UNIVERSITY OF TRENTO

Complexity, Cryptography, and Financial Technologies

Lecture 3 – Cryptocurrency Examples

Chan Nam Ngo

Examples of operational PTN

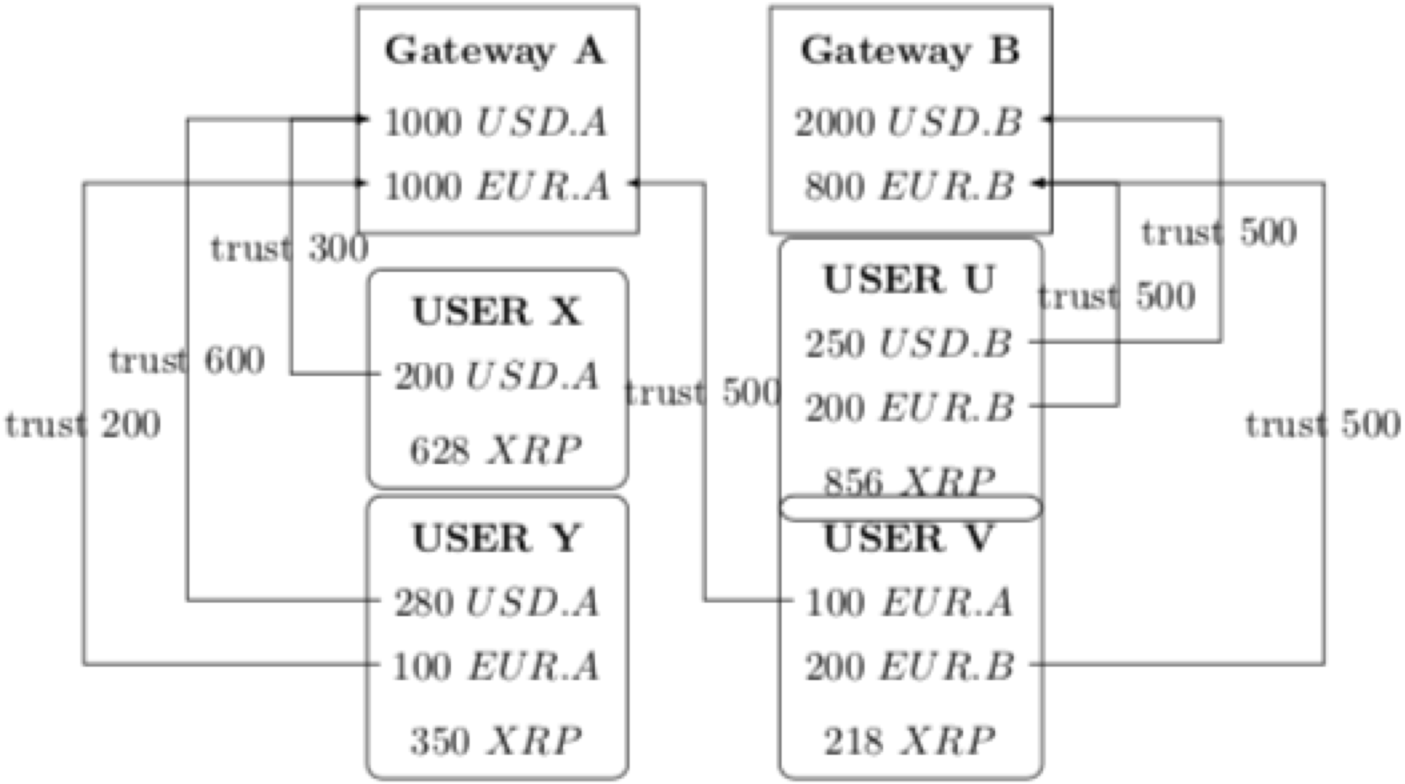
- **Ripple (https://ripple.com)**
  - Hybrid PTN
    - CAs for Value Creation
    - Brokers for Transaction Fulfillment
  - Supports multiple currencies
    - Including the native cryptocurrency XRP
  - History
    - https://en.wikipedia.org/wiki/Ripple_(payment_protocol)
- **ZeroCash (ZCash - https://z.cash)**
  - Decentralized PTN
  - Privacy-preserving cryptocurrency

Ripple Network Components

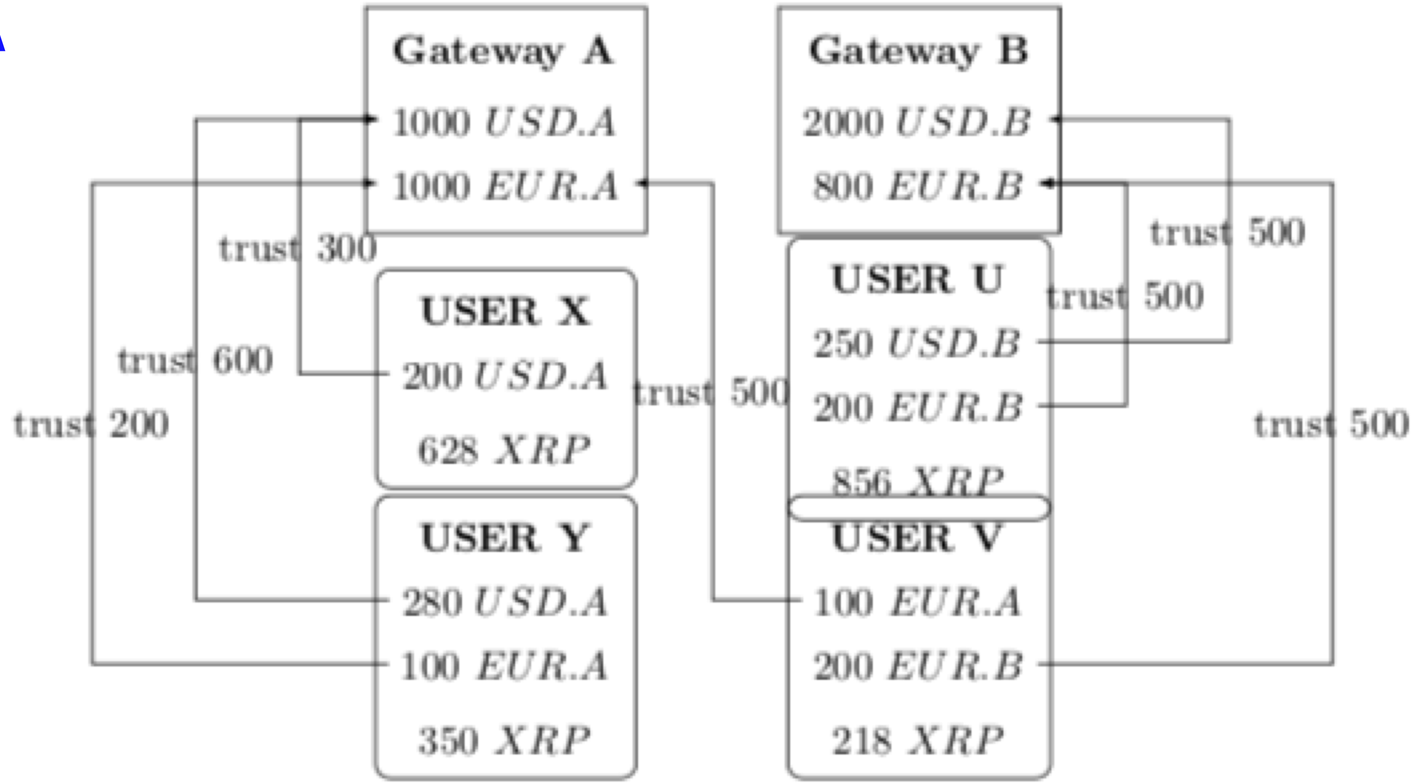
- **Gateways**
  - Create “issuances” that represent transferable value
  - Issuances are digital assets bound to gateways
    - Gateway A can create “USD.A” that corresponds to the USD currency that is issued by A
- **Brokers**
  - Validate and fulfill transactions
- **Users**
  - Out-of-band deposit the fiat money into the gateways in exchange for issuances
  - Declare ”trust-lines” to receive payments in issuances
    - “I accept at most 1000 USD.A”

