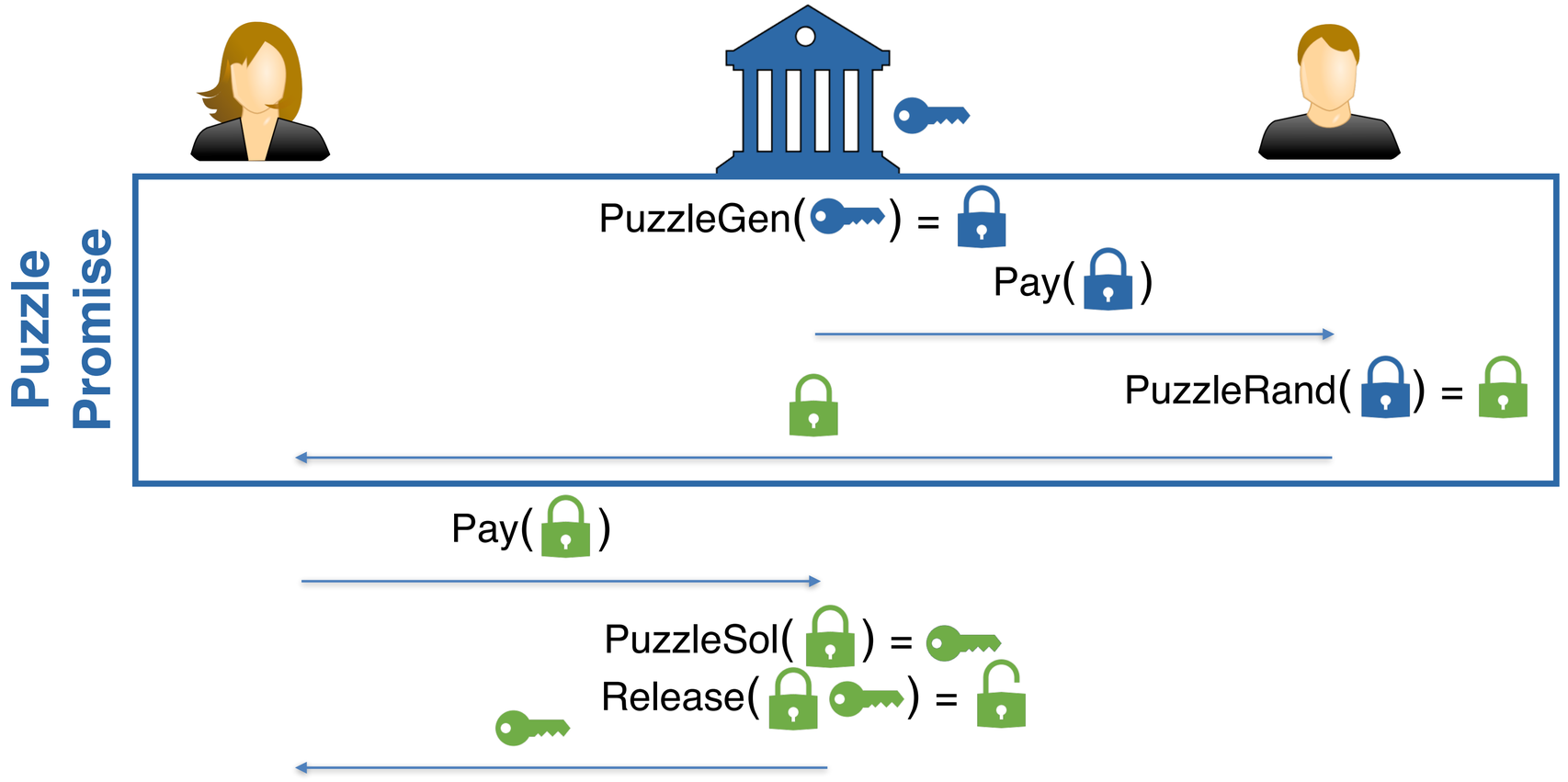
State-of-the-art

- TumbleBit is the state-of-the-art PCH introduced by Heilman et al. (NDSS'17), composed of *puzzle promise* and *puzzle solver* protocols



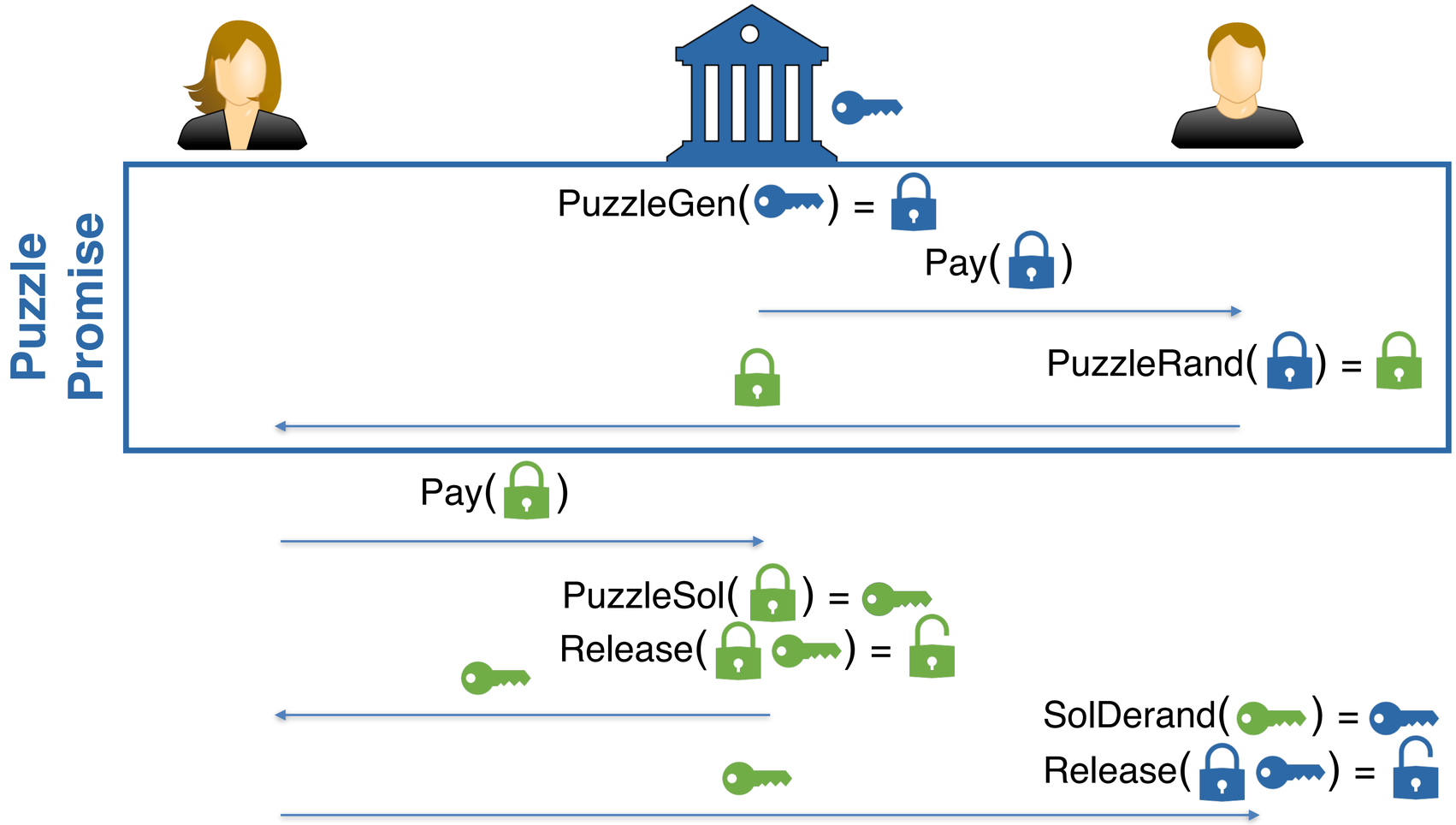
State-of-the-art

- TumbleBit is the state-of-the-art PCH introduced by Heilman et al. (NDSS'17), composed of *puzzle promise* and *puzzle solver* protocols



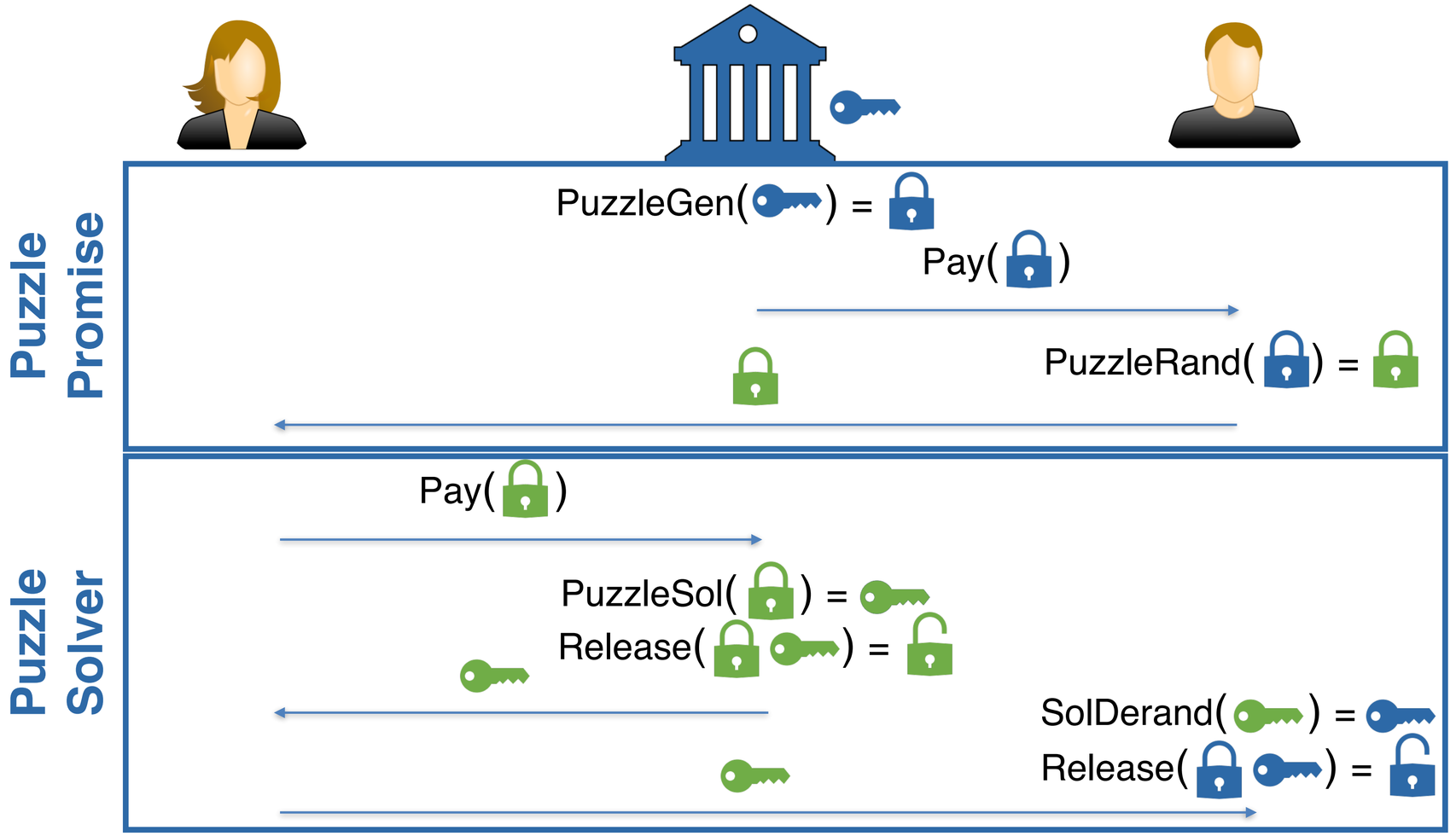
State-of-the-art

- TumbleBit is the state-of-the-art PCH introduced by Heilman et al. (NDSS'17), composed of *puzzle promise* and *puzzle solver* protocols



State-of-the-art

- TumbleBit is the state-of-the-art PCH introduced by Heilman et al. (NDSS'17), composed of *puzzle promise* and *puzzle solver* protocols



State-of-the-art

- Drawbacks of TumbleBit:
  - Lacks interoperability (only supports HTLC-based currencies,  is a hash)
  - Large communication overhead (due to the cut-and-choose proof technique needs to send large number of )
  - Susceptible to griefing attacks (i.e., asking the gateway for a large number of promises which never get released later)

State-of-the-art

- Drawbacks of TumbleBit:
  - Lacks interoperability (only supports HTLC-based currencies,  is a hash)
  - Large communication overhead (due to the cut-and-choose proof technique needs to send large number of )
  - Susceptible to griefing attacks (i.e., asking the gateway for a large number of promises which never get released later)

Is it possible to design a PCH that is efficient and that provides atomicity, unlinkability, and interoperability (with virtually all cryptocurrencies)?

State-of-the-art

- Drawbacks of TumbleBit:
  - Lacks interoperability (only supports HTLC-based currencies, 🔒 is a hash)
  - Large communication overhead (due to the cut-and-choose proof technique needs to send large number of 🔒)
  - Susceptible to griefing attacks (i.e., asking the gateway for a large number of promises which never get released later)

Is it possible to design a PCH that is efficient and that provides atomicity, unlinkability, and interoperability (with virtually all cryptocurrencies)?

Yes!

Building Block: Adaptor Signatures (AS)

- Formally defined by Aumayr et al. (https://eprint.iacr.org/2020/476)
- Goals:
  - Gateway can create an “incomplete-signature” that Bob can only finish using a secret value k. The condition of the signature corresponds to  , and the secret k to 
  - If Bob finishes the signature, Gateway learns k using the full and incomplete signatures

Building Block: Adaptor Signatures (AS)

- Formally defined by Aumayr et al. (https://eprint.iacr.org/2020/476)
- Goals:
  - Gateway can create an “incomplete-signature” that Bob can only finish using a secret value k. The condition of the signature corresponds to 🔒, and the secret k to 🔑
  - If Bob finishes the signature, Gateway learns k using the full and incomplete signatures

$pk_G = sk_G * G$

🔒 Condition: $C = k * G$

tx_G: Gateway pays 1 coin to Bob

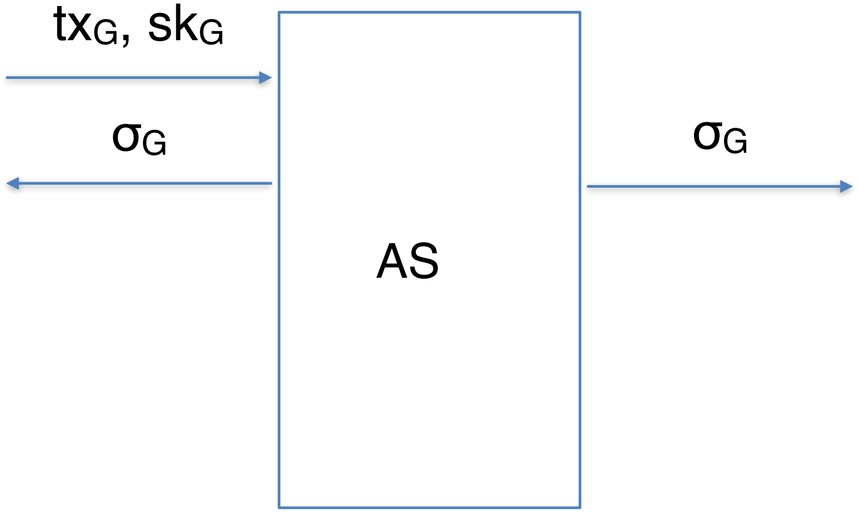
Building Block: Adaptor Signatures (AS)

