Scala Days, 13 June 2019

Compiling to Preserve Our Privacy

Manohar Jonnalagedda
Jakob Odersky

inpher

A

```
def avg(salaries: List[Int]) =
  salaries.sum / salaries.length

avg(A :: B :: C :: Nil)
```

B

C

Jakob

B = 100

Dimitar

A = 1000

Manohar

C = 70

A = 1000

$0 = d = d_1 + d_2 + d_3$

B = 100

$0 = j = j_1 + j_2 + j_3$

C = 70

$0 = m = m_1 + m_2 + m_3$

```
type SecretNum = Int // in this example
type SharedNum = List[SecretNum]

def createZeroShares(players: Int): SharedNum = {
  val rands =
    List.fill(players - 1)(util.Random.nextInt)
  val diff = -rands.sum
  diff :: rands
}

val d = createZeroShares(3) // locally by d
val j = createZeroShares(3) // locally by j
val m = createZeroShares(3) // locally by m
```

A = 1000

$0 = d = d_1 + d_2 + d_3$

B = 100

$0 = j = j_1 + j_2 + j_3$

C = 70

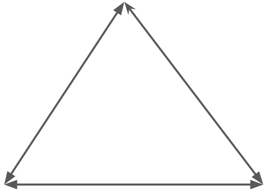
$0 = m = m_1 + m_2 + m_3$

```
type SecretNum = Int // in this example
type SharedNum = List[SecretNum]

def createZeroShares                      dNum = {
  val rands =
    List.fill(pl
  val diff = -r
  diff :: rands
}

val d = createZeroShares(3) // locally by d
val j = createZeroShares(3) // locally by j
val m = createZeroShares(3) // locally by m
```

Secret shares (elements of the list) reside with different players

A = 1000

$0 = d = d_1 + d_2 + d_3$



B = 100

$0 = j = j_1 + j_2 + j_3$

C = 70

$0 = m = m_1 + m_2 + m_3$

```
type SecretNum = Int // in this example
type SharedNum = List[SecretNum]

def createZeroShares(players: Int): SharedNum = {
  val rands =
    List.fill(players - 1)(util.Random.nextInt)
  val diff = -rands.sum
  diff :: rands
}


val d = createZeroShares(3) // locally by d
val j = createZeroShares(3) // locally by j
val m = createZeroShares(3) // locally by m
```

		players		
shared zeroes	d	d_1	d_2	d_3
	j	j_1	j_2	j_3
	m	m_1	m_2	m_3
data		A	B	C
shared sum		S_1	S_2	S_3
		$S = S_1 + S_2 + S_3$		

sum

		players		
shared zeroes	d	d_1	d_2	d_3
	j	j_1	j_2	j_3
	m	m_1	m_2	m_3
data		A	B	C
shared sum		S_1	S_2	S_3
		$S = S_1 + S_2 + S_3$		

sum

Secret shares for zero are **distributed**

		players		
shared zeroes	d	d_1	d_2	d_3
	j	j_1	j_2	j_3
	m	m_1	m_2	m_3
data		A	B	C
shared sum		S_1	S_2	S_3
		$S = S_1 + S_2 + S_3$		

sum

Shared sums are **revealed publicly**

		players		
shared zeroes	0	2000	3500	-5500
	0	-4000	4000	0
	0	4330	-1220	-3110
data		1000	100	70
shared sum		3330	6380	-8540
		S = 1170		

sum

1170

avg = 390

		players			
shared zeroes	0	2000	3500	-5500	
	0	-4000	4000	0	
	0	4330	-1220	-3110	sum
data		1000	100	70	1170
shared sum		3330	6380	-8540	
		S = 1170			avg = 390

		players		
shared zeroes	0	2000	3500	-5500
	0	-4000	4000	0
	0	4330	-1220	-3110
data		1000	100	70
shared sum		3330	6380	-8540
		S = 1170		

sum

1170

avg = 390

		players		
shared zeroes	0	2000	3500	-5500
	0	-4000	4000	0
	0	4330	-1220	-3110
data		1000	100	70
shared sum		3330	6380	-8540
		S = 1170		

sum

1170

avg = 390

		players		
shared zeroes	d	d_1	d_2	d_3
	j	j_1	j_2	j_3
	m	m_1	m_2	m_3
data		A	B	C
shared sum		S_1	S_2	S_3
		$S = S_1 + S_2 + S_3$		

```
def add(x: SharedNum, y: SharedNum): SharedNum =
  x.zip(y).map(addSecret)
def addSecret(x: SecretNum, y: SecretNum): SecretNum =
  x + y
def reveal(x: SharedNum): Int = x.sum

// shared zeroes
val d = createZeroShares(3)
val j = createZeroShares(3)
val m = createZeroShares(3)

val data: SharedNum = A :: B :: C :: Nil

val sharedSum       = add(add(d, j), add(m, data))


val sum             = reveal(sharedSum)
```

		players		
shared zeroes	d	d_1	d_2	d_3
	j	j_1	j_2	j_3
	m	m_1	m_2	m_3
data		A	B	C
shared sum		S_1	S_2	S_3
		$S = S_1 + S_2 + S_3$		

```
def add(x: SharedNum, y: SharedNum): SharedNum =
  x.zip(y).map(addSecret)
def addSecret(x: SecretNum, y: SecretNum): SecretNum =
  x + y
def reveal(x: SharedNum): Int = x.sum

// shared zeroes
val d = createZeroShares(3)
val j = createZeroShares(3)
val m = createZeroShares(3)

val data: SharedNum = A :: B :: C :: Nil

val sharedSum       = add(add(d, j), add(m, data))


val sum             = reveal(sharedSum)
```