Ripple Network Components - Gateways

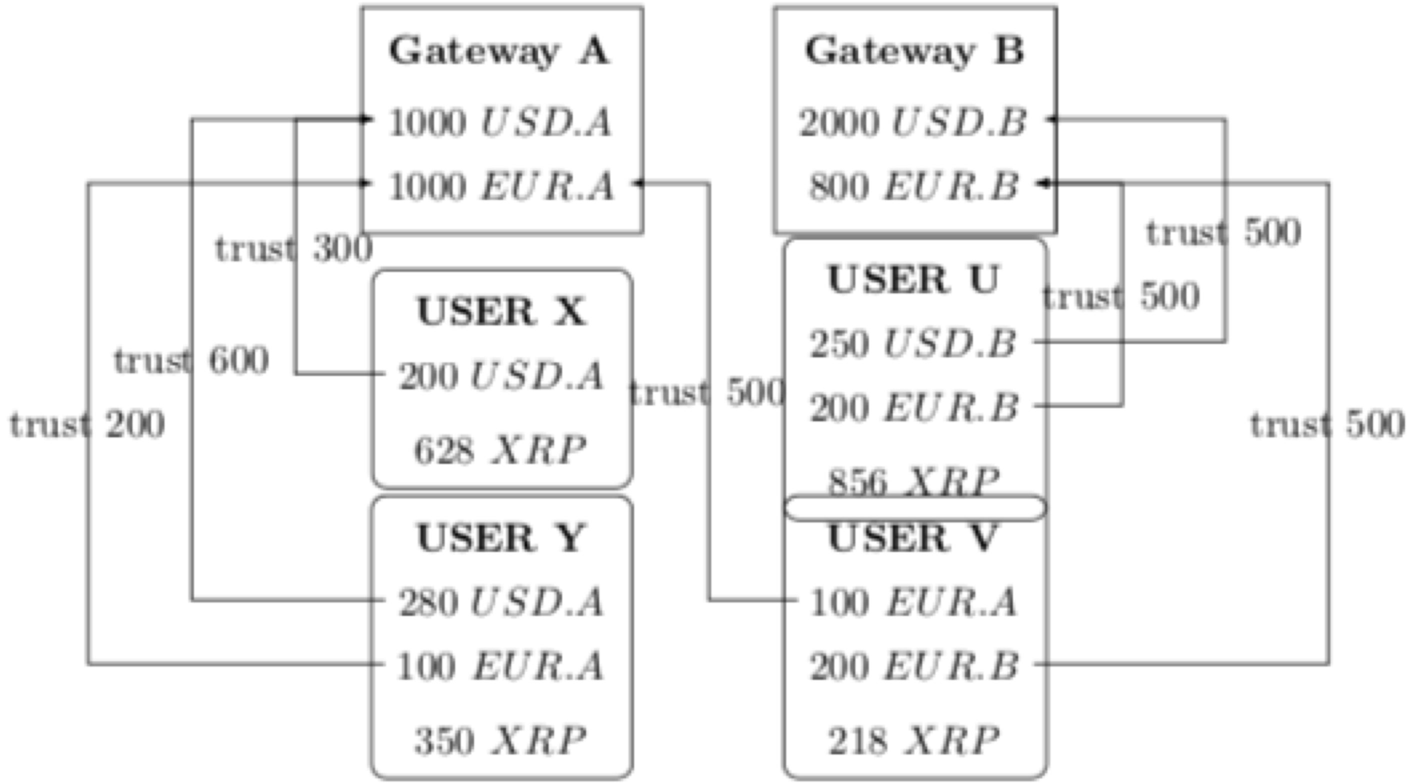
- **Gateway A issues 1000 USD.A and 1000 EUR.A**
- **Gateway B issues 2000 USD.B and 800 EUR.B**



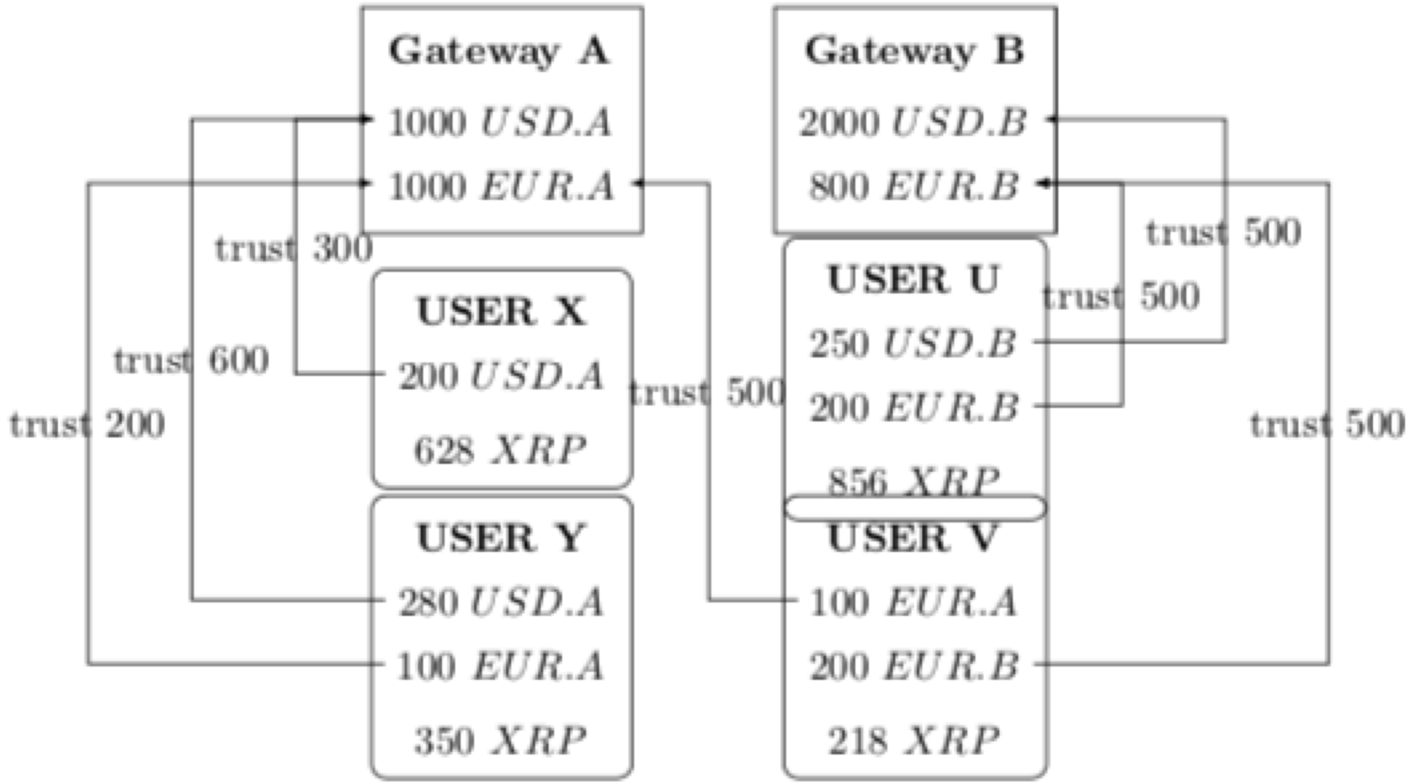
- **User X accepts at most 300 USD.A**
- **User Y accepts at most 600 USD.A and 200 EUR.A**



