Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography

Ivan Damgård[1] Yuval Ishai[2] Mikkel Krøigaard[3]

[1]University of Aarhus

[2]Technion and UCLA

[3]Eindhoven University of Technology

June 2, 2010

The Setting

- **n** players must evaluate an arithmetic (/boolean) circuit **C**.
- Up to **t** players may be corrupted.
- Perfect security (UC) against malicious and adaptive adversaries.
- Assume secure point-to-point synchronous communication.
- Elements are in a prime field $\mathbb{Z}_p$.

The Overhead of MPC

- Consider the total computational complexity for all players:

$$f \cdot |C| + g.$$

- f is the per-gate overhead.
- g should be dominated by $f \cdot |C|$ when $|C| >> n$.
- Best case: $f = O(1)$.
- Our result: $f = \mathrm{poly}(\log n, \log |C|)$.

Previous Work

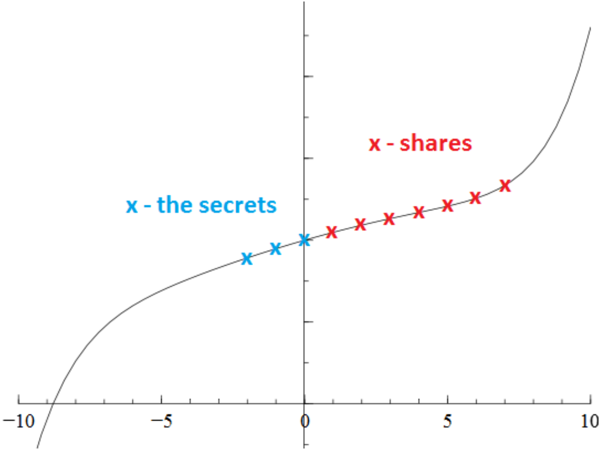
- [DIKNS08]: $\mathrm{poly}(k, \log n)$ overhead, but only computational security.
- [BH06,DI06,DN07,HN07]: $\tilde{O}(n)$ overhead.
- [DI06]: $\mathrm{poly}(\log n)$ overhead with a *constant* number of clients (i.e. players who give input or receive output).
- We want unconditional security *and* polylogarithmic overhead.

Overview

- Basis is a traditional Shamir-sharing based protocol with overhead $n^2 \log n$ [Shamir79,BGW88,GRR98].
- We utilize packed secret-sharing [FY92].
- Multiplication is sped up using prepared pairs and VSS protocols [BH08,DIKNS08].
- **New Technique 1**: Reorganize circuit for use of packed sharing.
- **New Technique 2**: New VSS protocols for dealing with permutations within shared blocks.
- Finally, boost the threshold with *player virtualization* [Bracha87,Chaum89,...,HM00,DIKNS08,Krøigaard10].

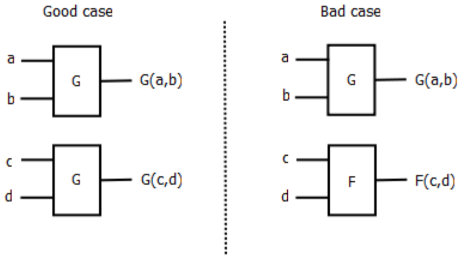
Packed Secret-Sharing

We use the idea from [FY92]:



Ramp scheme, but we avoid the ramp. Lowers the threshold.

Utilizing Packed Sharing

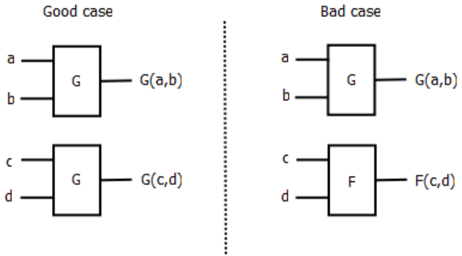


In the **bad** case:

1. Share single values $[a], [b], [c], [d]$.
2. Calculate $G([a], [b])$ *and* $F([c], [d])$.

Here there is no advantage to using packed sharing.

Utilizing Packed Sharing



In the **good** case:

1. Let $b_1 = (a, c)$ and $b_2 = (b, d)$. Share $[b_1], [b_2]$.
2. Calculate $G([b_1], [b_2]) = [(G(a, b), G(c, d))]$.

Idea: Set block size to $\Theta(n)$ and hope to divide the complexity by n.

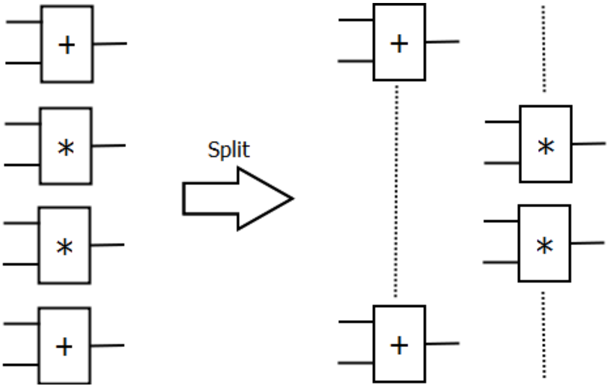
Technique 1: Transforming the Circuit

We transform C to get many good cases:

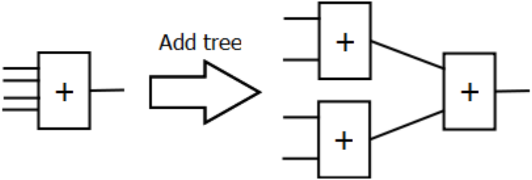
- Divide C into d layers in the natural way.
- Each layer depends only on previous layers for input.
- Transform C into C' in three steps.
- C' is computable in parallel, layer by layer.

