

Efficient Polynomial Operations in the Shared-Coefficients Setting

Payman Mohassel Matthew Franklin

Department of Computer Science
University of California, Davis

PKC 2006

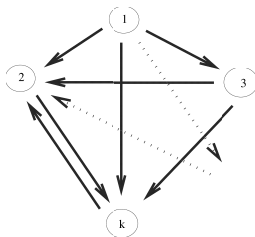
Outline

- 1 Introduction and Motivation
 - Secure Multiparty Computation
 - Motivation
- 2 Our Contribution
 - Overview
 - Protocols

Outline

- 1 Introduction and Motivation
 - Secure Multiparty Computation
 - Motivation
- 2 Our Contribution
 - Overview
 - Protocols

Secure Multiparty Computation



- Party i holds input x_i .
- Parties learn $f(x_1, x_2, \dots, x_k)$ but nothing else.
- **Computational** Setting
- **Information-theoretic** Setting

Secure MPC Cont'd

- Efficiency Criteria
 - Computational Complexity
 - **Communication** Complexity
 - **Round** Complexity
- Computational Setting:
 - C is the circuit computing f .
 - $O(1)$ rounds and $O(k|C|)$ comm. [Yao82], [BMR90].
- Information-Theoretic Setting:
 - $O(1)$ rounds for problems in NL , [FKN94], [IK97, 00].
 - No general *constant-round* protocol known.
 - $O(k|C|)$ comm. [CCD88], [BGW88].

Secure MPC Cont'd

- Efficiency Criteria
 - Computational Complexity
 - **Communication** Complexity
 - **Round** Complexity
- Computational Setting:
 - C is the circuit computing f .
 - $O(1)$ rounds and $O(k|C|)$ comm. [Yao82], [BMR90].
- Information-Theoretic Setting:
 - $O(1)$ rounds for problems in NL , [FKN94], [IK97, 00].
 - No general *constant-round* protocol known.
 - $O(k|C|)$ comm. [CCD88], [BGW88].

Secure MPC Cont'd

- Efficiency Criteria
 - Computational Complexity
 - **Communication** Complexity
 - **Round** Complexity
- Computational Setting:
 - C is the circuit computing f .
 - $O(1)$ rounds and $O(k|C|)$ comm. [Yao82], [BMR90].
- Information-Theoretic Setting:
 - $O(1)$ rounds for problems in **NL**, [FKN94], [IK97, 00].
 - No general *constant-round* protocol known.
 - $O(k|C|)$ comm. [CCD88], [BGW88].

Secure MPC Cont'd

- Efficiency Criteria
 - Computational Complexity
 - **Communication** Complexity
 - **Round** Complexity
- Computational Setting:
 - C is the circuit computing f .
 - $O(1)$ rounds and $O(k|C|)$ comm. [Yao82], [BMR90].
- Information-Theoretic Setting:
 - $O(1)$ rounds for problems in **NL**, [FKN94], [IK97, 00].
 - No general *constant-round* protocol known.
 - $O(k|C|)$ comm. [CCD88], [BGW88].

Secure MPC Cont'd

- Efficiency Criteria
 - Computational Complexity
 - **Communication** Complexity
 - **Round** Complexity
- Computational Setting:
 - C is the circuit computing f .
 - $O(1)$ rounds and $O(k|C|)$ comm. [Yao82], [BMR90].
- Information-Theoretic Setting:
 - $O(1)$ rounds for problems in **NL**, [FKN94], [IK97, 00].
 - No general *constant-round* protocol known.
 - $O(k|C|)$ comm. [CCD88], [BGW88].

Outline

- 1 Introduction and Motivation
 - Secure Multiparty Computation
 - Motivation
- 2 Our Contribution
 - Overview
 - Protocols

Polynomials in Secure Multiparty Computation

- Private matching of datasets, [FNP04].
- Privacy-preserving keyword search, [FIPR05].
- Privacy-preserving set operations, [KS05].
- $p(x) = \prod_{i=1}^n (x - x_i)$.
- Encrypted using Additively Homomorphic Cryptosystem.

