Berkeley CS276 & MIT 6.875

Specialized homomorphic encryption, commitments and applications

Lecturer: Raluca Ada Popa
Announcements

• Starting to record
Specialized/partial homomorphic encryption

• An encryption scheme that is homomorphic with respect to a specific function, and cannot compute arbitrary functions like FHE
• Usually faster than FHE due to specialization (but not always)
El Gamal encryption (1985)

A semantically secure public-key encryption scheme

**Setup($1^k$):**
- Generate large prime $p$ of size $k$
- Choose generator $1 < g < p - 1$
- Output $(p, g)$

**KeyGen($1^k$):**
- Choose random $0 \leq sk \leq p - 2$
- Let $pk = g^{sk} \mod p$
- Output $(sk, pk)$

**Enc($pk, m$):** $m \in [1, p - 1]$  Why?
- Choose random $0 \leq r \leq p - 2$
- Output $(g^r \mod p, m \times pk^r \mod p)$

**Dec($sk, (c_1, c_2)$):** How to decrypt?
- Output $c_2 c_1^{-sk} \mod p$

$c_2 c_1^{-sk} = m pk^r g^{-rsk} = m g^{sk} r g^{-r sk} = m$
DDH assumption

Enc(pk, m):
- Choose random $0 \leq r \leq p - 2$
- Output $(g^r \mod p, m \times pk^r \mod p)$

Diffie-Hellman key exchange in disguise + used as one time pad

Semantic security relies on the Decisional Diffie Hellman assumption:
For all nonuniform PPT A,

\[
\left| \Pr[(g, p) \leftarrow \text{Setup}(1^k); a, b \leftarrow [0, p - 2], A(p, g, g^a, g^b, g^{ab}) = 1] - \Pr[(g, p) \leftarrow \text{Setup}(1^k); a, b, c \leftarrow [0, p - 2], A(p, g, g^a, g^b, g^c) = 1]\right| < \text{negl}(k)
\]
Proof of security

Decisional Diffie Hellman assumption: \( \forall \) nonuniform PPT \( A \),

\[
\left| \Pr[(g, p) \leftarrow \text{Setup}(1^k); a, b \leftarrow [0, p - 2], A(p, g, g^a, g^b, g^{ab}) = 1] - \Pr[(g, p) \leftarrow \text{Setup}(1^k); a, b, c \leftarrow [0, p - 2], A(p, g, g^a, g^b, g^c) = 1] \right| < \text{negl}(k)
\]

Claim: If DDH holds, El Gamal is semantically secure.

Proof: Assume \( A \) can break El Gamal’s security, let’s show that \( B \) can break DDH.

\( B \) must distinguish between \( g^a, g^b, g^{ab} \) and \( g^a, g^b, g^c \)

\( A \) can distinguish between \( g^{sk}, g^r, m_0 g^{skr} \) and \( g^{sk}, g^r, m_1 g^{skr} \)

\( B \) feeds \( g^{ab} \) or \( g^c \) times \( m_b \) to \( A \) for \( b \) random. If it is \( g^c \), \( A \) cannot guess, else \( A \) guesses correctly.
## Other partially homomorphic encryption schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Homomorphism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goldwasser-Micali’82</td>
<td>XOR</td>
</tr>
<tr>
<td>Paillier’99</td>
<td>+</td>
</tr>
<tr>
<td>Boneh-Goh-Nissim’05</td>
<td>+, then one *, then + based on bilinear maps</td>
</tr>
<tr>
<td>PHE/SHE (partially homomorphic encryption)</td>
<td>Some polynomial</td>
</tr>
</tbody>
</table>
Recall: commitments

A commitment protocol is an efficient two-stage protocol between a sender S and a receiver R:
- **commitment stage**: S has private input $x$. At the end of the stage,
  - Both parties hold $com$ (commitment)
  - S holds $r$ (the randomness used for decommitment)
- **reveal stage**: S sends $(r, x)$ to R, which accepts or rejects

