Chair of
Applied Cryptography

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

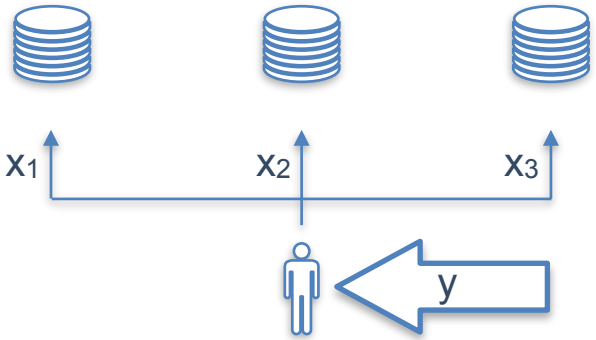
Homomorphic Secret Sharing for Low Degree Polynomials

Russell W. F. Lai, **Giulio Malavolta**, and Dominique Schröder

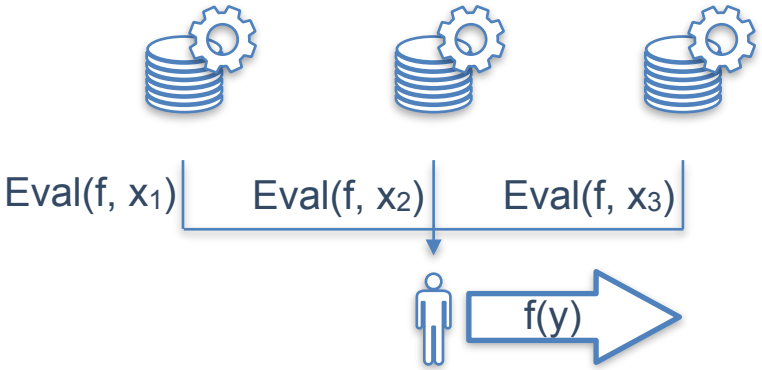
Friedrich-Alexander University Erlangen-Nürnberg

Homomorphic Secret Sharing

A secret-sharing scheme allows a client to share his data across several servers



A secret-sharing scheme is **homomorphic** if the servers can compute functions over the shares and the client can reconstruct the function output

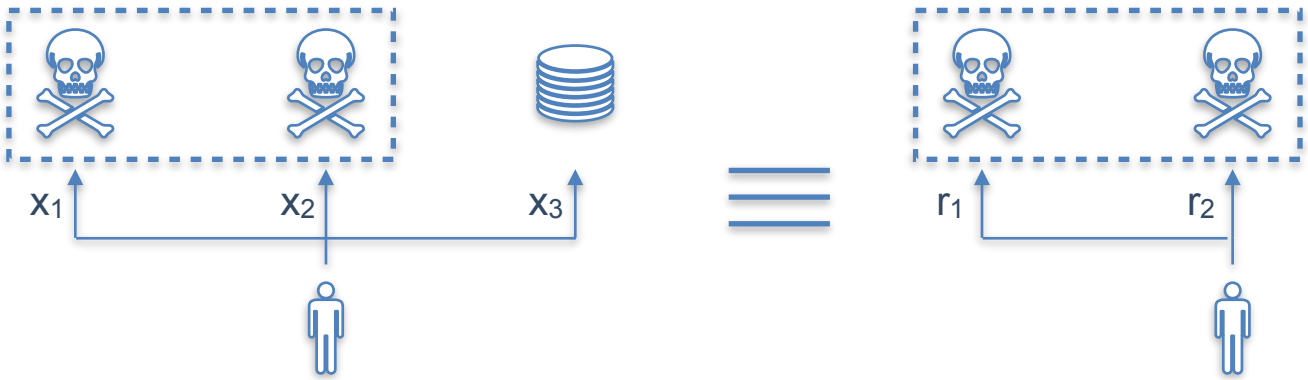


Efficiency: The communication must be **independent** from the size of the function

