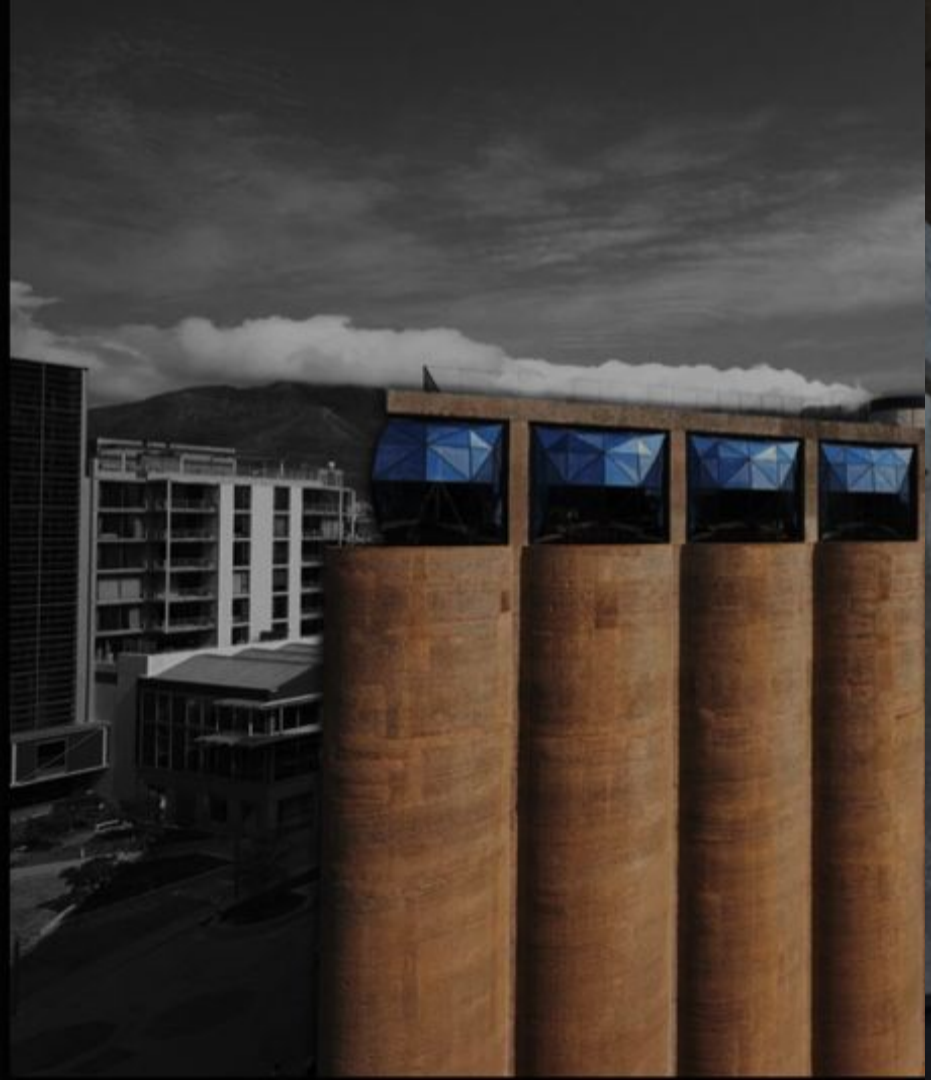
# Scalable Privacy-Preserving Computing with High Numerical Precision

Dimitar Jetchev

Chief Technology Officer
Inpher Inc./Sarl

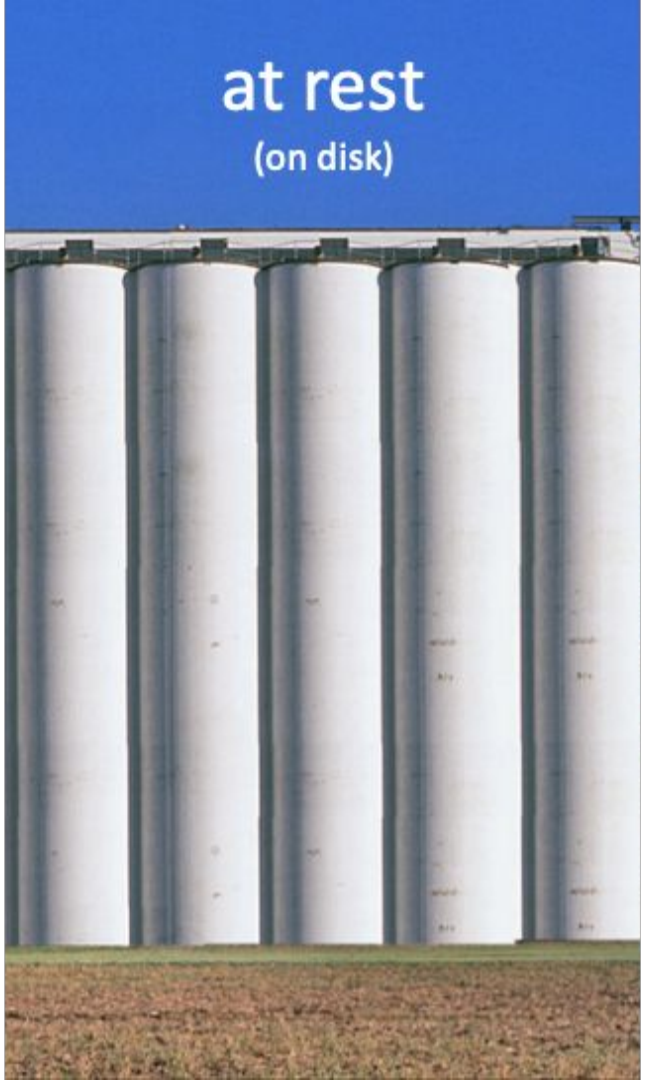
inpher

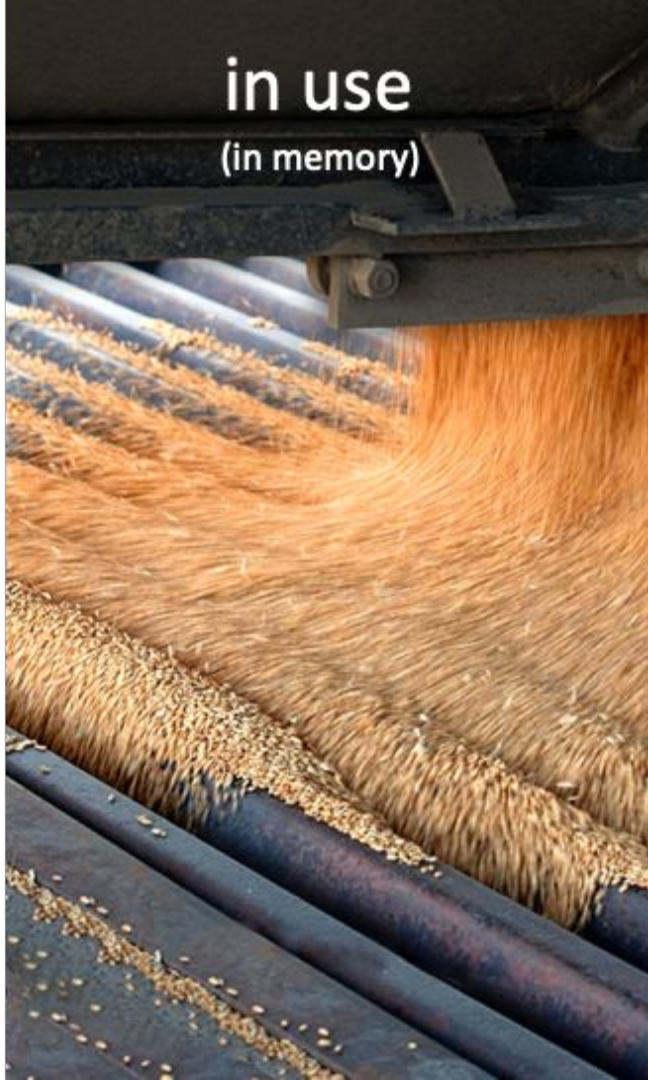


The biggest obstacle to using advanced data analysis isn't skill base or technology; it's plain old **access to the data.**

- Edd Wilder-James
Google TensorFlow

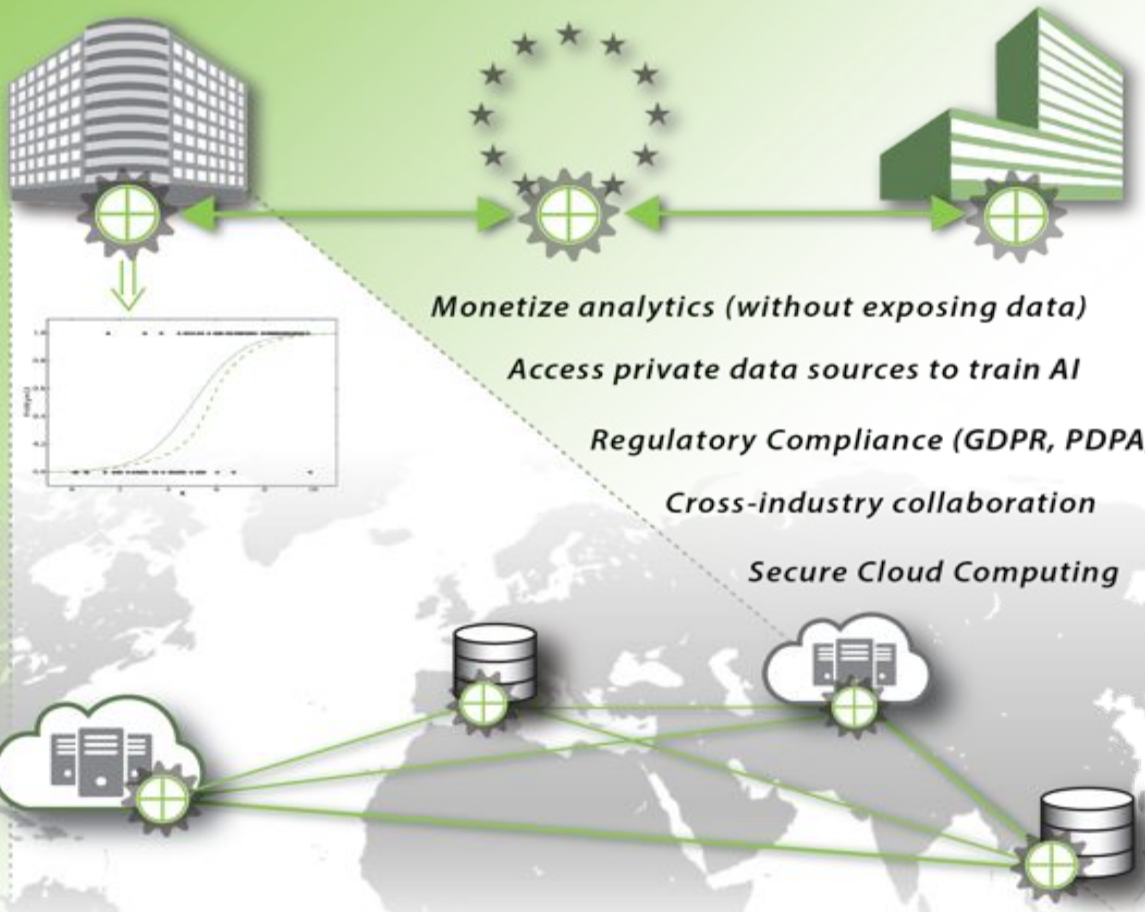

in transit
(https://)
©Inpher, Inc. 2018


at rest
(on disk)


in use
(in memory)

# Opportunities beyond data security.

+ Governments
+ Organizations
+ Jurisdictions
+ Departments
+ Datacenters



*Monetize analytics (without exposing data)*

*Access private data sources to train AI*

*Regulatory Compliance (GDPR, PDPA)*

*Cross-industry collaboration*

*Secure Cloud Computing*

THE WALL STREET JOURNAL.