Polynomials in Secure Multiparty Computation

- Private matching of datasets, [FNP04].
- Privacy-preserving keyword search, [FIPR05].
- Privacy-preserving set operations, [KS05].
- $p(x) = \prod_{i=1}^n (x - x_i)$.
- Encrypted using Additively Homomorphic Cryptosystem.

Polynomials in Secure Multiparty Computation

- Private matching of datasets, [FNP04].
- Privacy-preserving keyword search, [FIPR05].
- Privacy-preserving set operations, [KS05].
- $p(x) = \prod_{i=1}^n (x - x_i)$.
- Encrypted using Additively Homomorphic Cryptosystem.

Polynomials in Secure Multiparty Computation

- Private matching of datasets, [FNP04].
- Privacy-preserving keyword search, [FIPR05].
- Privacy-preserving set operations, [KS05].
- $p(x) = \prod_{i=1}^n (x - x_i)$.
- Encrypted using Additively Homomorphic Cryptosystem.

Polynomials in Secure Multiparty Computation

- Private matching of datasets, [FNP04].
- Privacy-preserving keyword search, [FIPR05].
- Privacy-preserving set operations, [KS05].
- $p(x) = \prod_{i=1}^n (x - x_i)$.
- Encrypted using Additively Homomorphic Cryptosystem.

Applications

- Sharing data without losing privacy.
 - Medical databases.
 - Sharing network information: traffic, bandwidth, topology, ...
 - Online recommendation services.

Large datasets!

Communication and round efficiency is critical.

Applications

- Sharing data without losing privacy.
 - Medical databases.
 - Sharing network information: traffic, bandwidth, topology, ...
 - Online recommendation services.

Large datasets!

Communication and round efficiency is critical.

Applications

- Sharing data without losing privacy.
 - Medical databases.
 - Sharing network information: traffic, bandwidth, topology, ...
 - Online recommendation services.

Large datasets!

Communication and round efficiency is critical.

Outline

- 1 Introduction and Motivation
 - Secure Multiparty Computation
 - Motivation
- 2 Our Contribution
 - Overview
 - Protocols

Contributions

- Secure protocols for operations on polynomials.
- Use techniques from [Symbolic Computation](#).
- [Constant-round](#) and [communication-efficient](#).
- Both computational and information-theoretic setting.
- Protocols are [composable](#).
- Applications to privacy-preserving set operations.

Contributions

- Secure protocols for operations on polynomials.
- Use techniques from **Symbolic Computation**.
- **Constant-round** and **communication-efficient**.
- Both computational and information-theoretic setting.
- Protocols are **composable**.
- Applications to privacy-preserving set operations.

Contributions

- Secure protocols for operations on polynomials.
- Use techniques from **Symbolic Computation**.
- **Constant-round** and **communication-efficient**.
- Both computational and information-theoretic setting.
- Protocols are **composable**.
- Applications to privacy-preserving set operations.

Contributions

- Secure protocols for operations on polynomials.
- Use techniques from **Symbolic Computation**.
- **Constant-round** and **communication-efficient**.
- Both computational and information-theoretic setting.
- Protocols are **composable**.
- Applications to privacy-preserving set operations.

Contributions

- Secure protocols for operations on polynomials.
- Use techniques from [Symbolic Computation](#).
- [Constant-round](#) and [communication-efficient](#).
- Both computational and information-theoretic setting.
- Protocols are [composable](#).
- Applications to privacy-preserving set operations.

Contributions

- Secure protocols for operations on polynomials.
- Use techniques from [Symbolic Computation](#).
- [Constant-round](#) and [communication-efficient](#).
- Both computational and information-theoretic setting.
- Protocols are [composable](#).
- Applications to privacy-preserving set operations.

Setting

- “Share” values in a finite field F .
 - Use a *Threshold Add. Homomorphic Cryptosystem* [Pai00, FP00].
 - Use a *Linear Secret Sharing Scheme* [Sha78].
- Add two shared values **without interaction**.
- Multiply a public value with a shared value **without interaction**
- Multiply two shared values with **constant rounds** of interaction.