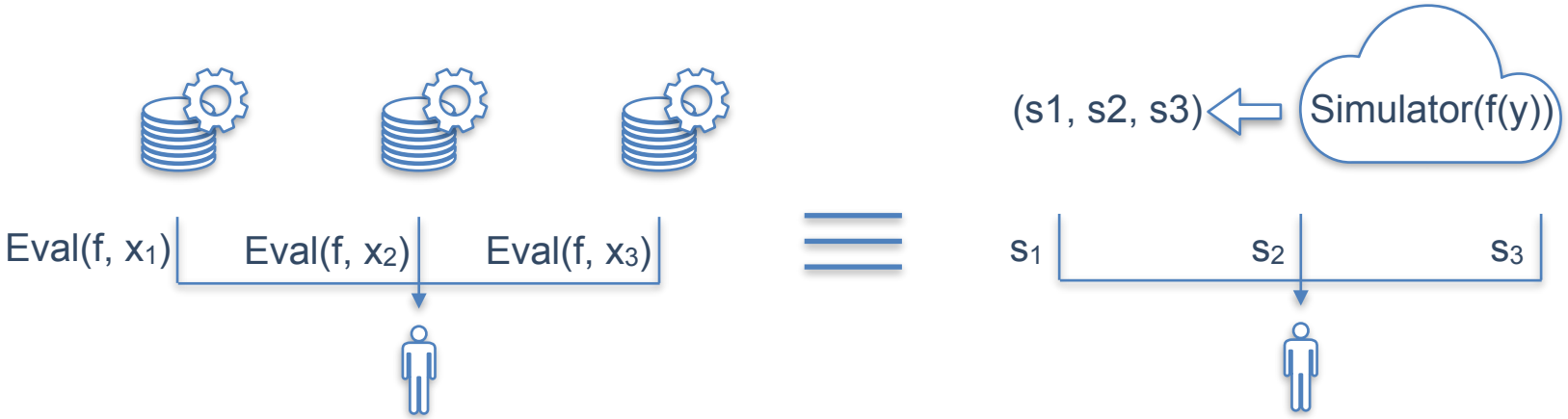
Analogy: “Distributed” FHE

Security Definitions

1) A corrupt set of servers should not learn anything about the data



2) The client should learn nothing beyond the **output** of the function



State-of-the-art

	# Clients	# Servers	# Corrupt	Function	Assump.	Model
[Sha79]	n	m	m - 1	$poly^{(m-1)}$	-	plain
[Ben87]	n	m	m - 1	affine	-	plain
[DHR+16]	n	m	m	P	LWE	plain
[BGI15]	n	2	1	point	OWF	plain
[BGI16]	n	2	1	NC^1	DDH	PKI (mult.)
[CF15]	n	2	1	$poly^{2k}$	k-HE	plain

Our Results

Theorem: For all integers n > 0, k >= 0, and m = O(log(n) / loglog(n)), if there exists a k-homomorphic public-key encryption scheme, then there exists a n-client m-server homomorphic secret sharing for polynomials of degree (k + 1) * m - 1.

Homomorphic encryption for

k = 1 => (lifted) ElGamal, Paillier
k = 2 => [BGN05] Pairings
k > 2 => Lattices

Example: Homomorphic secret-sharing for **degree-3 polynomials** from DDH (setting k = 1 and m = 2)

Randomized Encodings

Our Results

	# Clients	# Servers	# Corrupt	Function	Assump.	Model
[Sha79]	n	m	m - 1	$poly^{(m-1)}$	-	plain
[Ben87]	n	m	m - 1	affine	-	plain
[DHR+16]	n	m	m	P	LWE	plain
[BGI15]	n	2	1	point	OWF	plain
[BGI16]	n	2	1	NC^1	DDH	PKI (mult.)
[CF15]	n	2	1	$poly^{2k}$	k-HE	plain
THIS	n	m	1	$poly^{(k+1)m-1}$	k-HE	plain

Toy Example

A 2-server scheme from linearly homomorphic encryption to computed the function f(x,y,z) = x * y * z.

Sharing: Encode each input as

$$\begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \\ z_1 & z_2 \end{bmatrix} \text{ such that } \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \\ z_1 & z_2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Define the shares as

$$\begin{bmatrix} \text{Enc}(x_1), x_2 \\ \text{Enc}(y_1), y_2 \\ \text{Enc}(z_1), z_2 \end{bmatrix} \text{ and } \begin{bmatrix} x_1, \text{Enc}(x_2) \\ y_1, \text{Enc}(y_2) \\ z_1, \text{Enc}(z_2) \end{bmatrix}$$

Toy Example (continued)

Eval: Expand the product

$$x * y * z = (x_1 + x_2)\,(y_1 + y_2)\,(z_1 + z_2) = \Sigma_i\,\Sigma_j\,\Sigma_l\, x_i\, y_j\, z_l$$