Technique 1: Transforming the Circuit

Step 1: Make sure all layers have the same type of gate.

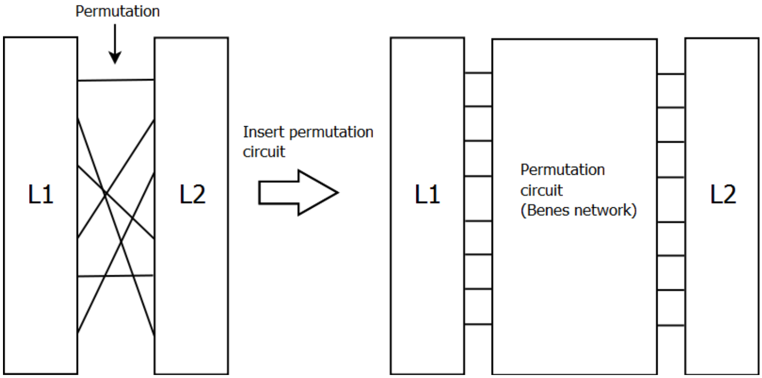


Step 2: Make sure all gates take two inputs.



Technique 1: Transforming the Circuit

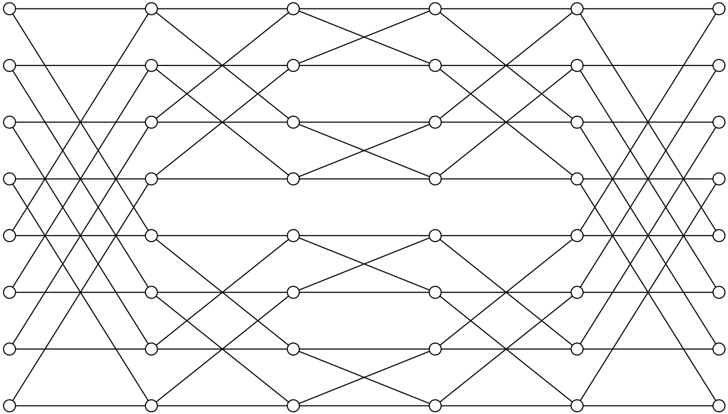
Step 3: Simplify permutations between layers.



No more permutations across blocks.

A Beneš Network

[Beneš64,Waksman68]:



Models any permutation, can easily be converted to a circuit.

Handling Permutations

- The Beneš network will handle permutations across blocks.
- Between each two layers, it requires a permutation.
- Either the permutation permutes all blocks (trivial).
- Or it performs the same permutation within each block.

Handling Permutations

Protocol to permute within a block:

1. Requires shared $[r], [\pi(r)]$ (from new VSS protocol).
2. Mask input:
$$[x+r] = [x] + [r].$$
3. Open to party i.
4. Party i VSS shares $[\pi(x+r)]$ using new VSS protocol that also proves correct permutation.
5. Unmask output:
$$[\pi(x)] = [\pi(x+r)] - [\pi(r)].$$

Note: new VSS protocols build on [BH08,DIKNS08].

Precise Result

- Security is *perfect*, *active* and *adaptive*.
- Threshold: For any $\varepsilon > 0$: $(1/3 - \varepsilon)n$.
- For circuit size s and depth d, no. arithmetic operations:

$$O(\log^2 n \log s \cdot s + \mathrm{poly}(n, \log s) \cdot d^2)$$

- No improvements for very narrow and deep circuits.
- For most natural circuits, the second term is linear in d.
- Additive term can be reduced to $\mathrm{poly}(n, \log s) \cdot d \log d$ for a factor $O(\log d)$ additional overhead.

Applications to Two-Party Cryptography

Possible applications in reducing the overhead of:

- Zero-knowledge proofs.
- Secure two-party computation.
- Other useful special cases.

Applications to Two-Party Cryptography

Possible applications in reducing the overhead of:

- Zero-knowledge proofs.
- Secure two-party computation.
- Other useful special cases.

[IKOS08] define the computational overhead of a primitive as:

$$\frac{\text{circuit size of secure impl.}}{\text{circuit size of correct but insecure impl.}}$$

where input size $\to \infty$.

Applications to Two-Party Cryptography

Consider the computational overhead:

- Standard implementations of encryption, signatures, ZK proofs, ..., have overhead poly(k).
- One could hope for $O(1)$.
- [IKOS08]: $O(1)$ overhead for 2PC in the *semihonest* model.
- Open: Improve on poly(k) for ZK and 2PC in the *malicious* model.
- New: Combining our protocol with [IKOS07,IKOS08,IPS08,ACPS09] we get:
  1. ZK with poly($\log k$) overhead under standard assumptions.
  2. 2PC with poly($\log k$) overhead under a non-standard assumption or preprocessing.

Thank you!

Questions?