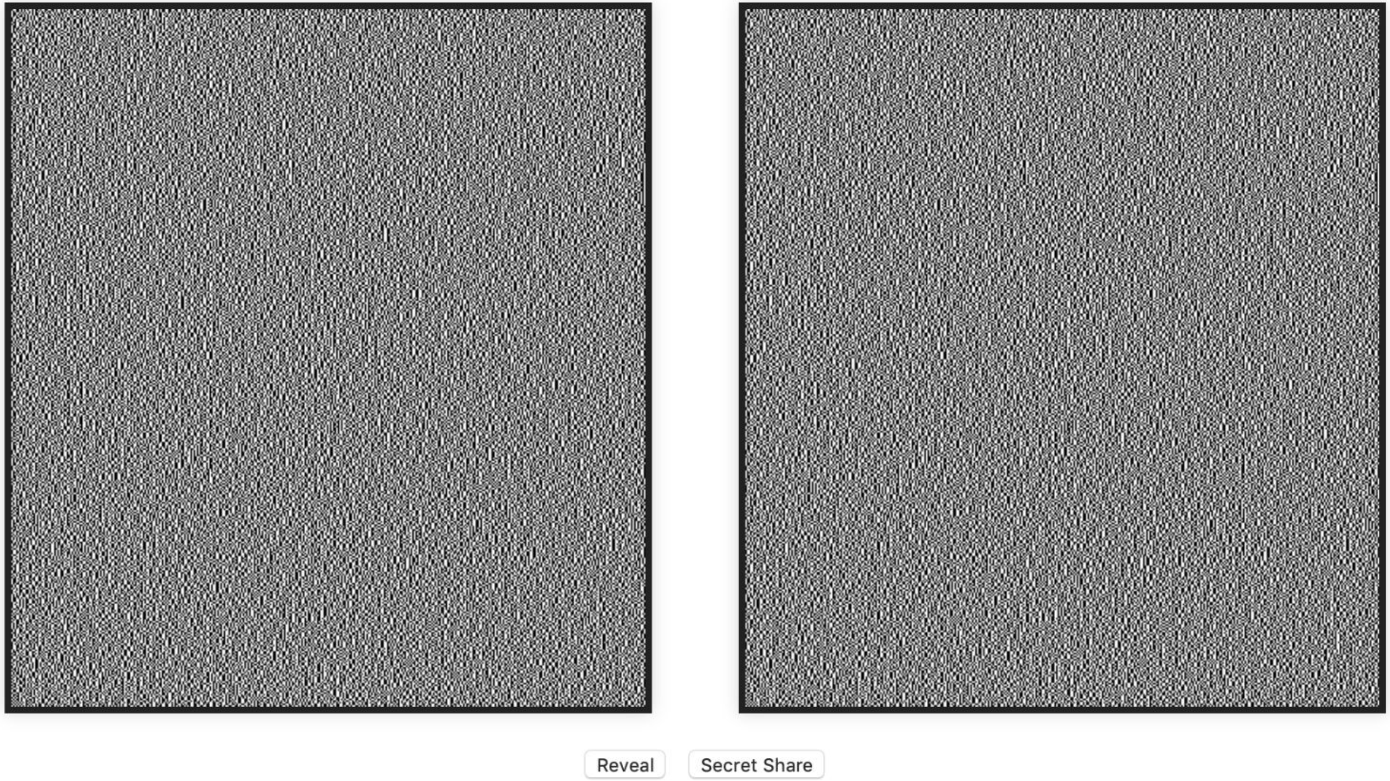
d + j + m + data

Secure multi-party computation (MPC)

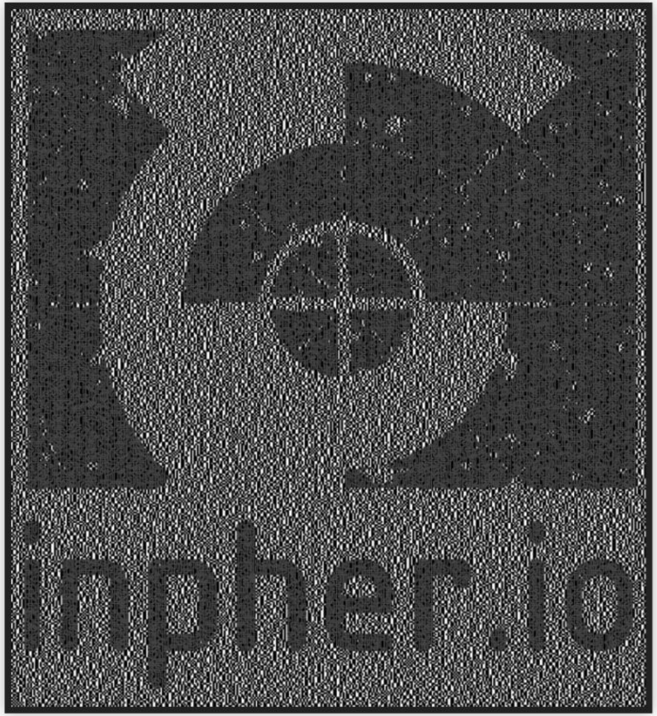
Secure multi-party computation (MPC)

subfield of cryptography with the goal of creating methods for parties to **jointly compute a function** over their inputs **while keeping those inputs private**.

Secret Shared Data



Reveal Secret Shared Data



Reveal Secret Share

Further operations on SharedNum ?

1. Addition
2. …
3. …
4. …

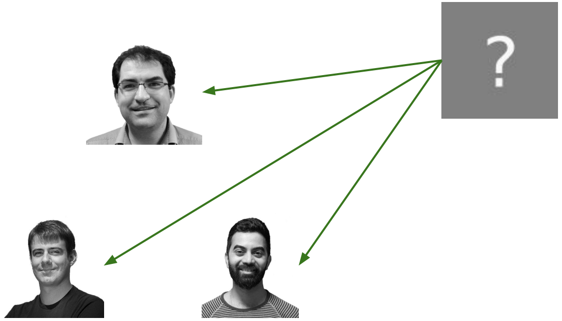
Further operations on SharedNum ?

1. Addition
2. Multiplication
3. …
4. …

Multiplications on SharedNum

```
def mult(x: SharedNum, y: SharedNum) : SharedNum = ???
```

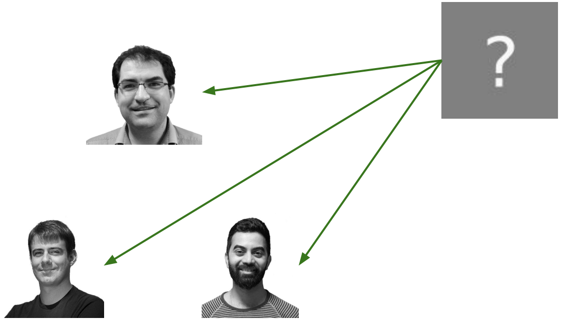
- is more efficient with a **trusted dealer**



Multiplications on SharedNum

```
def mult(x: SharedNum, y: SharedNum) : SharedNum = ???
```

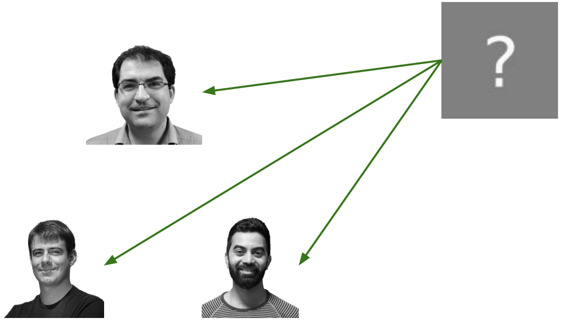
- is more efficient with a **trusted dealer**
- requires some precomputed random values
  - **Beaver triplets**
  - Used for **mask and reveal**



Multiplications on SharedNum

```
def mult(x: SharedNum, y: SharedNum) : SharedNum = ???
```

- is more efficient with a **trusted dealer**
- requires some precomputed random values
  - **Beaver triplets**
  - Used for **mask and reveal**
- come talk to us to know more!