Communication measure = number of multiplications.

- Share a polynomial by sharing its coefficients.

Setting

- “Share” values in a finite field F .
 - Use a *Threshold Add. Homomorphic Cryptosystem* [Pai00, FP00].
 - Use a *Linear Secret Sharing Scheme* [Sha78].
- Add two shared values **without interaction**.
- Multiply a public value with a shared value **without interaction**
- Multiply two shared values with **constant rounds** of interaction.

Communication measure = number of multiplications.

- Share a polynomial by sharing its coefficients.

Setting

- “Share” values in a finite field F .
 - Use a *Threshold Add. Homomorphic Cryptosystem* [Pai00, FP00].
 - Use a *Linear Secret Sharing Scheme* [Sha78].
- Add two shared values **without interaction**.
- Multiply a public value with a shared value **without interaction**
- Multiply two shared values with **constant rounds** of interaction.

Communication measure = number of multiplications.

- Share a polynomial by sharing its coefficients.

Setting

- “Share” values in a finite field F .
 - Use a *Threshold Add. Homomorphic Cryptosystem* [Pai00, FP00].
 - Use a *Linear Secret Sharing Scheme* [Sha78].
- **Add** two shared values **without interaction**.
- **Multiply** a public value with a shared value **without interaction**
- **Multiply** two shared values with **constant rounds** of interaction.

Communication measure = number of multiplications.

- Share a polynomial by sharing its coefficients.

Setting

- “Share” values in a finite field F .
 - Use a *Threshold Add. Homomorphic Cryptosystem* [Pai00, FP00].
 - Use a *Linear Secret Sharing Scheme* [Sha78].
- **Add** two shared values **without interaction**.
- **Multiply** a public value with a shared value **without interaction**
- **Multiply** two shared values with **constant rounds** of interaction.

Communication measure = number of multiplications.

- Share a polynomial by sharing its coefficients.

Setting

- “Share” values in a finite field F .
 - Use a *Threshold Add. Homomorphic Cryptosystem* [Pai00, FP00].
 - Use a *Linear Secret Sharing Scheme* [Sha78].
- **Add** two shared values **without interaction**.
- **Multiply** a public value with a shared value **without interaction**
- **Multiply** two shared values with **constant rounds** of interaction.

Communication measure = number of multiplications.

- Share a polynomial by sharing its coefficients.

Setting

- “Share” values in a finite field F .
 - Use a *Threshold Add. Homomorphic Cryptosystem* [Pai00, FP00].
 - Use a *Linear Secret Sharing Scheme* [Sha78].
- **Add** two shared values **without interaction**.
- **Multiply** a public value with a shared value **without interaction**
- **Multiply** two shared values with **constant rounds** of interaction.

Communication measure = number of multiplications.

- Share a polynomial by sharing its coefficients.

Outline

- 1 Introduction and Motivation
 - Secure Multiparty Computation
 - Motivation
- 2 Our Contribution
 - Overview
 - Protocols

Polynomial Multiplication

Input: Shares of $p(x), q(x) \in F[x]$, $\deg(p) = \deg(q) = n$.
Output: Shares of $h(x) = p(x) \times q(x)$.

- 1 Parties compute shares of $p(i)$ and $q(i)$ for $0 \leq i \leq 2n$.
 - 2 Parties do $2n$ multiplications to get shares of $h(i) = p(i) \times q(i)$ for $0 \leq i \leq 2n$.
 - 3 Parties use Lagrange Interpolation to compute shares of $h(x)$.
- $O(1)$ round, $O(n)$ multiplication.
 - Efficient **multiset-union** protocol.

Polynomial Multiplication

Input: Shares of $p(x), q(x) \in F[x]$, $\deg(p) = \deg(q) = n$.
Output: Shares of $h(x) = p(x) \times q(x)$.