Subscribe Now | Sign In

Home World U.S. Politics Economy **Business** Tech Markets Opinion Life & Arts Real Estate WSJ. Magazine Search

Farmers on Drenched Land Confront Tough Choice on Planting

More Universities Find Full-Time M.B.A. Programs Aren't Worth It

Bristol R&D Chief to Leave After Celgene Acquisition

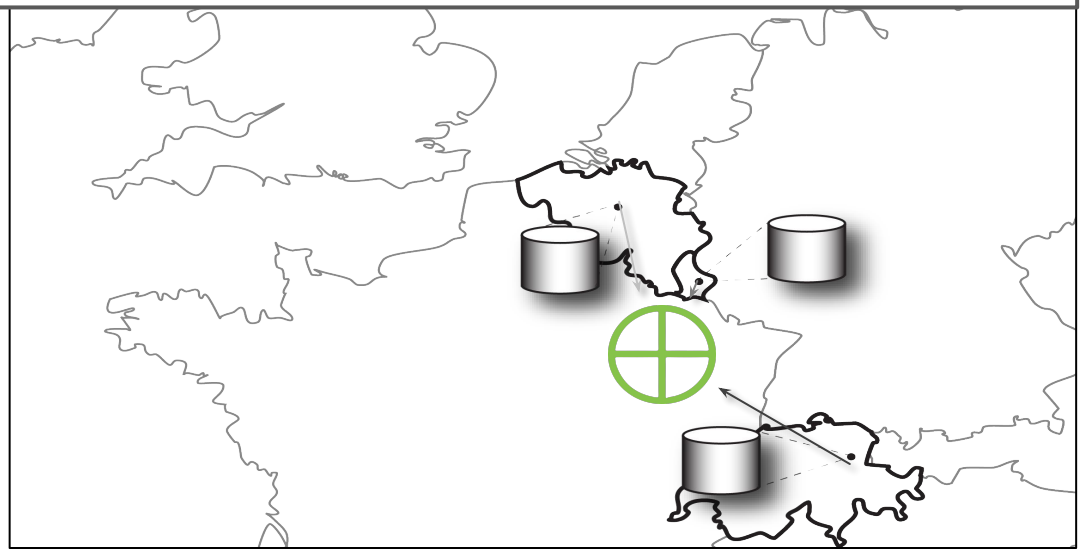
Jaguar Land Rover's Turnaround Ends in China

Sale Campbel Strength

CIO JOURNAL.

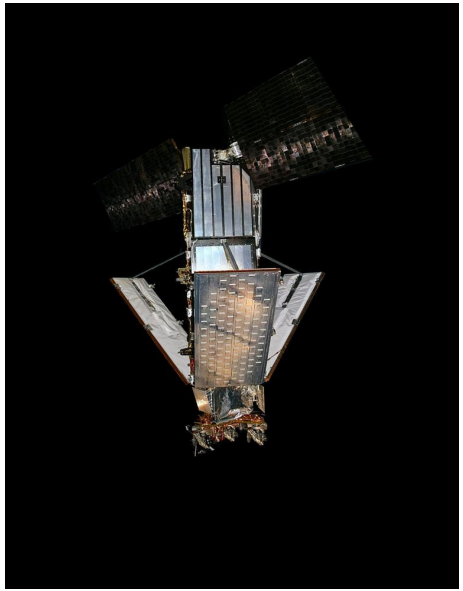
# ING Belgium Sees Opportunities for 'Secret' Sharing of Encrypted Data

Zero-knowledge computing would let companies analyze encrypted information without revealing any secret information

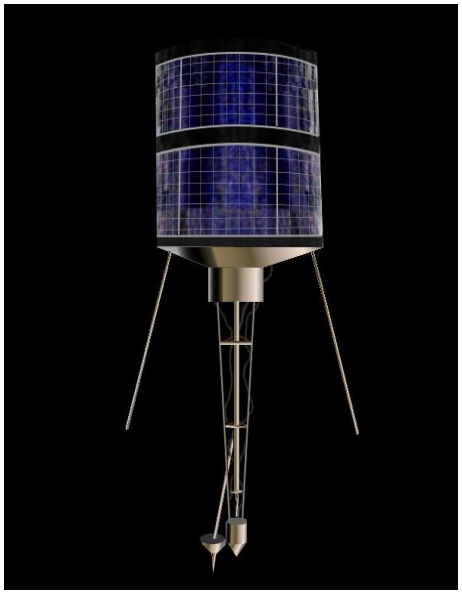


# High-Precision Use Cases

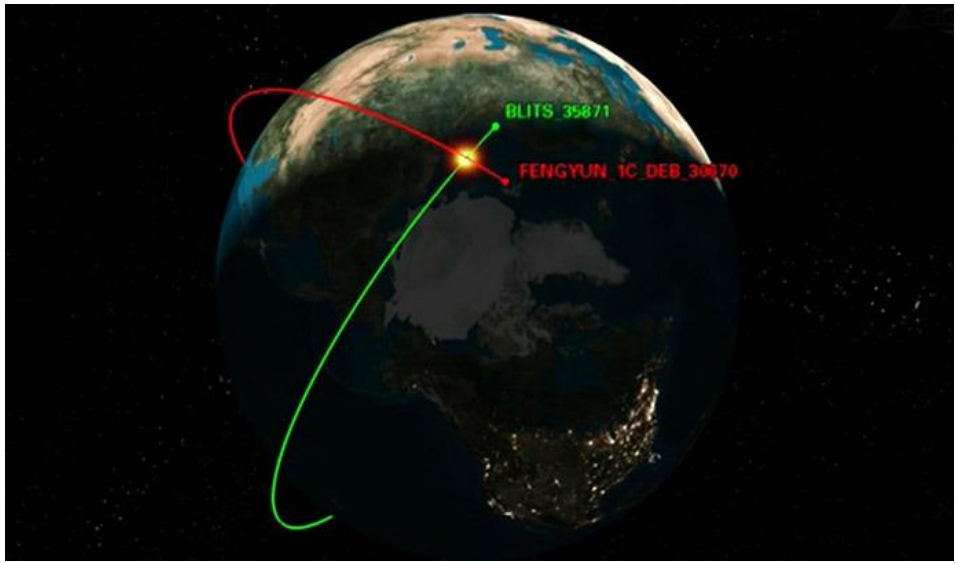
# Iridium 33 and Kosmos-2251 Satellite Collision



- Collision - 2009
- 11,700 m/s
- 789 km above Syberia
- More than 2000 debris
- ISS special maneuvers



# High-Precision Privacy-Preserving Compute




- Predicting collisions of satellites
- Satellite trajectories are private
- Satellite operators nonetheless perform **conjunction analysis**
- Need to evaluate non-linear functions with high numerical precision

# Detecting Rare Events with Private Data

*see FC'18 paper: **High-Precision Privacy-Preserving Real-Valued Function Evaluation**

Problem: detect a rare event via a classification algorithm (e.g., a fraudulent activity or a rare market event)

- 1 out of 10,000 bank customers performs fraudulent activity
- A classifier predicting always negative has 99,99% accuracy

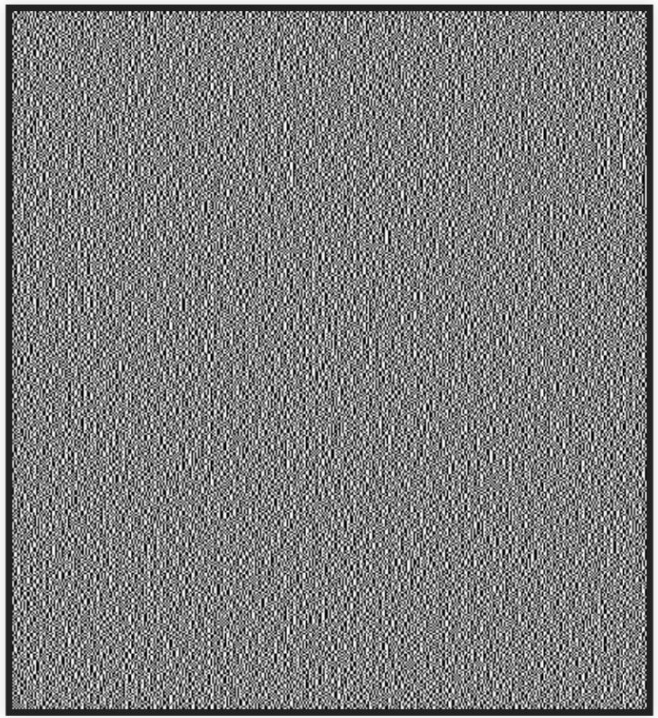
| | **PREDICTED 1** | **PREDICTED 0** |
|---|---|---|
| **TRUE 1** | $TP = 0$ | $FN = 1$ |
| **TRUE 0** | $FP = 0$ | $TN = 9{,}999$ |

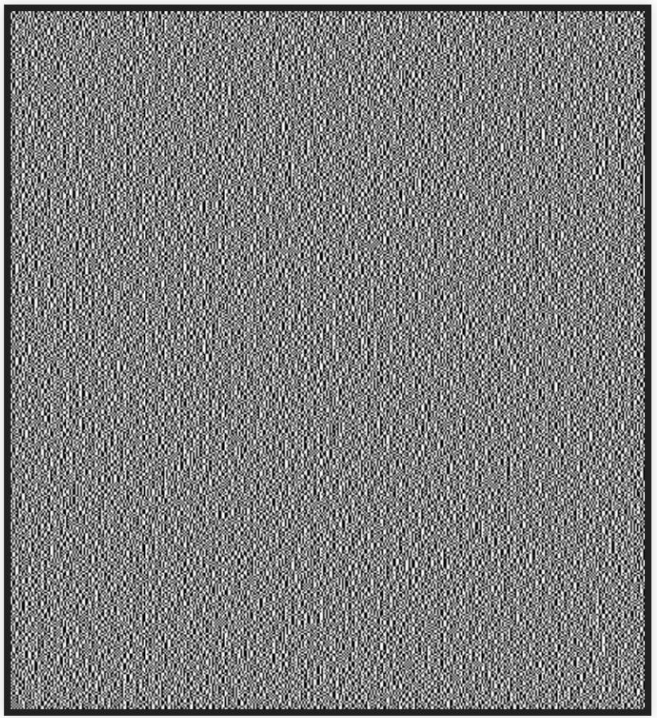
accuracy $\approx 1$, recall $= 0$, F1-score $= 0$