- X has 200 USD.A, wants to pay 100 USD to Y
- X can just send 100 USD.A to Y

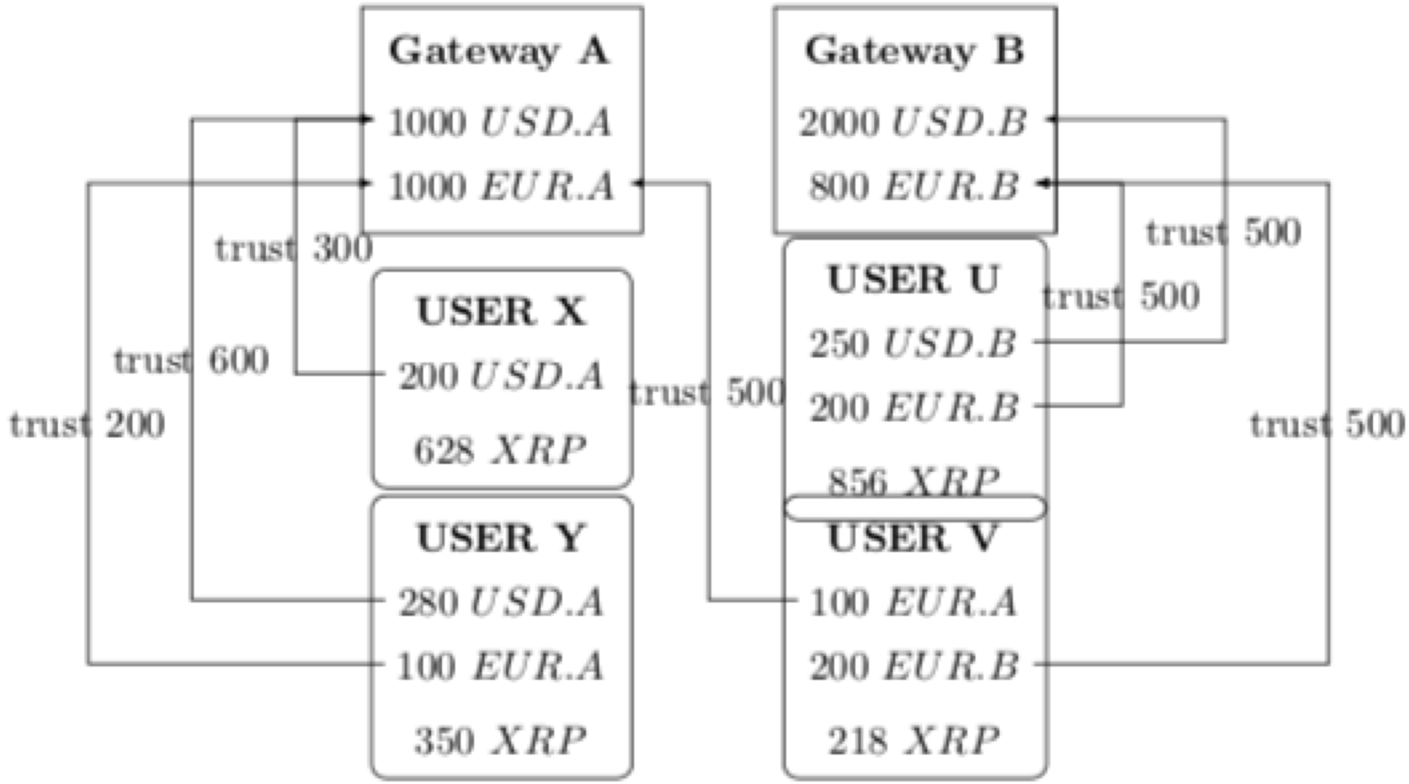


- User V accepts at most 500 EUR.A and 500 EUR.B

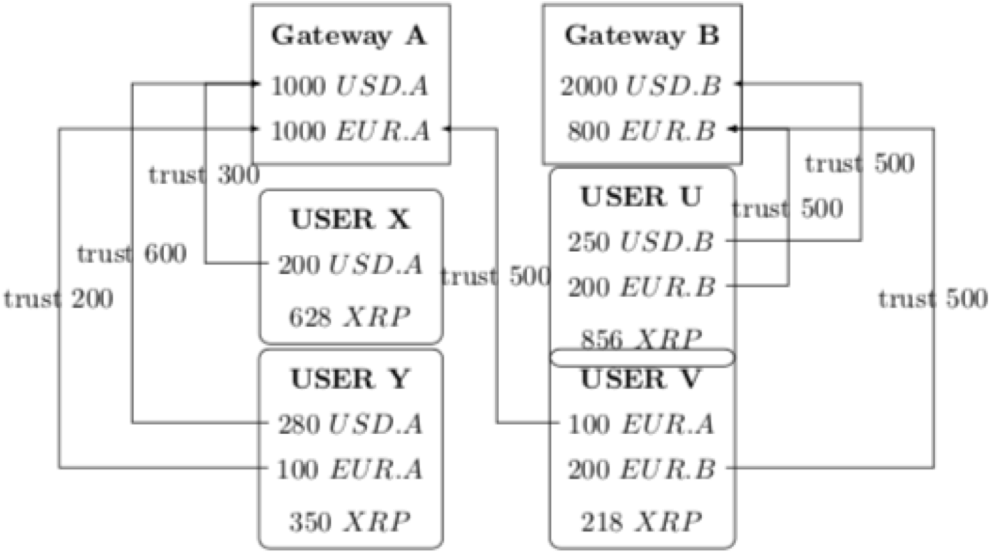


UNIVERSITY OF TRENTO

- **Y has 100 EUR.A, wants to pay V 50 Euros**
- **Y can just send 50 EUR.A to V**



- **Y wants to pay U 50 Euros**
  - U doesn't accept USD.A or EUR.A
- **V must act as a "middle-man"**
  - Y sends 50 EUR.A to V
  - V sends 50 EUR.B to U
- **Further complications**
  - https://developers.ripple.com/paths.html