- Formally defined by Aumayr et al. (https://eprint.iacr.org/2020/476)
- Goals:
  - Gateway can create an “incomplete-signature” that Bob can only finish using a secret value k. The condition of the signature corresponds to 🔒, and the secret k to 🔑
  - If Bob finishes the signature, Gateway learns k using the full and incomplete signatures

$pk_G = sk_G * G$

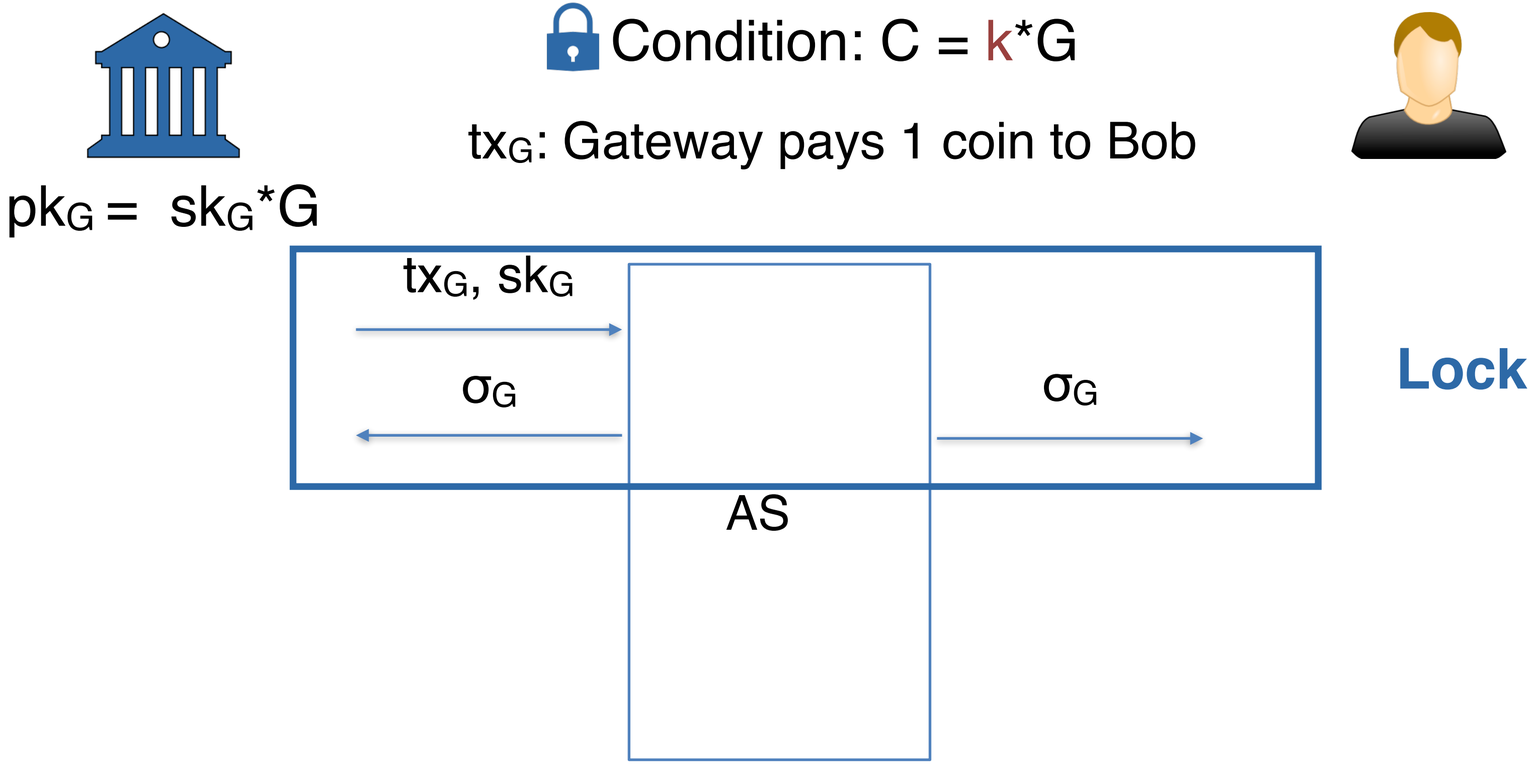
🔒 Condition: $C = k * G$

tx_G: Gateway pays 1 coin to Bob



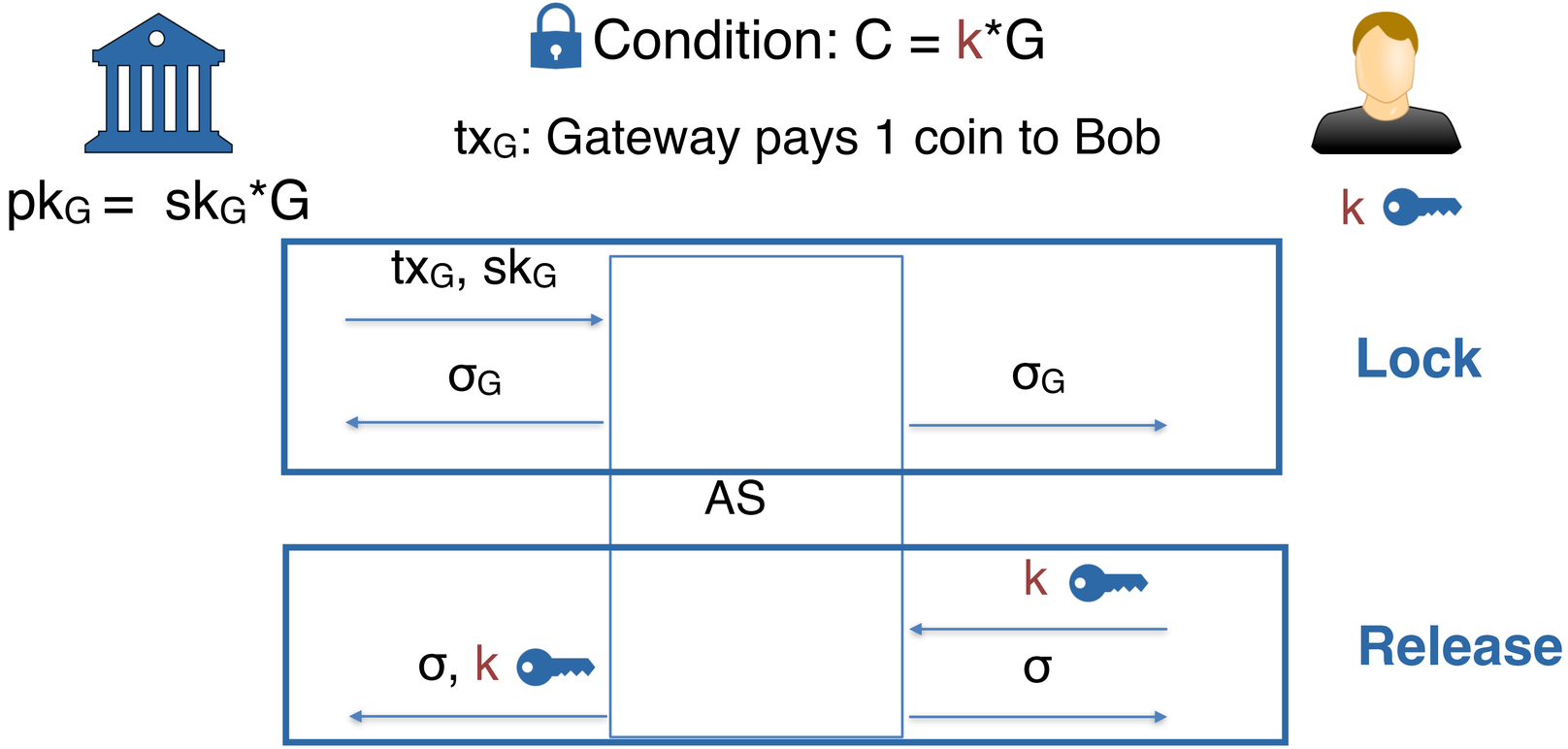
Building Block: Adaptor Signatures (AS)

- Formally defined by Aumayr et al. (https://eprint.iacr.org/2020/476)
- Goals:
  - Gateway can create an “incomplete-signature” that Bob can only finish using a secret value k. The condition of the signature corresponds to 🔒, and the secret k to 🔑
  - If Bob finishes the signature, Gateway learns k using the full and incomplete signatures



Building Block: Adaptor Signatures (AS)

- Formally defined by Aumayr et al. (https://eprint.iacr.org/2020/476)
- Goals:
  - Gateway can create an “incomplete-signature” that Bob can only finish using a secret value k. The condition of the signature corresponds to 🔒, and the secret k to 🔑
  - If Bob finishes the signature, Gateway learns k using the full and incomplete signatures



Payment in PCH: First Attempt

(pk_A, sk_A)

$C = k*G$

(pk_G, sk_G, k)

$C = k*G$

(pk_B, sk_B)

AS

AS

Payment in PCH: First Attempt

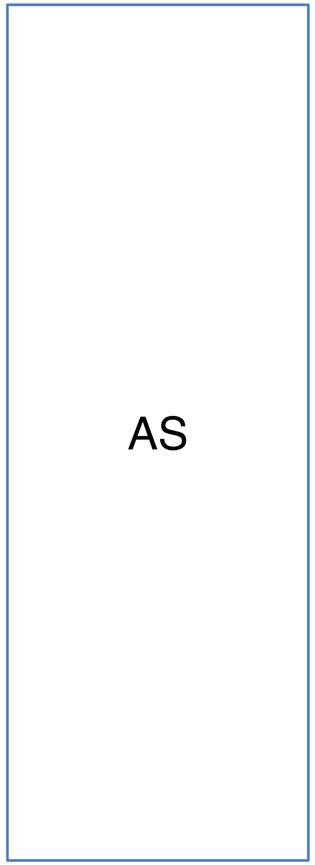
(pk_A, sk_A)

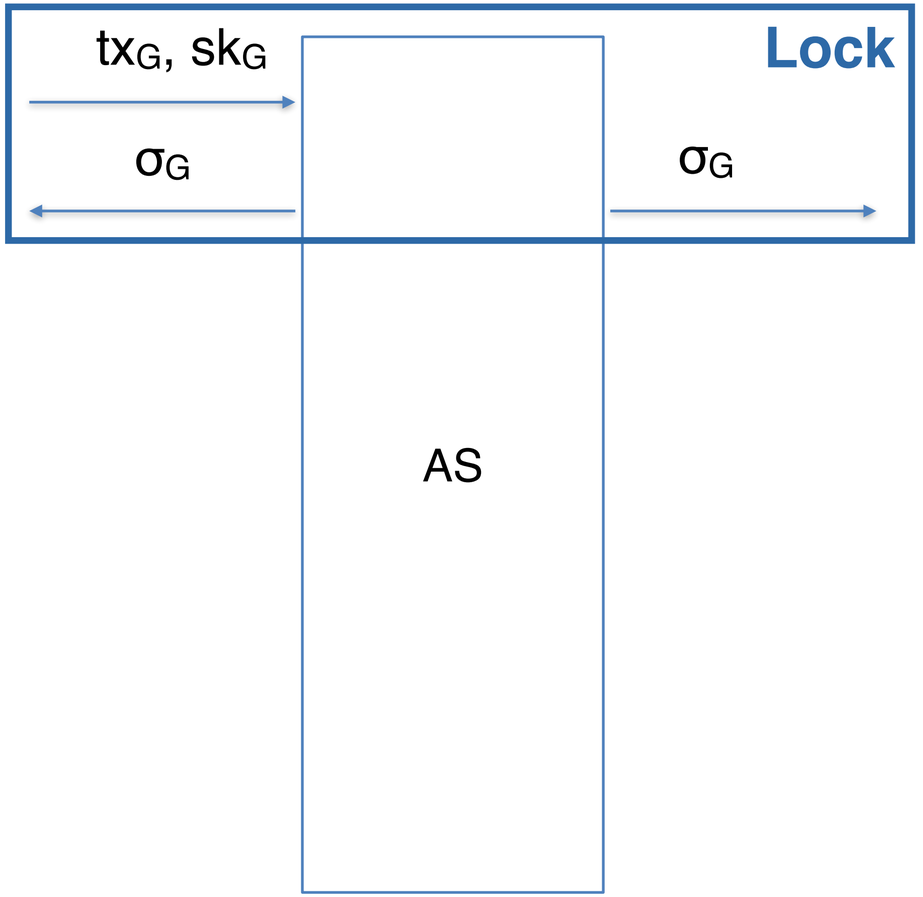
$C = k*G$

(pk_G, sk_G, k)

$C = k*G$

(pk_B, sk_B)





Payment in PCH: First Attempt

(pk_A, sk_A)

$C = k*G$

(pk_G, sk_G, k)

$C = k*G$

(pk_B, sk_B)

AS

Puzzle Promise

AS

Payment in PCH: First Attempt

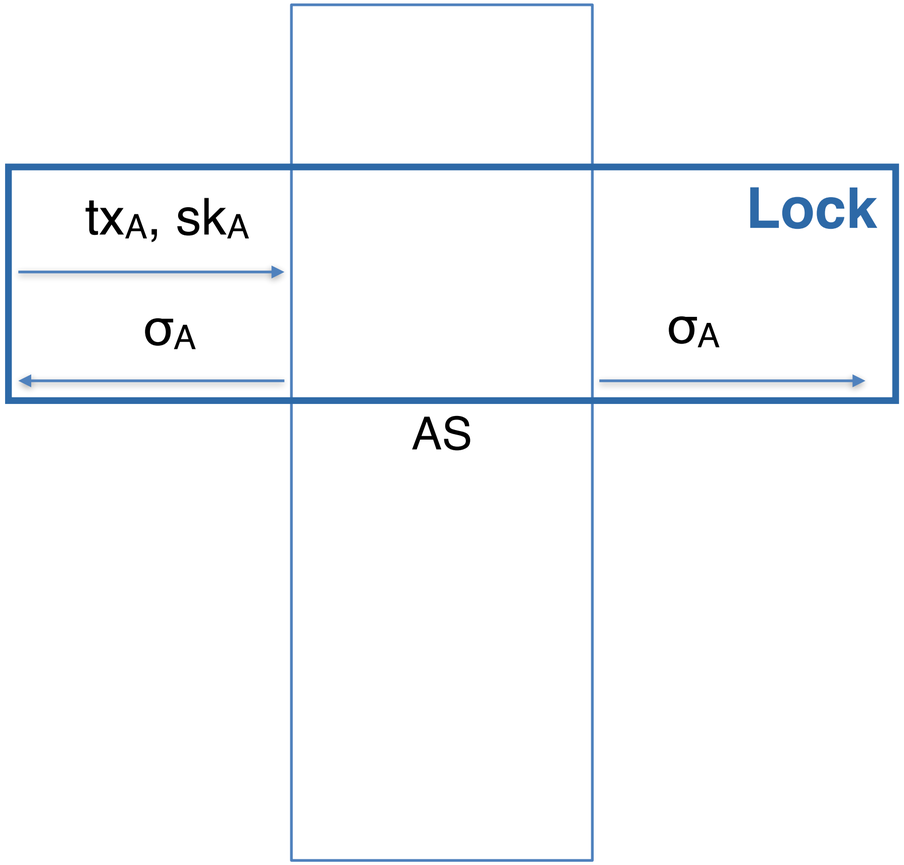
(pk_A, sk_A)