Further operations on SharedNum ?

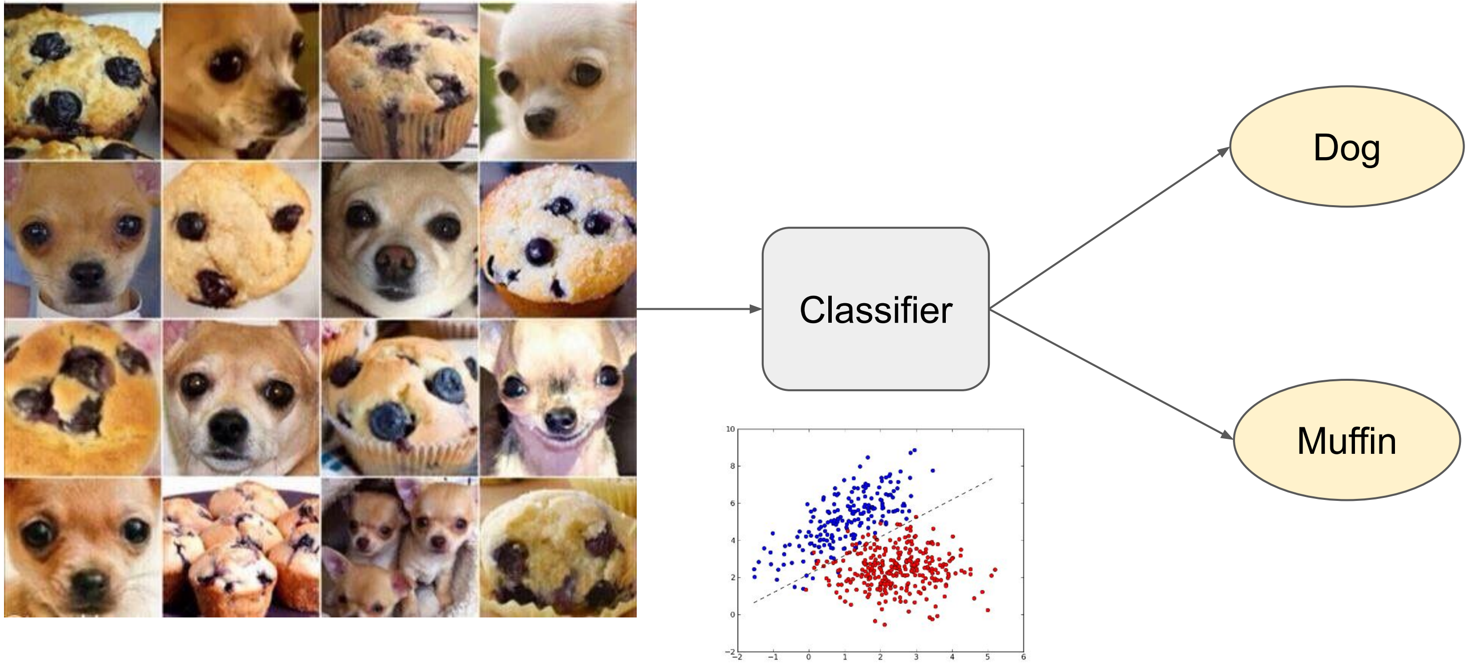
1. Addition
2. Multiplication
3. ???
4. ...

LinReg: y = ax + b

ML

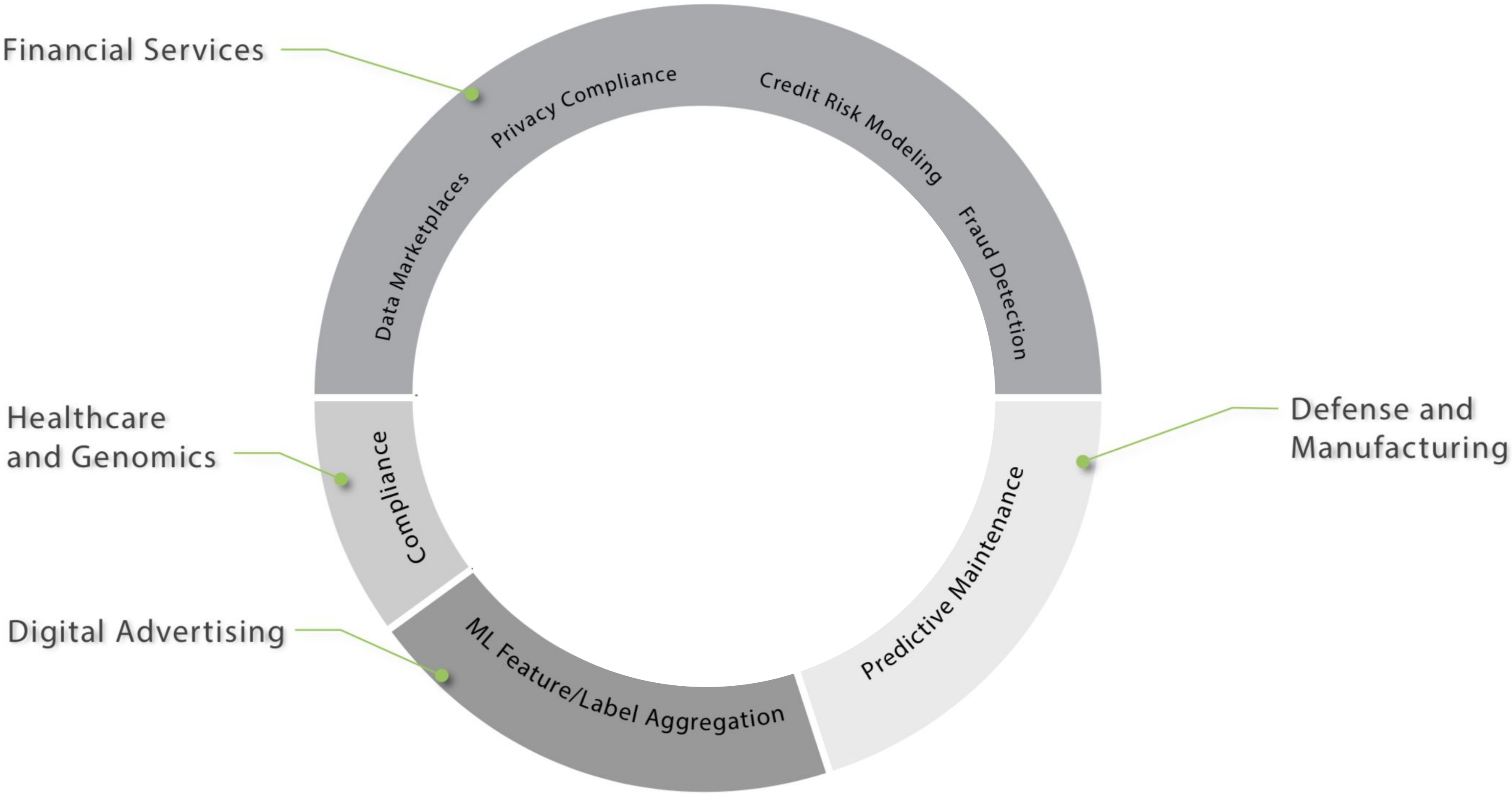
LogReg: y = 1 / 1 + exp(-ax - b)