By the pigeonhole principle, for all (i, j, l) there exists at least one server that can compute the corresponding monomial by treating the plaintexts as constants, e.g.,

$$Enc(x_1) * (y_2 * z_2) = Enc(x_1 * y_2 * z_2)$$

Let A be the set of monomials computable by the first server and B the set computable by the second

$$c_1 = Enc(\Sigma_A\, m_A) \qquad \text{and} \qquad c_2 = Enc(\Sigma_B\, m_B)$$

Toy Example (continued)

Decode: Decrypt c_1 and c_2 and sum the plaintexts to obtain

$$\Sigma_A\, m_A + \Sigma_B\, m_B = \Sigma_i\, \Sigma_j\, \Sigma_l\, x_i\, y_j\, z_l = x * y * z$$

Increasing the degree: Increasing the number of servers also increases the degree of the polynomial the can be computed, setting the i-th share as

$$\begin{bmatrix} x_1, \ldots, x_{i-1}, \mathrm{Enc}(x_i), x_{i+1}, \ldots x_m \\ \ldots \\ z_1, \ldots, z_{i-1}, \mathrm{Enc}(z_i), z_{i+1}, \ldots z_m \end{bmatrix}$$

allows one to compute polynomials of degree m-1

Main Construction and Efficiency

Important to choose a suitable **Split** function to split the monomials across the servers to avoid duplicates

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.KGen}(1^\lambda)$

return $(\mathsf{pk}, \mathsf{sk})$

$y \leftarrow \mathsf{Dec}(\mathsf{sk}, y_1, \ldots, y_m)$

$c \leftarrow \mathsf{HE.Eval}(\mathsf{pk}, f_{\mathrm{Add}}, (y_1, \ldots, y_m))$

$y \leftarrow \mathsf{HE.Dec}(\mathsf{sk}, c)$

return y

$(s_{i,1}, \ldots, s_{i,m}) \leftarrow \mathsf{Share}(\mathsf{pk}, i, x_i)$

$(x_{i,1}, \ldots, x_{i,m}) \leftarrow R^m \; s.t. \sum_{j \in [m]} x_{i,j} = x_i$

$(z_{i,1}, \ldots, z_{i,m}) \leftarrow R^m \; s.t. \sum_{j \in [m]} z_{i,j} = 0$

$\tilde{x}_{i,j} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, x_{i,j}) \; \forall j \in [m]$

$x_i^{-j} := (x_{i,1}, \ldots, x_{i,j-1}, x_{i,j+1}, \ldots, x_{i,m})$

$s_{i,j} := (x_i^{-j}, \tilde{x}_{i,j}, z_{i,j})$

return $(s_{i,1}, \ldots, s_{i,m})$

$y_j \leftarrow \mathsf{Eval}(j, f, (s_{1,j}, \ldots, s_{n,j}))$

parse $s_{i,j}$ **as** $(x_i^{-j}, \tilde{x}_{i,j}, z_{i,j})$

$f_j := \mathsf{Split}_d(j, f, (x_1^{-j}, \ldots, x_n^{-j})) + \sum_{i \in [n]} z_{i,j}$

$y_j \leftarrow \mathsf{HE.Eval}(\mathsf{pk}, f_j, (\tilde{x}_{1,j}, \ldots, \tilde{x}_{n,j}))$

return y_j

Greedy: Each server computes as many monomials as he can (taking care of avoiding duplicates)

=> Efficient for m = O(log(n) / loglog(n))

Fair: Weights are assigned to each monomial

=> For k = 1, efficient for m = O(log(n))

Multi-Key and Collusion Resistance

Multi-Key: Our construction naturally extends to support function evaluation over shares from different client

=> Replace the homomorphic encryption with a multi-key homomorphic encryption

Collusion Resistance: The vanilla version of our construction is resilient against the corruption of a single server

=> We show how to trade expressiveness for corruption threshold t

Applications

Our scheme has several appealing features:

- Simple assumptions
- Perfect correctness
- Efficient output client

Outsourced Computation: Our scheme can be used off-the-shelf to compute statistical measure over encrypted data (e.g., mean and variance)

Multi-Server PIR: An m-server PIR with communication dominated by a factor $|DB|/2^d$ (where d depends on k, m, and t)

Round-Optimal MPC: Applying the generic transform of [BGI+18] we can turn a homomorphic secret sharing for degree-3 polynomials into a 2-round semi-honest MPC (in a weak corruption model)

Open Problems

1) Other applications of our techniques?

2) Increasing the degree of the polynomials? Better Split functions? Bootstrapping?

3) Homomorphic secret-sharing for P from more assumptions? (only known from lattices)

Thank you for your attention!

Questions?