**Completeness**: R always accepts in an honest execution of S.

**Hiding**: Hiding: $\forall R^*, x \neq x'$, in commit stage

$$
\{ \text{View}(S(x), R^*)(1^k) \} \approx_c \{ \text{View}(S(x'), R^*)(1^k) \}.
$$

**Binding**: Let $com$ be output of commit stage, $\forall S^*$

$$
\text{Prob}[S^* \text{ can reveal two pairs } (r, x) \& (r', x') \text{ s.t. } R(\text{com}, r, x) = R(\text{com}, r', x') = \text{Accept}] < \text{negl}(k)
$$
Pedersen commitment

Setup ($1^k$) - at the receiver:
- select large primes $p$ and $q$ of size $k$ such that $q$ divides $p - 1$
- select a generator $g$ of the order-$q$ subgroup of $\mathbb{Z}_p^*$
- generate randomly $a \leftarrow \mathbb{Z}_q$
- let $h = g^a \mod p$
- output $(g, h, p)$

Commit($g, h, p, x$) - by the sender:
- choose random $r \leftarrow \mathbb{Z}_q$
- output $comm = g^x h^r \mod p$

Reveal - by the sender:
- send $x$ and $r$ to receiver
- the receiver verifies that $comm = g^x h^r \mod p$ and accepts if so, else rejects
Perfectly hiding

Commit\((g, h, p, x)\) - by the sender:
- choose random \(r \leftarrow Z_q\)
- output \(comm = g^x h^r \text{ mod } p\)

• For a commitment \(comm\), every \(x\) could have been committed to in \(comm\)
• Given \(x, r\) and any \(x'\), \(\exists r'\) such that \(g^x h^r = g^{x'} h^{r'}\)
  \[ r' = (x - x')a^{-1} + r \mod q \]
Computationally binding

- Assume the sender can find $x', r'$, s.t $x' \neq x$ and
  \[ \text{comm} = g^x h^r = g^{x'} h^{r'} \]
- $h = g^a \mod p$ implies $x + ar = x' + ar' \mod q$
- The sender can compute $a = (x' - x)(r - r')^{-1}$

=> Sender solved discrete logarithm of h base g!!
Why is Pedersen homomorphic?

Commit($g, h, p, x$) - by the sender:
- choose random $r \leftarrow Z_q$
- output $\text{comm}(x, r) = g^x h^r \text{ mod } p$

$\text{comm}(x_1, r_1) \ast \text{comm}(x_2, r_2) = g^{x_1+x_2} h^{r_1+r_2} \text{ mod } p$

The sender reveals this commitment by showing $x_1 + x_2$ and $r_1 + r_2$
Application: zkLedger

• Privacy-preserving auditing for distributed ledgers
• A cryptographic system built out of:
  – Pedersen commitments and their homomorphism
  – Zero-knowledge proofs
First: the use case

(all cryptographic systems should have a use case)
structure of the financial system

• Dozens of large investment banks

• Trading:
  – Securities
  – Currencies
  – Commodities
  – Derivatives

• Trillions of dollars
A ledger records financial transactions

Assume a trusted ledger: append-only, immutable, consistent & visible to everyone

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>$</td>
<td>Citibank</td>
<td>Goldman Sachs</td>
<td>1,000,000</td>
</tr>
<tr>
<td>91</td>
<td>€</td>
<td>JP Morgan</td>
<td>UBS</td>
<td>200,000</td>
</tr>
<tr>
<td>92</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>3,000,000</td>
</tr>
</tbody>
</table>
Can verify important financial invariants

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>$</td>
<td>Citibank</td>
<td>Goldman Sachs</td>
<td>1,000,000</td>
</tr>
<tr>
<td>91</td>
<td>€</td>
<td>JP Morgan</td>
<td>UBS</td>
<td>200,000</td>
</tr>
<tr>
<td>92</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>3,000,000</td>
</tr>
</tbody>
</table>

**Verify**
- Consent to transfer
- Has assets to transfer
- Assets neither created nor destroyed

Examining ledger
Banks care about privacy

Trades reveal sensitive strategy information
Verifying invariants are maintained with privacy