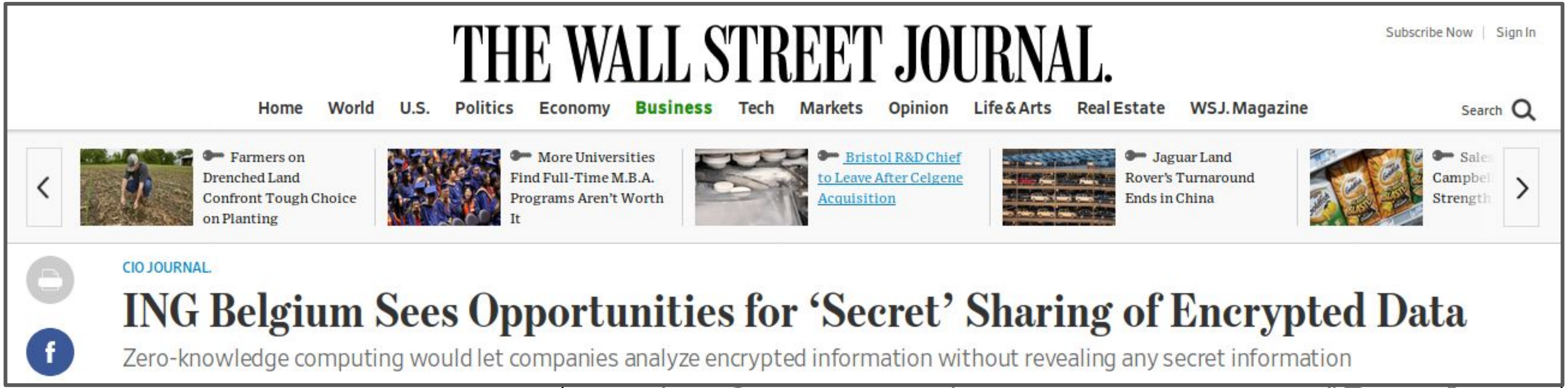
Classification: Chihuahua or Muffin ?



Further operations on SharedNum ?

1. Addition
2. Multiplication
3. ???
4. Real-world (i.e. profit)


Financial Services
Data Marketplaces
Privacy Compliance
Credit Risk Modeling
Fraud Detection
Healthcare and Genomics
Compliance
Digital Advertising
ML Feature/Label Aggregation
Defense and Manufacturing
Predictive Maintenance



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

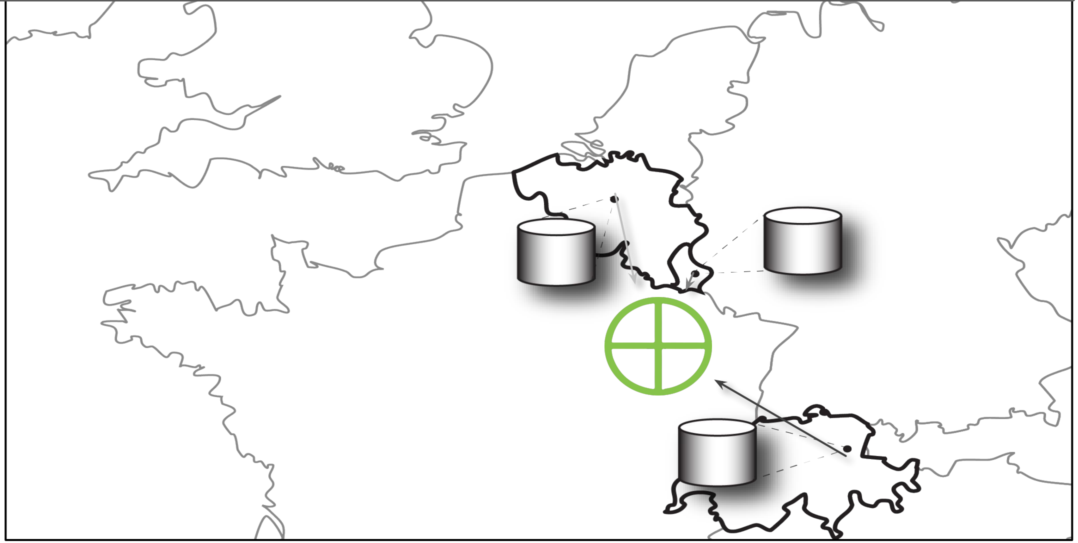
Jaguar Land Rover's Turnaround Ends in China