- 1 Parties compute shares of $p(i)$ and $q(i)$ for $0 \leq i \leq 2n$.
- 2 Parties do $2n$ multiplications to get shares of $h(i) = p(i) \times q(i)$ for $0 \leq i \leq 2n$.
- 3 Parties use Lagrange Interpolation to compute shares of $h(x)$.

- $O(1)$ round, $O(n)$ multiplication.
- Efficient **multiset-union** protocol.

Polynomial Multiplication

Input: Shares of $p(x), q(x) \in F[x]$, $\deg(p) = \deg(q) = n$.
Output: Shares of $h(x) = p(x) \times q(x)$.

- 1 Parties compute shares of $p(i)$ and $q(i)$ for $0 \leq i \leq 2n$.
 - 2 Parties do $2n$ multiplications to get shares of $h(i) = p(i) \times q(i)$ for $0 \leq i \leq 2n$.
 - 3 Parties use Lagrange Interpolation to compute shares of $h(x)$.
- $O(1)$ round, $O(n)$ multiplication.
 - Efficient **multiset-union** protocol.

Polynomial Multiplication

Input: Shares of $p(x), q(x) \in F[x]$, $\deg(p) = \deg(q) = n$.
Output: Shares of $h(x) = p(x) \times q(x)$.

- 1 Parties compute shares of $p(i)$ and $q(i)$ for $0 \leq i \leq 2n$.
- 2 Parties do $2n$ multiplications to get shares of $h(i) = p(i) \times q(i)$ for $0 \leq i \leq 2n$.
- 3 Parties use Lagrange Interpolation to compute shares of $h(x)$.

- $O(1)$ round, $O(n)$ multiplication.
- Efficient **multiset-union** protocol.

Polynomial Multiplication

Input: Shares of $p(x), q(x) \in F[x]$, $\deg(p) = \deg(q) = n$.
Output: Shares of $h(x) = p(x) \times q(x)$.

- 1 Parties compute shares of $p(i)$ and $q(i)$ for $0 \leq i \leq 2n$.
 - 2 Parties do $2n$ multiplications to get shares of $h(i) = p(i) \times q(i)$ for $0 \leq i \leq 2n$.
 - 3 Parties use Lagrange Interpolation to compute shares of $h(x)$.
- $O(1)$ round, $O(n)$ multiplication.
 - Efficient **multiset-union** protocol.

Polynomial Division with Remainder

Input: Shares of $f(x), g(x) \in F[x]$.

Output: Shares of $r(x), q(x)$; $f(x) = g(x)q(x) + r(x)$.

- $\deg(f) = n, \deg(g) = m, m \leq n$.
- Let $\text{rev}_k(a) = x^k a(1/x)$ for $a(x) \in F[x]$.
- Then:

$$\bullet \text{rev}_{n-m}(q) = \text{rev}_n(f) \times \text{rev}_m(g)^{-1} \bmod x^{n-m+1}$$

- Need to compute $\text{rev}_m(g)^{-1} \bmod x^{n-m+1}$ efficiently.
- Everything else is done using the multiplication protocol.

Polynomial Division with Remainder

Input: Shares of $f(x), g(x) \in F[x]$.

Output: Shares of $r(x), q(x)$; $f(x) = g(x)q(x) + r(x)$.

- $\deg(f) = n, \deg(g) = m, m \leq n$.
- Let $\text{rev}_k(a) = x^k a(1/x)$ for $a(x) \in F[x]$.
- Then:

$$\bullet \text{rev}_{n-m}(q) = \text{rev}_n(f) \times \text{rev}_m(g)^{-1} \bmod x^{n-m+1}$$

- Need to compute $\text{rev}_m(g)^{-1} \bmod x^{n-m+1}$ efficiently.
- Everything else is done using the multiplication protocol.

Polynomial Division with Remainder

Input: Shares of $f(x), g(x) \in F[x]$.

Output: Shares of $r(x), q(x)$; $f(x) = g(x)q(x) + r(x)$.