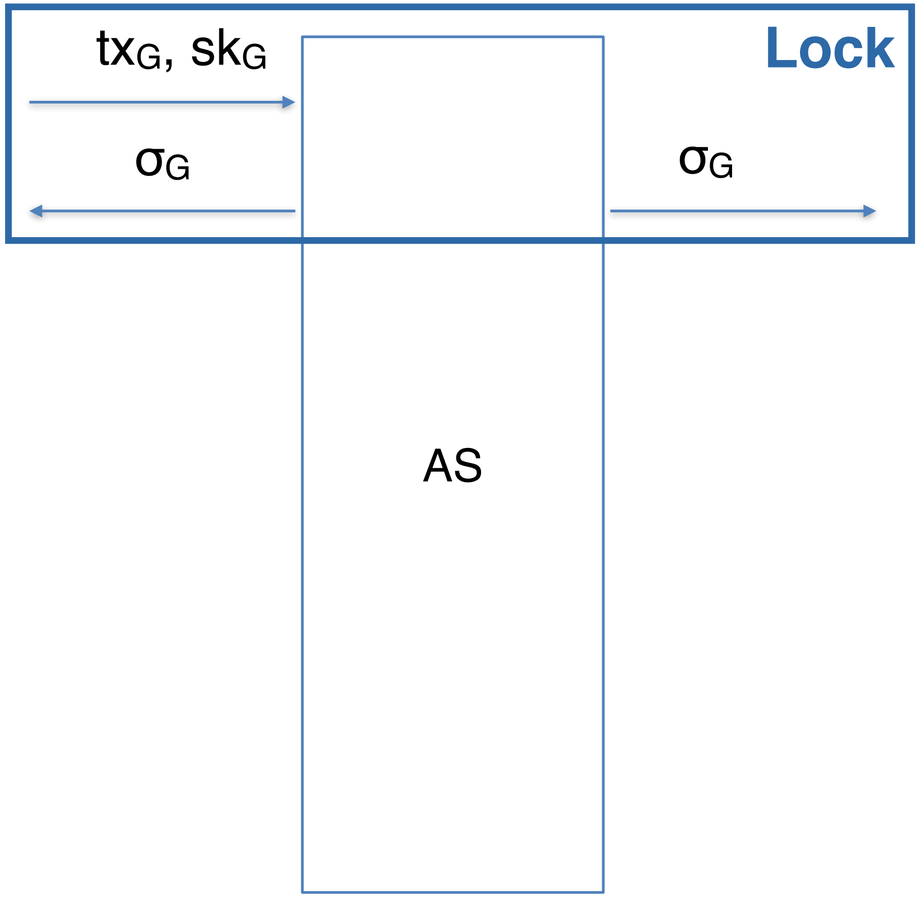
$C = k*G$

(pk_G, sk_G, k)

$C = k*G$

(pk_B, sk_B)





Payment in PCH: First Attempt

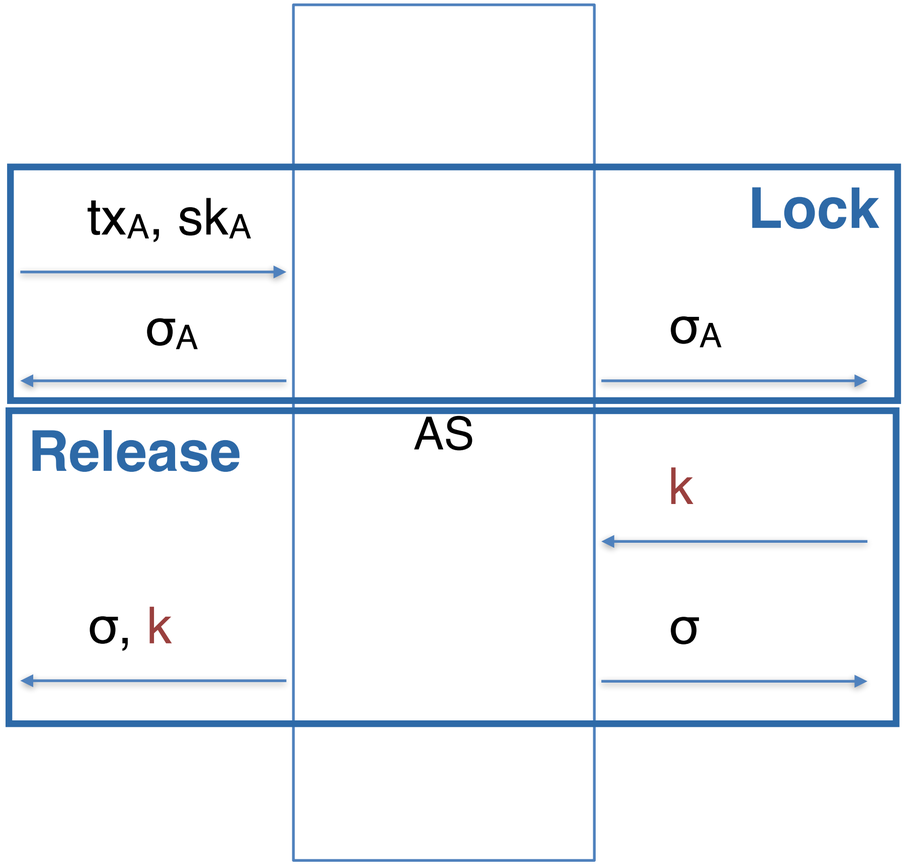
(pk_A, sk_A)

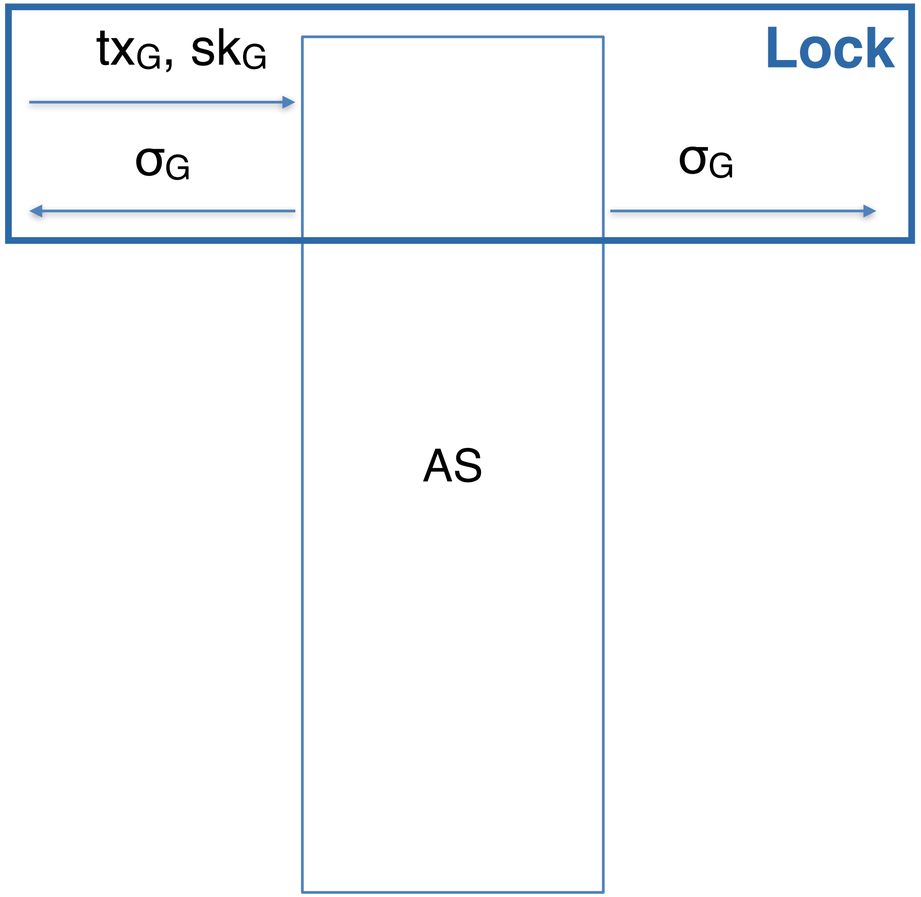
$C = k*G$

(pk_G, sk_G, k)

$C = k*G$

(pk_B, sk_B)





Payment in PCH: First Attempt

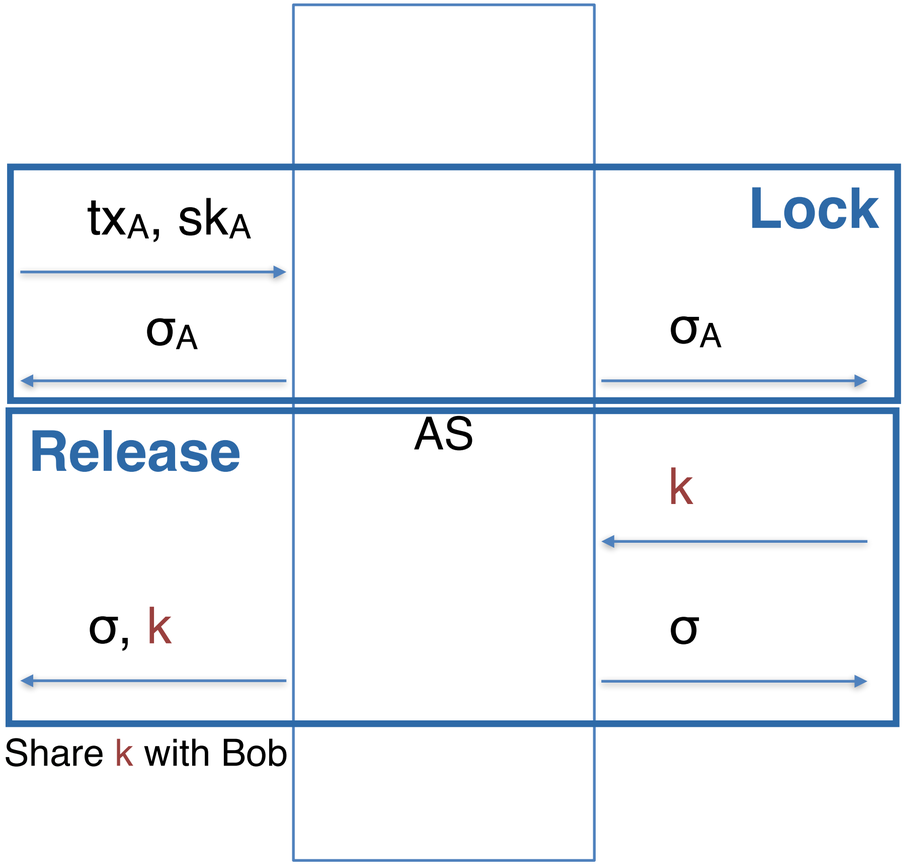
(pk_A, sk_A)

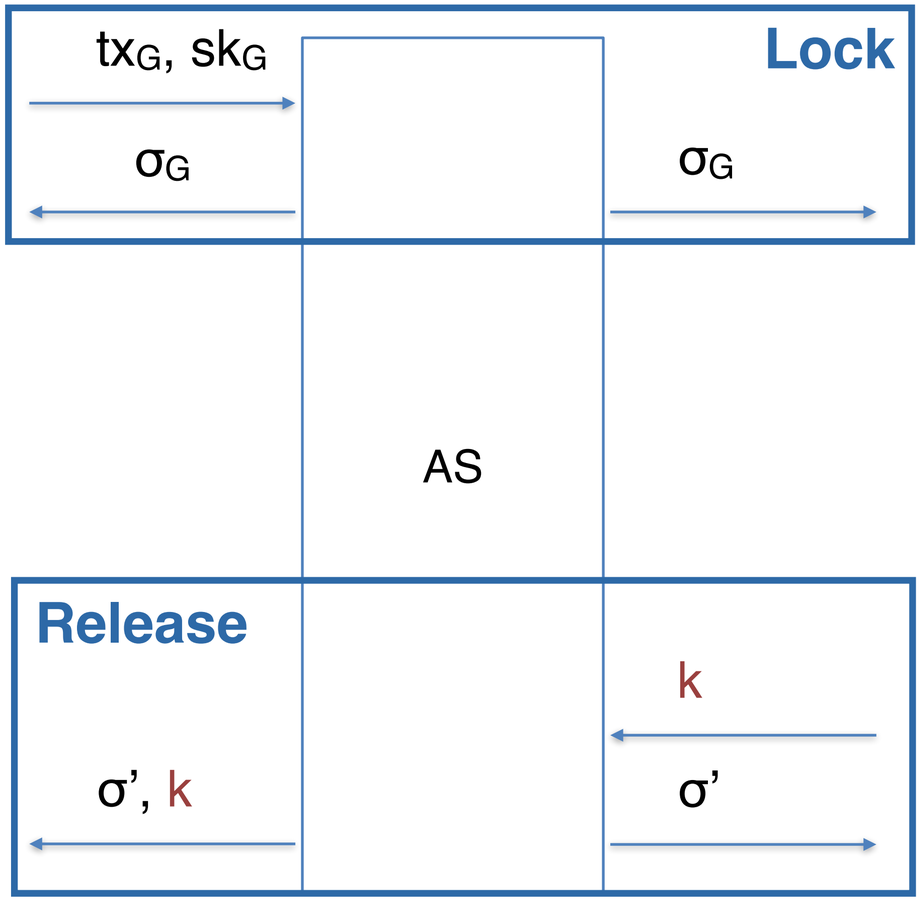
$C = k*G$

(pk_G, sk_G, k)

$C = k*G$

(pk_B, sk_B)





Payment in PCH: First Attempt

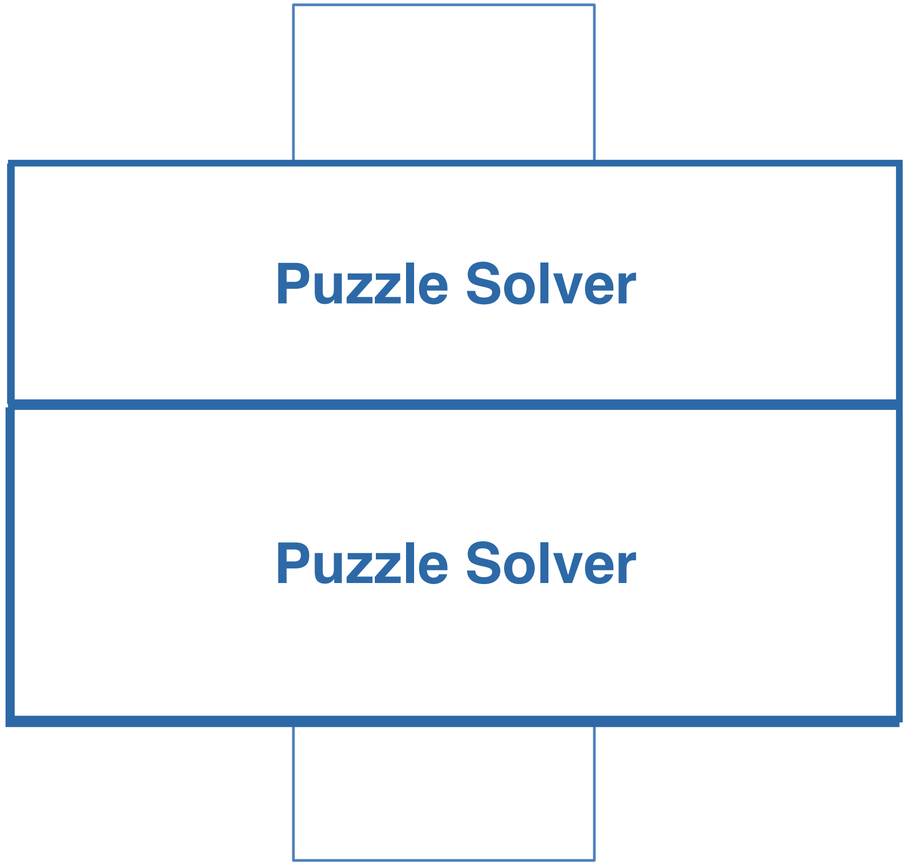
(pk_A, sk_A)