Sale Campbel Strength

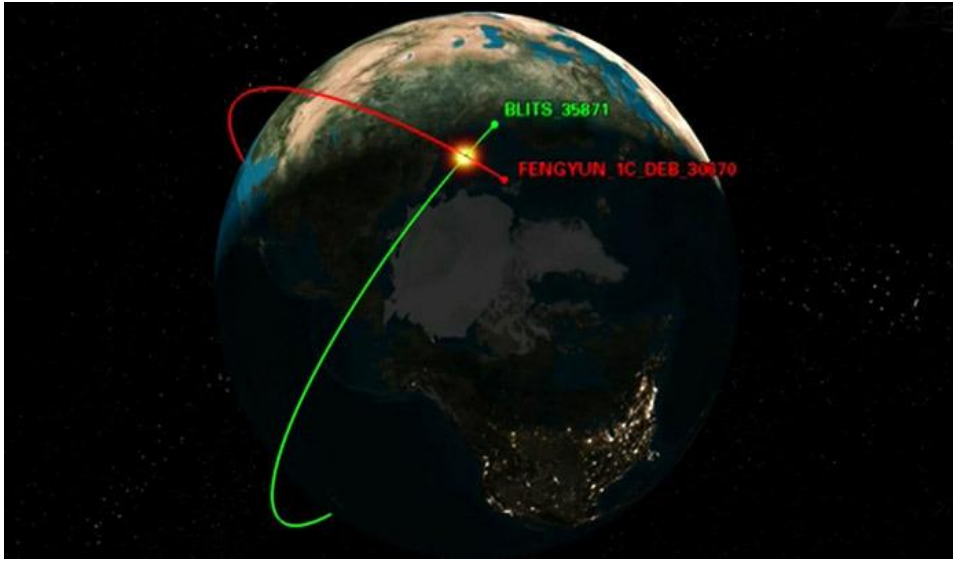
CIO JOURNAL.

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

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

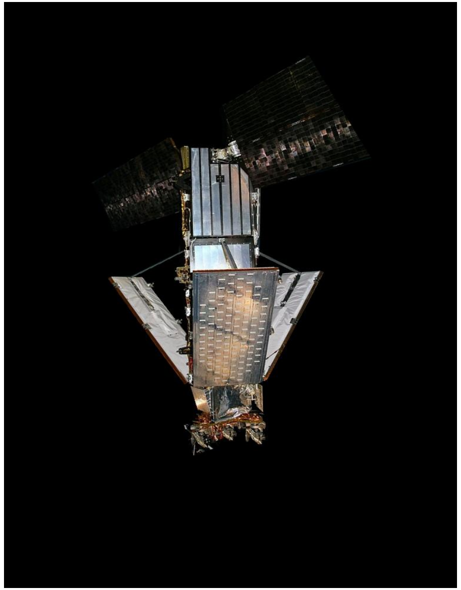


Privacy-preserving satellite collision detection

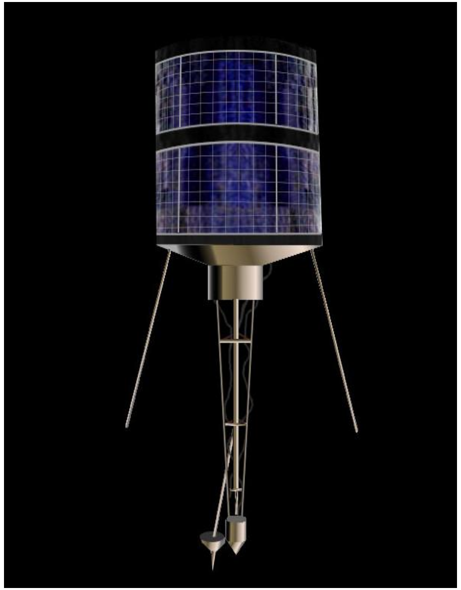


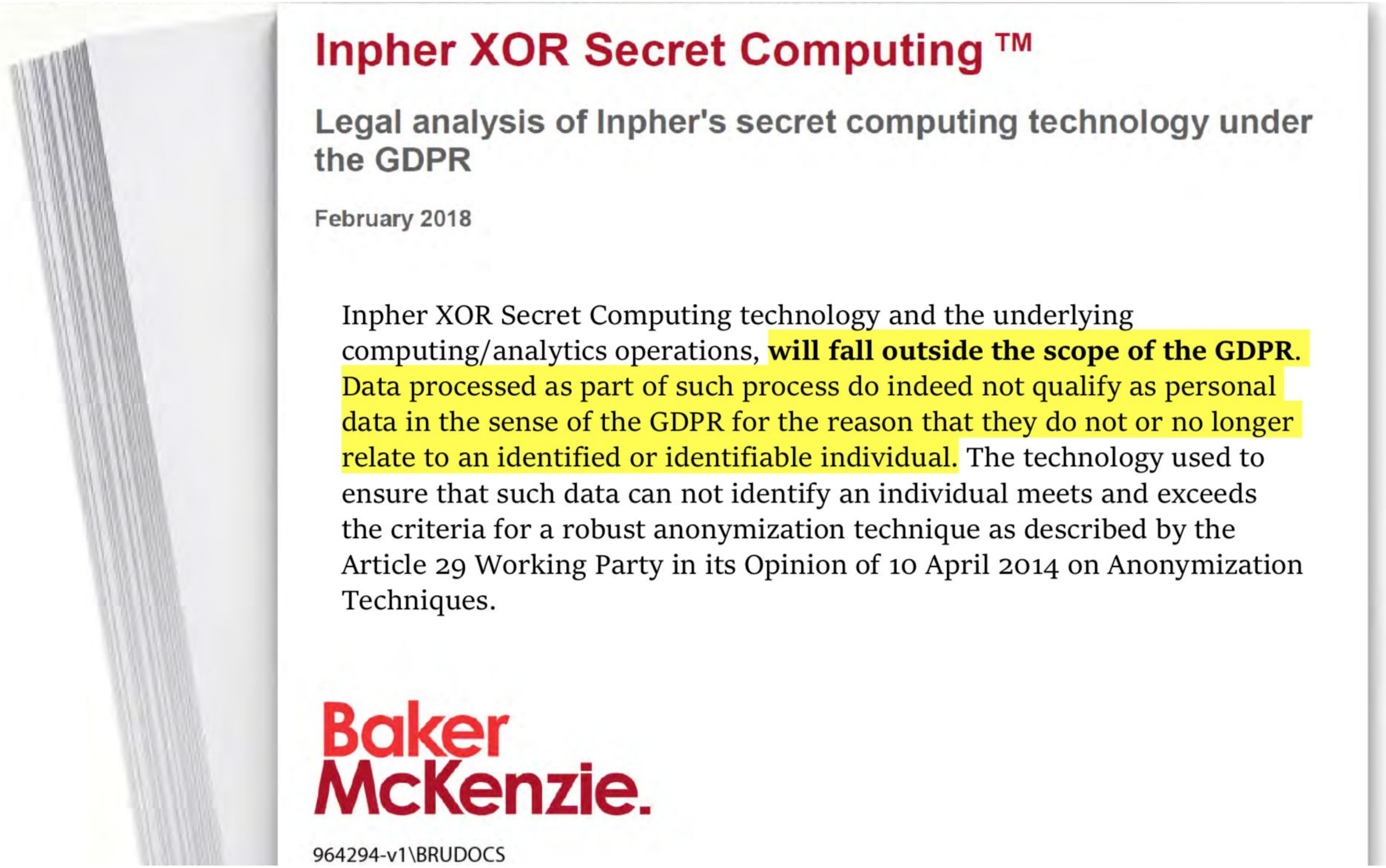
- Predicting collisions of satellites
- Satellite trajectories are private
- Satellite operators nonetheless perform **conjunction analysis**
- Need to evaluate non-linear functions with high numerical precision