**For classifying rare events, numerical accuracy does matter!**

# Secure Multiparty Computation

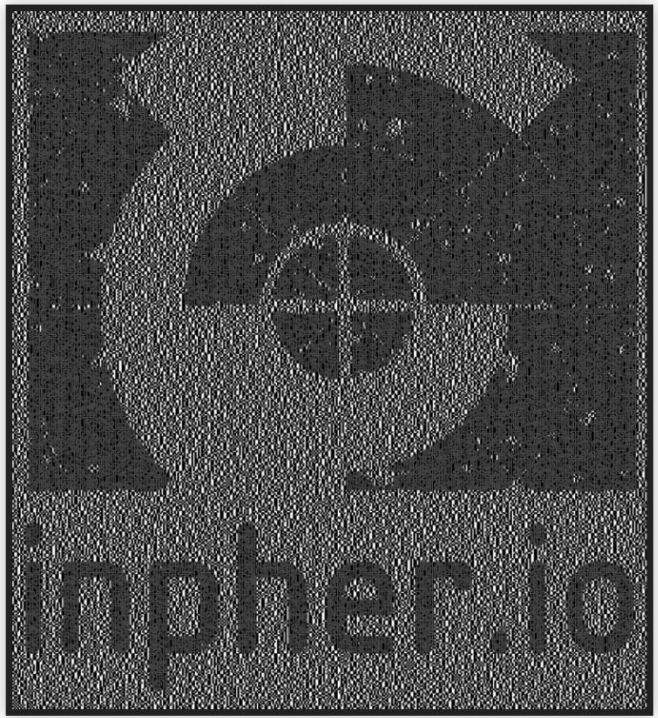
# Secret Shared Data





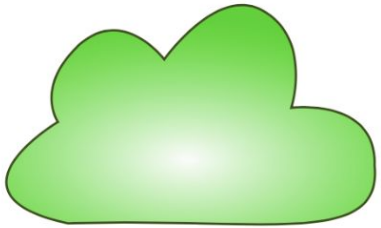
Reveal Secret Share

# Reveal Secret Shared Data

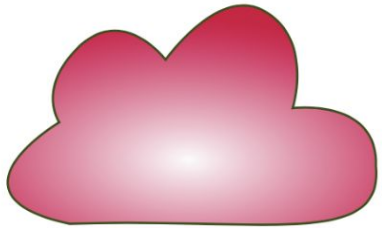



Reveal Secret Share

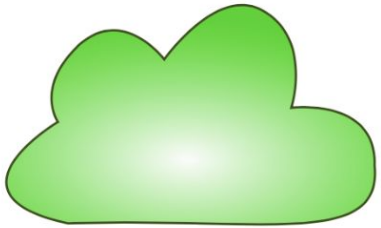
# Multiplications



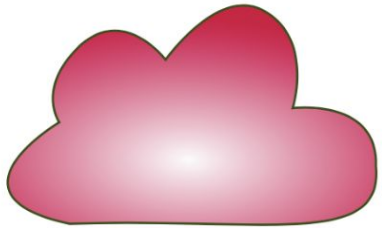
6 x 7



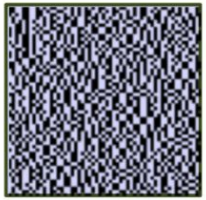
# Multiplications

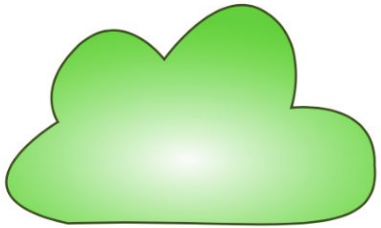


x 7

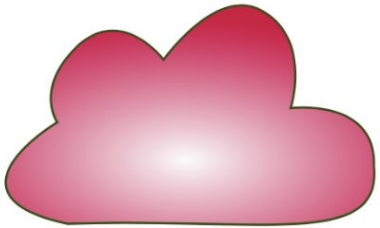


# Multiplications

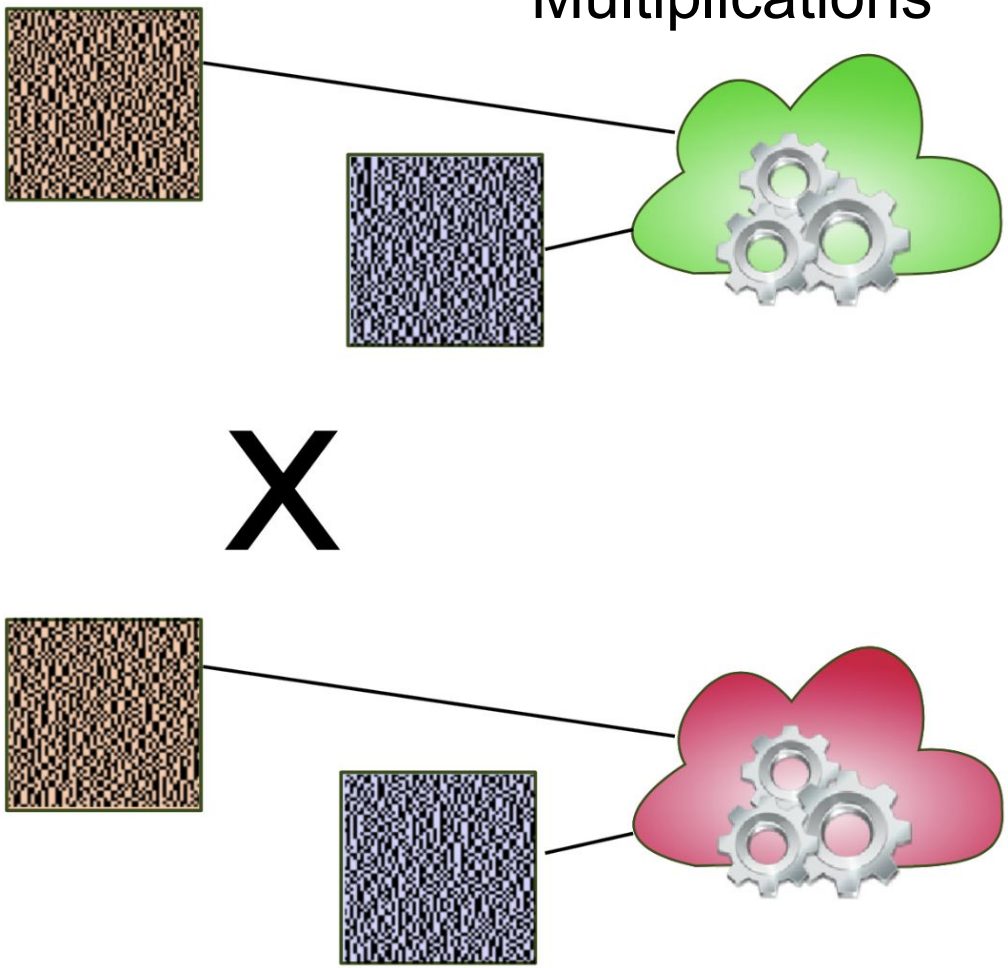




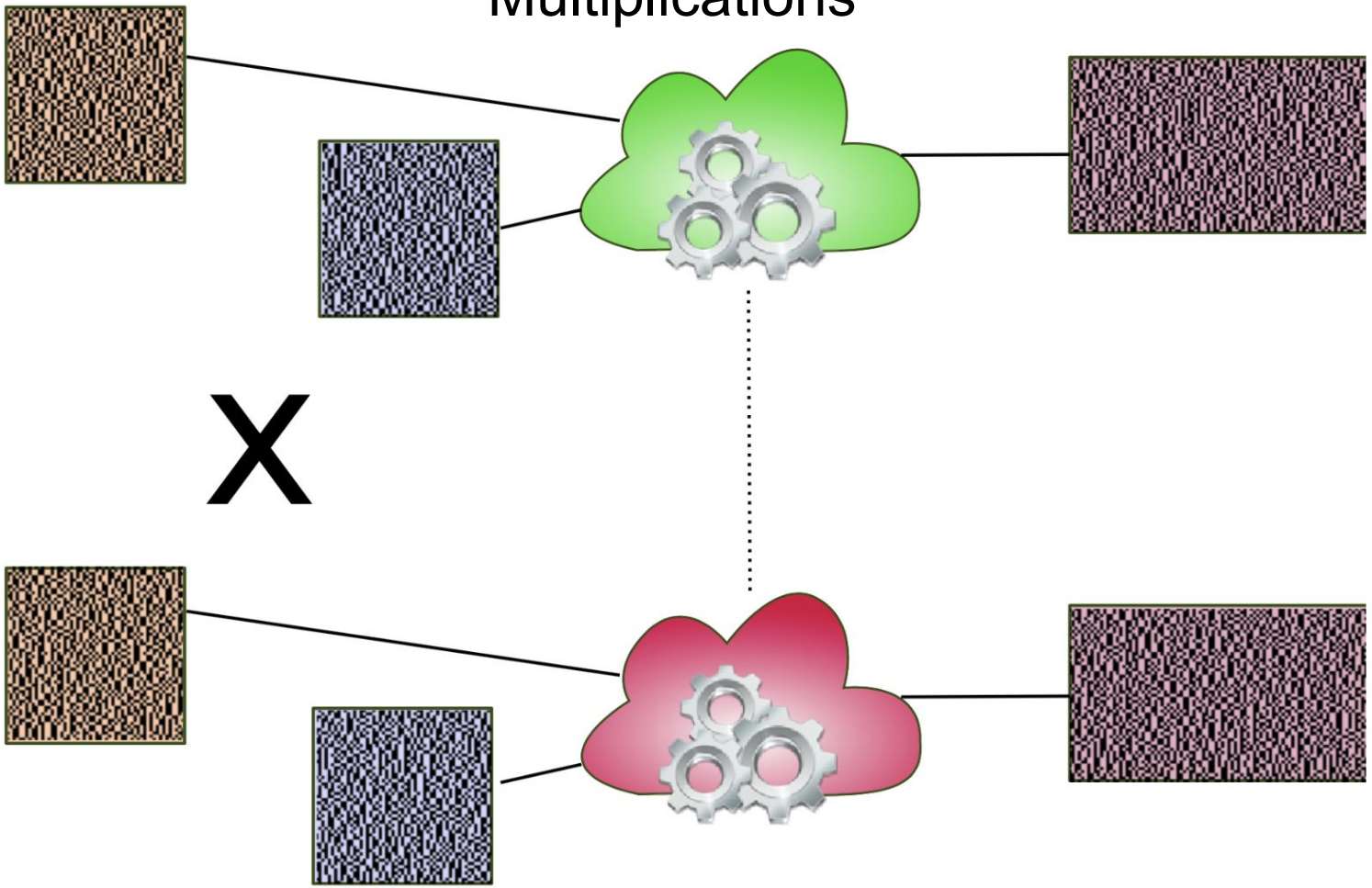
X



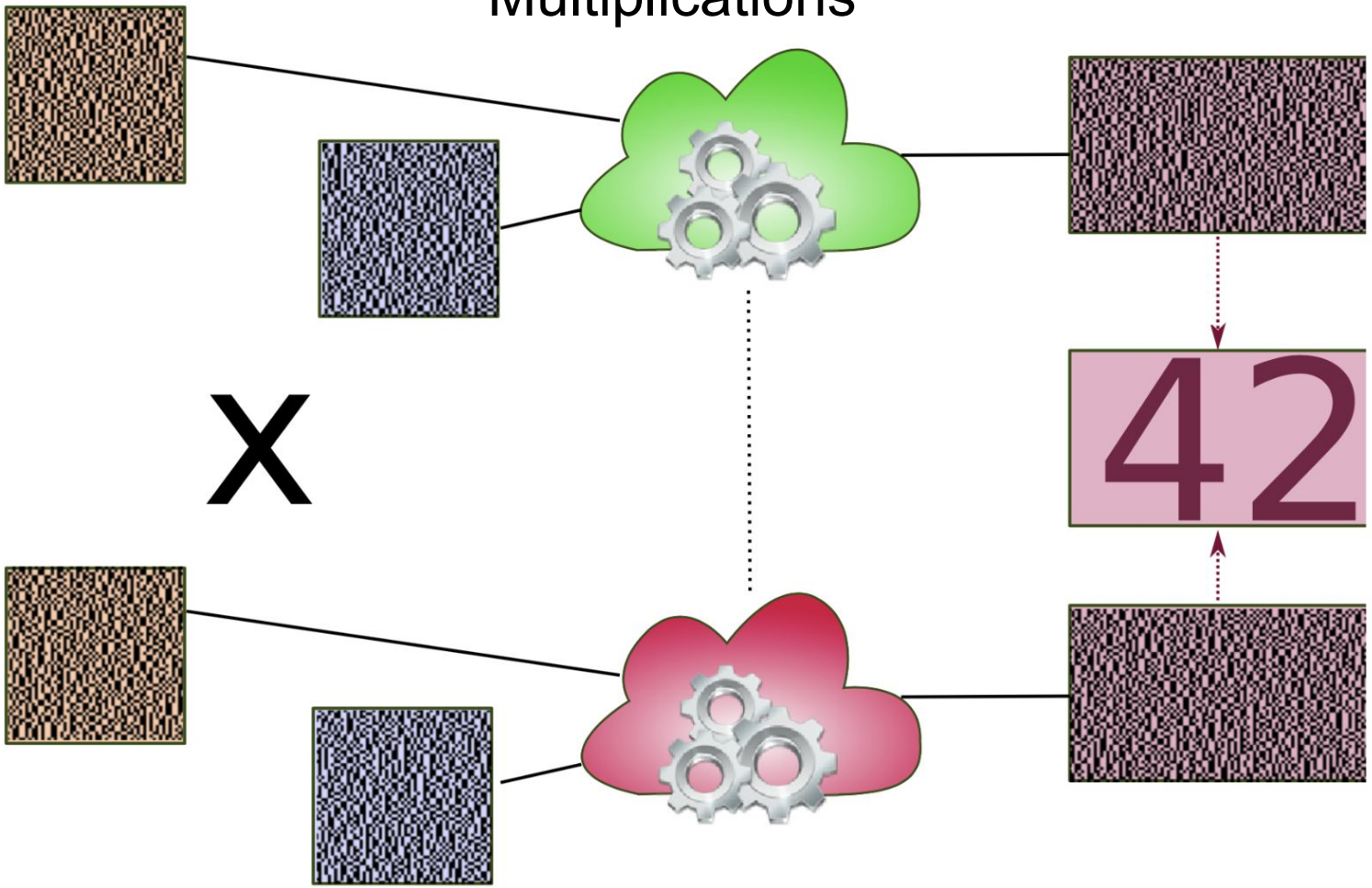
# Multiplications



# Multiplications



# Multiplications



# Beaver Multiplication

## Goal

- Given $x, y \in G_1, G_2$, compute $x \times y \in G_3$
  - where $\times : G_1 \times G_2 \to G_3$ is bilinear and public

# Beaver Multiplication

## Goal

- Given $x, y \in G_1, G_2$, compute $x \times y \in G_3$
