Additively Homomorphic UC Commitments With Optimal Amortized Overhead

Ignacio Cascudo, Ivan Damgård, Bernardo David, Irene Giacomelli, Jesper Buus Nielsen, Roberto Trifiletti
Aarhus University
Structure

1. Introduction
Structure

1. Introduction
2. Our Contributions
Structure

1. Introduction
2. Our Contributions
3. A general framework
Structure

1. Introduction
2. Our Contributions
3. A general framework
4. Open Questions
Commitment Schemes
Commitment Schemes
Multiparty Computation

- The Millionaires’ Problem
Universal Composability

• Protocols remain secure in parallel concurrent executions and arbitrary composition.
Universal Composability

• Protocols remain secure in parallel concurrent executions and arbitrary composition.

• Commitments require setup assumptions [CF01].
Universal Composability

- Protocols remain secure in parallel concurrent executions and arbitrary composition.

- Commitments require setup assumptions [CF01].
- Commitments are complete [CLOS02].
Related Works

• DDH based fast UC commitments: [Lindell11, BCPV13].
  – Use a Common Reference String (CRS).
  – High asymptotic communication and computational complexity.
Related Works

• DDH based fast UC commitments: [Lindell11,BCPV13].
  – Use a Common Reference String (CRS).
  – High asymptotic communication and computational complexity.

• UC commitments with optimal rate: [DDGN14,GIKW14].
  – Use Oblivious Transfer as a setup assumption.
  – Require PRGs and general Linear Secret Sharing.
What do we do in theory?
What do we do in theory?

• Optimal communication
What do we do in theory?

• Optimal communication
• Additively Homomorphic
What do we do in theory?

- Optimal communication
- Additively Homomorphic
- Optimal computation • NEW!
What do we do in theory?

- Optimal communication
- Additively Homomorphic
- Optimal computation
- No need for general secret sharing
What do we do in theory?

- Optimal communication
- Additively Homomorphic
- Optimal computation
- No need for general secret sharing

How do we do it?

ECC + PRG + OT
What do we do in practice?

- Online Phase:

\[ \text{BCH} \ [796,256,\geq121] + \text{PRG} \]

2 Encodings: 1.5 \( \mu s \)
What do we do in practice?

- Online Phase:

\[
\text{BCH} \ [796, 256, \geq 121] + \text{PRG}
\]

2 Encodings: 1.5\,\mu s

VS.

[Lindell11, BCPV13] \rightarrow 22\,\text{exponentiations}: 8250\,\mu s
What do we do in practice?

• Online Phase:

\[ \text{BCH} \ [796,256,\geq121] \ + \ \text{PRG} \]

2 Encodings: 1.5 µs

\[ [\text{Lindell11,BCPV13}] \rightarrow 22 \ \text{exponentiations:} \ 8250 \ \mu s \]

• Practical scheme runs 5500 times faster
Practical Trade Offs...
Practical Trade Offs...

- No additive homomorphism.
Practical Trade Offs...

- No additive homomorphism.
- Setup phase cost:
  - 796 OTs
  - 8756 exponentiations using [PVW08]
  - 398 [Lindell11,BCPV13] commitments
Building Blocks

• Error correcting codes:
  − Linear-time encodable codes
    [GI01, GI02, GI03, GI05, Spi96, DI14].
Building Blocks

• Error correcting codes:
  – Linear-time encodable codes
    [GI01, GI02, GI03, GI05, Spi96, DI14].

• UC Oblivious Transfer:
  – Any UC Oblivious Transfer protocol, e.g. [PVW08]
Building Blocks

• Error correcting codes:
  – Linear-time encodable codes
    [GI01, GI02, GI03, GI05, Spi96, DI14].