- $\deg(f) = n, \deg(g) = m, m \leq n$.
- Let $\text{rev}_k(a) = x^k a(1/x)$ for $a(x) \in F[x]$.
- Then:

$$\bullet \text{rev}_{n-m}(q) = \text{rev}_n(f) \times \text{rev}_m(g)^{-1} \text{ mod } x^{n-m+1}$$

- Need to compute $\text{rev}_m(g)^{-1} \text{ mod } x^{n-m+1}$ efficiently.
- Everything else is done using the multiplication protocol.

Polynomial Division with Remainder

Input: Shares of $f(x), g(x) \in F[x]$.

Output: Shares of $r(x), q(x)$; $f(x) = g(x)q(x) + r(x)$.

- $\deg(f) = n, \deg(g) = m, m \leq n$.
- Let $\text{rev}_k(a) = x^k a(1/x)$ for $a(x) \in F[x]$.
- Then:

$$\bullet \text{rev}_{n-m}(q) = \text{rev}_n(f) \times \text{rev}_m(g)^{-1} \bmod x^{n-m+1}$$

- Need to compute $\text{rev}_m(g)^{-1} \bmod x^{n-m+1}$ efficiently.
- Everything else is done using the multiplication protocol.

Polynomial Division cont'd

Input: Parties share $f(x)$.

Output: Parties Share $f(x)^{-1} \bmod x^t$:

- Parties share a random $s(x) \in F[x]/x^t$.
- Parties compute shares of $f(x) \times s(x) \bmod x^t$.
- Parties publicly announce $f(x) \times s(x) \bmod x^t$.
- Parties compute $g(x) = (f(x) \times s(x))^{-1} \bmod x^t = s(x)^{-1} \times f(x)^{-1} \bmod x^t$.
- Parties compute shares of $f(x)^{-1} \bmod x^t = s(x) \times g(x) \bmod x^t$.

- $O(1)$ round and $O(n)$ multiplication.
- Efficient **Subset-Deletion** protocol.

Polynomial Division cont'd

Input: Parties share $f(x)$.

Output: Parties Share $f(x)^{-1} \bmod x^t$:

- 1 Parties share a random $s(x) \in F[x]/x^t$.
- 2 Parties compute shares of $f(x) \times s(x) \bmod x^t$.
- 3 Parties publicly announce $f(x) \times s(x) \bmod x^t$.
- 4 Parties compute $g(x) = (f(x) \times s(x))^{-1} \bmod x^t = s(x)^{-1} \times f(x)^{-1} \bmod x^t$.
- 5 Parties compute shares of $f(x)^{-1} \bmod x^t = s(x) \times g(x) \bmod x^t$.

- $O(1)$ round and $O(n)$ multiplication.
- Efficient **Subset-Deletion** protocol.

Polynomial Division cont'd

Input: Parties share $f(x)$.

Output: Parties Share $f(x)^{-1} \bmod x^t$:

- 1 Parties share a random $s(x) \in F[x]/x^t$.
- 2 Parties compute shares of $f(x) \times s(x) \bmod x^t$.
- 3 Parties publicly announce $f(x) \times s(x) \bmod x^t$.
- 4 Parties compute $g(x) = (f(x) \times s(x))^{-1} \bmod x^t = s(x)^{-1} \times f(x)^{-1} \bmod x^t$.
- 5 Parties compute shares of $f(x)^{-1} \bmod x^t = s(x) \times g(x) \bmod x^t$.

- $O(1)$ round and $O(n)$ multiplication.
- Efficient **Subset-Deletion** protocol.

Polynomial Division cont'd

Input: Parties share $f(x)$.

Output: Parties Share $f(x)^{-1} \bmod x^t$:

- 1 Parties share a random $s(x) \in F[x]/x^t$.
- 2 Parties compute shares of $f(x) \times s(x) \bmod x^t$.
- 3 Parties publicly announce $f(x) \times s(x) \bmod x^t$.
- 4 Parties compute $g(x) = (f(x) \times s(x))^{-1} \bmod x^t = s(x)^{-1} \times f(x)^{-1} \bmod x^t$.
- 5 Parties compute shares of $f(x)^{-1} \bmod x^t = s(x) \times g(x) \bmod x^t$.