  - where $\times : G_1 \times G_2 \to G_3$ is bilinear and public

## Offline phase (independent on the data value)

- Secret share $(\lambda, \mu, \lambda \times \mu)$ for random $\lambda, \mu$

# Beaver Multiplication

## Goal

- Given $x, y \in G_1, G_2$, compute $x \times y \in G_3$
  - where $\times : G_1 \times G_2 \to G_3$ is bilinear and public

## Offline phase (independent on the data value)

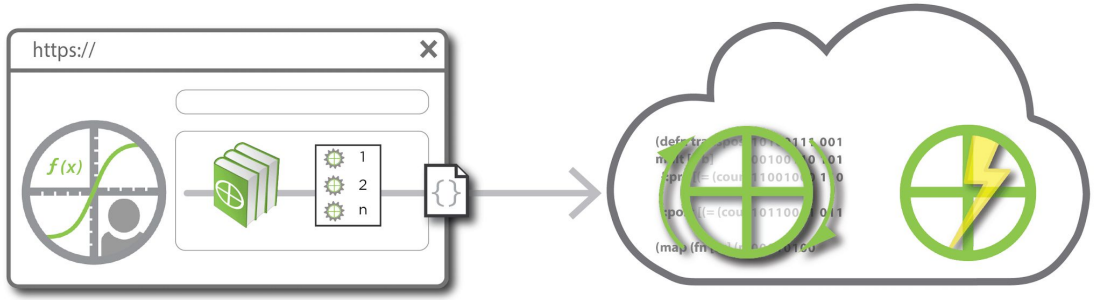
- Secret share $(\lambda, \mu, \lambda \times \mu)$ for random $\lambda, \mu$

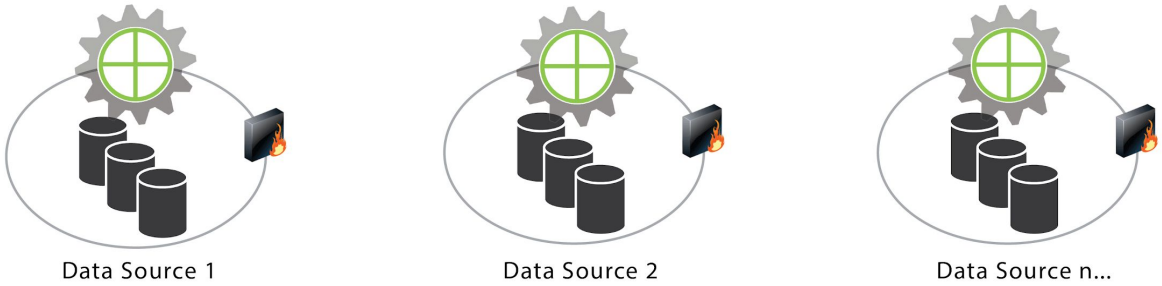
## Online phase

- Jointly reveal $a = x + \lambda$
- Jointly reveal $b = y + \mu$
- Now, everything becomes linear:
  $x \times y = a \times b - \lambda \times b - a \times \mu - \lambda \times \mu$
- And **never ever re-use** the same triplet twice!

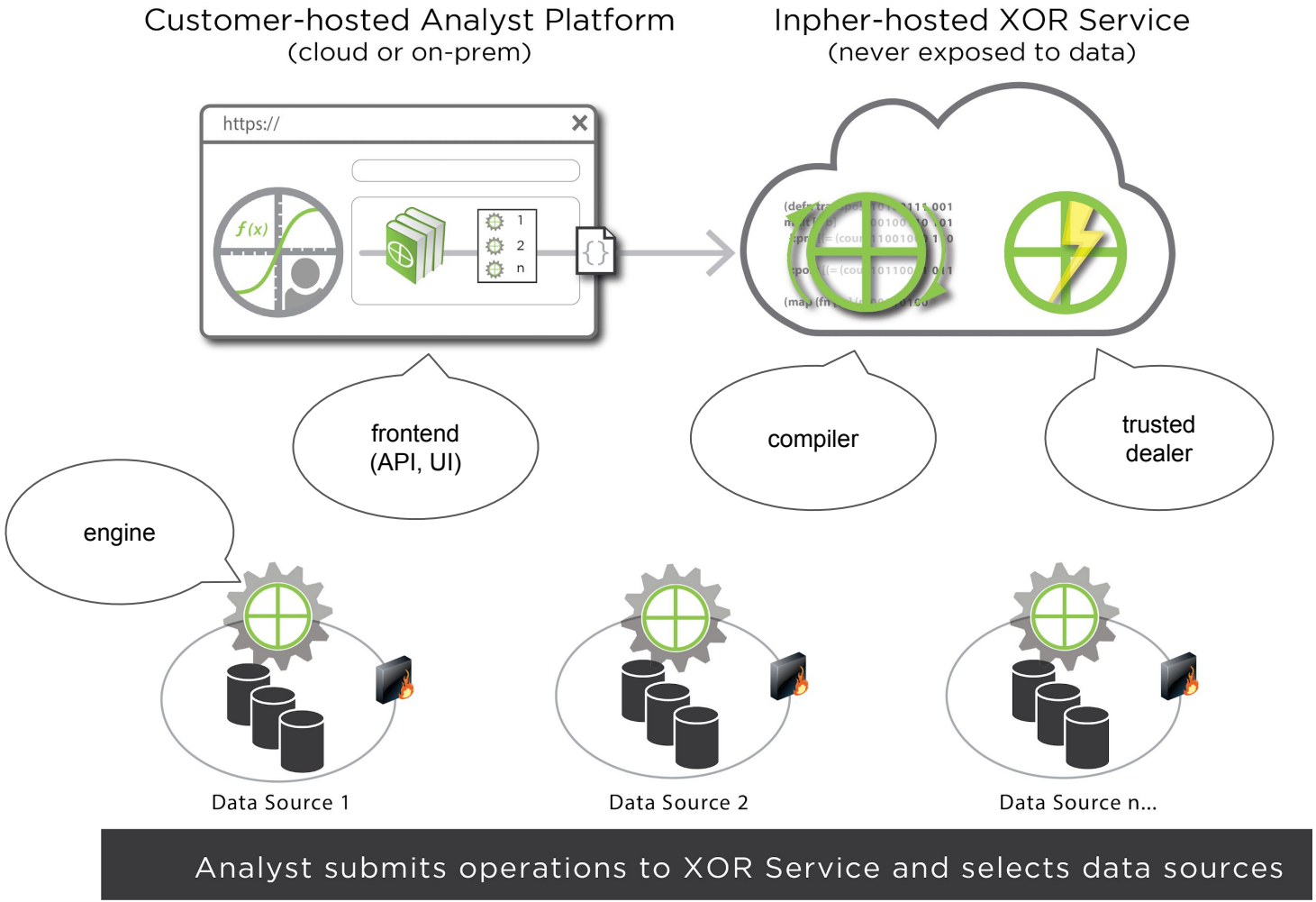
Customer-hosted Analyst Platform
(cloud or on-prem)

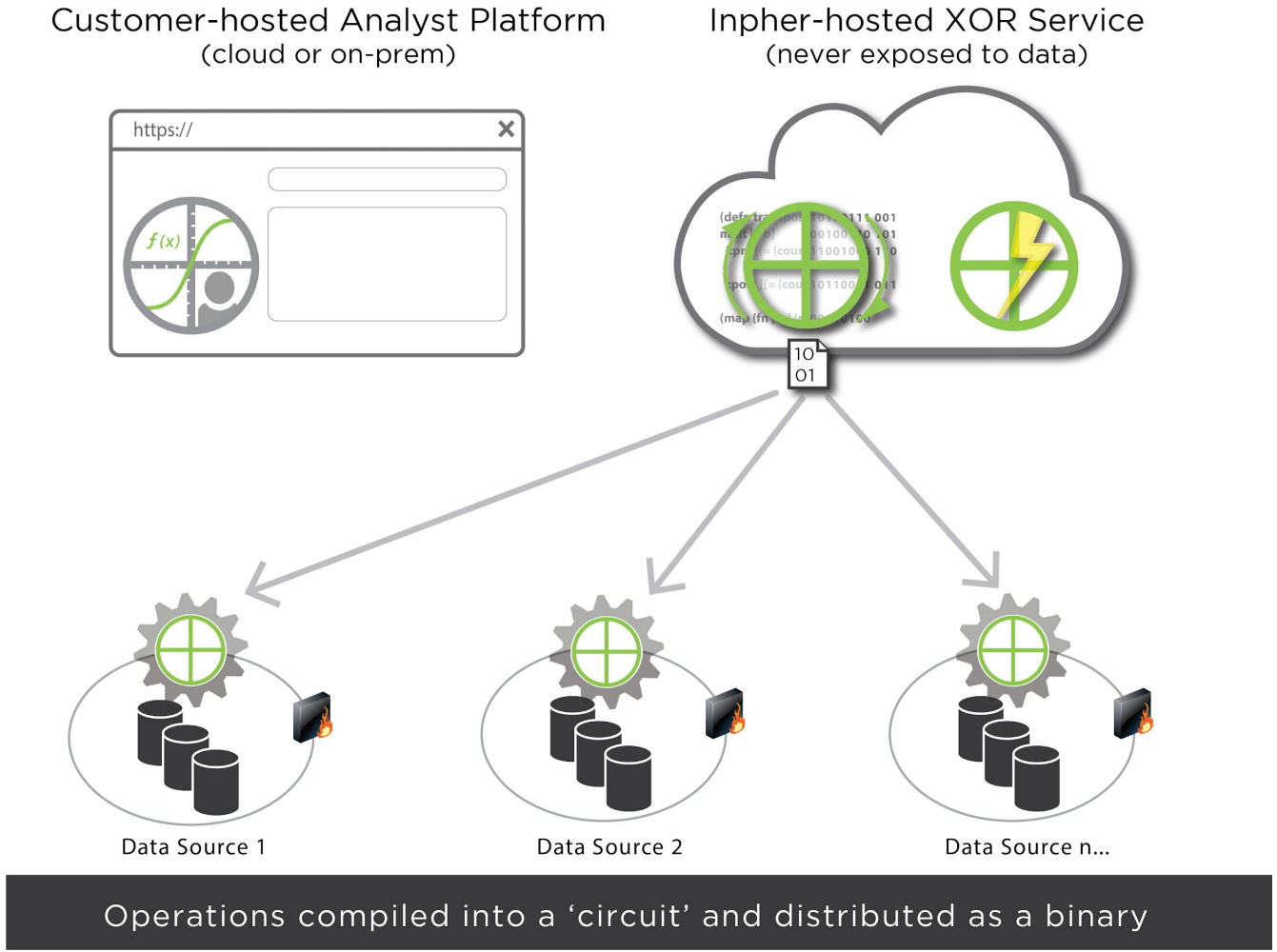
Inpher-hosted XOR Service
(never exposed to data)