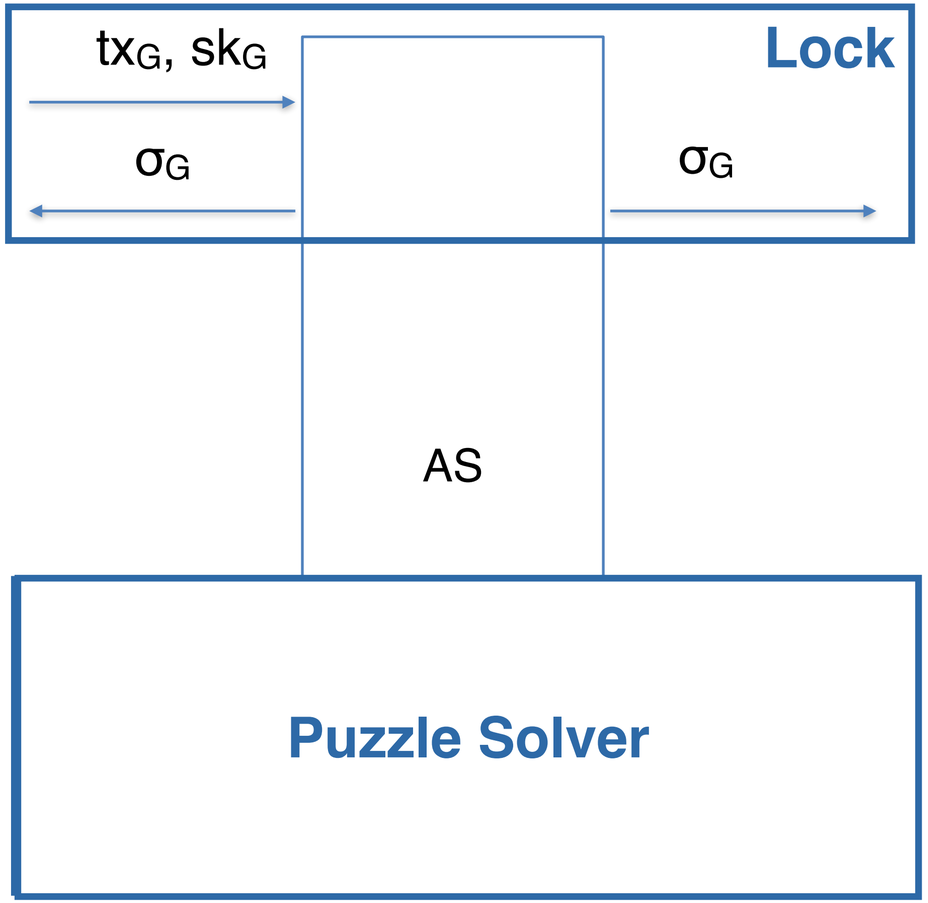
$C = k*G$

(pk_G, sk_G, k)

$C = k*G$

(pk_B, sk_B)





Payment in PCH: First Attempt

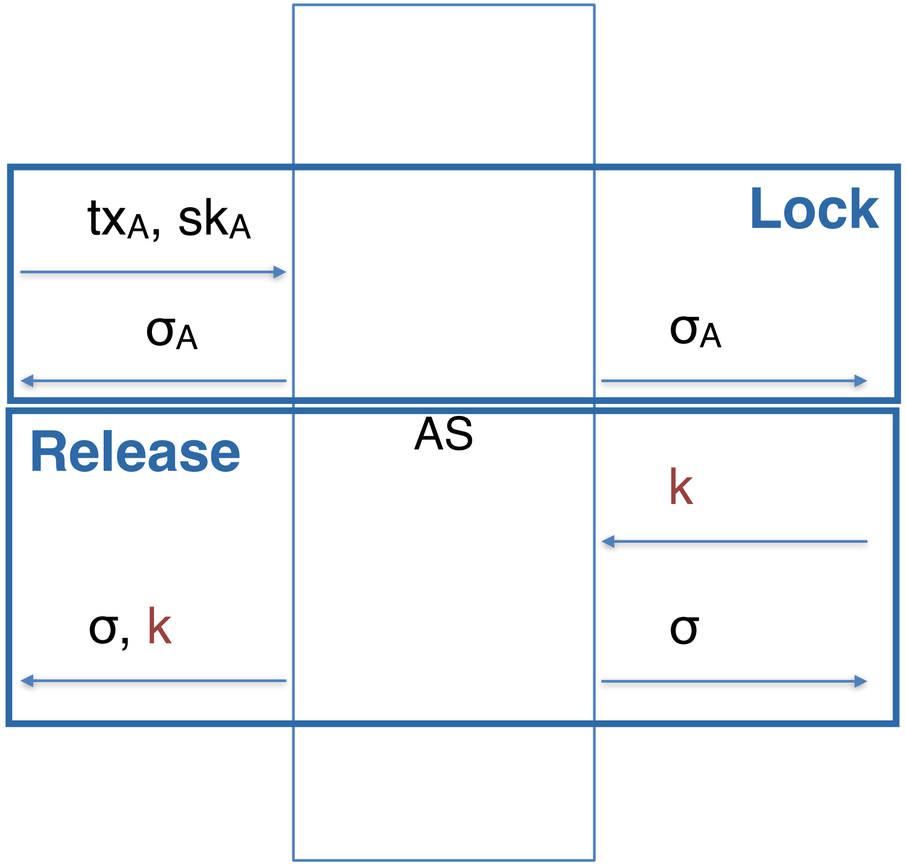
(pk_A, sk_A)

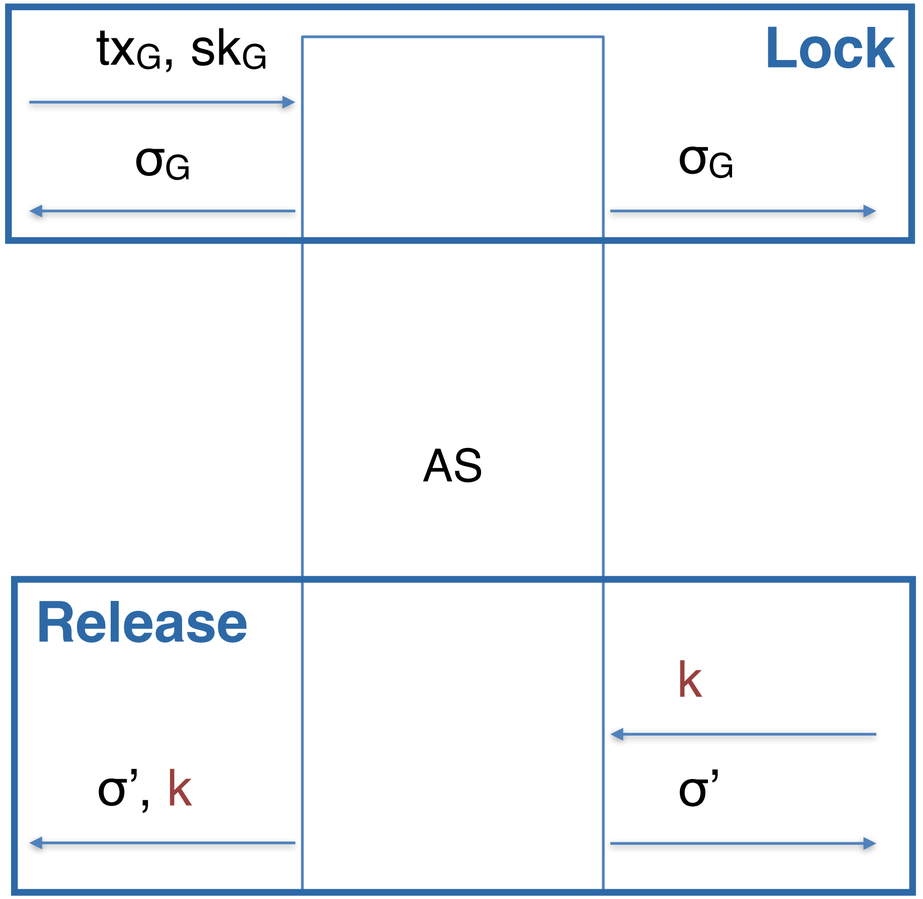
$C = k*G$

(pk_G, sk_G, k)

$C = k*G$

(pk_B, sk_B)





Privacy Issue

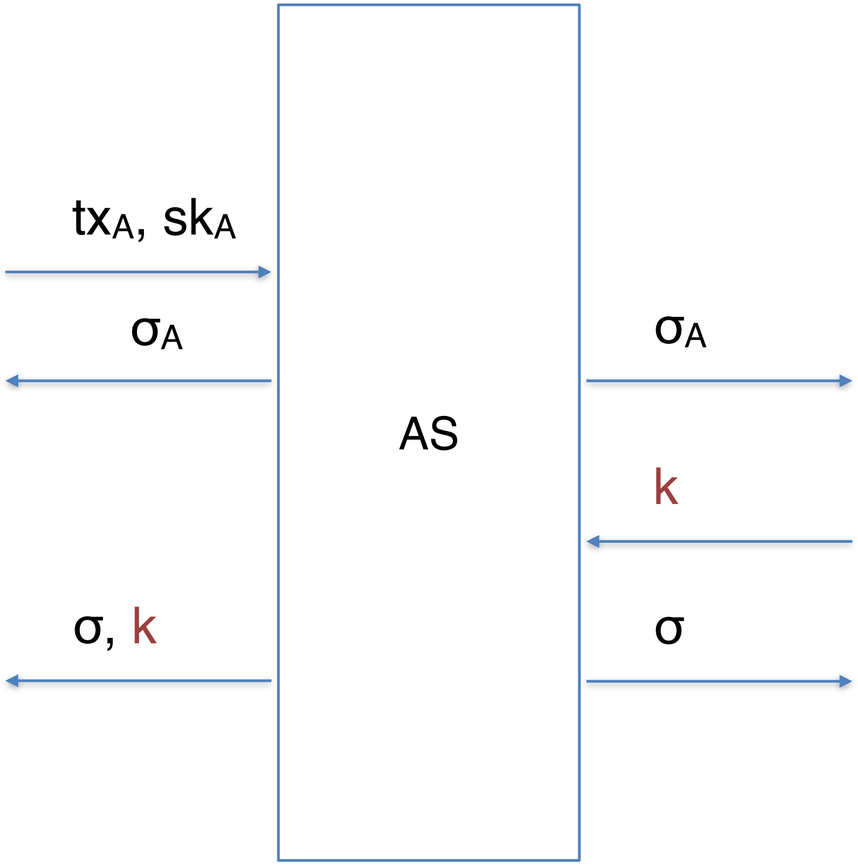
(pk_A, sk_A)

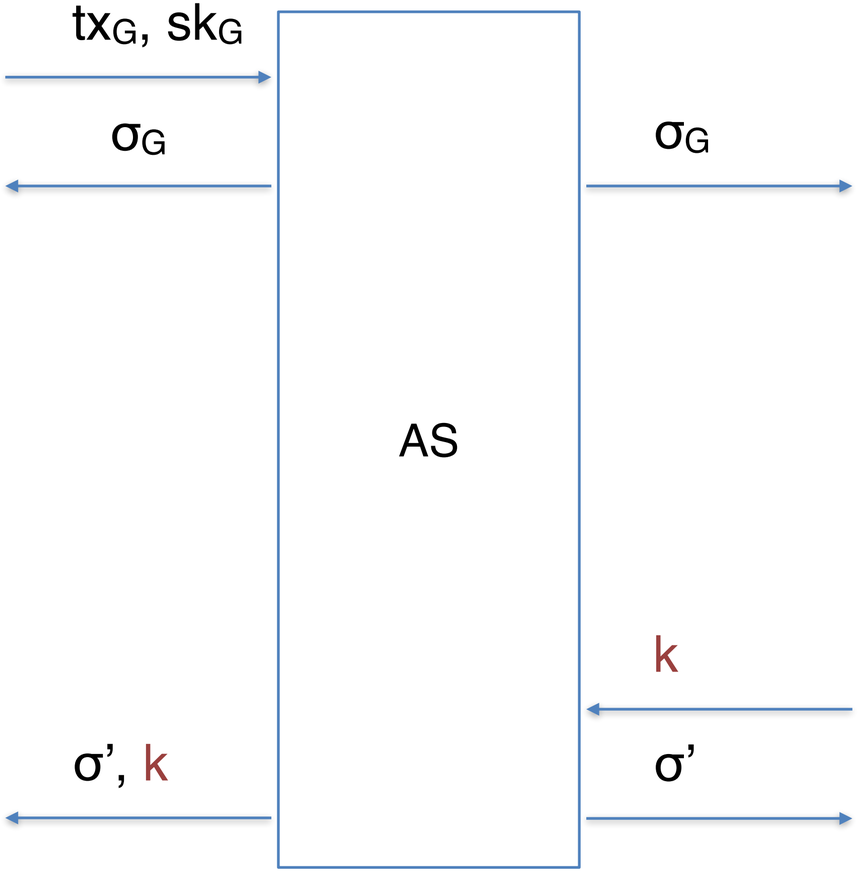
$C = k*G$

(pk_G, sk_G, k)

$C = k*G$

(pk_B, sk_B)





Privacy Issue

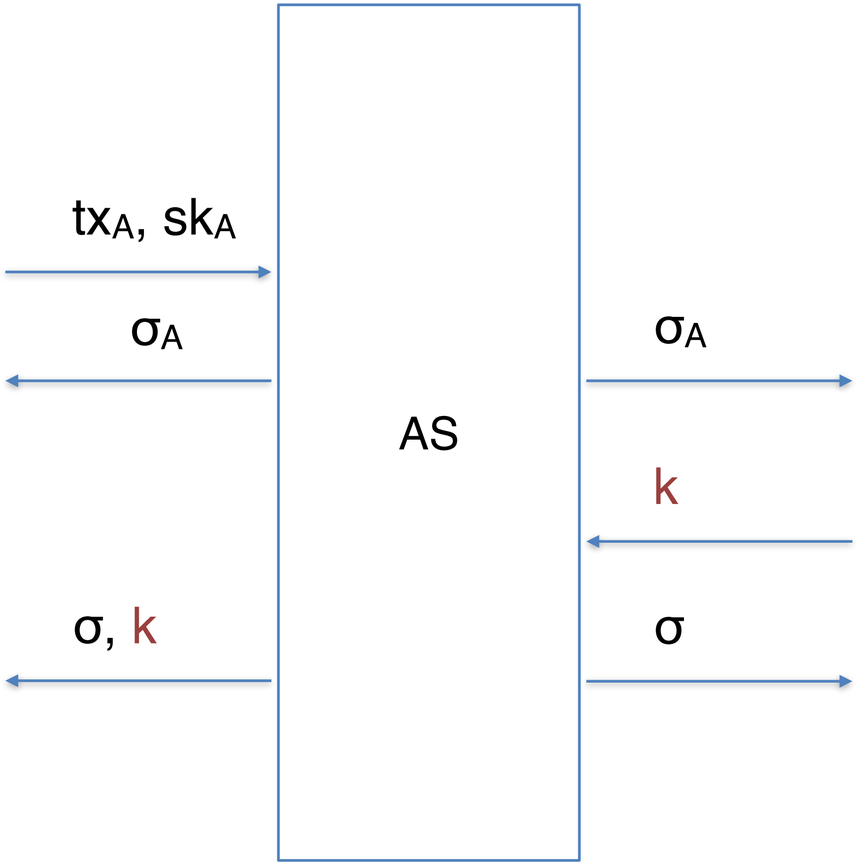
$C = k*G$

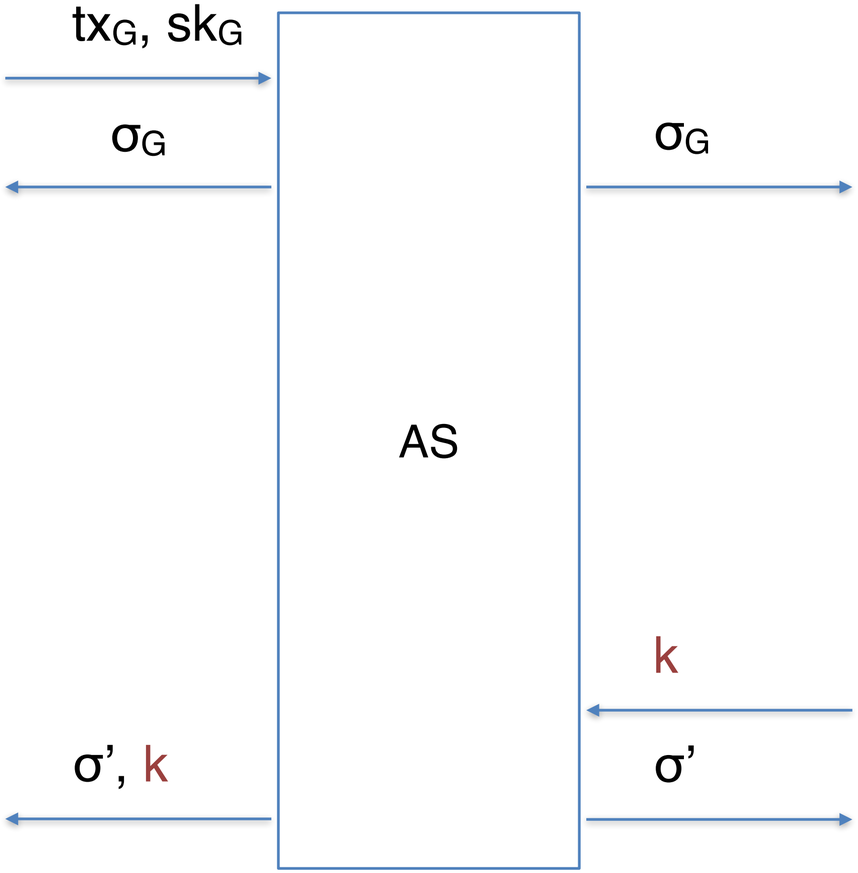
$C = k*G$

(pk_A, sk_A)

