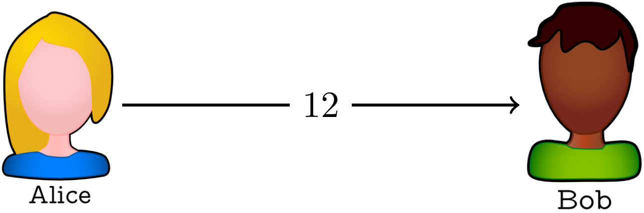
Zef: Low-latency, Scalable, Private Payments

Mathieu Baudet
mathieu.baudet@zefchain.com

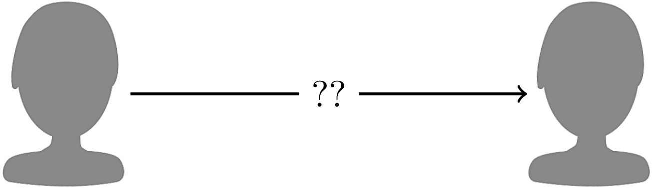
Joint work with Alberto Sonnino, Mahimna Kelkar, and George Danezis

Wire Transfers



Transferring funds between **accounts** with identifiable owners

Anonymous Payments



Transferring funds between **accounts** designated by **addresses**

We want to hide

- account balances and payment amounts (**opacity**)
- the link between sending and receiving addresses (**unlinkability**)

Building Scalable Decentralized Systems

- Faster blockchain (e.g. Solana)
- Blockchain + sharding + Layer 2 (e.g. Ethereum 2.0 with ZK rollups)
- Sidechain with 2/3 honest validators
  - BFT consensus (e.g. Cosmos/Tendermint)
  - BFT consistent broadcast (**this talk**)

Towards Decentralized Anonymous Payments at Scale 1/2

ZCash, Monero

- **anonymous**
- high confirmation time (~30min)
- throughput limited by hardware (perhaps 10..500 TPS)
- blockchain with PoW

FastPay (AFT'2020)

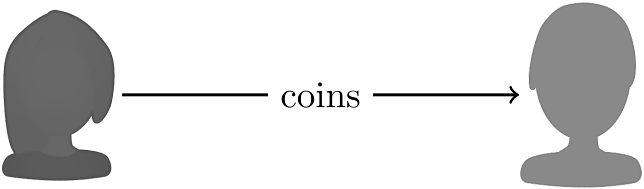
- **linearly scalable**
- quick BFT finality (200ms)
- not anonymous
- sidechain with 2/3 honest validators

Towards Decentralized Anonymous Payments at Scale 2/2

Zef = FastPay + opaque coins + removable accounts

- https://arxiv.org/abs/2201.05671
- Opaque coins are based on the **Coconut scheme** [Sonnino et al. NDSS'19]
- Deleting accounts to optimize (hot) storage requires generating **non-replayable addresses** (aka UIDs)

Anonymous Payments - Opaque coins



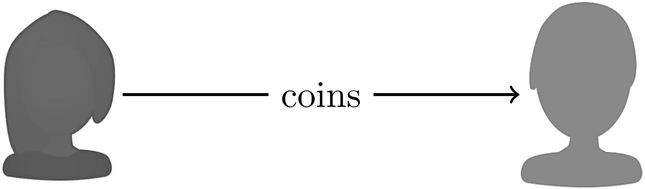
Accounts hold **coins** – whose face values are secret.

Users reveal some of their addresses and keep others secret.

Ok to leak account activity:

- source address of a transfer
- #coins per transfer
- #coins in an account

Anonymous Payments - More Disclaimers

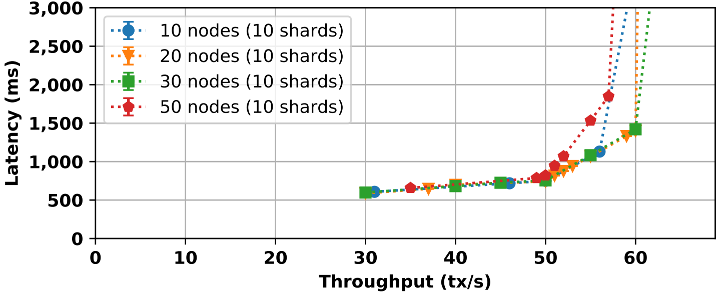


Also probably ok:

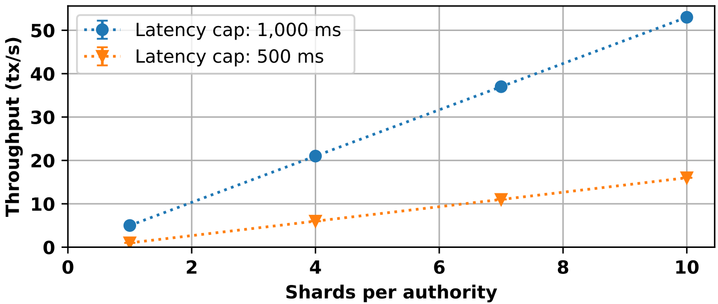
- public fees
- corrupt senders can reveal the addresses of receivers
- a private network (Tor) is needed to operate accounts secretly

Performance of Anonymous Payments with Zef

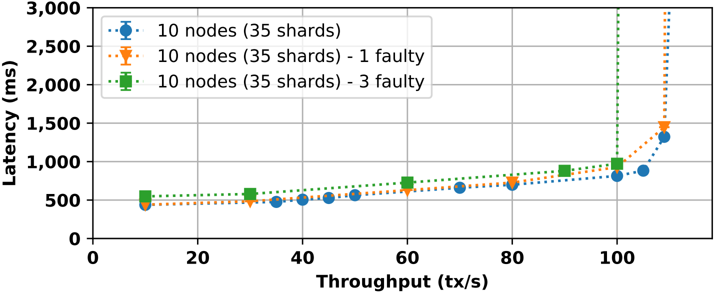
Benchmarks - Latency



Benchmarks - Linear Scalability



Benchmarks - Fault Tolerance



The FastPay Protocol

The FastPay/Zef Security Model

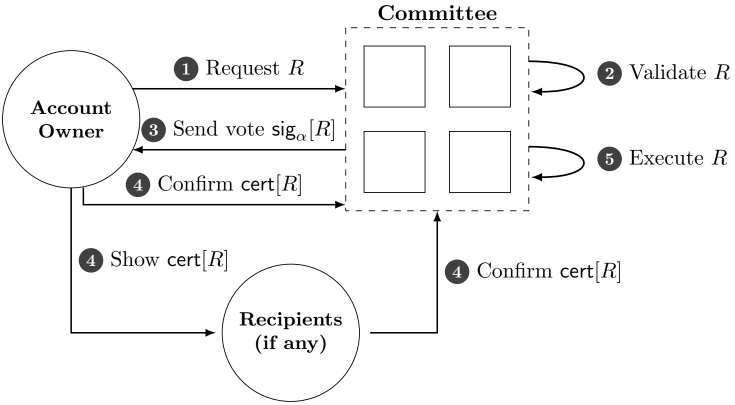
- $N = 3f + 1$ validators (aka “authorities” or “the committee”)
- At most f validators are malicious
- Asynchronous network

A statement S signed by a **quorum** of validators ($2f + 1$ of them) is called a **certificate**: $C = \text{cert}[S]$.

The FastPay/Zef Communication Model

- FastPay/Zef validators are **sharded services**
- Validators do not interact with each other.
- Clients query every validator in parallel and wait for a quorum of answers.
- No mempool
- During execution, shards may send asynchronous messages to other shards of the **same** validator

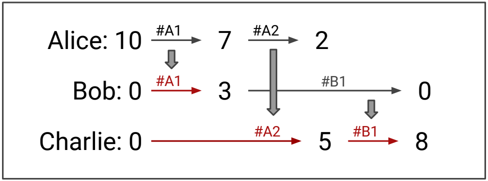
Account Operations in FastPay/Zef



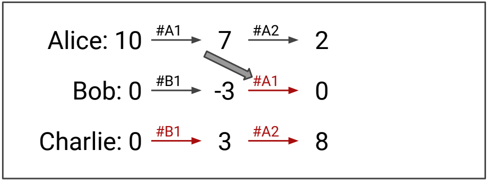
Example: $R = \mathsf{Transfer}\{\mathsf{from} : \mathit{Alice}, \mathsf{seq} : 1, \mathsf{to} : \mathit{Bob}, \mathsf{amount} : 3\}$