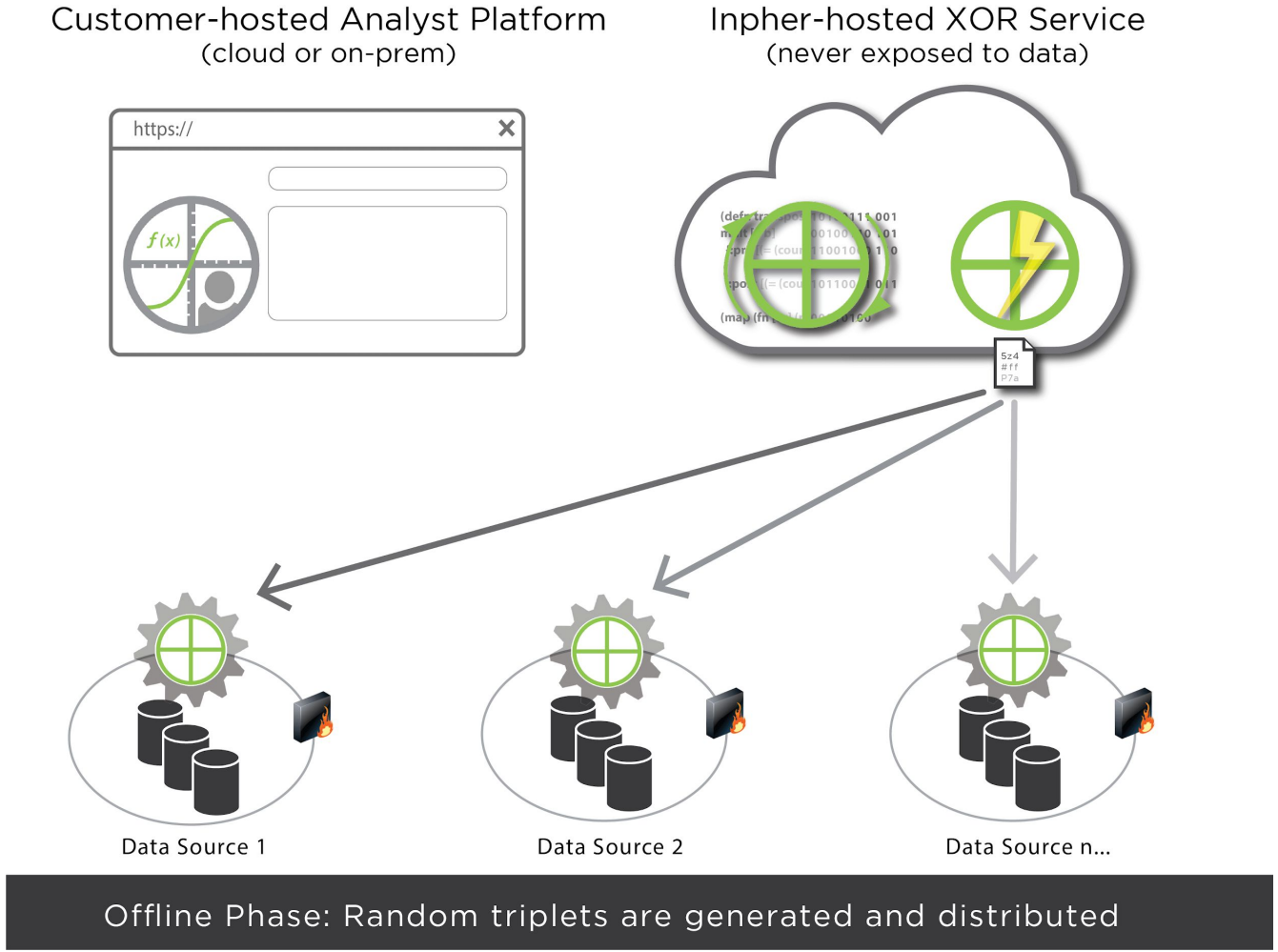


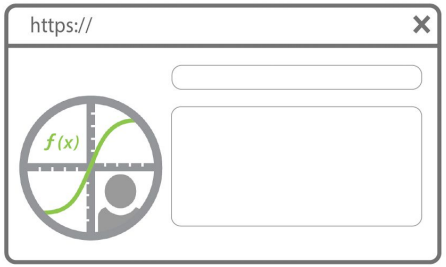

Analyst submits operations to XOR Service and selects data sources


Customer-hosted Analyst Platform
(cloud or on-prem)
Inpher-hosted XOR Service
(never exposed to data)
https://
f(x)
1
2
n
frontend
(API, UI)
compiler
trusted
dealer
engine
Data Source 1
Data Source 2
Data Source n...
Analyst submits operations to XOR Service and selects data sources

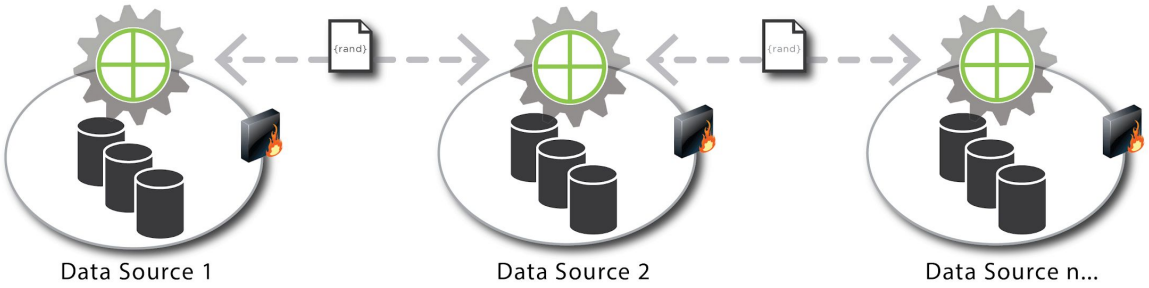

Customer-hosted Analyst Platform
(cloud or on-prem)
Inpher-hosted XOR Service
(never exposed to data)
https://
f(x)
10
01
Data Source 1
Data Source 2
Data Source n...
Operations compiled into a ‘circuit’ and distributed as a binary


Customer-hosted Analyst Platform
(cloud or on-prem)
Inpher-hosted XOR Service
(never exposed to data)
https://
f(x)
Data Source 1
Data Source 2
Data Source n...
Offline Phase: Random triplets are generated and distributed

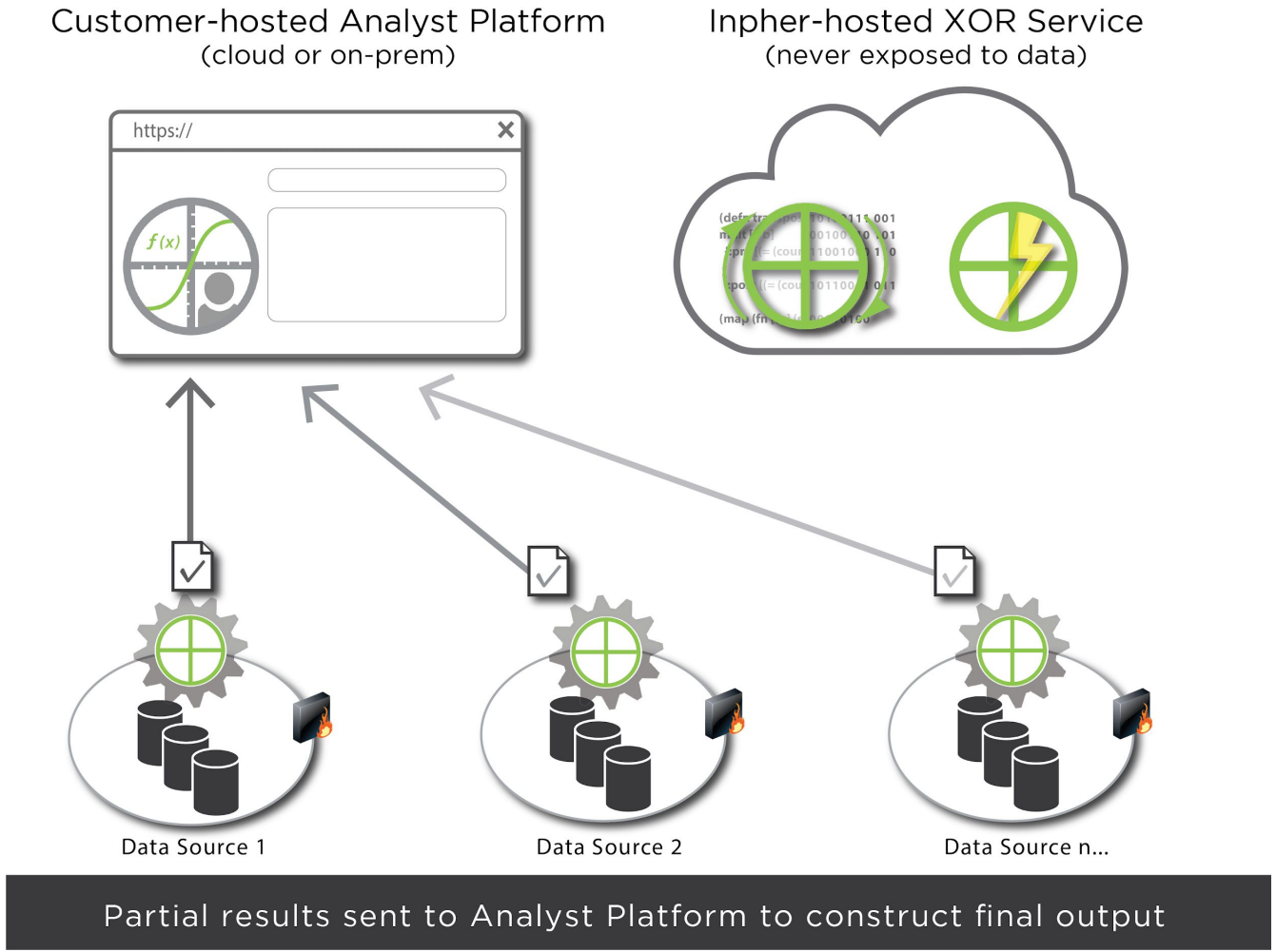
Customer-hosted Analyst Platform
(cloud or on-prem)




Inpher-hosted XOR Service
(never exposed to data)