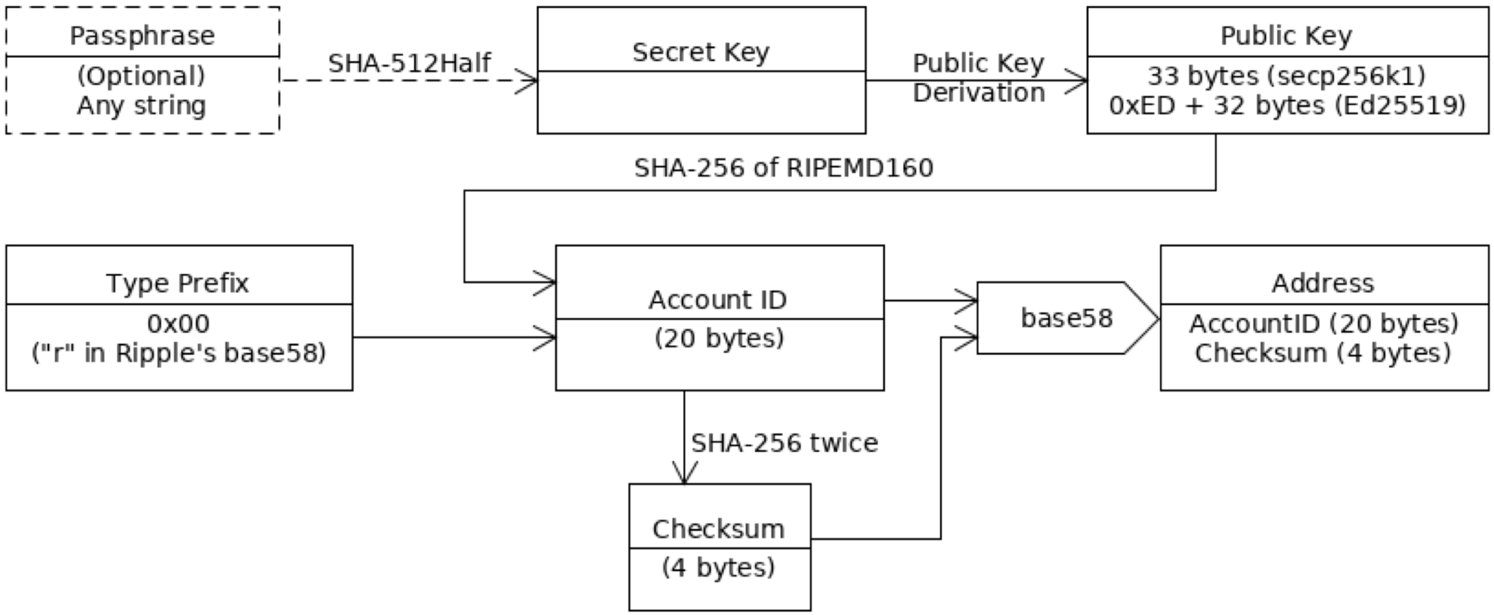
Digital Signature for Authenticity

UNIVERSITY OF TRENTO

- **Key Generation**
  - (vk, sk) $\leftarrow$ KeyGen()
    - vk is called the (public) verifying key
    - sk is called the (private) signing key
- **Signature Generation**
  - s $\leftarrow$ Sign(sk,m)
    - m is the message to be signed
    - s is the signature on m
- **Signature Verification**
  - {0,1} $\leftarrow$ Verify(vk,s,m)

Payment Address from Verifying Key

- **address = AdressGen(vk)**
- **Requires SHA-256, RIPEMD160, and base58**



Source: https://developers.ripple.com/accounts.html#address-encoding

UNIVERSITY OF TRENTO

- **Practical Byzantine Fault Tolerant**
  - Technical details → Distributed Systems 2
  - We only use it as a black-box for now
- **DL is an ordered list that is**
  - Distributed – many nodes have the same copy
  - Append-Only – can only insert into the end of the list
    - No delete, no update
- **For simplicity we assume**
  - no conflict nor disagreement
  - no transaction fee for Brokers

Setup – DL and Keys

- **Brokers**
  - Initialize an empty DL
- **Gateway G**
  - $(vk_g, sk_g) \leftarrow \text{KeyGen}()$
  - $addr_g = \text{AddressGen}(vk_g)$
  - Broadcasts vk_g and $addr_g$
- **User U**
  - $(vk_u, sk_u) \leftarrow \text{KeyGen}()$
  - $addr_u = \text{AddressGen}(vk_u)$
  - Receives then stores vk_g and $addr_g$

Setup – trust-lines

- **User U**
  - Creates a trust-line
    - tl = ($addr_u$, $addr_g$, limit, currency)
  - tls ← Sign(sk_u, tl)
  - Sends tl and tls to Brokers
- **Brokers**
  - Receives tl and tls from U
  - b ← Verify(vk_u, tls, tl)
  - If b = 1, add tl into DL

Setup - deposit

- **User U**
  - Out-of-band deposits x USD (or another currency) into G
  - Sends $addr_u$ to G
- **Gateway G**
  - Receives $addr_u$ from U
  - Creates an "issued" transaction
    - it = ($addr_g$, $addr_u$, x, USD)
  - its $\leftarrow$ Sign(sk_g, it)
  - Sends it and its to Brokers
- **Brokers**
  - Receives its and it from G
  - b $\leftarrow$ Verify(vk_g, its, it)
  - If b = 1, cont.
  - look into the DL to see if there is a trust-line l =($addr_u$, $addr_g$, limit > x, USD)
  - if YES, add it into DL

Ripple Payment

- **Payee V**
  - We assume V has done the Setup and added a trust-line on USD.G
  - Sends $addr_v$ and amount p to U
- **Payer U**
  - Receives $addr_v$ and amount p from V
  - Creates a transaction
    - $t = (addr_u, addr_v, p, USD.G)$
  - ts $\leftarrow$ Sign(sk_g, i)
  - Sends t and ts to Brokers
- **Brokers**
  - Receives t and ts from U
  - b $\leftarrow$ Verify(vk_u, ts, t)
  - If b = 1, cont.
  - Look through the transaction history of U on DL to see if U has more than p USD.G
  - look into the DL to see if there is a trust-line tl = ($addr_v$, $addr_g$, limit > p, USD.G)
  - if YES, add t into DL
  - Any check fails, do nothing

Exercise time!!!