Eventual* Consistency

Validator α



Validator β



(*) with clients' help

Replicated State of a FastPay Account

- Owner's public key, used as **address**
- Public **balance**
- Next sequence number
- Current pending request (possibly $\perp$)
- Logs for executed requests (sent and received)

Validation, Sequencing, and Execution

Let $R = \text{Transfer}\{\text{from} : \mathit{Alice}, \text{seq} : n, \text{to} : \mathit{Bob}, \text{amount} : x\}$

- R is **valid** iff Alice's account satisfies $\mathit{pending} \in \{\perp, R\}$, $\mathit{nextseq} = n$, $\mathit{balance} \geq x$.
- **Voting** on a valid R sets $\mathit{pending} \leftarrow R$
- **Executing** $C = \text{cert}[R]$
  - ensure that $n = \mathit{nextseq}(\mathit{Alice})$
  - in Alice's account: let $\mathit{pending} \leftarrow \perp$, $\mathit{nextseq} \leftarrow \mathit{nextseq} + 1$, $\mathit{balance} \leftarrow \mathit{balance} - x$
  - in Bob's account: $\mathit{balance} \leftarrow \mathit{balance} + x$
  - in both accounts: $\mathit{logs} \leftarrow \mathit{logs} :: C$

Analysis of FastPay/Zef Account Operations

- Under BFT assumption, two certified requests for the same account and same sequence number are equal.
- Every honest validator eventually* executes the same certified requests
  - in the same order for senders
  - in arbitrary order for receivers
- If one honest validator validates a transaction, then it will eventually* "look valid" for everybody.
- To receive/spend money, clients may have to obtain missing certs (from available logs) and update lagging validators.

(*) with clients' help (unless the protocol is modified so that validator interacts)

Adding Opaque Coins

New cryptographic primitives

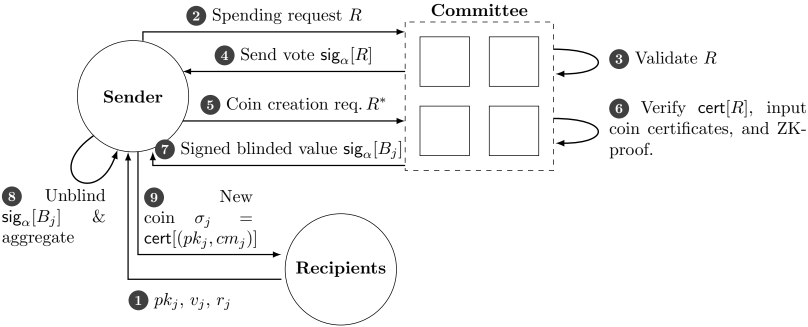
- Random commitment: $cm = \mathrm{commit}(v, r)$
- Blind signatures:
  $M' = \mathrm{blind}(M, u) \Rightarrow \mathrm{unblind}(\mathrm{sig}_\alpha[M'], u) = \mathrm{sig}_\alpha[M]$
- NIZK proofs: $\exists$*secrets* s.t. *predicate*(*inputs*, *secrets*)
- Threshold signature (optional):
  $\mathrm{aggregate}((\mathrm{sig}_\alpha[M])_{\alpha \in quorum}) = \mathrm{cert}[M]$

Our implementation uses a high-level library based on Coconut and Bulletproofs over BLS12-381 instead of abstract primitives.

Opaque Coin in FastPay++

- An **opaque coin** $\sigma = \text{cert}[(pk, cm)]$ binds a commitment $cm = \text{commit}(v, r)$ to some address pk
- $v \geq 0$ is the **value** and r is a secret random **seed**
- The owner of σ must know v and r and own the address pk.