Online Phase: Data sources secretly compute with random numbers


Customer-hosted Analyst Platform
(cloud or on-prem)
Inpher-hosted XOR Service
(never exposed to data)
https://
f(x)
Data Source 1
Data Source 2
Data Source n...
Partial results sent to Analyst Platform to construct final output

# Arithmetic with Real Numbers

# Computations with Real Numbers

## Floating-point representation of the reals

- Addition is generally inefficient compared to multiplication
- Boolean circuit approach is slow in MPC

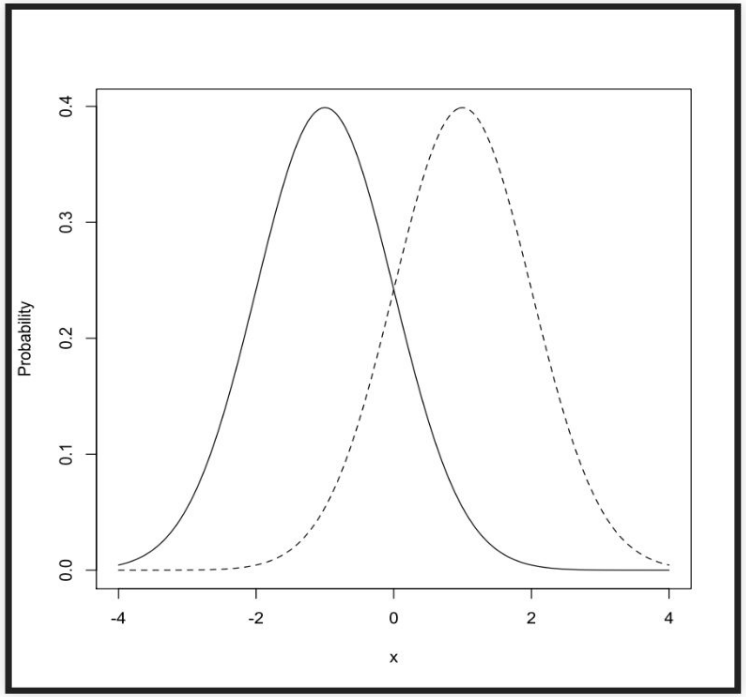
## Fixed-point representation

- Overflows often blindly destroy the result

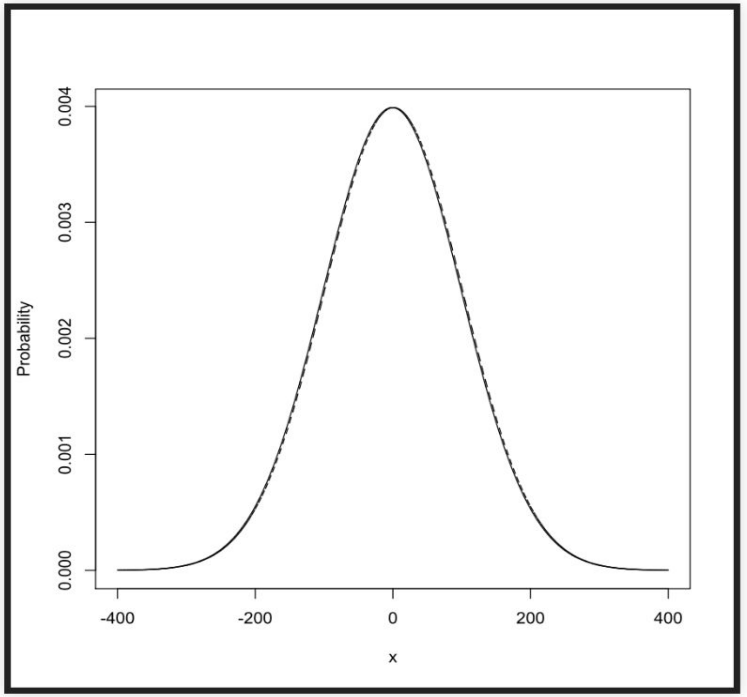
## Fixed-point arithmetic with native FP backend

- Pros
  - Overflows do not destroy the result
  - **float128** addition and multiplication: native on some hardware
- Cons
  - Small impact on security
  - Statistical masking instead of perfect masking

# Statistical Masking




Masking $x = \pm 1, \sigma = 1.$




Masking $x = \pm 1, \sigma = 100.$

# Disadvantages of Real Arithmetic with Floating Point Backend

- Large-depth arithmetic circuits require multi-precision floating-point numbers (masking sizes grow with depth)
- **Computational security** as opposed to **information-theoretic security**
- Memory overhead in both offline and online phases.
- Matrix operations over multi-precision floating types may be expensive

# Representing Real Numbers

## Floating-point representation

- $x = m \cdot 2^{\tau}$, $m \in 2^{-\rho} \cdot \mathbb{Z}$, $1/2 \leq |m| \leq 1$
- $\tau = \lceil \log_2 |x| \rceil$ - exponent; data dependent, not MPC-friendly, not public

## Fixed-point representation

- $x = m \cdot 2^{\tau}$ with $m \in 2^{-\rho} \cdot \mathbb{Z}$, $0 \leq |m| < 1$,
- $\tau$ is public, MPC-friendly
- Might overflow if $\tau$ is too small
- Might underflow if $\tau$ is too large

## Addition is more subtle than you might think!

- Given $(m_1, \tau_1)$ and $(m_2, \tau_2)$ as well as $\tau$
- Compute $m \cdot 2^{\tau} = m_1 \cdot 2^{\tau_1} + m_2 \cdot 2^{\tau_2}$ with a numerical window of size $\rho$
- Addition requires right-shift and roundings (non-linear operations)

# Overflows and Underflows

The class of a value is known in compile/runtime, but not the value itself.

## Plaintext Overflow

We know that $3 \in \mathcal{P}_{2,-2}$, but neither $3+3$, nor $3 \times 3$ is in $\mathcal{P}_{2,-2}$, i.e.,

- $\mathcal{P}_{\mathrm{pmsb,plsb}}$ is closed neither under addition nor under multiplication.
- Compute (at compile time) a sharp upper bound $\mathrm{pmsb}$ for the result.

## Plaintext Underflow

We know $x = 1 \times 10^{-3}, y = 6 \times 10^{-3} \in \mathcal{P}_{0,-14}$.

- If resulting class is still $\mathcal{P}_{0,-14}$, we lose the result

$$0.0010 \times 0.0060 = 0.0000$$

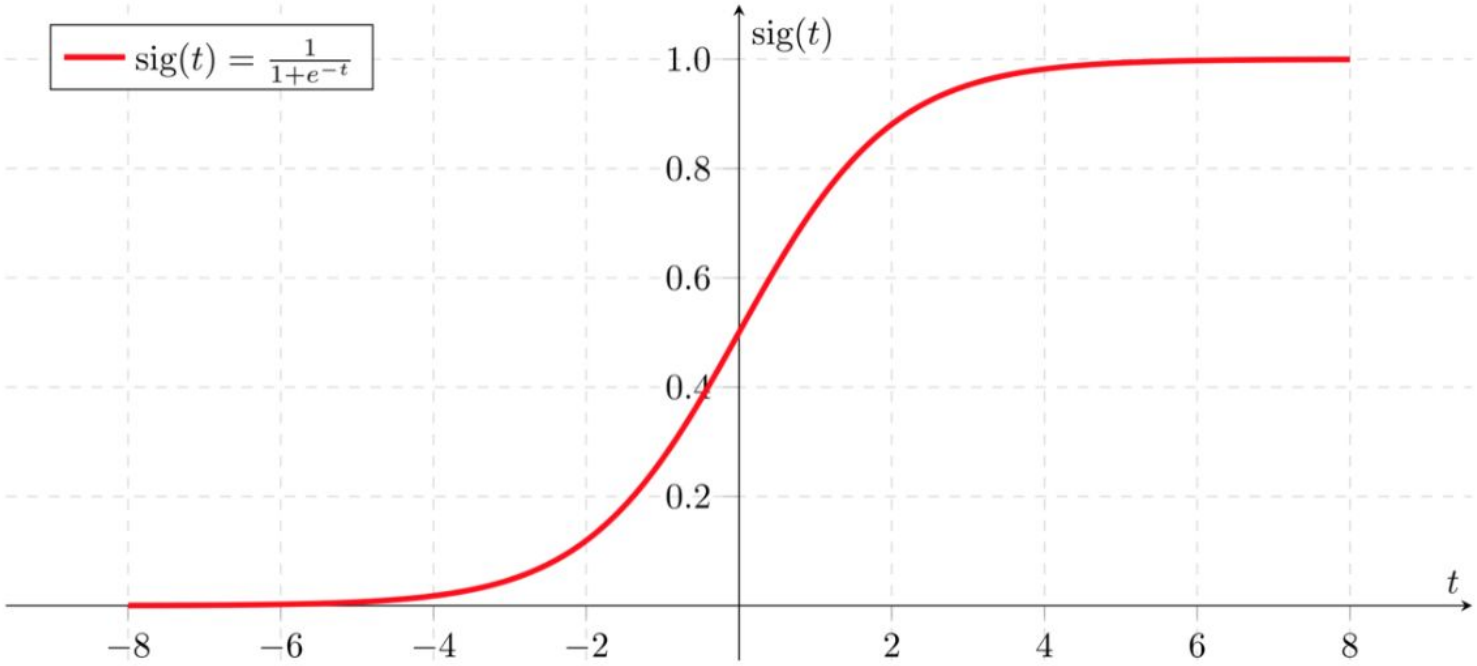
- Need for computing a suitable $\mathrm{plsb}$ at compile time

# Casting Between Classes and Arithmetic Operations

- One of the same real number can be represented with different parameters
  - mantissa, exponent, numerical window
- Need methods to cast from one representation to another
- Casts are key for implementing addition and multiplication

# Fourier Approximation of Real-Valued Functions

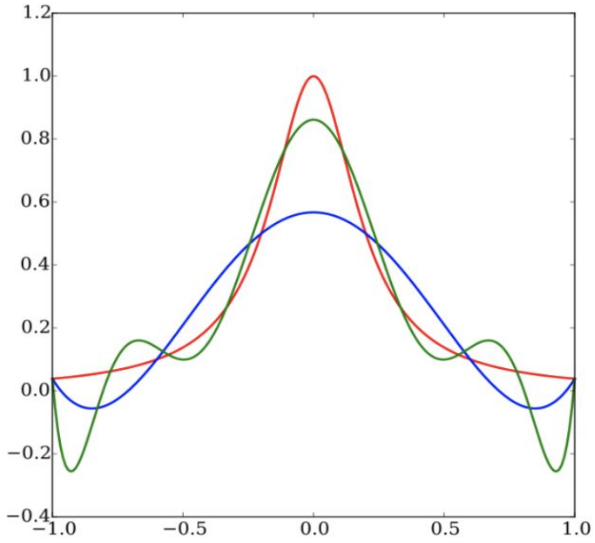
# Approximation of Non-Linear Functions



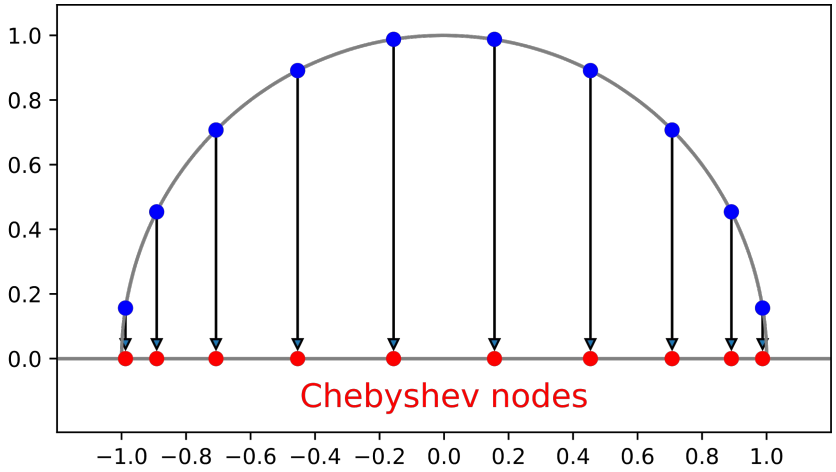
# Deficiency of Polynomial Approximation