- Identify the steps that are relevant to the 4 high-level conceptual steps
- Identify the steps that mitigate the threats
- Let’s go back and see the protocol again

Value Creation

- **User U**
  - Out-of-band deposits x USD (or another currency) into G
  - Sends $addr_u$ to G
- **Gateway G**
  - Receives $addr_u$ from U
  - Creates an "issued" transaction
    - it = ($addr_g$, $addr_u$, x, USD)
  - its ← Sign(sk_g, it)
  - Sends it and its to Brokers
- **Brokers**
  - Receives its and it from G
  - b ← Verify(vk_g, its, it)
  - If b = 1, cont.
  - look into the DL to see if there is a trust-line l =($addr_u$, $addr_g$, limit > x, USD)
  - if YES, add it into DL

Payment Promise

- **Payee V**
  - We assume V has done the Setup and added a trust-line on USD.G
  - Sends $addr_v$ and amount p to U
- **Payer U**
  - Receives $addr_v$ and amount p from V
  - Creates a transaction
    - t = ($addr_u$, $addr_v$, p, USD.G)
  - ts ← Sign(sk_g, i)
  - Sends t and ts to Brokers
- **Brokers**
  - Receives t and ts from U
  - b ← Verify(vk_u, ts, t)
  - If b = 1, cont.
  - Look through the transaction history of U on DL to see if U has more than p USD.G
  - look into the DL to see if there is a trust-line tl = ($addr_v$, $addr_g$, limit > p, USD.G)
  - if YES, add t into DL
  - Any check fails, do nothing

Transaction Fulfillment

- **Payee V**
  - We assume V has done the Setup and added a trust-line on USD.G
  - Sends $addr_v$ and amount p to U
- **Payer U**
  - Receives $addr_v$ and amount p from V
  - Creates a transaction
    - $t = (addr_u, addr_v, p, USD.G)$
  - ts ← Sign(sk_g, i)
  - Sends t and ts to Brokers
- **Brokers**
  - Receives t and ts from U
  - b ← Verify(vk_u, ts, t)
  - If b = 1, cont.
  - Look through the transaction history of U on DL to see if U has more than p USD.G
  - look into the DL to see if there is a trust-line tl = ($addr_v$, $addr_g$, limit > p, USD.G)
  - if YES, add t into DL
  - Any check fails, do nothing

Recap: mapping Ripple protocol into High-level Steps

- **Creation of Value**
  - Payer out-of-band deposits into Gateway
  - Payer/Payee creates and sends a signed trust-line tl to Brokers
  - Gateway creates and sends a signed issued transaction it to Brokers
  - Brokers validate and accepts the signed trust-line from Payer/Payee and the signed issued transaction from Gateway
- **Promise of Payment**
  - Payee sends payment address and amount to Payer
  - Payer creates and sends a signed transaction t to Brokers

Recap: mapping Ripple protocol into High-level Steps (2)

- **Fulfillment of Transactions**
  - Brokers validate the payment transactions
    - A payment transaction must be signed by the Payer and the DL history shows that the Payer owns more than the transacted amount
    - Payee must have a trustline for the incoming issuances
  - then add them into DL
- **Preservation of Value**
  - Gateway and Payer/Payee stores the keypairs and payment addresses
  - Brokers store the trust-line history (in DL)
  - Brokers store the transaction history (in DL)

Over-drafting countermeasure

UNIVERSITY OF TRENTO

- **Payee V**
  - We assume V has done the Setup and added a trust-line on USD.G
  - Sends $addr_v$ and amount p to U
- **Payer U**
  - Receives $addr_v$ and amount p from V
  - Creates a transaction
    - $t = (addr_u, addr_v, p, USD.G)$
  - $ts \leftarrow Sign(sk_g, i)$
  - Sends t and ts to Brokers
- **Brokers**
  - Receives t and ts from U
  - $b \leftarrow Verify(vk_u, ts, t)$
  - If b = 1, cont.
  - Look through the transaction history of U on DL to see if U has more than p USD.G
  - look into the DL to see if there is a trust-line tl = $(addr_v, addr_g, limit > p, USD.G)$
  - if YES, add t into DL
  - Any check fails, do nothing

Countermeasures for Ripple Integrity

- **Over-Drafting**
  - Brokers look through the transaction history to determine current available fund and check against the transacted amount
  - Brokers look into the trust-line history to determine the current limit for incoming issuances
- **Double-Spending**
  - N/A
- **Unauthorized-Spending**
  - Brokers check the Payer signature when validating the transactions
- **Individual Loss**
  - N/A
- **Systemic Loss**
  - N/A