(pk_G, sk_G, k)

(pk_B, sk_B)



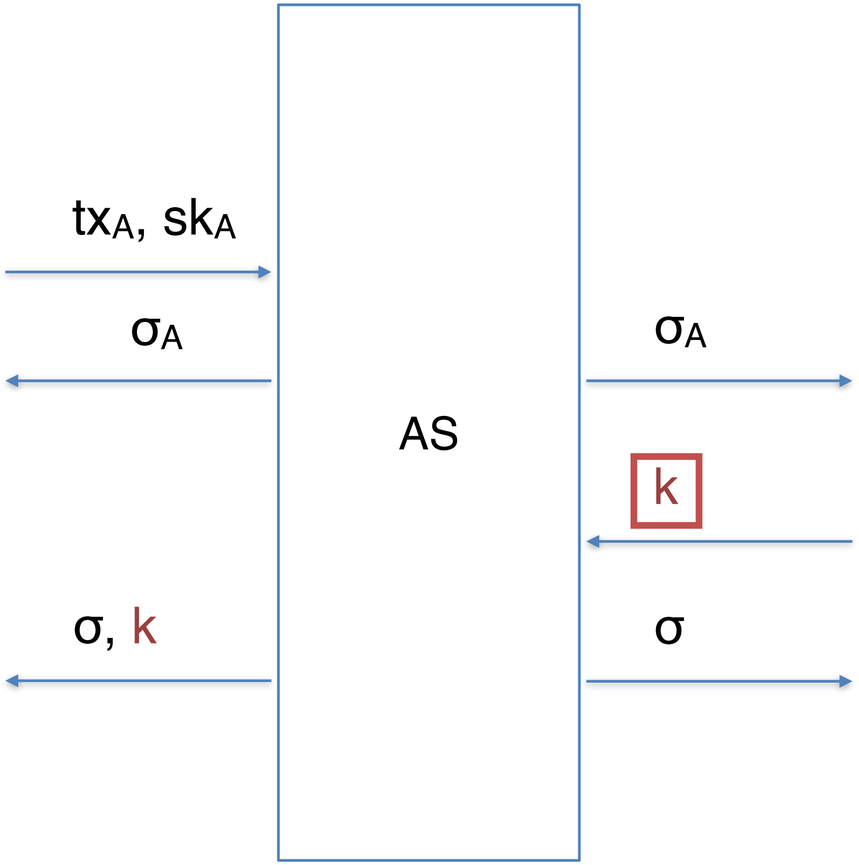


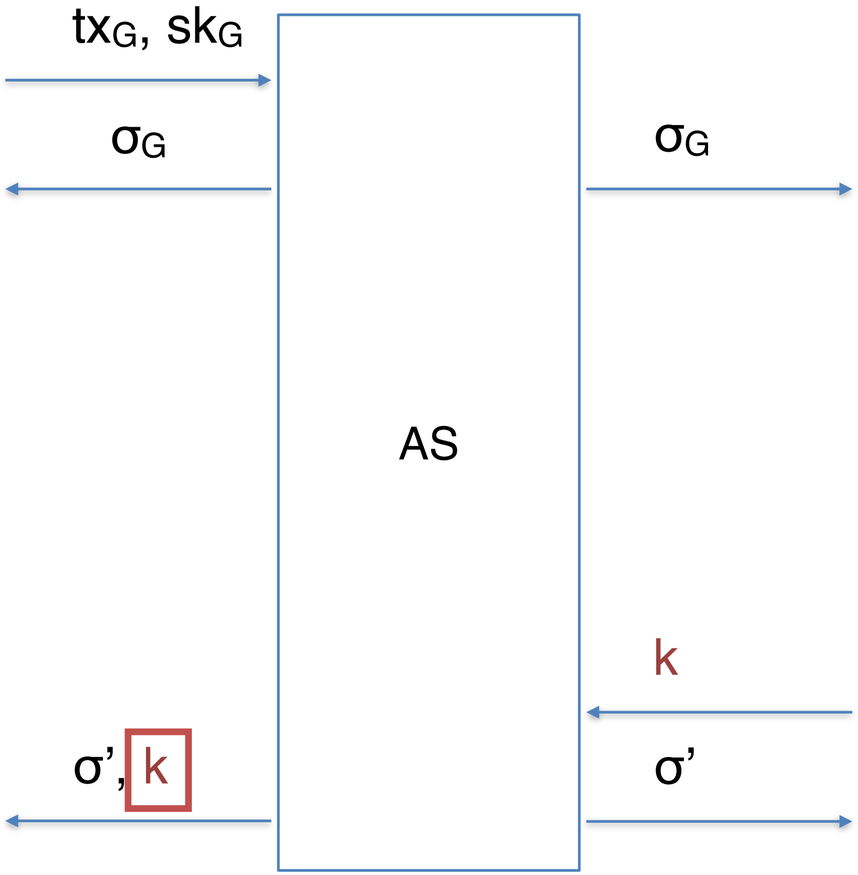
Privacy Issue

C = k*G

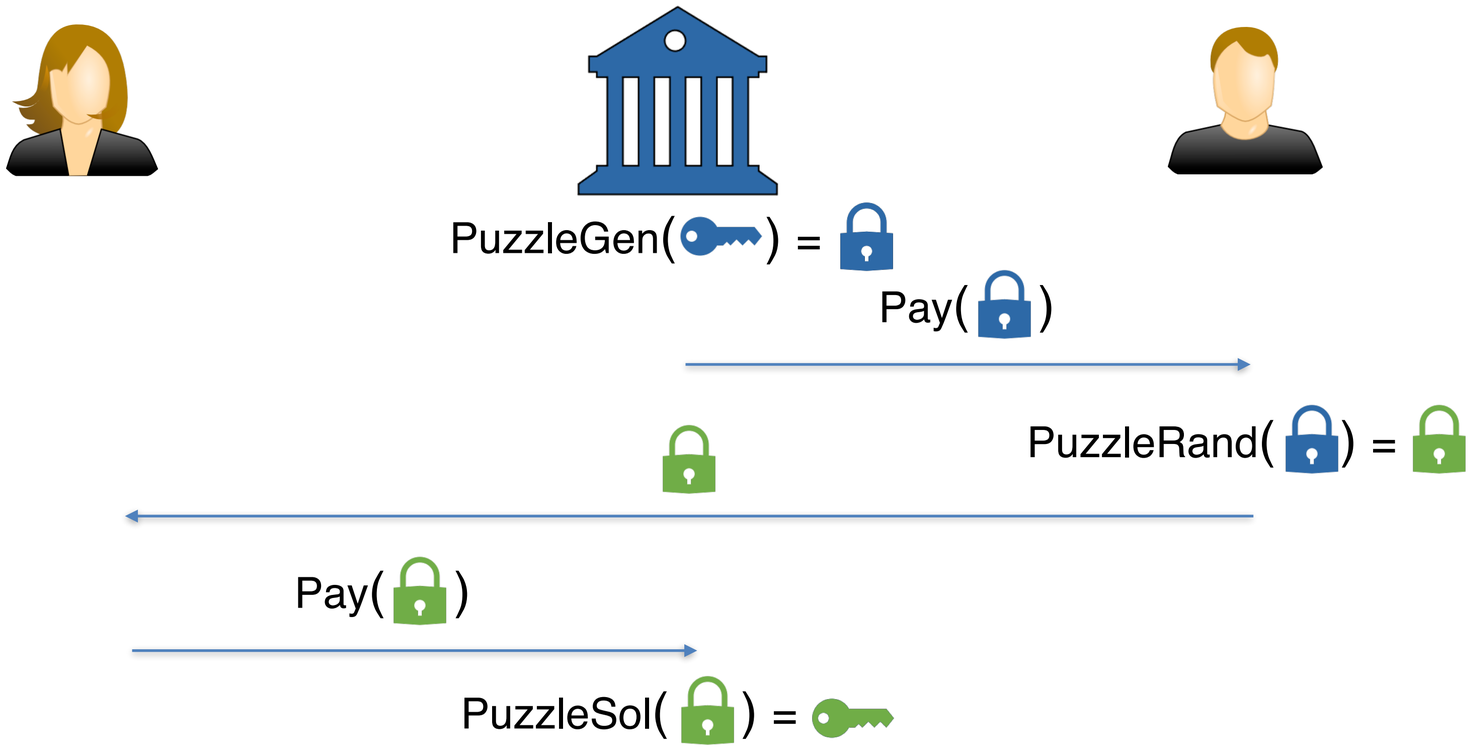
C = k*G

(pk_A, sk_A) (pk_G, sk_G, k) (pk_B, sk_B)

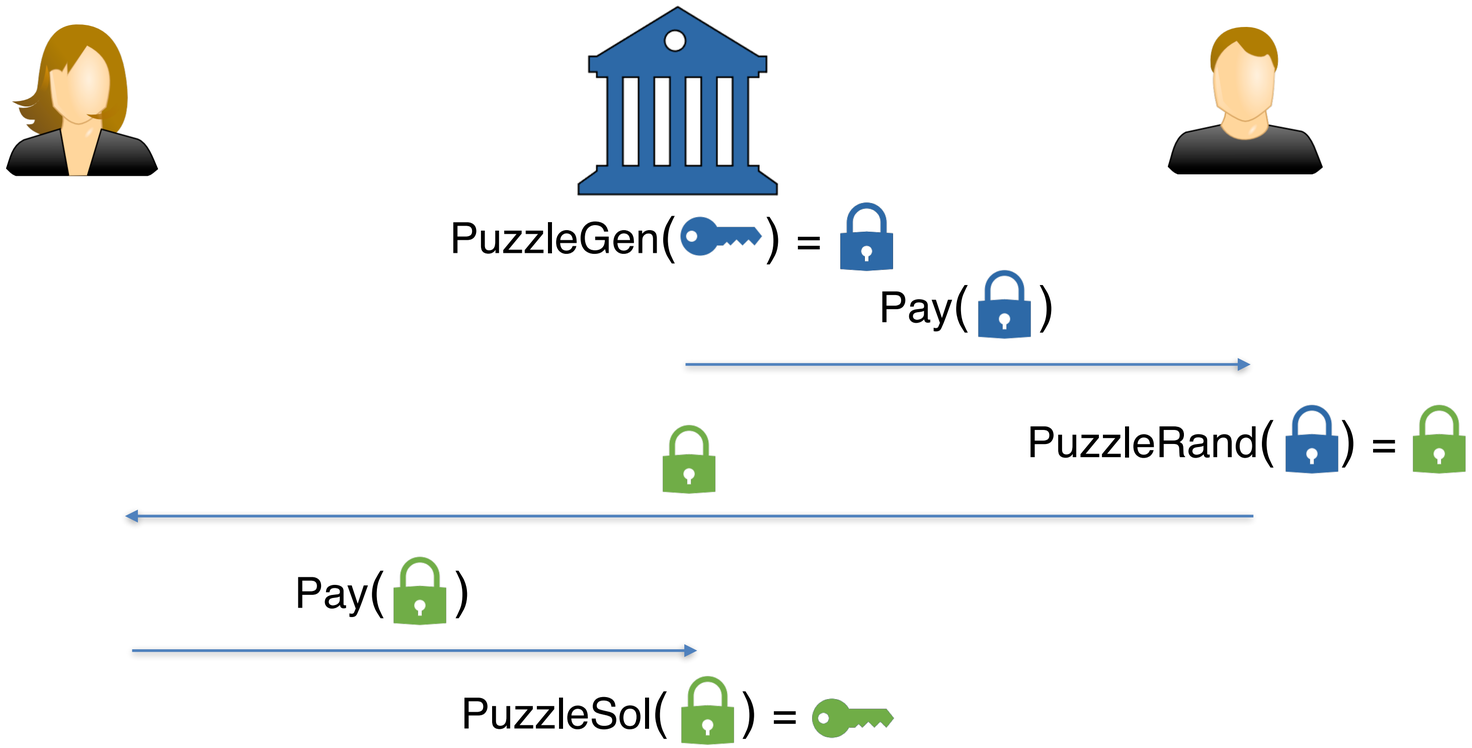




Privacy Solution

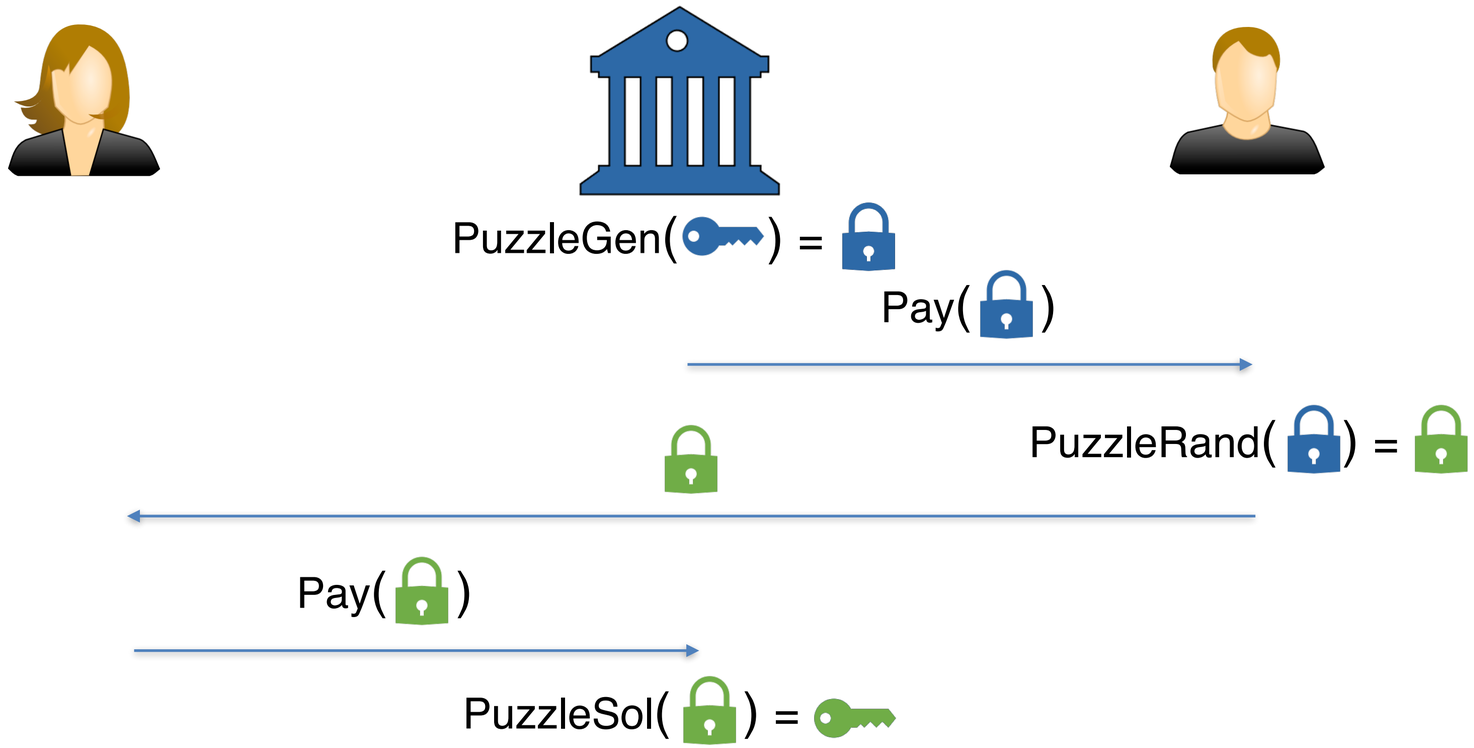


Privacy Solution



- Recall in our case the puzzle is the condition C = k*G, and the solution is the secret k. Hence, the randomized puzzle would correspond to computing C' = r*k*G, for a random r