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>$</td>
<td>Citibank</td>
<td>Goldman Sachs</td>
<td>1,000,000</td>
</tr>
<tr>
<td>91</td>
<td>€</td>
<td>JP Morgan</td>
<td>UBS</td>
<td>200,000</td>
</tr>
<tr>
<td>92</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>3,000,000</td>
</tr>
</tbody>
</table>

Verify

Consent to transfer
Has assets to transfer
Assets neither created nor destroyed
Verifying invariants are maintained with privacy

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From, To, Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>€</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>€</td>
<td></td>
</tr>
</tbody>
</table>

Zerocash (zk-SNARKs) [S&P 2014]
Solidus (PVORM) [CCS 2017]

Verify
- ✓ Consent to transfer
- ✓ Has assets to transfer
- ✓ Assets neither created nor destroyed
Problem

Regulators need insight into markets to maintain financial stability and protect investors

Participants would like to measure counterparty risk

- Leverage
- Exposure
- Overall market concentration
How to confidently audit banks to determine risk?

What fraction of your assets are in Euros?

3 million / 100 million

How exposed is this bank to a drop in the Euro?
zkLedger
A private, auditable transaction ledger

• **Privacy:** Hides transacting banks and amounts

• **Integrity with public verification:** *Everyone* can verify transactions are well-formed

• **Auditing:** Compute provably-correct linear functions over transactions
Outline

• System & threat model
• zkLedger design
  – Pedersen commitments
  – Ledger table format
  – Zero-knowledge proofs
• Evaluation
Outline

• **System & threat model**
• zkLedger design
  – Pedersen commitments
  – Ledger table format
  – Zero-knowledge proofs
• Evaluation
zkLedger system model

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>Transaction details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td></td>
</tr>
</tbody>
</table>
An auditor can obtain correct answers on ledger contents

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>Transaction details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td></td>
</tr>
</tbody>
</table>

What fraction of your assets are in Euros?

\[
\frac{3 \text{ million}}{100 \text{ million}}
\]

π

 Auditor
Measurements zkLedger supports

- Ratios and percentages of holdings
- **Sums**, averages, variance, skew
- Outliers
- Approximations and orders of magnitude
- Changes over time
- Well-known financial risk measurements (Herfindahl-Hirschmann index)
Security goals

• The auditor and non-involved parties cannot see transaction participants or amounts

• Banks cannot lie to the auditor or omit transactions

• Banks cannot violate financial invariants
  – Honest banks can always convince the auditor of a correct answer

• A malicious bank cannot block other banks from transacting
Threat model

Banks might attempt to steal or hide assets, manipulate balances, or lie to the auditor

Banks can arbitrarily collude

Banks or the auditor might try to learn transaction contents

Out of scope:

- A ledger that omits transactions or is unavailable
- An adversary watching network traffic
- Banks leaking their own transactions
Outline

• System & threat model

• zkLedger design
  – Pedersen commitments
  – Ledger table format
  – Zero-knowledge proofs

• Evaluation
## Example public transaction ledger

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor</td>
<td>Goldman Sachs</td>
<td>30,000,000</td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>Goldman Sachs</td>
<td>JP Morgan</td>
<td>10,000,000</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>1,000,000</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>2,000,000</td>
</tr>
</tbody>
</table>
Depositor injects assets to the ledger

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor</td>
<td>Goldman Sachs</td>
<td>30,000,000</td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>Goldman Sachs</td>
<td>JP Morgan</td>
<td>10,000,000</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>1,000,000</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>2,000,000</td>
</tr>
</tbody>
</table>
## Goals: auditing + privacy

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor</td>
<td>Goldman Sachs</td>
<td>30,000,000</td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>Goldman Sachs</td>
<td>JP Morgan</td>
<td>10,000,000</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>1,000,000</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>2,000,000</td>
</tr>
</tbody>
</table>