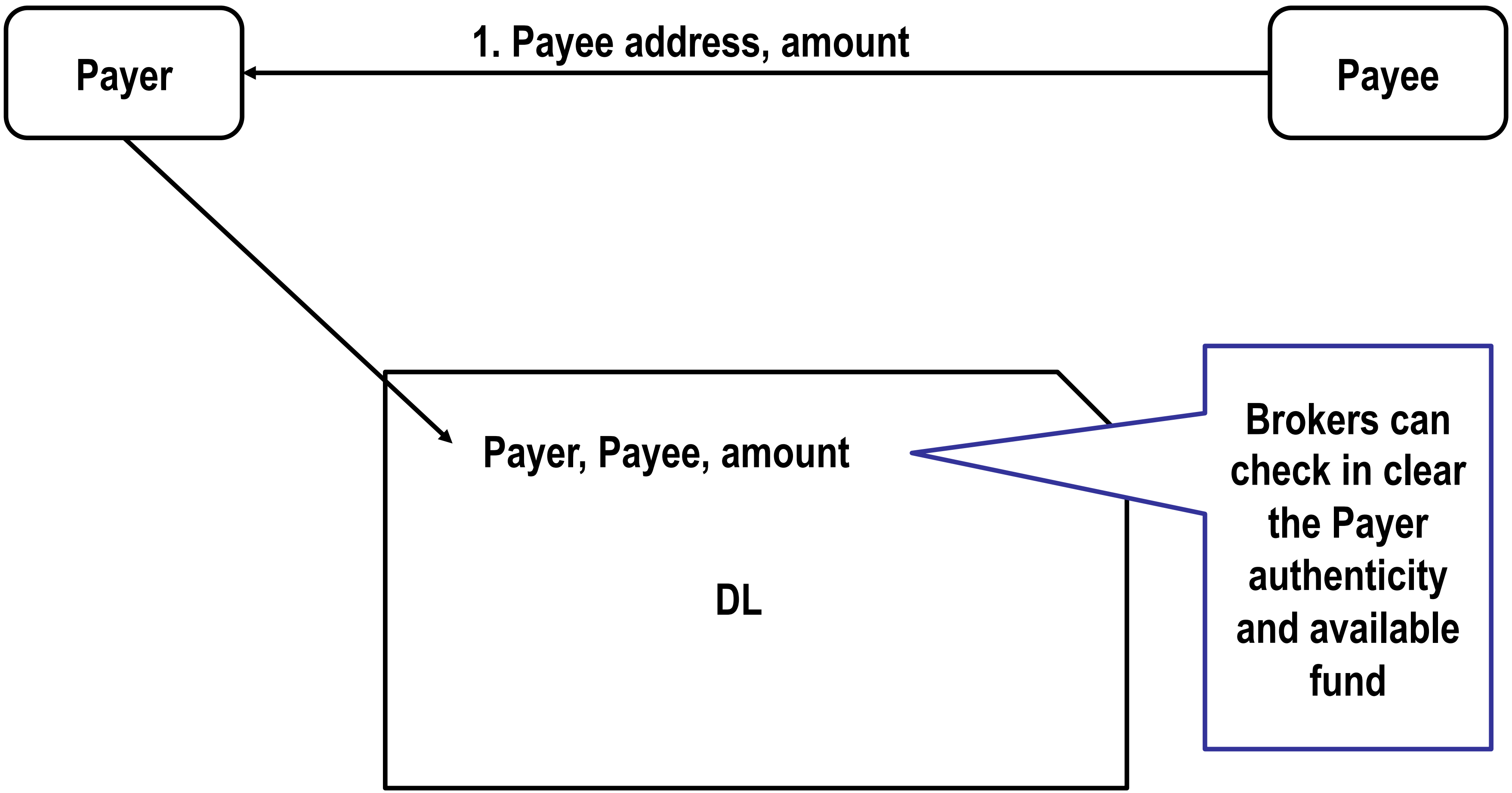
Confidentiality vs Anonymity

UNIVERSITY OF TRENTO

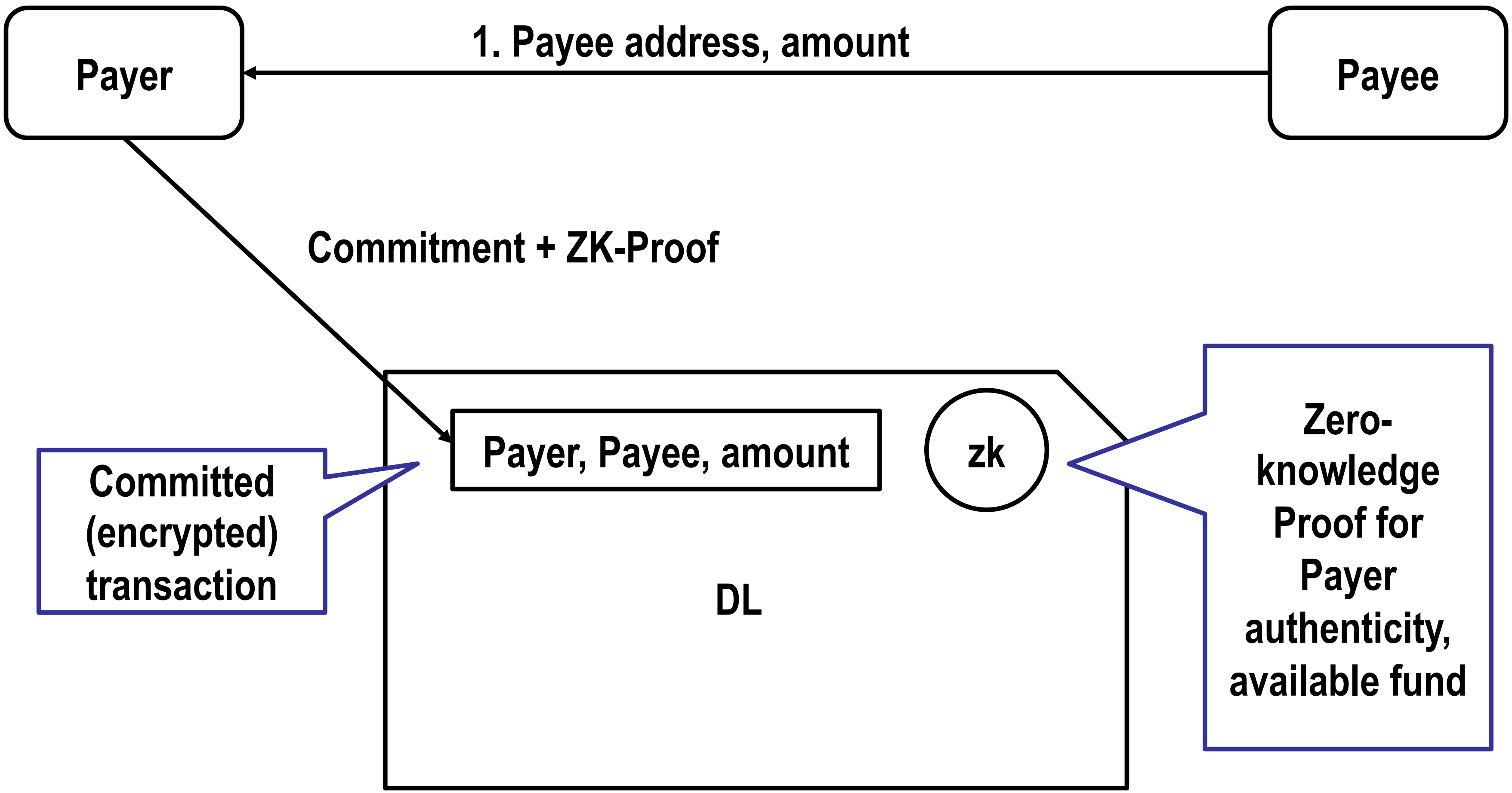
- **Involves answering three questions**
  - Instantaneous Networth
    - At time t, can we identify the total value v of a nominal identity I? YES
  - Transient Value
    - At time t, can we know about a transaction of value v between two nominal identities I_1 and I_2? YES
  - Persistent Identity
    - Can we link two nominal identities I_1 at time t and I_2 at time t'? YES
- **There is no Confidentiality nor Anonymity in Ripple**

- **ZeroCash**
  - is a decentralized PTN
  - that offers privacy of transactions
    - ZeroCash payments are published on a public blockchain,
    - but the sender, recipient, and amount of a transaction remain private.
- **Technology**
  - DL
  - Commitment Scheme
  - Zero-knowledge Proof
- **We only consider a simplified version of ZeroCash due to its very high technical complexity**

Ripple – High-level Overview



ZeroCash – High-level Overview



UNIVERSITY OF TRENTO

- **Commitment scheme**
  - Secret value v
  - Randomness r
  - Commitment c = Com(v;r)
- **Security Properties**
  - Data hiding
    - Cannot know v upon seeing c
  - Data binding
    - Cannot find (v'; r') $\neq$ (v;r) s.t. c = com(v';r')

- **Zero-knowledge Proof**
  - allows a Prover to convince a Verifier that a statement is true without leaking any information beyond that fact.
- **3 algorithms**
  - crs ← ZSetup(R)
    - crs is called the Common Reference String
    - R is called the relation
      - R(st,w) = 1 means “st is true”
      - st is called the statement
      - w is called the witness
  - π ← ZProve(crs, st, w)
    - π is the zk-proof
  - {0,1} ← ZVerify(crs, st, π)