Privacy Solution



- Recall in our case the puzzle is the condition C = k*G, and the solution is the secret k. Hence, the randomized puzzle would correspond to computing C' = r*k*G, for a random r
- Gateway cannot solve the puzzle now as it does not know r. The solution is to extend the puzzle with the encryption of the secret k under the gateway's key

Building Block: Randomizable Puzzle

- Randomizable puzzle combines the condition of adaptor signature with an encryption under additively homomorphic encryption scheme
- Goals:
  - Gateway can create a puzzle  that can be solved using a trapdoor (e.g., secret key)
  - The puzzle can be randomized to create a fresh looking version 

Building Block: Randomizable Puzzle

- Randomizable puzzle combines the condition of adaptor signature with an encryption under additively homomorphic encryption scheme
- Goals:
  - Gateway can create a puzzle that can be solved using a trapdoor (e.g., secret key)
  - The puzzle can be randomized to create a fresh looking version

pp, td, k

r

Building Block: Randomizable Puzzle

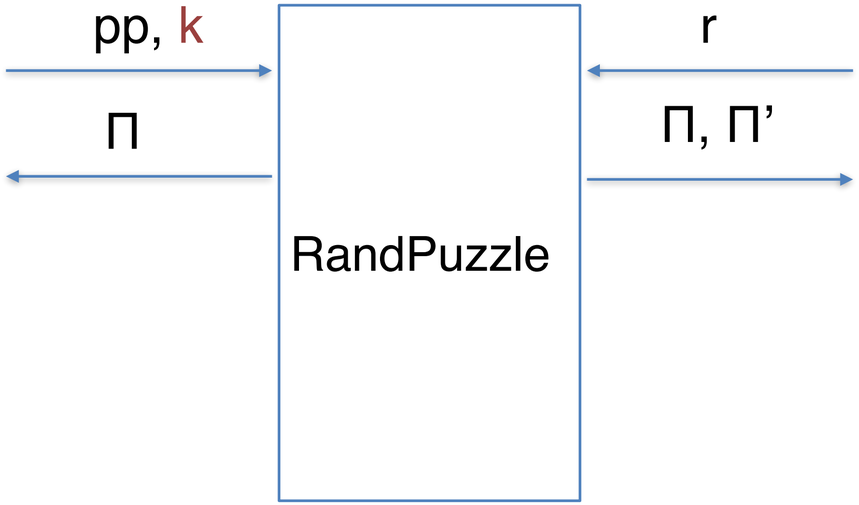
- Randomizable puzzle combines the condition of adaptor signature with an encryption under additively homomorphic encryption scheme
- Goals:
  - Gateway can create a puzzle that can be solved using a trapdoor (e.g., secret key)
  - The puzzle can be randomized to create a fresh looking version

pp, td, k

$\Pi = (C = k*G,\ c = Enc(pk_G,\ k))$

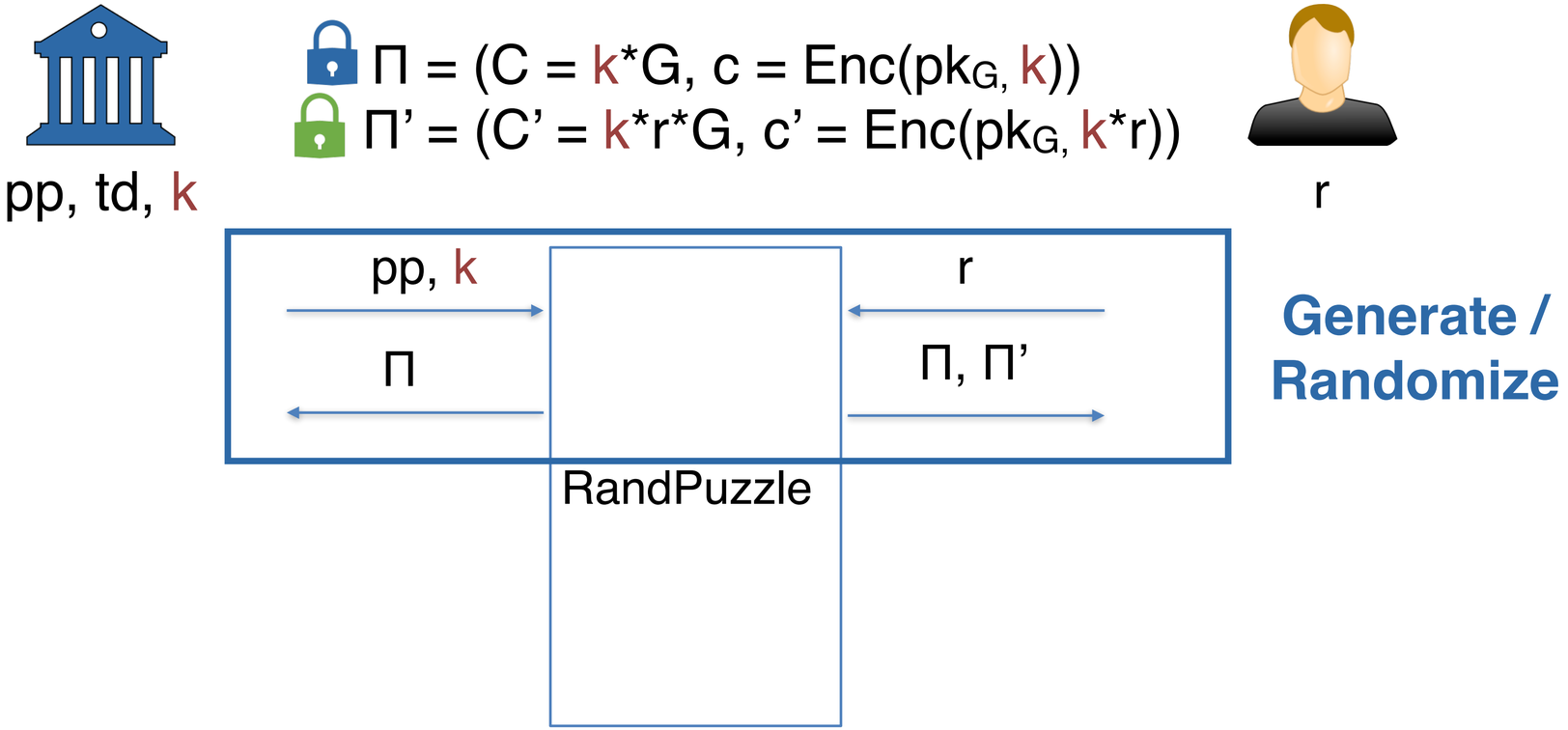
$\Pi' = (C' = k*r*G,\ c' = Enc(pk_G,\ k*r))$

r



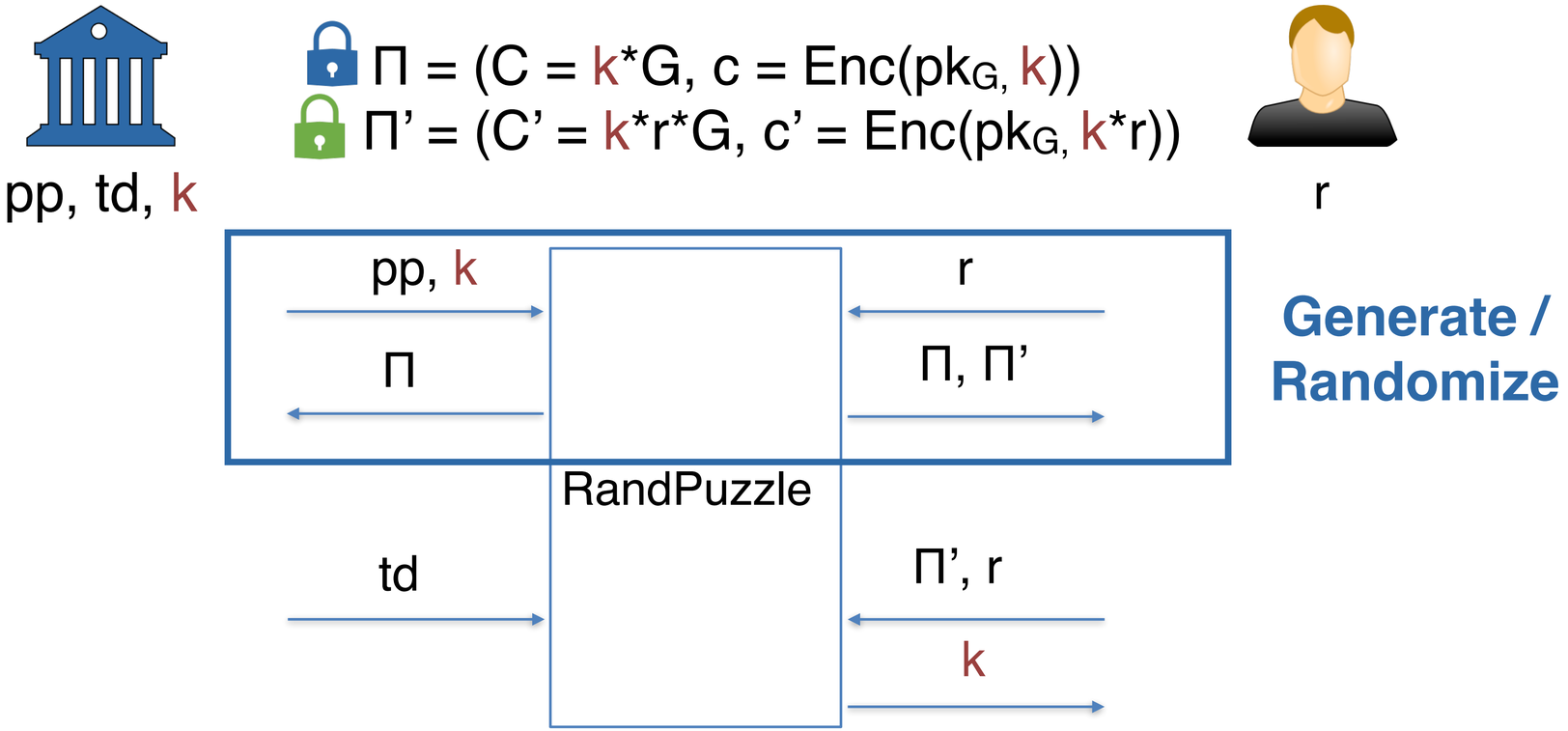
Building Block: Randomizable Puzzle

- Randomizable puzzle combines the condition of adaptor signature with an encryption under additively homomorphic encryption scheme
- Goals:
  - Gateway can create a puzzle that can be solved using a trapdoor (e.g., secret key)
  - The puzzle can be randomized to create a fresh looking version



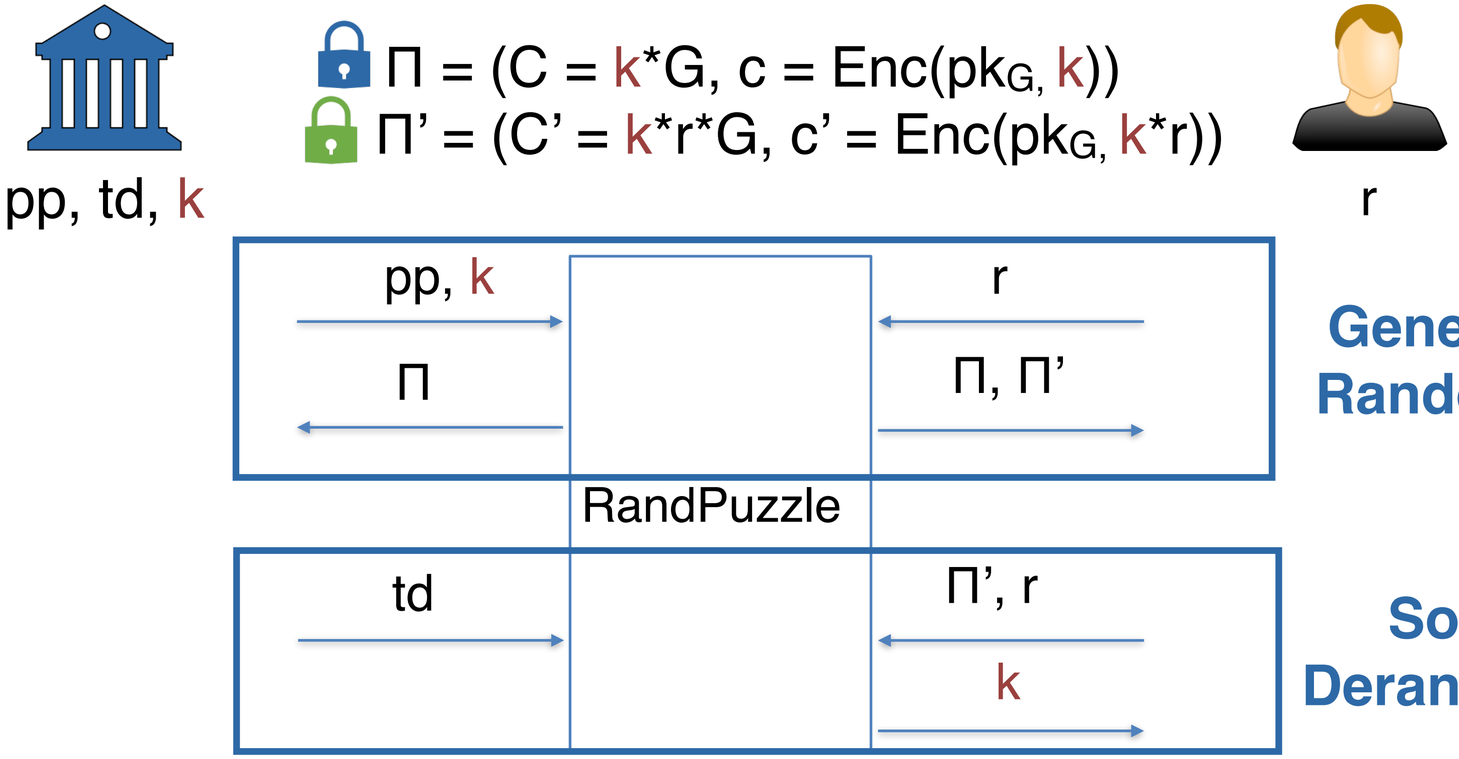
Building Block: Randomizable Puzzle

- Randomizable puzzle combines the condition of adaptor signature with an encryption under additively homomorphic encryption scheme
- Goals:
  - Gateway can create a puzzle that can be solved using a trapdoor (e.g., secret key)
  - The puzzle can be randomized to create a fresh looking version