**Goals:**

- Provably audit Barclays to find Euro holdings
- Hide participants, amounts, and transaction graph
Hide amounts with commitments

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor</td>
<td>Goldman Sachs</td>
<td>30M</td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>Goldman Sachs</td>
<td>JP Morgan</td>
<td>comm(10M) ×</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(1M) ×</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(2M) ×</td>
</tr>
</tbody>
</table>

= comm(13M)
Hide participants with other techniques

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor</td>
<td>Goldman Sachs</td>
<td>30M</td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>Goldman Sachs</td>
<td>JP Morgan</td>
<td>comm(10M)</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(1M)</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(2M)</td>
</tr>
</tbody>
</table>
Strawman: audit by opening up combined commitments

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor</td>
<td>Goldman Sachs</td>
<td>30M</td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>Goldman Sachs</td>
<td>JP Morgan</td>
<td>comm(10M)</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(1M)</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(2M)</td>
</tr>
</tbody>
</table>

- Auditor reveals transactions.
- How many Euros do you hold? 3 million.
- Open comm(1M) × comm(2M) to 3M.
- Problems?
A malicious bank could omit transactions

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor</td>
<td>Goldman Sachs</td>
<td>30M</td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>Goldman Sachs</td>
<td>JP Morgan</td>
<td>comm(10M)</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(1M)</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(2M)</td>
</tr>
</tbody>
</table>

How many Euros do you hold?

1 million

Open comm(1M) to 1M
A malicious bank could omit transactions

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>From</th>
<th>To</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor</td>
<td>Goldman Sachs</td>
<td>30M</td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>Goldman Sachs</td>
<td>JP Morgan</td>
<td>comm(10M)</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(1M)</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>JP Morgan</td>
<td>Barclays</td>
<td>comm(2M)</td>
</tr>
</tbody>
</table>
zkLedger design: an entry for every bank in every transaction

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>Goldman Sachs</th>
<th>JP Morgan</th>
<th>Barclays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor, Goldman Sachs, 30M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>comm(-10M)</td>
<td>comm(10M)</td>
<td>comm(0)</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>comm(0)</td>
<td>comm(-1M)</td>
<td>comm(1M)</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>comm(0)</td>
<td>comm(-2M)</td>
<td>comm(2M)</td>
</tr>
</tbody>
</table>

Depositor transactions are public
Spender’s column commits to negative value, receiver’s positive value
For non-involved banks, entries commit to 0
Indistinguishable from commitments to non-zero values
Key insight: auditor audits every transaction

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>Goldman Sachs</th>
<th>JP Morgan</th>
<th>Barclays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor, Goldman Sachs, 30M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>comm(-10M)</td>
<td>comm(10M)</td>
<td>comm(0)</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>comm(0)</td>
<td>comm(-1M)</td>
<td>comm(1M)</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>comm(0)</td>
<td>comm(-2M)</td>
<td>comm(2M)</td>
</tr>
</tbody>
</table>

How many Euros do you hold?

3 million

Open \[\text{comm}(0) \times \text{comm}(1M) \times \text{comm}(2M)\] to 3M
A malicious bank can’t produce a proof for a different answer

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>Goldman Sachs</th>
<th>JP Morgan</th>
<th>Barclays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor, Goldman Sachs, 30M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>comm(-10M)</td>
<td>comm(10M)</td>
<td>comm(0)</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>comm(0)</td>
<td>comm(-1M)</td>
<td>comm(1M)</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>comm(0)</td>
<td>comm(-2M)</td>
<td>comm(2M)</td>
</tr>
</tbody>
</table>

How many Euros do you hold?
Security goals

- The auditor and non-involved parties cannot see transaction participants, amounts, or transaction graph.

- Banks cannot lie to the auditor or omit transactions.
  - Banks cannot violate financial invariants.
    - Honest banks can always convince the auditor of a correct answer.

- A malicious bank cannot block other banks from transacting.
How to maintain financial invariants?