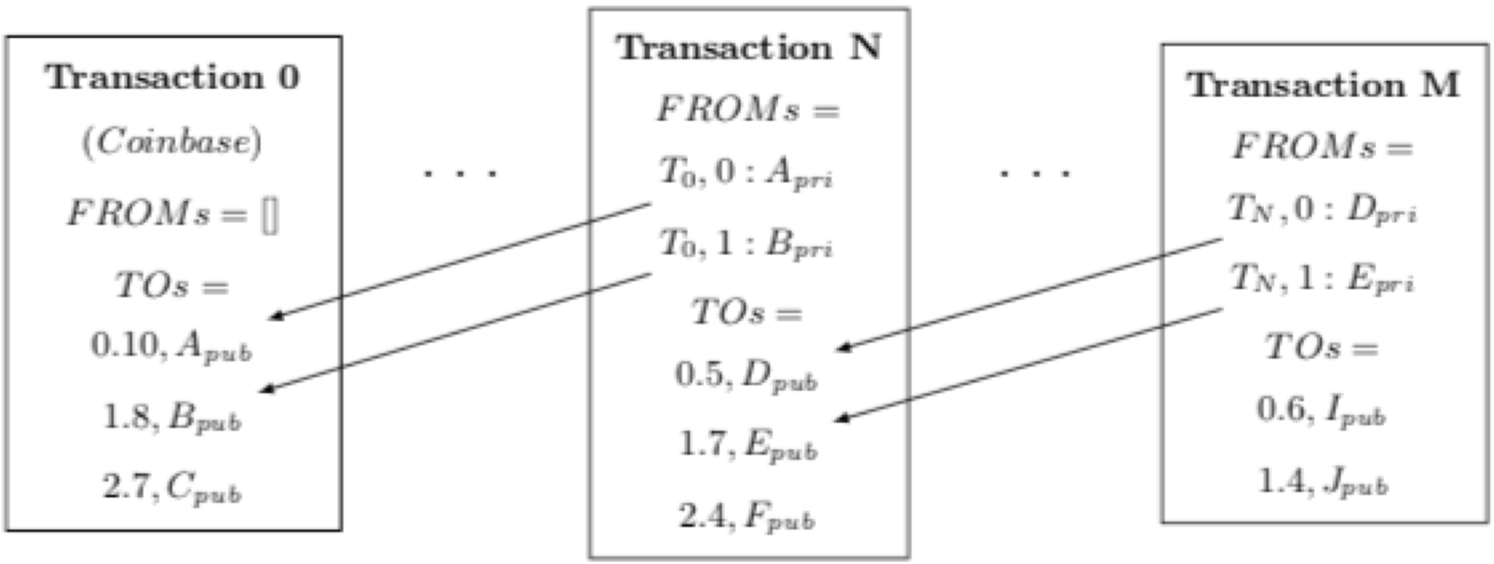
A simple Commitment and ZK-Proof example

- **Commitment**
  - Prover
    - Randomly selects v and r
    - c = Com(v;r)
    - Sends c to the Verifier
  - Verifier
    - Receives and stores c
- **Now Prover wants to convince Verifier that c is the commitment of a value v such that v > 0**

- **ZK-Proof**
  - Verifier
    - R(c,(v;r)) means
      - c = Com(v;r)
      - v > 0
    - crs ← ZSetup(R)
    - Sends crs to Prover
    - Receives π from Prover
    - b ← ZVerify(crs, c, π)
    - accepts π if b = 1
  - Prover
    - Receives crs from Verifier
    - π ← ZProve(crs,c,(v;r))
    - Sends π to Verifier

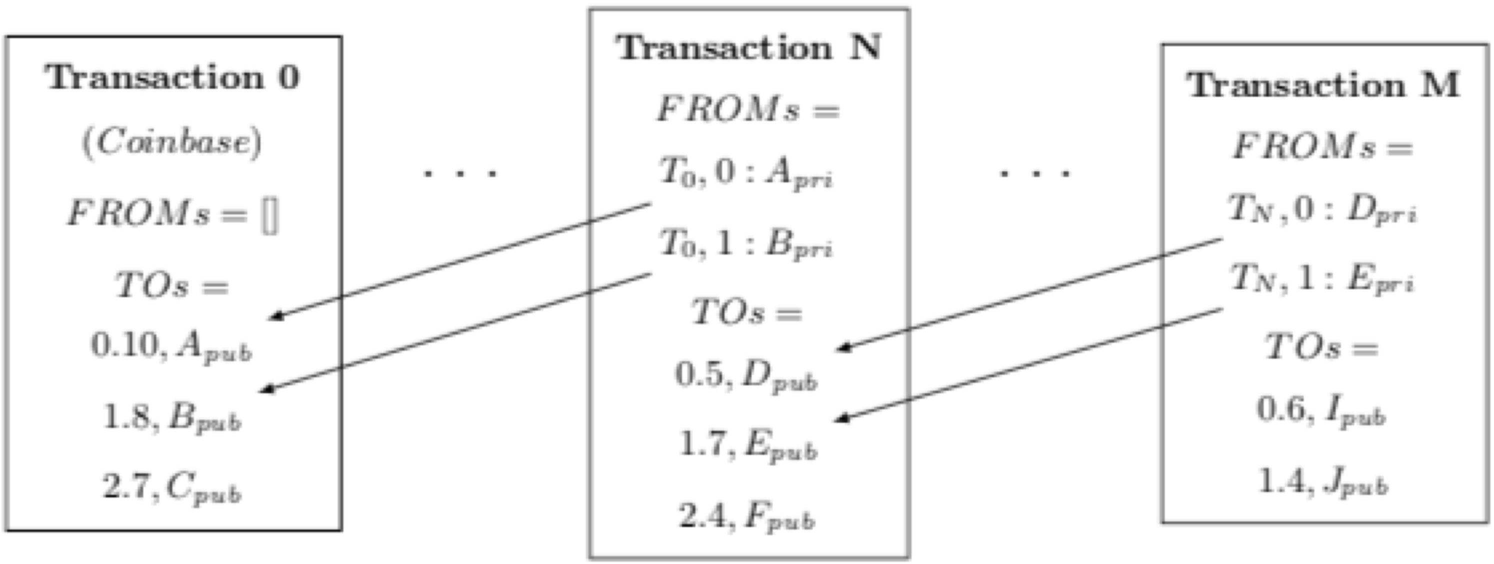
ZeroCash Transaction

- **A coinbase transaction has no input coin but requires a Proof-of-Work PoW**
- **A regular transaction needs unspent input coins**
- **Output coins ≤ (unspent) input coins (or PoW)**