Coin Creation in FastPay++



Replicated State of a FastPay++ Account

- Public key *pk* ("address")
- Public balance
- ...
- **Spent list** = set of all coin commitments *cm* that have been spent by this account.

How to Spend a Coin ...

- New account operation
  $R = \text{SpendInto}\{\text{from} : \mathit{Alice}, \text{seq} : 2, \text{coin} : \sigma, \text{into} : h\}$
- R is valid only if $\sigma = \text{cert}[(\mathit{Alice}, \mathit{cm})]$ and cm is not in the spent list of Alice's account
- Executing $C = \text{cert}[R]$ adds cm to the spent list

. . . and Make New Coins (R^*)

- Let $B_j = \text{blind}((pk_j, cm_j), u_j)$ for the j-th output coin
- Assume $h = \text{hash}(\pi, cm, (B_j))$ was used to produce $C = \text{cert}[R]$ for some $\sigma = \text{cert}[(Alice, cm)]$ and $R = \text{SpendInto}\{\text{from} : Alice, \text{seq} : n, \text{coin} : \sigma, \text{into} : h\}$
- Upon receiving a valid **coin creation request** $R^* = \text{CreateCoins}\{\text{proof} : \pi, \text{input} : C, \text{outputs} : (B_j)\}$ each validator returns a signature on B_j
- Unblinding and aggregating the signatures on B_j gives the new coin $\sigma_j = \text{cert}[(pk_j, cm_j)]$

. . . and Make New Coins (ZK proof)

π is **valid** iff it is a NIZK-proof that $\exists v, r, v_j, r_j, pk_j, cm_j, u_j$ s.t. all of the following hold on $(cm, (B_j))$

- $v \geq 0$ and $v_j \geq 0$
- $\sum_j v_j = v$
- $cm = \text{commit}(v, r)$ and $cm_j = \text{commit}(v_j, r_j)$
- $B_j = \text{blind}((pk_j, cm_j), u_j)$

Analysis of Opaque Coins

- The total coin value of an account is the sum of over **distinct** coins
- Coins are burnt first then new coins are created for an equivalent value
- When coins are burnt, h commits to a particular coin creation operation R^*. Replaying R^* just creates the same blinded signatures, hence the same coins again.
- Blinding factors u_j and NIZK proof π keep information on output coins secret from validators (and the rest of the network)

Generalization

- **Multiple input coins**:
  - $R^* = \text{CreateCoins}\{\text{proof} : \pi, \text{inputs} : (C_i), \text{outputs} : (B_j)\}$
  - $h = \text{hash}(\pi, (cm_i), (B_j))$
  - Input coins (pk_i, cm_i) must be mutually distinct
- **Transparent inputs**:
  - $R = \text{SpendInto}\{\text{from} : Alice, \text{seq} : 2, \text{amount} : v, \text{into} : h\}$
- **Transparent output**:
  - $R = \text{SpendAndTransfer}\{\text{from} : Alice, \text{seq} : 2, \text{coinvalue} : v, \text{coinseed} : r, \text{to} : Bob\}$
- Inputs must always be controlled by the same participant
- Transparent coins also possible: $\sigma = \text{cert}[(pk, v, nonce)]$

The Storage Problem

- FastPay accounts (indexed by users' pk) can never be removed
- Except for public address (e.g. crowdfunding), there is an incentive not to re-use accounts.
- Storage to prevent replay attacks is not cold storage
- High throughput $\Rightarrow$ high storage cost