Iridium 33 and Kosmos-2251 Satellite Collision



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





Inpher XOR Secret Computing ™

Legal analysis of Inpher's secret computing technology under the GDPR

February 2018

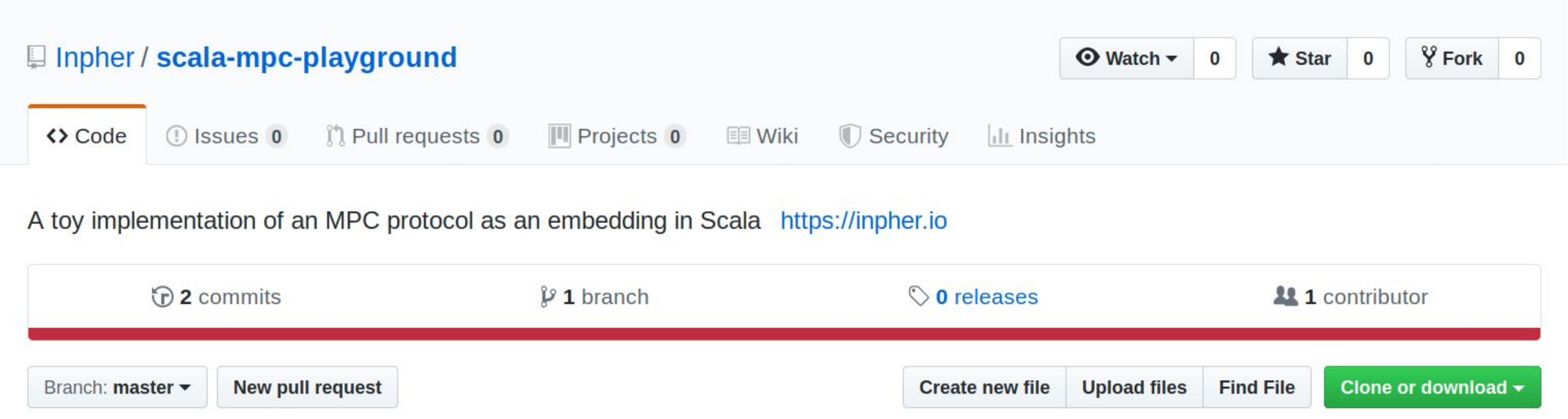
Inpher XOR Secret Computing technology and the underlying computing/analytics operations, **will fall outside the scope of the GDPR**. Data processed as part of such process do indeed not qualify as personal data in the sense of the GDPR for the reason that they do not or no longer relate to an identified or identifiable individual. The technology used to ensure that such data can not identify an individual meets and exceeds the criteria for a robust anonymization technique as described by the Article 29 Working Party in its Opinion of 10 April 2014 on Anonymization Techniques.

Baker
McKenzie.

964294-v1\BRUDOCS

A Scala embedding

- Write code in linear algebra style, get secret compute with it for (almost) free!
- More Scala goodness for seamless DSL (Numeric, implicit Ops etc...)
- https://github.com/inpher/scala-mpc-playground



A toy implementation of an MPC protocol as an embedding in Scala https://inpher.io

From library to full-blown compiler

- Computations **distributed** in reality
  - Must target a runtime that distributes computation and deploys

From library to full-blown compiler

- Computations **distributed** in reality
  - Must target a runtime that distributes computation and deploys
- Require **static analysis** of programs
  - Compute **statistical distributions** of intermediate values
  - **Optimize** for memory usage and communication

The DSL and the compiler

- The DSL: allows composable code in a linear algebra style
- Generates low-level, MPC-specific instructions

The DSL and the compiler

- The DSL: allows composable code in a linear algebra style
- Generates low-level, MPC-specific instructions
- Various static analysis passes
  - The **usual** (type-checking, dimension-checking)
  - The **MPC specific** (infer numerical parameters, optimise for memory/communication)