## Limitations of polynomial approximation

- Evaluation requires multiple rounds
- Overflows are difficult to manage
- Approximation diverges quickly outside of the domain
- Uniform approximation fails (Runge's phenomenon)



# Chebyshev Polynomials and Polynomial Approximations



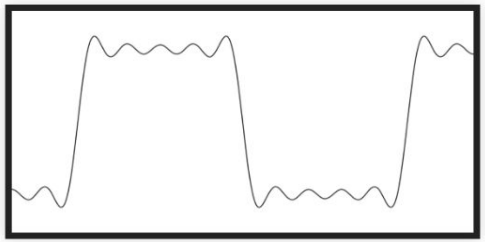

## Chebyshev Polynomials

- Polynomial $T_n$ of degree $n$ such that $T_n(\cos\theta) = \cos n\theta$
- Optimal solution to the Runge phenomenon in degree $n$

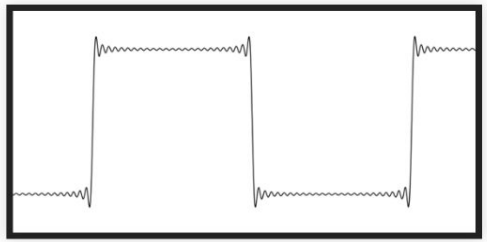
# Naive Fourier Approximation

## Naive approach via Fourier series

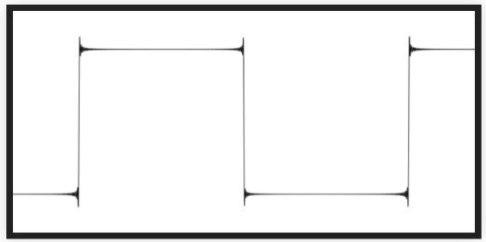
- $a_n = \frac{1}{2B} \int_{-B}^{+B} f(x) e^{\frac{\pi i n x}{B}} \, dx$
- Consider $f(x) = \sum_{k=-n}^{n} a_n e^{\frac{\pi i n x}{B}}$
- Gibbs phenomenon is an issue



$n = 5$



$n = 25$



$n = 125$

# Privacy-Preserving Evaluation of Fourier Series

## Fourier Triplets

- Fourier series can be evaluated in MPC
- $h(x) = \sum_{i=-n}^{n} a_k e^{ikx} \Rightarrow$ triplets replaced $(\lambda, e^{\pm i\lambda}, \ldots, e^{\pm in\lambda})$

## Periodic extensions on larger intervals

To evaluate a function $f$ on an interval $[-B, B]$:

- Extend $f$ to a periodic function on $[-2B, 2B]$
- Consider the Fourier series of the extension

# Rapidly Convergent Uniform Approximation

$f\colon [-\pi/2, \pi/2) \to \mathbb{R}$ - not necessarily smooth or periodic (square-integrable)

**The approximation problem**

Given $n$, define

$$G_n = \left\{ g(x) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \sin kx + \sum_{k=1}^{n} b_k \cos kx \right\}.$$

of $2\pi$-periodic functions. Consider

$$g_n(x) = \operatorname{argmin}_{g \in G_n} \|f - g\|_{L^2_{[-\pi/2, \pi/2)}}$$

- Coefficients with respect to the standard basis are numerically unstable (diverge as $n \to \infty$)
- Coefficients with respect to a basis of Chebyshev polynomials (first and second kind) is stable; exponential convergence

# Building Logistic Regression Models

Requires a privacy-preserving evaluation of sigmoid function:

$$h_\theta(\mathbf{x}) = \frac{1}{1 + e^{-\theta \cdot \mathbf{x}}},$$

as well as minimization of the cost function:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{cost}(h_\theta(\mathbf{x}^{(i)}, y^{(i)})),$$

where

$$\mathbf{cost}(h_\theta(\mathbf{x}, y)) = -y \log(h_\theta(\mathbf{x})) - (1 - y) \log(1 - h_\theta(\mathbf{x})).$$

# Logistic Regression - 1M x 50 - 3 Players

| | [FC2018] | Xor v2. |
|---|---|---|
| **Abs precision** | 2.3e-5 | 1.7e-8 ✔ |
| **Offline Time** | 04:06:43 | 00:36:40 |
| **Offline RAM** | 276G | 6.87G |
| **Triplets pp.** | 51.2G | 92.0G |
| **Online (100 MBps)** | 03:18:19 | 00:24:30 |
| **Online RAM** | 192G | 10.1G |
| **Online Comm pp.** | 20G | 66.8 |

# Methods for Compiling Privacy-Preserving Programs

# Why a Compiler?

**AUTOMATION AND STATIC ANALYSIS**

- Computation requires auxiliary random masking data (offline phase)
- Random data generated from distributions with specific parameters
- **===>** need for **static analysis of distributions (statistical calculator)** (compile time)
- Optimizations (memory/communication)

**CONSTRUCTS SPECIFIC TO MPC**

- Type checking, ANF, SSA (standard phases of compilation)
- Inline function definitions
- Unroll **for** loops with bounded number of iterations

# Linear Regression - Source

```
def solve(A: Matrix, b: Vector): Vector {
 var nrows: Int = xor.rows(A);
 var ncols: Int = xor.cols(A);

 var P: Matrix = xor.orthrand(nrows, ncols, -6);
 var Q: Matrix = xor.orthrand(nrows, ncols, -6);

 var PAQ: Matrix = P * A * Q;
 var Pb: Vector = P * b;

 xor.reveal(PAQ);
 xor.reveal(Pb);
 var r: Vector = xor.publicSolve(PAQ, Pb);
 return Q * r;
}
```

```
def linreg(y: Vector, X: Matrix): Vector {
 var A: Matrix = xor.transpose(X) * X;
 var b: Vector = xor.transpose(X) * y;
 return solve(A, b);
}

def main() {
 var X: Matrix = xor.input("X");
 var y: Vector = xor.input("y");
 var theta: Vector = linreg(y, X);
 xor.output(theta, "thetas");
}
```

# Linear Regression - Source

```
def solve(A: Matrix, b: Vector): Vector {
 var nrows: Int = xor.rows(A);
 var ncols: Int = xor.cols(A);

 var P: Matrix = xor.orthrand(nrows, ncols, -6);
 var Q: Matrix = xor.orthrand(nrows, ncols, -6);

 var PAQ: Matrix = P * A * Q;
 var Pb: Vector = P * b;

 xor.reveal(PAQ);
 xor.reveal(Pb);
 var r: Vector = xor.publicSolve(PAQ, Pb);
 return Q * r;
}
```

builtin

builtin

```
def linreg(y: Vector, X: Matrix): Vector {
 var A: Matrix = xor.transpose(X) * X;
 var b: Vector = xor.transpose(X) * y;
 return solve(A, b);
}

def main() {
 var X: Matrix = xor.input("X");
 var y: Vector = xor.input("y");
 var theta: Vector = linreg(y, X);
 xor.output(theta, "thetas");
}
```

builtin

builtin

```
manojoport in ~/workspace/inpher/xor-compiler
± |master {4} U:1 ?:5 ✗| → xorc -L
```

| Index | Phase | Brief |
|---|---|---|
| 0 | parse | Parses the program. |
| 1 | namer | Enters symbols for all top-level functions. |
| 2 | typer | Checks that the program is well typed, and performs some basic type inference. |
| 3 | anf | Transform the program in A-normal form. |
| 4 | ssa | Transforms the tree into its Static Single-Assignment (SSA) form. |
| 5 | inline | Inlines all functions called by the main function into its body. |
| 6 | tuple-elimination | Eliminates tuples creation and projections. |
| 7 | copy-propagation | Removes intermediate variables from assign-chains, and other unused variables. |
| 8 | constant-fold | Performs basic partial evaluation and folds constants through the program. |
| 9 | dimension-checking | Checks the dimensions of all matrices, and ensures that the operations are valid. |
| 10 | desugaring | Transforms a user-level program into one that operates on MPC-level primitives only. |
| 11 | visibility | Tracks and sets the visibility of all values. |
| 12 | plaintext-param | Computes plaintext parameters pMsb and pLsb. |
| 13 | mask-resolution | Resolves masking parameters from plaintext parameters. |
| 14 | builtin-params-resolution | Computes extra parameters specific to builtins. |
| 15 | codegen | Generates a compiled-program.bin that can be executed by an appropriate backend. |

```
      -e, --process <phase>
            compile until given <phase> and print program

      -E, --process-each <phase>
            compile and print each phase until <phase>

      -m, --print-symbols
            include symbols when printing tree

      -f, --print-full-type
            include the full type when printing tree

      -c, --print-code
            print human friendly representation of generated code

      -L, --print-phases
            print all phases, in the order in which they are executed, and exit

  Control options
      -W, --warning <warn>
            enable/disable warning <warn>

      --disable-warnings
            disable all warnings

      -O, --optimize <level>
            enable optimizations level <level>

      -D, --define:<name>=<value>
            define a global variable <name> of specified <value>

      -v, --verbose
Manual page (stdin) line 1/104 62% (press h for help or q to quit)
```

```
      • 1 otherwise

ENVIRONMENT
      JAVA_OPTS
            command-line options to pass to the underlying JVM

EXAMPLES
      Assume file main.xor with contents:

            def main() {
              var v1: Vector = xor.input("v1");
              var v2: Vector = xor.input("v2");
              var corr: Float = v1 * v1;
              xor.output(corr, "product");
            }

      Assume file iodb.csv with contents:

            # Placeholder name, visibility, rows, cols, msb, lsb
            v1, secret, 5, 1, 5, -2
            v2, secret, 5, 1, 5, -2

      Run the xor compiler with the following invocation:

            xorc --iodb iodb.csv main.xor -o main.xbin

      This will produce a compiled program in main.xbin.

AUTHORS
      Inpher, Inc.

                                                June 2019                XORC(1)
Manual page (stdin) line 47/104 (END) (press h for help or q to quit)
```

# Linear Regression - Assembly

```
  :  ...
 6:  CreateContainer(V6, FlMR<2,2,9,7,-43>);
 7:  BeaverMod(PriV1, PriV1, V6, AW=(29,-20), BW=(29,-20), W=(9,-40), Pairing=4);
 8:  CreateContainer(V8, FlMR<2,1,15,13,-37>);
 9:  BeaverMod(PriV1, PriV3, V8, AW=(35,-20), BW=(35,-20), W=(15,-40), Pairing=4);
10:  {
11:        CreateContainer(V11, FlMR<2,2,2,0,-6>);
12:        RandomOrthogonalMatrix(V11);
13:        CreateContainer(V13, FlMR<2,2,2,0,-6>);
14:        RandomOrthogonalMatrix(V13);
15:        CreateContainer(V15, FlMR<2,2,14,12,-38>);
16:        BeaverMod(PriV11, PriV6, V15, AW=(52,-6), BW=(20,-38), W=(14,-44), Pairing=3);
17:        CreateContainer(V17, FlMR<2,2,15,13,-37>);
18:        BeaverMod(PriV15, PriV13, V17, AW=(21,-37), BW=(52,-6), W=(15,-43), Pairing=3);
19:        CreateContainer(V19, FlMR<2,1,16,14,-36>);
20:        BeaverMod(PriV11, PriV8, V19, AW=(52,-6), BW=(22,-36), W=(16,-42), Pairing=3);
21:        Reveal(V17);
22:        Reveal(V19);
23:        CreateContainer(V23, FlMR<2,1,26,24,-26>);
24:        PublicSolve(V17, V19, V23);
25:        CreateContainer(V25, FlMR<2,1,27,25,-25>);
26:        BeaverMod(PriV13, PubV23, V25, AW=(52,-6), BW=(33,-25), W=(27,-31), Pairing=3);
27:  }
  :  ...
```