- $O(1)$ round and $O(n)$ multiplication.
- Efficient **Subset-Deletion** protocol.

Polynomial Division cont'd

Input: Parties share $f(x)$.

Output: Parties Share $f(x)^{-1} \bmod x^t$:

- 1 Parties share a random $s(x) \in F[x]/x^t$.
- 2 Parties compute shares of $f(x) \times s(x) \bmod x^t$.
- 3 Parties publicly announce $f(x) \times s(x) \bmod x^t$.
- 4 Parties compute $g(x) = (f(x) \times s(x))^{-1} \bmod x^t = s(x)^{-1} \times f(x)^{-1} \bmod x^t$.
- 5 Parties compute shares of $f(x)^{-1} \bmod x^t = s(x) \times g(x) \bmod x^t$.

- $O(1)$ round and $O(n)$ multiplication.
- Efficient **Subset-Deletion** protocol.

Polynomial Division cont'd

Input: Parties share $f(x)$.

Output: Parties Share $f(x)^{-1} \bmod x^t$:

- 1 Parties share a random $s(x) \in F[x]/x^t$.
- 2 Parties compute shares of $f(x) \times s(x) \bmod x^t$.
- 3 Parties publicly announce $f(x) \times s(x) \bmod x^t$.
- 4 Parties compute $g(x) = (f(x) \times s(x))^{-1} \bmod x^t = s(x)^{-1} \times f(x)^{-1} \bmod x^t$.
- 5 Parties compute shares of $f(x)^{-1} \bmod x^t = s(x) \times g(x) \bmod x^t$.

- $O(1)$ round and $O(n)$ multiplication.
- Efficient **Subset-Deletion** protocol.

More Protocols

- Multiplying *many* polynomials

Input: Parties share $f_1, f_2, \dots, f_l \in F[x]$.
Output: Parties share $\prod_{i=1}^l f_i$.

- Polynomial Interpolation.

Input: Parties share $x_i, y_i \in F$ for $i \in \{1, \dots, n\}$.
Output: Parties share $f \in F[x]$, where $f(x_i) = y_i$ for $i \in \{1, \dots, n\}$.

-

More Protocols

- Multiplying *many* polynomials

Input: Parties share $f_1, f_2, \dots, f_l \in F[x]$.
Output: Parties share $\prod_{i=1}^l f_i$.

- Polynomial Interpolation.

Input: Parties share $x_i, y_i \in F$ for $i \in \{1, \dots, n\}$.
Output: Parties share $f \in F[x]$, where $f(x_i) = y_i$ for $i \in \{1, \dots, n\}$.

More protocols

- Are two polynomials coprime?

Input: Parties share $f(x), g(x) \in F[x]$.

Output: Parties share the bit b ; $b = 1$ if $\gcd(f, g) = 1$, $b = 0$ otherwise.

- gcd of two polynomials

Input: Parties share $f(x), g(x) \in F[x]$.

Output: Parties share $\gcd(f, g)$.



- Please see the paper for more detail.

More protocols

- Are two polynomials coprime?

Input: Parties share $f(x), g(x) \in F[x]$.

Output: Parties share the bit b ; $b = 1$ if $\gcd(f, g) = 1$, $b = 0$ otherwise.

- gcd of two polynomials

Input: Parties share $f(x), g(x) \in F[x]$.

Output: Parties share $\gcd(f, g)$.

- Please see the paper for more detail.

More protocols

- Are two polynomials coprime?

Input: Parties share $f(x), g(x) \in F[x]$.

Output: Parties share the bit b ; $b = 1$ if $\gcd(f, g) = 1$, $b = 0$ otherwise.

- gcd of two polynomials

Input: Parties share $f(x), g(x) \in F[x]$.

Output: Parties share $\gcd(f, g)$.

- Please see the paper for more detail.

Summary

- Protocols for polynomial operations.
- Improving on general MPC solutions.
- Solutions for computational and information-theoretic settings.
- Applications to Privacy-preserving set operations.

Questions

Thank you!
Questions?