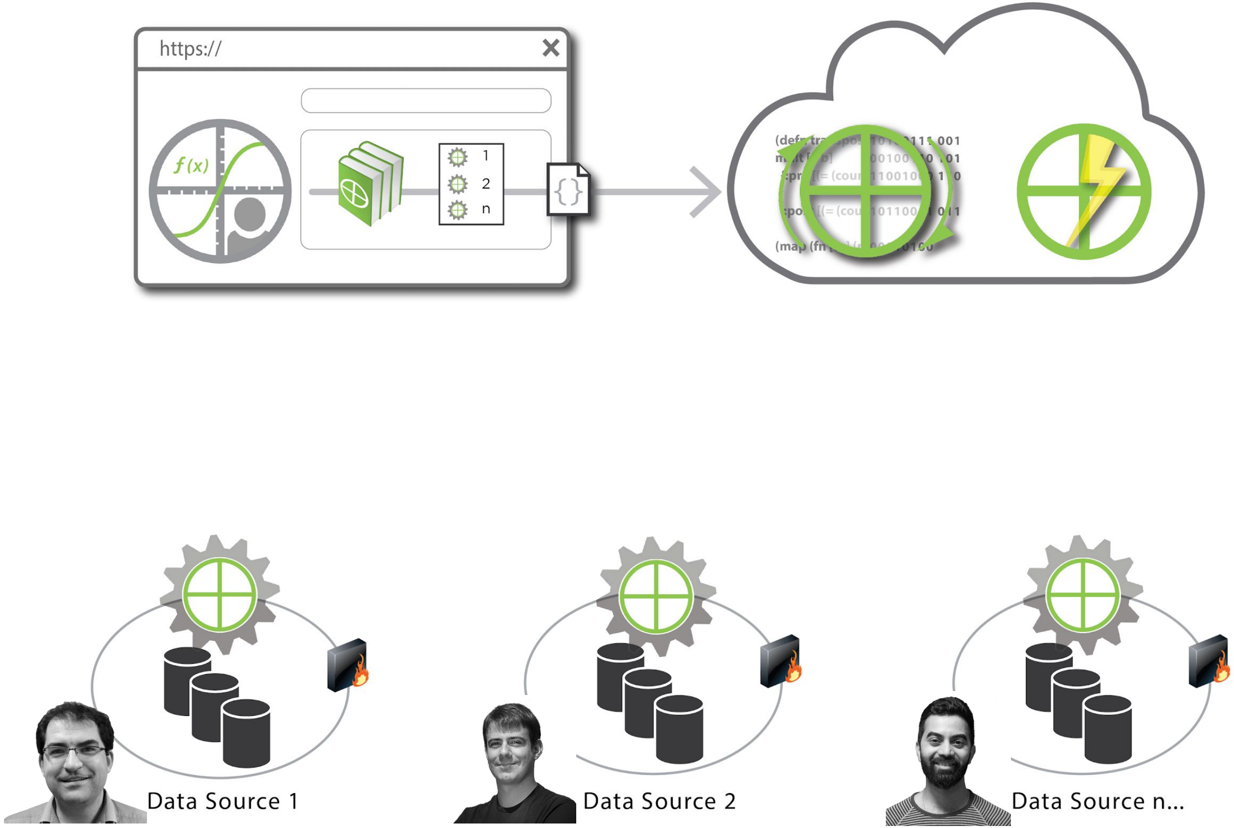
From library to full-blown compiler

- Computations **distributed** in reality
  - Must target a runtime that distributes computation and deploys
- Require **static analysis** of programs
  - Compute **statistical distributions** of intermediate values
  - **Optimize** for memory usage and communication

The complete architecture

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

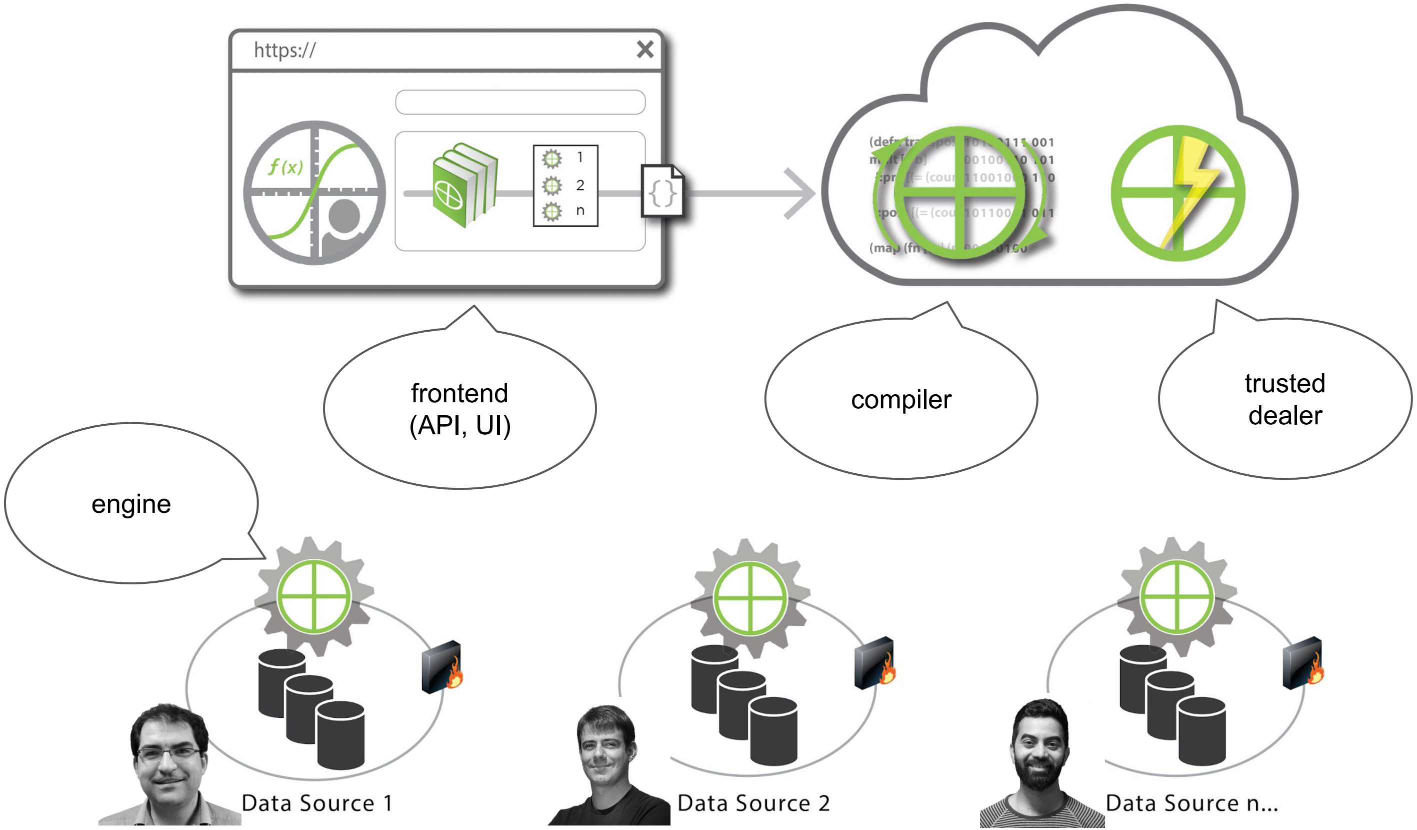
Inpher-hosted XOR Service
(never exposed to data)