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>Goldman Sachs</th>
<th>JP Morgan</th>
<th>Barclays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor, Goldman Sachs, 30M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>comm(-10M)</td>
<td>comm(10M)</td>
<td>comm(0)</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>comm(0)</td>
<td>comm(-1M)</td>
<td>comm(1M)</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>comm(0)</td>
<td>comm(-2M)</td>
<td>comm(2M)</td>
</tr>
</tbody>
</table>

use non-interactive zero-knowledge proofs (NIZKs)!

\text{comm}(\text{sig}_{GS}) \quad \text{comm}(\text{sig}_{JP})
What are the NIZK proof statements?

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>Goldman Sachs</th>
<th>JP Morgan</th>
<th>Barclays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>€</td>
<td>Depositor, Goldman Sachs, 30M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>€</td>
<td>(\text{comm}(-10M))</td>
<td>(\text{comm}(10M))</td>
<td>(\text{comm}(0))</td>
</tr>
<tr>
<td>3</td>
<td>€</td>
<td>(\text{comm}(0))</td>
<td>(\text{comm}(-1M))</td>
<td>(\text{comm}(1M))</td>
</tr>
<tr>
<td>4</td>
<td>€</td>
<td>(\text{comm}(0))</td>
<td>(\text{comm}(-2M))</td>
<td>(\text{comm}(2M))</td>
</tr>
</tbody>
</table>

Sender proves in zero knowledge that it knows \(sk\) for signing, values committed to in row, and decommitment randomness for all of them such that:

- Values in the transaction row sum to zero
- Signature verifies with the PK of sending bank on that amount
- One bank receives, all others are zero
- Bank has assets to transfer from previous transactions
Preliminaries

- Anyone can compute the aggregate commitment for every bank $i$ (over all transactions including this new transaction): $comm_{agg,i}$
- Let $n$ be the number of banks $
- comm_{n+1}$ contains the signature on the transaction
- Let $PK_i$ be the verification key of bank $i$ with signing key $SK_i$
- Assume that the receiver obtains the decommitment values from the spender using an out-of-band channel
The spender proves in zero-knowledge that it knows
- $s$ the index of spending bank, $\ell$ the index of receiving bank,
- decommitment values $r_i$ and values $v_i$
- signature randomness $r$ and $sk$,
- $r_{agg}, v_{agg}$ for $comm_{agg,s}$,

such that:
- $comm_i$ opens up with $r_i$ and $v_i$,
- $v_{n+1}$ is $sig$ produced with $r, sk$ and $sig$ verifies with $PK_s$ on transaction content
  - [transaction is authorized]
- $v_s \leq 0, v_s = -v_\ell, v_i = 0$ for $i \in [1, n] \neq \ell, s$,
  - [spender loses money, receiver gains same money, the rest have zero]
- $comm_{agg,s}$ opens up with $r_{agg}$ and $v_{agg}$ and $v_{agg} \geq 0$
  - [spender spends no more than resources]

Instead of one monolithic proof enforcing these properties, zkLedger does a set of more efficient things but they are less relevant here
Outline

• System model
• zkLedger design
  – Hiding commitments
  – Ledger table format
  – Zero-knowledge proofs
• Evaluation
Implementation

• zkLedger written in Go
• Elliptic curve library: btcec, secp256k1
• ~4000 loc
Evaluation

• How fast is auditing?
• How does zkLedger scale with the number of banks?

Experiments on 12 4 core Intel Xeon 2.5Ghz VMs, 24 GB RAM
Simple auditing is fast and independent of ledger size

Pedersen commitments + table design amenable to caching

Auditing 4 banks measuring market concentration
Cost in a transaction per bank

- Entry size: **4.5KB**
- Creating an entry: **8ms**
- Verifying an entry: **7ms**

Highly parallelizable

Significant opportunities for compression and speedup
Summary

- Specialized/partial homomorphic encryption enables specific functionalities and tend to be faster than FHE at computing these
- Pedersen commitment is also homomorphic
- zkLedger provides privacy and auditing on transaction ledgers using Pedersen commitments, their homomorphism and NIZKs