Building Block: Randomizable Puzzle

- Randomizable puzzle combines the condition of adaptor signature with an encryption under additively homomorphic encryption scheme
- Goals:
  - Gateway can create a puzzle that can be solved using a trapdoor (e.g., secret key)
  - The puzzle can be randomized to create a fresh looking version



A²L: Protocol Overview

C = k*G

(pk_A, sk_A, r_A)

$(pk_G, sk_G, pp_G, td_G, k)$

r_B

AS + RandPuzzle

AS + RandPuzzle

A²L: Protocol Overview

(pk_A, sk_A, r_A)

$C = k*G$

$(pk_G, sk_G, pp_G, td_G, k)$

r_B

AS + RandPuzzle

tx_G, sk_G, pp_G

σ_G, Π

r_B

σ_G, Π, Π'

Share Π' with Alice

AS + RandPuzzle

A²L: Protocol Overview

$C'' = r_A * r_B * k * G$

$C = k * G$

(pk_A, sk_A, r_A)

$(pk_G, sk_G, pp_G, td_G, k)$

r_B

tx_A, sk_A, Π', r_A

σ_A

AS + RandPuzzle

σ_A, Π''

tx_G, sk_G, pp_G

σ_G, Π

AS + RandPuzzle

r_B

σ_G, Π, Π'

Share Π' with Alice

A²L: Protocol Overview

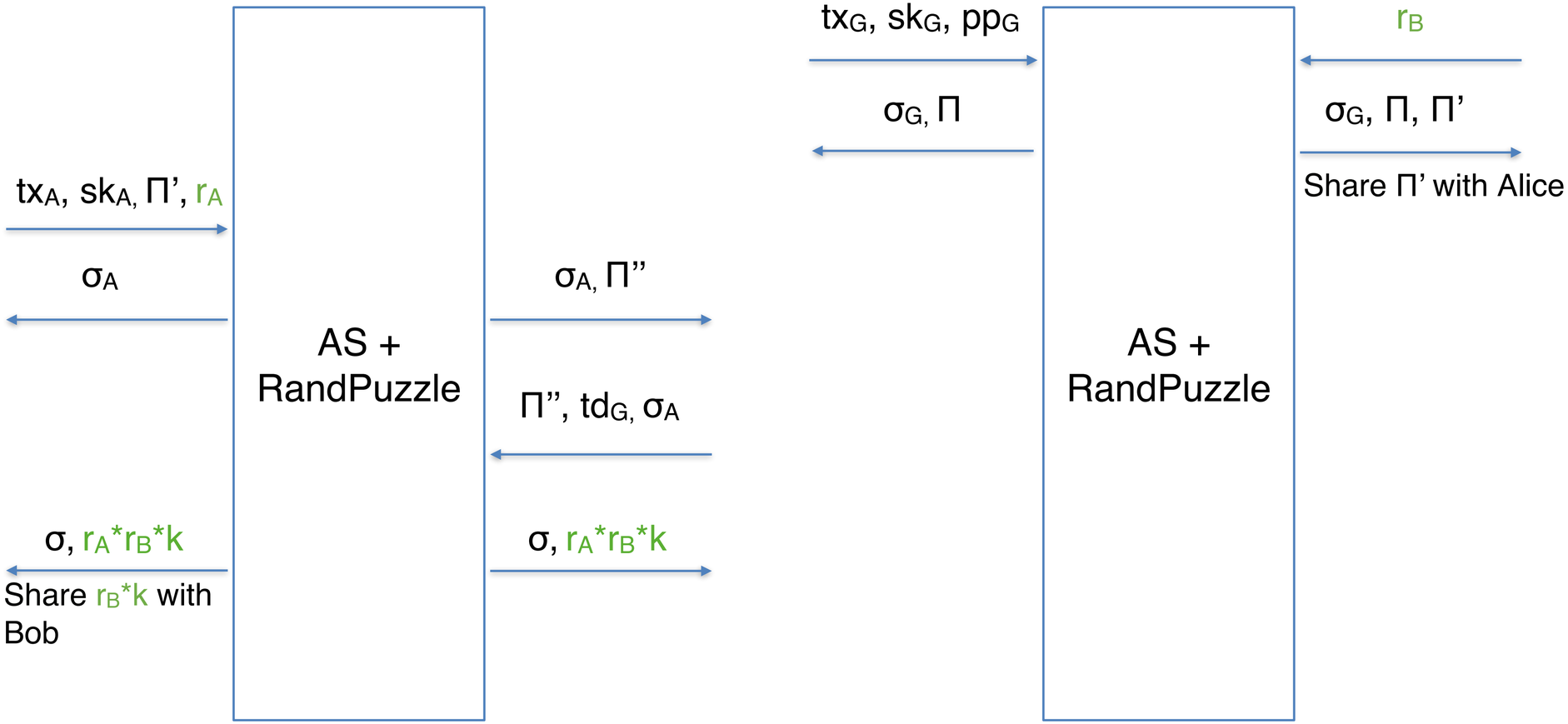
$C'' = r_A * r_B * k * G$

$C = k * G$

(pk_A, sk_A, r_A)

$(pk_G, sk_G, pp_G, td_G, k)$

r_B



A²L: Protocol Overview

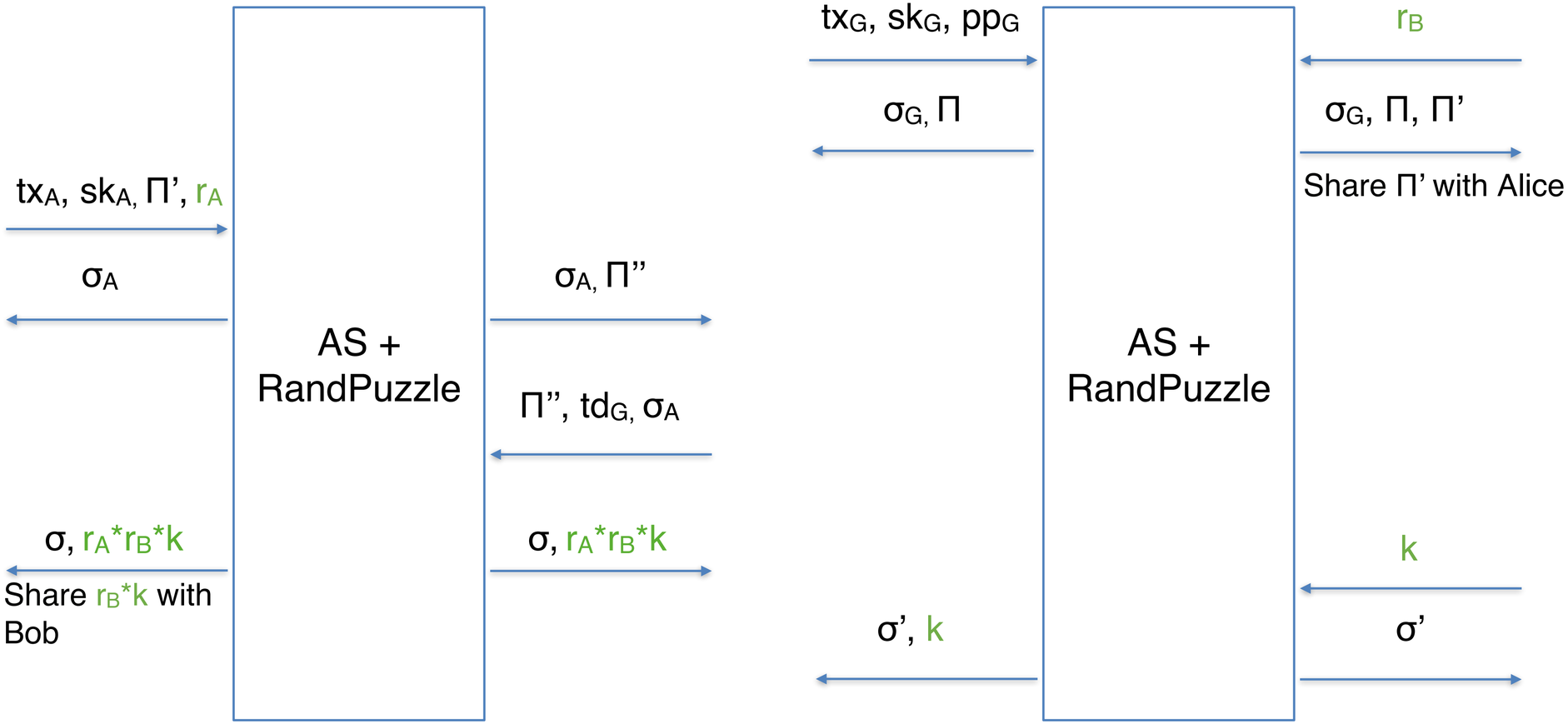
$C'' = r_A * r_B * k * G$

$C = k * G$

(pk_A, sk_A, r_A)

$(pk_G, sk_G, pp_G, td_G, k)$

r_B



A²L: Protocol Overview

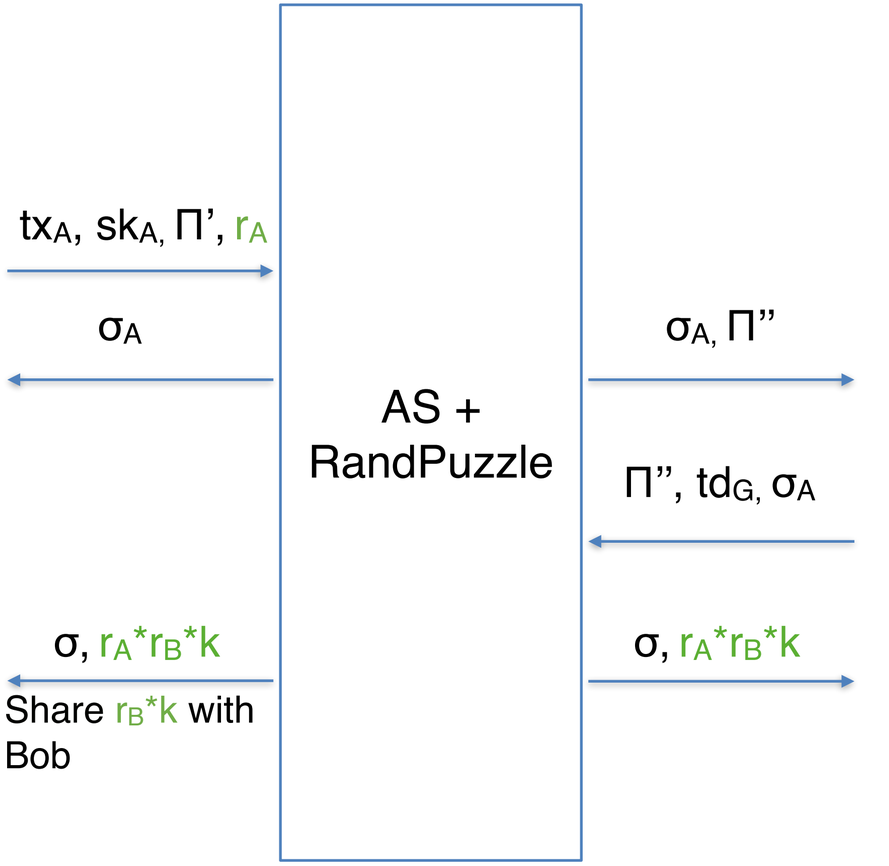
$C'' = r_A * r_B * k * G$

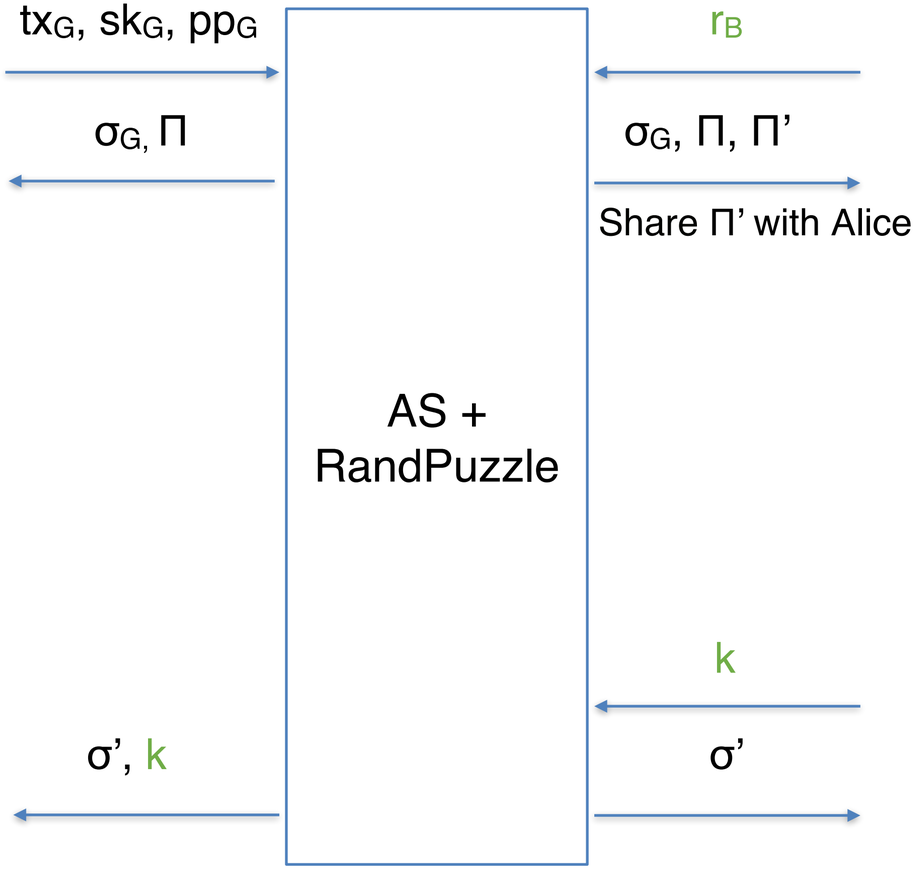
$C = k * G$

(pk_A, sk_A, r_A)