Adding Non-Replayable Addresses

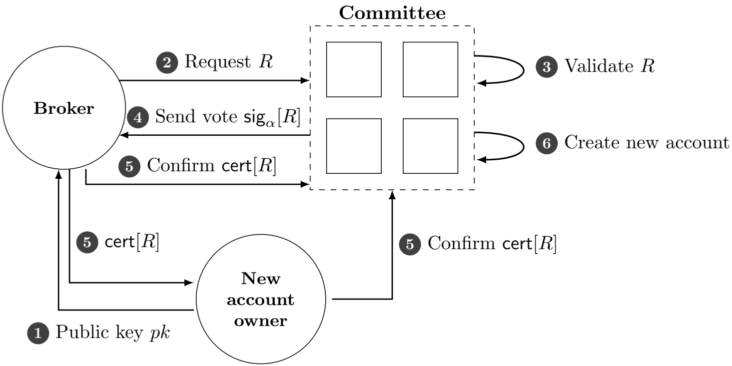
Replicated State of a Zef Account

- A unique identifier **uid**
  - used as an address
  - "unique" = account creation for this UID cannot be replayed
- The owner's public key **pk**
  - for authentication purposes only
  - can change over time
  - $\perp$ for inactive account
- ... (same as before)

Unique Identifiers ("UIDs")

- A Zef address is a non-empty list of sequence numbers: $uid = [1, 3, 5]$
- The **parent** of $[1, 3, 5]$ is $[1, 3]$
- $[2]$ is a **root**
- Roots are used for initial accounts given to validators
- **Zef uses existing (parent) accounts to derive fresh UIDs**

Activation of New Accounts



$$R = \mathsf{OpenAccount}\{\mathsf{from} : [1,0], \mathsf{seq} : 2, \mathsf{for} : pk\}$$

The new account has $uid = [1,0,2]$ and initial public key pk.

Benefits

Deactivated accounts cannot validate/execute requests and can never be reactivated, therefore do not need to remain in hot storage.

- In practice, we may limit #operations per account and incentivize users to deactivate unused accounts voluntarily.
- Some coordination between validators may also be needed to ensure that accounts are deactivated for every honest validator.

Additional account operations:

- $R = \text{ChangeOwner}\{\text{from} : [1, 0, 2], \text{seq} : 7, \text{for} : pk\}$
- $R = \text{CloseAccount}\{\text{from} : [1, 0, 2], \text{seq} : 7\}$
  (this sets $pk \leftarrow \bot$)

One Last Difficulty

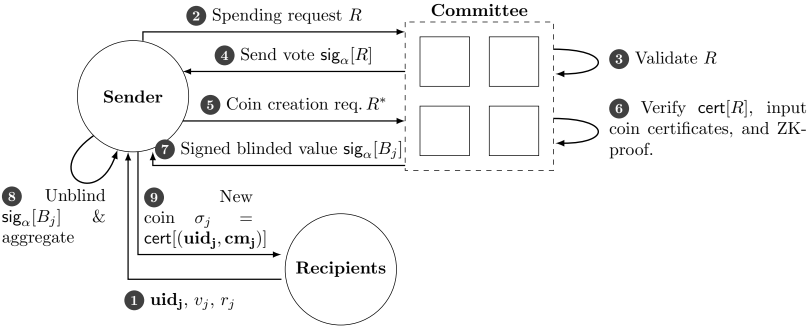
What happens if we transfer funds to $uid = [1, 0, 2]$ and this account does not exist yet in some validators?

- $R = \text{Transfer}\{\text{from} : [3], \text{seq} : 1, \text{to} : [1, 0, 2], \text{amount} : 5\}$
- Executing R may create a not-yet-active account with $uid = [1, 0, 2]$, balance 5, and $pk = \perp$
- Later, $R' = \text{OpenAccount}\{\text{from} : [1, 0], \text{seq} : 2, \text{for} : pk\}$ updates pk but keeps the balance 5.

Analysis of the Protocol

- *OpenAccount* is the only operation that can transition an account public key from $\perp$ to $pk \neq \perp$
- Inactive accounts cannot create or execute requests
- $\Rightarrow$ By induction on $|uid|$, every validator may only execute *uid* operations in sequential order and the deactivation of an account *uid* is final
- Account “brokers” do not have to be trusted for safety
- … but clients must check the certificate of account creation before using an account

Wrapping up: Coin Creation in Zef



Conclusion

- New point in the design space of decentralized systems for anonymous payments
- **Linear scalability**
- Strong **anonymity** properties
- We did not try to optimize NIZK proofs (e.g. Bulletproofs $\rightarrow$ transparent SNARKs?)
- More extensions of Zef to follow (e.g. Atomic Swaps)

Thanks!