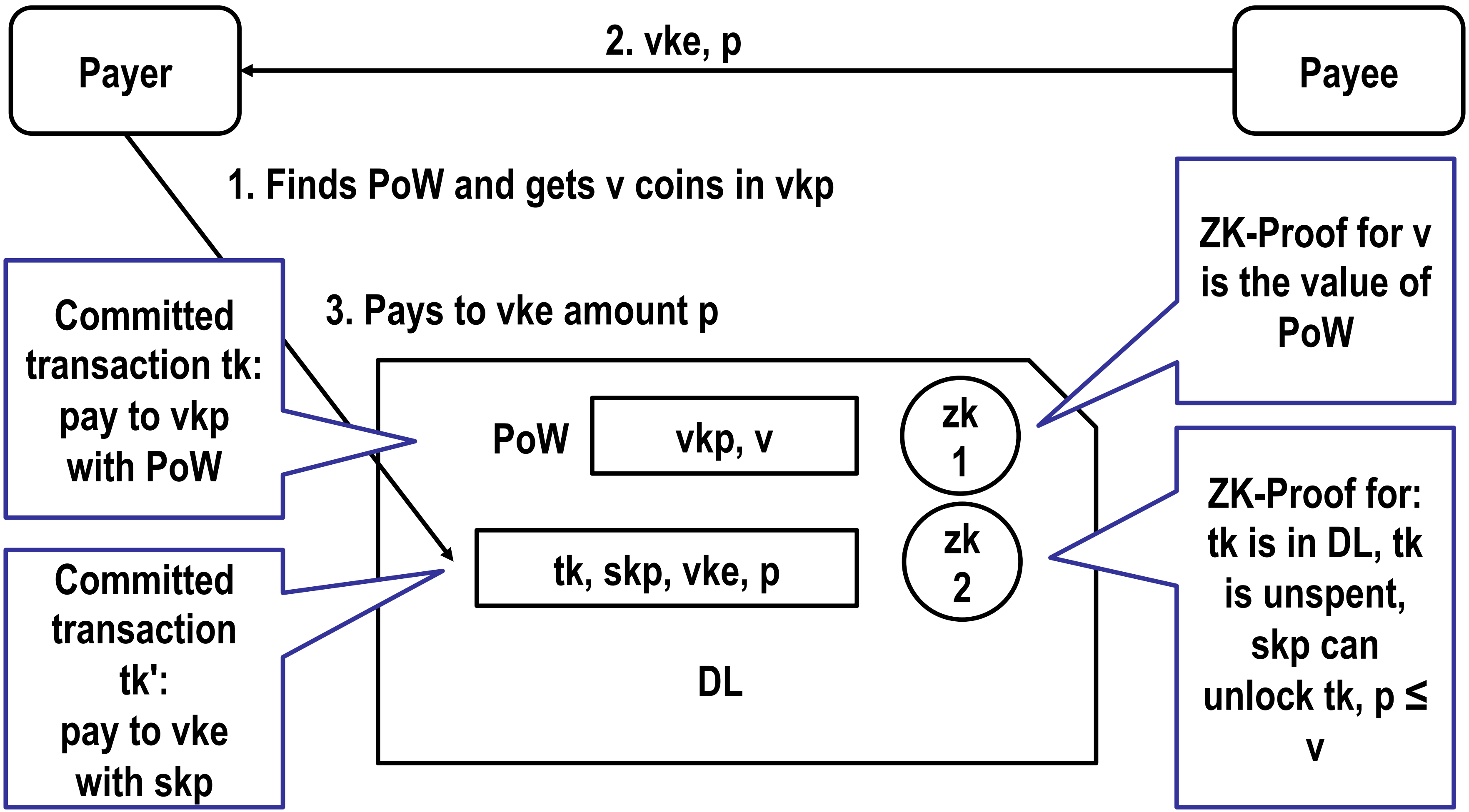


ZeroCash Transaction (2)

- **PoW worths 25 coins**
- **A, B, C gets 10, 8, 7 coins from PoW**
- **D, E, F gets 5, 7, 4 coins from A and B**
- **I, J gets 6, 4 coins from D, E**



ZeroCash – High-level Overview (2)



ZeroCash - Setup

- **Brokers**
  - initialize an empty DL
  - crs ← ZSetup(R)
    - The ZCash ceremony https://youtu.be/D6dY-3x3teM
    - R(st,w) for a transaction tk' that takes tk as input
      - st = DL, (PoW), tk'
      - w = tk, skp, vke, p, r'
      - R(st,w) = 1 requires
        - » tk is in DL and tk's output coins are unspent
        - » skp is the signing key that is required to spend tk's output coins
        - » p ≤ tk's output coins value
        - » tk' = Com((tk, skp, vke, p);r')
      - requires PoW in st in case of coinbase transactions
        - » p ≤ PoW's value
        - » ...
  - broadcast the crs
- **Payer/Payee**
  - Store the crs

ZeroCash – Coinbase Transaction

- **Payer**
  - Out-of-band finds the PoW
  - $(vk_p, sk_p) \leftarrow$ KeyGen()
  - Creates a coinbase transaction
    - Supposed a PoW worths v coins
    - $ct = (PoW, vk_p, v)$
  - Commits to ct
    - $tk = Com(ct;r)$
  - $\pi \leftarrow ZProve(crs,(DL, tk, PoW),(vk_p,v;r))$
  - Sends PoW, tk and π to Brokers
- **Brokers**
  - Receives PoW, tk and π from Prover
  - $b \leftarrow ZVerify(crs, (DL, tk, PoW), \pi)$
  - accepts π if $b = 1$ and add (tk, π) into DL

ZeroCash – Regular Transaction

- **Payee**
  - $(vk_e, sk_e) \leftarrow$ KeyGen()
  - Sends vk_e and amount p to Payer
    - Supposed p = v
- **Payer**
  - Receives vk_e and p from Payee
  - Creates a regular transaction
    - $t = (tk, sk_p, vk_e, p)$
  - Commits to t
    - $tk' = Com(t;r')$
  - $\pi' \leftarrow ZProve(crs,(DL,tk'),(tk,sk_p,vk_e,p;r'))$
  - Sends tk' and π' to Brokers
- **Brokers**
  - Receives tk' and π' from Prover
  - $b \leftarrow ZVerify(crs, (DL, tk'), \pi)$
  - accepts π if b = 1 and add (tk', π') into DL

ZeroCash – Coinbase Transaction - Confidentiality and Anonymity

- **Payer**
  - Out-of-band finds the PoW
  - $(vk_p, sk_p) \leftarrow$ KeyGen()
  - Creates a coinbase transaction
    - Supposed a PoW worths v coins
    - ct = (PoW, vk_p, v)
  - Commits to ct
    - tk = Com(ct;r)
  - $\pi \leftarrow$ ZProve(crs,(DL, tk, PoW),(vk_p,v;r))
  - Sends PoW, tk and π to Brokers
- **Brokers**
  - Receives PoW, tk and π from Prover
  - $b \leftarrow$ ZVerify(crs, (DL, tk, PoW), π)
  - accepts π if b = 1 and add (tk, π) into DL

Brokers only see tk but not ct

ZeroCash – Regular Transaction - Confidentiality and Anonymity

UNIVERSITY OF TRENTO

- **Payee**
  - $(vk_e, sk_e) \leftarrow$ KeyGen()
  - Sends vk_e and amount p to Payer
    - Supposed p = v
- **Payer**
  - Receives vk_e and p from Payee
  - Creates a regular transaction
    - $t = (tk, sk_p, vk_e, p)$
  - Commits to t
    - tk' = Com(t;r')
  - $\pi' \leftarrow$ ZProve(crs,(DL,tk'),(tk,sk_p,vk_e,p;r'))
  - Sends tk' and π' to Brokers
- **Brokers**
  - Receives tk' and π' from Prover
  - b $\leftarrow$ ZVerify(crs, (DL, tk'), π)
  - accepts π if b = 1 and add (tk', π') into DL

Brokers only see tk' but not tk and t

Confidentiality vs Anonymity

- **Instantaneous Networth**
  - At time t, can we identify the total value v of a nominal identity I? NO
- **Transient Value**
  - At time t, can we know about a transaction of value v between two nominal identities I_1 and I_2? NO
- **Persistent Identity**
  - Can we link two nominal identities I_1 at time t and I_2 at time t'? NO

Suggested Readings

UNIVERSITY OF TRENTO

- **PTN survey**
  - Massacci, Fabio, Chan-Nam Ngo, and Julian M. Williams. "Decentralized Financial Intermediation Beyond Blockchains." (2018).
- **Ripple**
  - https://developers.ripple.com/trust-lines-and-issuing.html
  - https://developers.ripple.com/paths.html
  - https://developers.ripple.com/transaction-basics.html
  - https://developers.ripple.com/intro-to-consensus.html
  - Schwartz, David, Noah Youngs, and Arthur Britto. "The Ripple protocol consensus algorithm." *Ripple Labs Inc White Paper* 5 (2014).
- **ZeroCash**
  - https://z.cash/technology/index.html
  - Sasson, Eli Ben, et al. "Zerocash: Decentralized anonymous payments from bitcoin." *2014 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2014.