$(pk_G, sk_G, pp_G, td_G, k)$

r_B





A²L: Protocol Overview

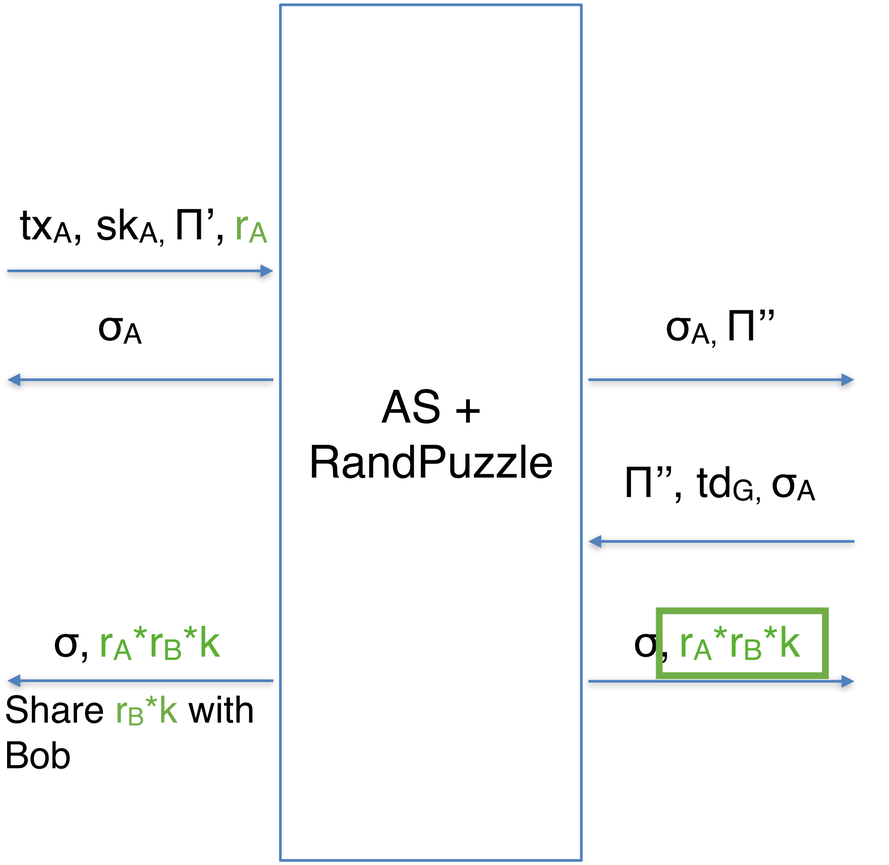
$C'' = r_A * r_B * k * G$

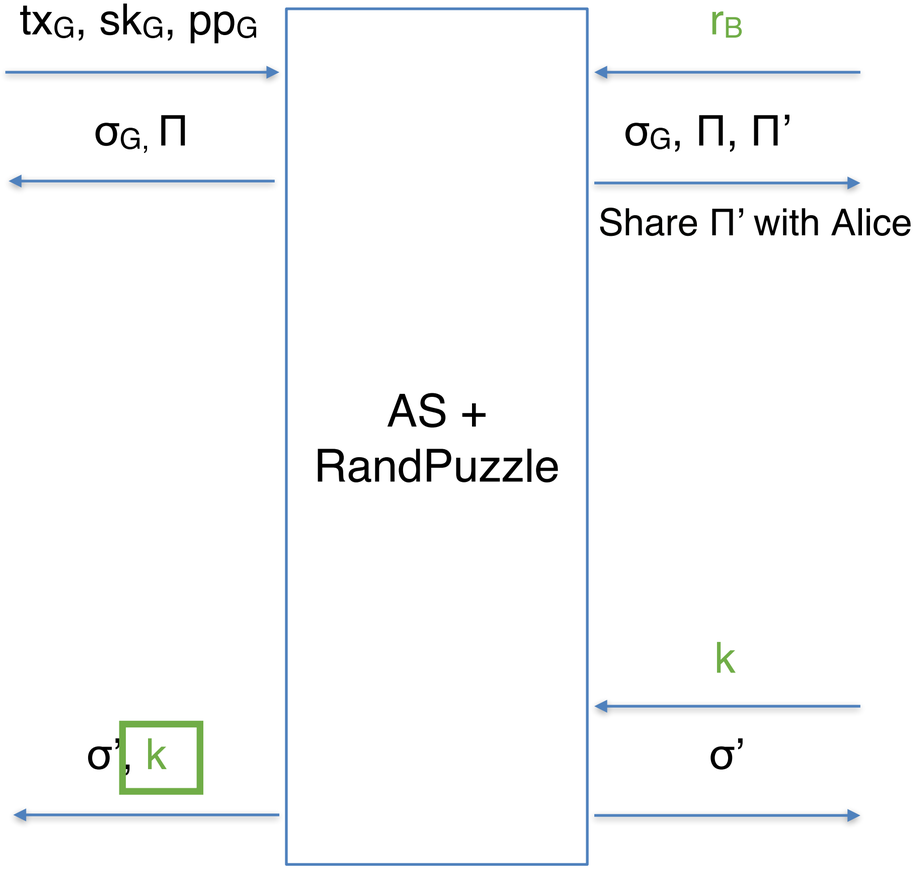
$C = k * G$

(pk_A, sk_A, r_A)

$(pk_G, sk_G, pp_G, td_G, k)$

r_B



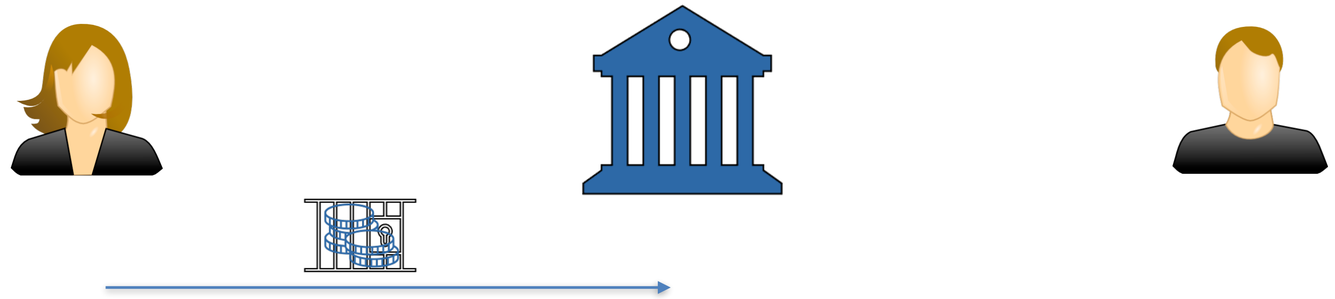


Discussion

- Privacy-preserving registration protocol to protect against griefing attacks

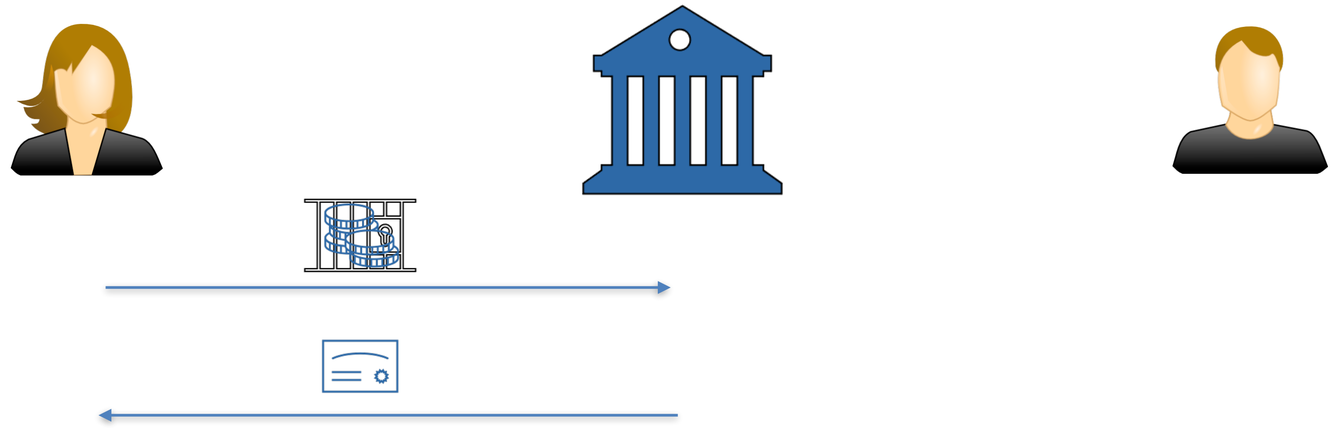
Discussion

- Privacy-preserving registration protocol to protect against griefing attacks



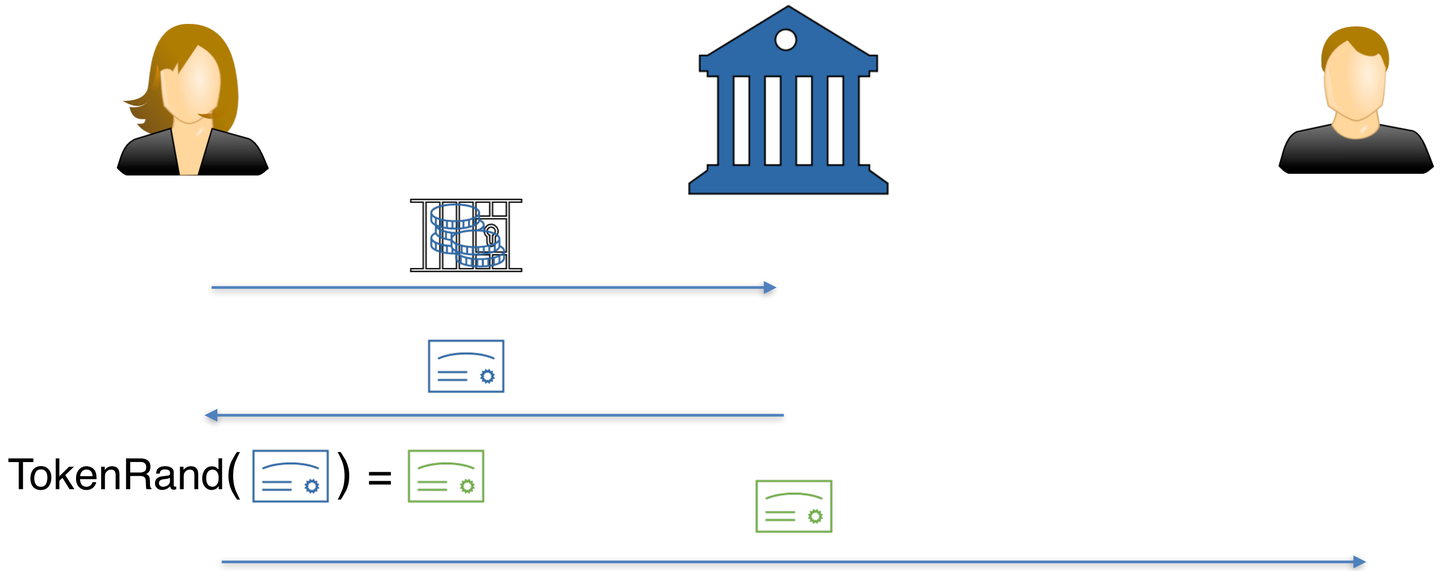
Discussion

- Privacy-preserving registration protocol to protect against griefing attacks



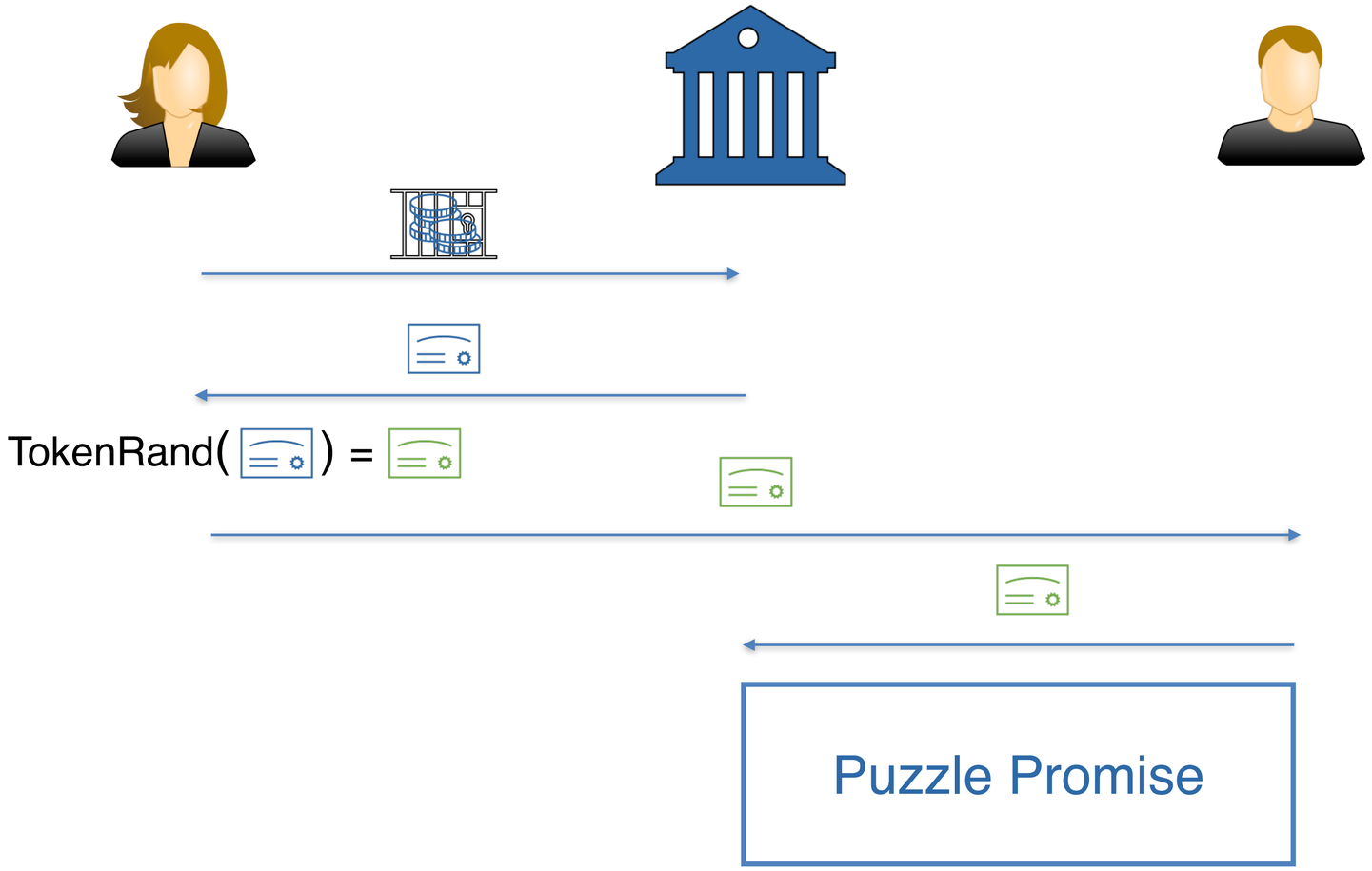
Discussion

- Privacy-preserving registration protocol to protect against griefing attacks



Discussion

- Privacy-preserving registration protocol to protect against griefing attacks



Discussion

- A²L incurs 33x less bandwidth overhead and provides 2x speedup in computation compared to TumbleBit

	Registration	Puzzle Promise	Puzzle Solver	Total
WAN[1]	0.726	1.251	1.076	3.053
LAN	0.008	0.475	0.118	0.601
LAN (pre-processing)	0.008	0.194	0.118	0.320
Bandwidth (KB)	0.30	7.31	2.31	9.92

Performance of A²L instantiated with ECDSA signature. Time shown in seconds.
[1]Payment hub over Amazon AWS machines in Oregon-Frankfurt-Singapore.

Conclusion

- A^2L is a cryptographic protocol for PCHs that achieves security, unlinkability and interoperability

Conclusion

- A^2L is a cryptographic protocol for PCHs that achieves security, unlinkability and interoperability
- Formally specified and proven secure (in the UC framework)

Conclusion

- A^2L is a cryptographic protocol for PCHs that achieves security, unlinkability and interoperability
- Formally specified and proven secure (in the UC framework)
- Advantages:
  - Fully backwards compatible with Bitcoin, and scriptless cryptocurrencies (e.g., Monero)
  - Protection against griefing attacks
  - The most efficient Bitcoin-compatible PCH

Conclusion

- A^2L is a cryptographic protocol for PCHs that achieves security, unlinkability and interoperability
- Formally specified and proven secure (in the UC framework)
- Advantages:
  - Fully backwards compatible with Bitcoin, and scriptless cryptocurrencies (e.g., Monero)
  - Protection against griefing attacks
  - The most efficient Bitcoin-compatible PCH
- Paper available at https://eprint.iacr.org/2019/589.pdf
- Our C implementation available at https://github.com/etairi/A2L
- Rust implementation by COMIT Network available at https://github.com/comit-network/a2l-poc

Conclusion

- A²L is a cryptographic protocol for PCHs that achieves security, unlinkability and interoperability

- Formally specified and proven secure (in the UC framework)

- Advantages:
  - Fully backwards compatible with Bitcoin, and scriptless cryptocurrencies (e.g., Monero)
  - Protection against griefing attacks
  - The most efficient Bitcoin-compatible PCH

- Paper available at https://eprint.iacr.org/2019/589.pdf
- Our C implementation available at https://github.com/etairi/A2L
- Rust implementation by COMIT Network available at https://github.com/comit-network/a2l-poc

Thank You!

@erkantairi