• UC Oblivious Transfer:
  – Any UC Oblivious Transfer protocol, e.g. [PVW08]

• Pseudorandom Generator:
  – Linear-time PRG, e.g. [VZ12]
Oblivious Transfer

\[ s_0, s_1 \in \{0,1\}^l \]

\[ c \in \{0,1\} \]

Does not learn \( c \)

Learns either \( s_0 \) OR \( s_1 \)
Oblivious Transfer

\[ s_0, s_1 \in \{0,1\}^l \]

\[ c \in \{0,1\} \]

Alice

\[ b_0 \rightarrow 1-2 \text{ OT} \rightarrow i \]

\[ b_1 \rightarrow \]

Bob (Cat)

Does not learn \( c \)

Learns either \( s_0 \) OR \( s_1 \)
Encoding Scheme

M → ENC
Encoding Scheme

M → ENC

ECC

Codeword:

\[ c[1] \]

\[ c[2] \]

\[ \ldots \]

\[ c[n] \]
Encoding Scheme

M \rightarrow ENC

ECC

Codeword: \[ c[1], c[2], \ldots, c[n] \]

Randomness: \[ s_2[1], s_2[2], \ldots, s_2[n] \]
Encoding Scheme

M → ENC

ECC

Codeword: $s_2[1], s_2[2], \ldots, s_2[n]$
Randomness: $s_1[1], s_1[2], \ldots, s_1[n]$

$M \rightarrow \text{ENC}$
Encoding Scheme

M → ENC

ECC

Codeword:

Randomness:

M → ECC

ECC

\[ c[1] + s_2[1] = s_1 \]

\[ c[2] + s_2[2] = s_1 \]

\[ … \]

\[ c[n] + s_2[n] = s_1 \]

s_1[1]

s_1[2]

s_2[1]

s_2[2]
General Framework

• Setup phase:

• Commitment/Open Phases:
General Framework

• Setup phase:
  – Independent from the inputs

• Commitment/Open Phases:
General Framework

• Setup phase:
  – Independent from the inputs
  – Constant number of OTs for unbounded number of commitments.

• Commitment/Open Phases:
General Framework

• Setup phase:
  – Independent from the inputs
  – Constant number of OTs for unbounded number of commitments.
  – Constant communication complexity.

• Commitment/Open Phases:
General Framework

• Setup phase:
  – Independent from the inputs
  – Constant number of OTs for unbounded number of commitments.
  – Constant communication complexity.

• Commitment/Open Phases:
  – Linear communication complexity.
General Framework

• Setup phase:
  – Independent from the inputs
  – Constant number of OTs for unbounded number of commitments.
  – Constant communication complexity.

• Commitment/Open Phases:
  – Linear communication complexity.
  – Only require a PRG and the encoding scheme.
General Framework

• Setup phase:
  – Independent from the inputs
  – Constant number of OTs for unbounded number of commitments.
  – Constant communication complexity.

• Commitment/Open Phases:
  – Linear communication complexity.
  – Only require a PRG and the encoding scheme.
  – Non interactive.
Setup Phase

Sender

Random Seeds:

$\text{k}_1$
$\text{k}_2$
$\text{k}_3$
$\text{k}_4$

$\cdots$

$\text{k}_{n-1}$
$\text{k}_n$

Receiver
Setup Phase

Sender

Random Seeds:

- $k_1$
- $k_2$
- $k_3$
- $k_4$
- ...
- $k_{n-1}$
- $k_n$

Random Choices:

- $1-2$ OT
- $1-2$ OT
- $1-2$ OT

Received Seeds:

- $k_{1+c1}$
- $k_{3+c2}$
- ...
- $k_{n-1+cn}$

Receiver
Commitment Phase (Sender)

Generate one-time pads:

\[ k_1 \rightarrow \text{PRG} \rightarrow P_1 \]
\[ k_2 \rightarrow \text{PRG} \rightarrow P_2 \]
\[ k_3 \rightarrow \text{PRG} \rightarrow P_3 \]
\[ k_4 \rightarrow \text{PRG} \rightarrow P_4 \]
\[ \vdots \]
\[ k_{n-1} \rightarrow \text{PRG} \rightarrow P_{n-1} \]
\[ k_n \rightarrow \text{PRG} \rightarrow P_n \]
Commitment Phase (Sender)

Generate one-time pads:

\[ k_1 \rightarrow \text{PRG} \rightarrow P_1 \]
\[ k_2 \rightarrow \text{PRG} \rightarrow P_2 \]
\[ k_3 \rightarrow \text{PRG} \rightarrow P_3 \]
\[ k_4 \rightarrow \text{PRG} \rightarrow P_4 \]
\[ \vdots \]
\[ k_{n-1} \rightarrow \text{PRG} \rightarrow P_{n-1} \]
\[ k_n \rightarrow \text{PRG} \rightarrow P_n \]