# Linear Regression - Assembly

```
  :  ...
 6:  CreateContainer(V6, FlMR<2,2,9,7,-43>);
 7:  BeaverMod(PriV1, PriV1, V6, AW=(29,-20), BW=(29,-20), W=(9,-40), Pairing=4);
 8:  CreateContainer(V8, FlMR<2,1,15,13,-37>);
 9:  BeaverMod(PriV1, PriV3, V8, AW=(35,-20), BW=(35,-20), W=(15,-40), Pairing=4);
10:  {
11:        CreateContainer(V11, FlMR<2,2,2,0,-6>);
12:        RandomOrthogonalMatrix(V11);
13:        CreateContainer(V13, FlMR<2,2,2,0,-6>);
14:        RandomOrthogonalMatrix(V13);
15:        CreateContainer(V15, FlMR<2,2,14,12,-38>);
16:        BeaverMod(PriV11, PriV6, V15, AW=(52,-6), BW=(20,-38), W=(14,-44), Pairing=3);
17:        CreateContainer(V17, FlMR<2,2,15,13,-37>);
18:        BeaverMod(PriV15, PriV13, V17, AW=(21,-37), BW=(52,-6), W=(15,-43), Pairing=3);
19:        CreateContainer(V19, FlMR<2,1,16,14,-36>);
20:        BeaverMod(PriV11, PriV8, V19, AW=(52,-6), BW=(22,-36), W=(16,-42), Pairing=3);
21:        Reveal(V17);
22:        Reveal(V19);
23:        CreateContainer(V23, FlMR<2,1,26,24,-26>);
24:        PublicSolve(V17, V19, V23);
25:        CreateContainer(V25, FlMR<2,1,27,25,-25>);
26:        BeaverMod(PriV13, PubV23, V25, AW=(52,-6), BW=(33,-25), W=(27,-31), Pairing=3);
27:  }
  :  ...
```

# Linear Regression - Assembly

```
  :  ...
 6:  CreateContainer(V6, FlMR<2,2,9,7,-43>);
 7:  BeaverMod(PriV1, PriV1, V6, AW=(29,-20), BW=(29,-20), W=(9,-40), Pairing=4);
 8:  CreateContainer(V8, FlMR<2,1,15,13,-37>);
 9:  BeaverMod(PriV1, PriV3, V8, AW=(35,-20), BW=(35,-20), W=(15,-40), Pairing=4);
10:  {
11:        CreateContainer(V11, FlMR<2,2,2,0,-6>);
12:        RandomOrthogonalMatrix(V11);
13:        CreateContainer(V13, FlMR<2,2,2,0,-6>);
14:        RandomOrthogonalMatrix(V13);
15:        CreateContainer(V15, FlMR<2,2,14,12,-38>);
16:        BeaverMod(PriV11, PriV6, V15, AW=(52,-6), BW=(20,-38), W=(14,-44), Pairing=3);
17:        CreateContainer(V17, FlMR<2,2,15,13,-37>);
18:        BeaverMod(PriV15, PriV13, V17, AW=(21,-37), BW=(52,-6), W=(15,-43), Pairing=3);
19:        CreateContainer(V19, FlMR<2,1,16,14,-36>);
20:        BeaverMod(PriV11, PriV8, V19, AW=(52,-6), BW=(22,-36), W=(16,-42), Pairing=3);
21:        Reveal(V17);
22:        Reveal(V19);
23:        CreateContainer(V23, FlMR<2,1,26,24,-26>);
24:        PublicSolve(V17, V19, V23);
25:        CreateContainer(V25, FlMR<2,1,27,25,-25>);
26:        BeaverMod(PriV13, PubV23, V25, AW=(52,-6), BW=(33,-25), W=(27,-31), Pairing=3);
27:  }
  :  ...
```

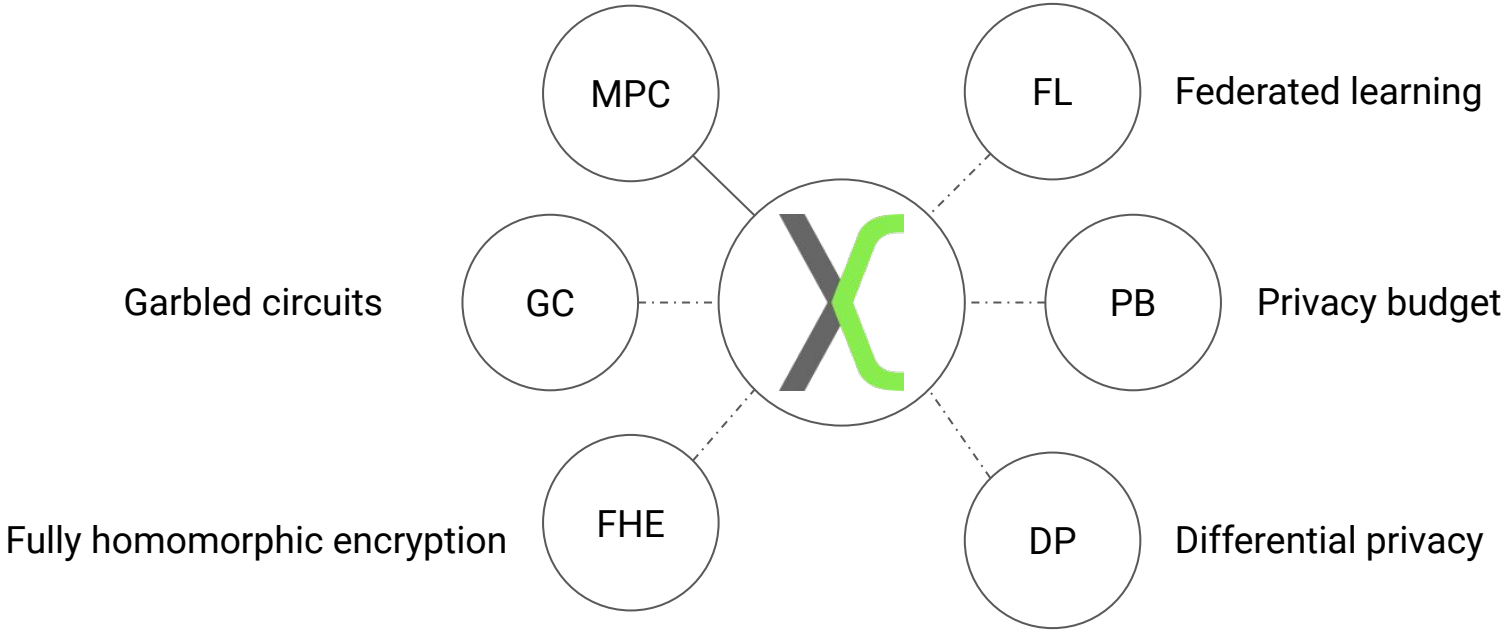
# Linear Regression - Assembly

```
  :  ...
 6:  CreateContainer(V6, FlMR<2,2,9,7,-43>);
 7:  BeaverMod(PriV1, PriV1, V6, AW=(29,-20), BW=(29,-20), W=(9,-40), Pairing=4);
 8:  CreateContainer(V8, FlMR<2,1,15,13,-37>);
 9:  BeaverMod(PriV1, PriV3, V8, AW=(35,-20), BW=(35,-20), W=(15,-40), Pairing=4);
10:  {
11:        CreateContainer(V11, FlMR<2,2,2,0,-6>);
12:        RandomOrthogonalMatrix(V11);
13:        CreateContainer(V13, FlMR<2,2,2,0,-6>);
14:        RandomOrthogonalMatrix(V13);
15:        CreateContainer(V15, FlMR<2,2,14,12,-38>);
16:        BeaverMod(PriV11, PriV6, V15, AW=(52,-6), BW=(20,-38), W=(14,-44), Pairing=3);
17:        CreateContainer(V17, FlMR<2,2,15,13,-37>);
18:        BeaverMod(PriV15, PriV13, V17, AW=(21,-37), BW=(52,-6), W=(15,-43), Pairing=3);
19:        CreateContainer(V19, FlMR<2,1,16,14,-36>);
20:        BeaverMod(PriV11, PriV8, V19, AW=(52,-6), BW=(22,-36), W=(16,-42), Pairing=3);
21:        Reveal(V17);
22:        Reveal(V19);
23:        CreateContainer(V23, FlMR<2,1,26,24,-26>);
24:        PublicSolve(V17, V19, V23);
25:        CreateContainer(V25, FlMR<2,1,27,25,-25>);
26:        BeaverMod(PriV13, PubV23, V25, AW=(52,-6), BW=(33,-25), W=(27,-31), Pairing=3);
27:  }
  :  ...
```

# Linear Regression - Assembly

```
  :  ...
 6:  CreateContainer(V6, FlMR<2,2,9,7,-43>);
 7:  BeaverMod(PriV1, PriV1, V6, AW=(29,-20), BW=(29,-20), W=(9,-40), Pairing=4);
 8:  CreateContainer(V8, FlMR<2,1,15,13,-37>);
 9:  BeaverMod(PriV1, PriV3, V8, AW=(35,-20), BW=(35,-20), W=(15,-40), Pairing=4);
10:  {
11:        CreateContainer(V11, FlMR<2,2,2,0,-6>);
12:        RandomOrthogonalMatrix(V11);
13:        CreateContainer(V13, FlMR<2,2,2,0,-6>);
14:        RandomOrthogonalMatrix(V13);
15:        CreateContainer(V15, FlMR<2,2,14,12,-38>);
16:        BeaverMod(PriV11, PriV6, V15, AW=(52,-6), BW=(20,-38), W=(14,-44), Pairing=3);
17:        CreateContainer(V17, FlMR<2,2,15,13,-37>);
18:        BeaverMod(PriV15, PriV13, V17, AW=(21,-37), BW=(52,-6), W=(15,-43), Pairing=3);
19:        CreateContainer(V19, FlMR<2,1,16,14,-36>);
20:        BeaverMod(PriV11, PriV8, V19, AW=(52,-6), BW=(22,-36), W=(16,-42), Pairing=3);
21:        Reveal(V17);
22:        Reveal(V19);
23:        CreateContainer(V23, FlMR<2,1,26,24,-26>);
24:        PublicSolve(V17, V19, V23);
25:        CreateContainer(V25, FlMR<2,1,27,25,-25>);
26:        BeaverMod(PriV13, PubV23, V25, AW=(52,-6), BW=(33,-25), W=(27,-31), Pairing=3);
27:  }
  :  ...
```

# Compiling to Preserve Privacy - Beyond MPC



# Other Applications - Compiling FHE Programs

Bootstrapping: an operation used to reduce noise in the plaintext in order to continue processing the ciphertext in a computation

## Ring-LWE-based FHE Schemes

- `TFHE` - suitable for boolean circuits, automata evaluations
- `B/FV` - suitable for integer arithmetic, arithmetic circuits
- `HEAAN` - suitable for vectorized operations with real numbers
- `CHIMERA` - scheme switching method

## Numerical Parameters in the Context of FHE

- $L$ - level exponent of ciphertext - related to the maximal number of homomorphic operations before performing a bootstrapping
- $\rho$ - numerical window (as in the MPC setting)
- $\tau$ - exponent (only know a sufficiently good bound at compile time)

# Thank you!