Encode messages and encrypt with one-time pads:

\[ M \rightarrow \text{ENC} \rightarrow s_1[1] \]
\[ \text{\vdots} \]
\[ \text{\vdots} \]
\[ \text{\vdots} \]
\[ \text{\vdots} \]
\[ k_{n-1} \rightarrow \text{PRG} \rightarrow P_{n-1} \]
\[ k_n \rightarrow \text{PRG} \rightarrow P_n \]
Commitment Phase (Sender)

Generate one-time pads:

\[ k_1 \rightarrow \text{PRG} \rightarrow P_1 \]
\[ k_2 \rightarrow \text{PRG} \rightarrow P_2 \]
\[ k_3 \rightarrow \text{PRG} \rightarrow P_3 \]
\[ k_4 \rightarrow \text{PRG} \rightarrow P_4 \]
\[ \vdots \]
\[ k_{n-1} \rightarrow \text{PRG} \rightarrow P_{n-1} \]
\[ k_n \rightarrow \text{PRG} \rightarrow P_n \]

Encode messages and encrypt with one-time pads:

\[ M \rightarrow \text{ENC} \rightarrow \]
\[ s_1[1] \]
\[ s_2[1] \]
\[ s_1[2] \]
\[ s_2[2] \]
\[ \vdots \]
\[ s_1[n] \]
\[ s_2[n] \]

\[ P_1 \rightarrow \text{PRG} \rightarrow C_1 \]
\[ P_2 \rightarrow \text{PRG} \rightarrow C_2 \]
\[ P_3 \rightarrow \text{PRG} \rightarrow C_3 \]
\[ P_4 \rightarrow \text{PRG} \rightarrow C_4 \]
\[ P_{n-1} \rightarrow \text{PRG} \rightarrow C_{n-1} \]
\[ P_n \rightarrow \text{PRG} \rightarrow C_n \]
Open Phase (Receiver)

Opening Message:

\[ M \]

\[ s_1[1] \]

\[ s_2[1] \]

\[ s_1[2] \]

\[ s_2[2] \]

\[ \ldots \]

\[ s_1[n] \]

\[ s_2[n] \]
Open Phase (Receiver)

Opening Message:

- $M$
- $s_1[1]$
- $s_2[1]$
- $s_1[2]$
- $s_2[2]$
- $s_1[n]$
- $s_2[n]$

Generate one-time pads:

- $k_{1+c1}$ → PRG → $P_{1+c1}$
- $k_{3+c2}$ → PRG → $P_{3+c2}$
- $k_{n-1+cn}$ → PRG → $P_{n-1+cn}$
Opening Message:

M

\[ s_1[1] \]
\[ s_2[1] \]
\[ s_1[2] \]
\[ s_2[2] \]

…

\[ s_1[n] \]
\[ s_2[n] \]

Open Phase (Receiver)

Generate one-time pads:

\[ k_{1+c1} \rightarrow \text{PRG} \rightarrow P_{1+c1} \]
\[ k_{3+c2} \rightarrow \text{PRG} \rightarrow P_{3+c2} \]

…

\[ k_{n-1+cn} \rightarrow \text{PRG} \rightarrow P_{n-1+cn} \]

Check known shares:

\[ C_1 \]
\[ C_2 \]
\[ C_3 \]
\[ C_4 \]

…

\[ C_{n-1} \]
\[ C_n \]

\[ P_{1+c1} \]
\[ P_{3+c2} \]

…

\[ P_{n-1+cn} \]

\[ s_1[1] \]
\[ s_2[1] \]
\[ s_1[2] \]
\[ s_2[2] \]

…

\[ s_1[n] \]
\[ s_2[n] \]
Open Phase (Receiver)

Reconstruct ECC codeword:
Open Phase (Receiver)

Reconstruct ECC codeword:

Check that codewords match:
Open Problems

• Can we get optimal rate?
• Can optimal fully homomorphic commitments be constructed without general LSSS?
• Can we get additive homomorphism in this construction without VSS?
• Can we increase concrete efficiency in both setup and online phases?
THANK YOU!

READ THE FULL PAPER:

https://eprint.iacr.org/2014/829