Analyst submits operations to XOR Service and selects data sources

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

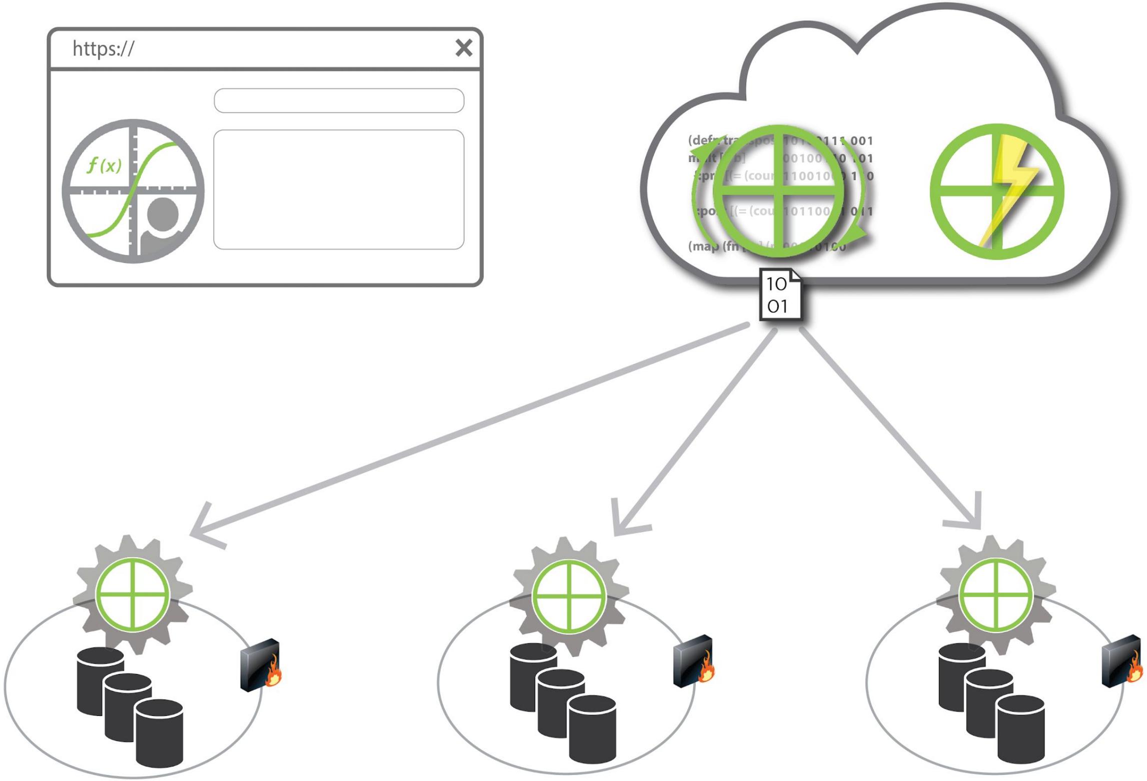
Inpher-hosted XOR Service
(never exposed to data)



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)



Data Source 1

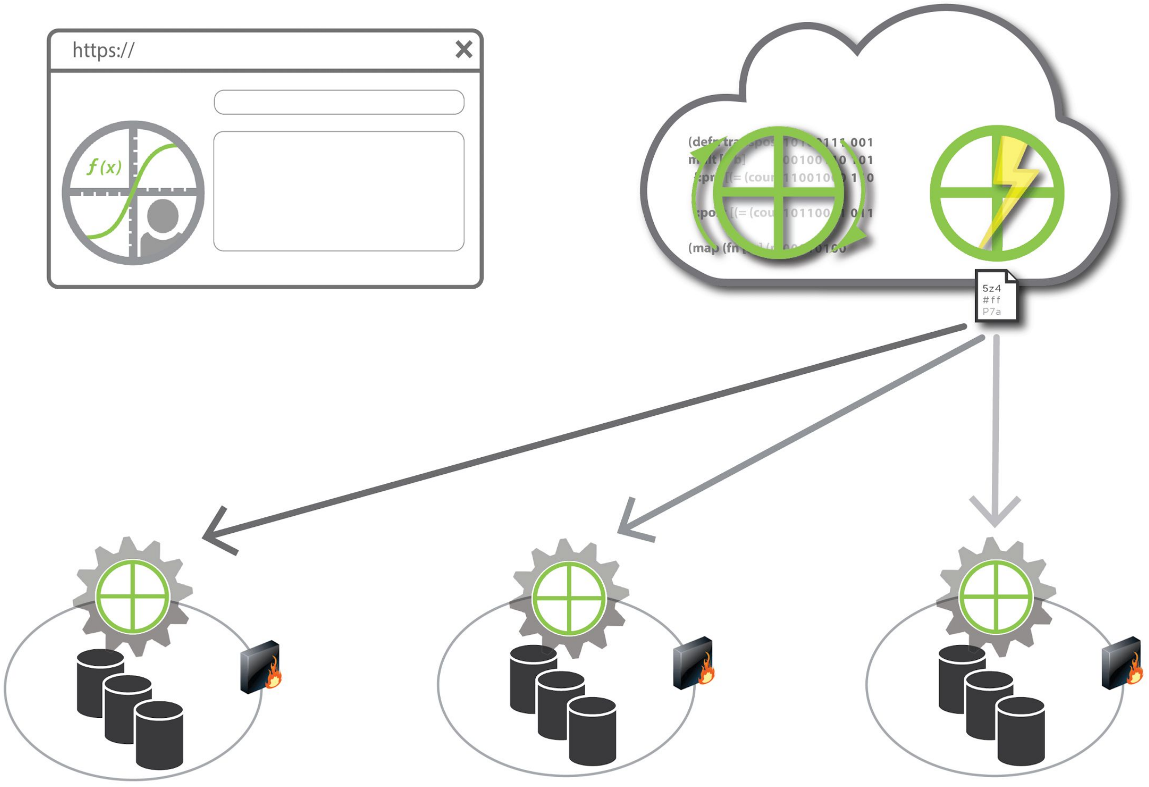
Data Source 2

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)



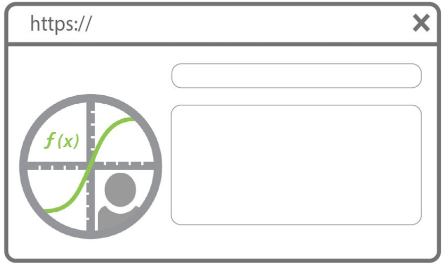
Data Source 1

Data Source 2

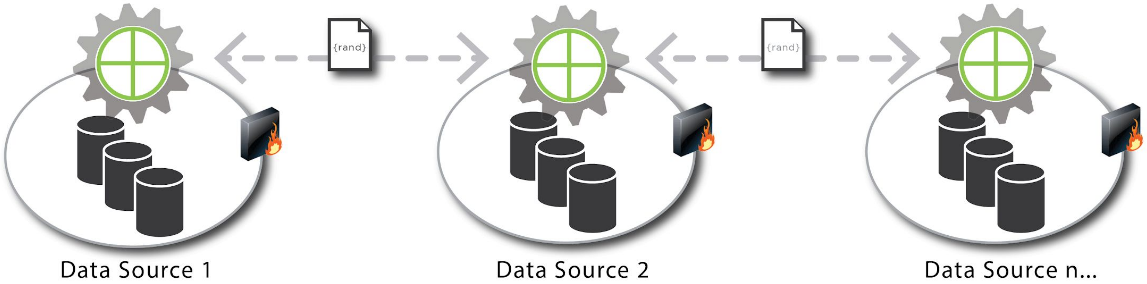
Data Source n...

Offline Phase: Random triplets are generated and distributed

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



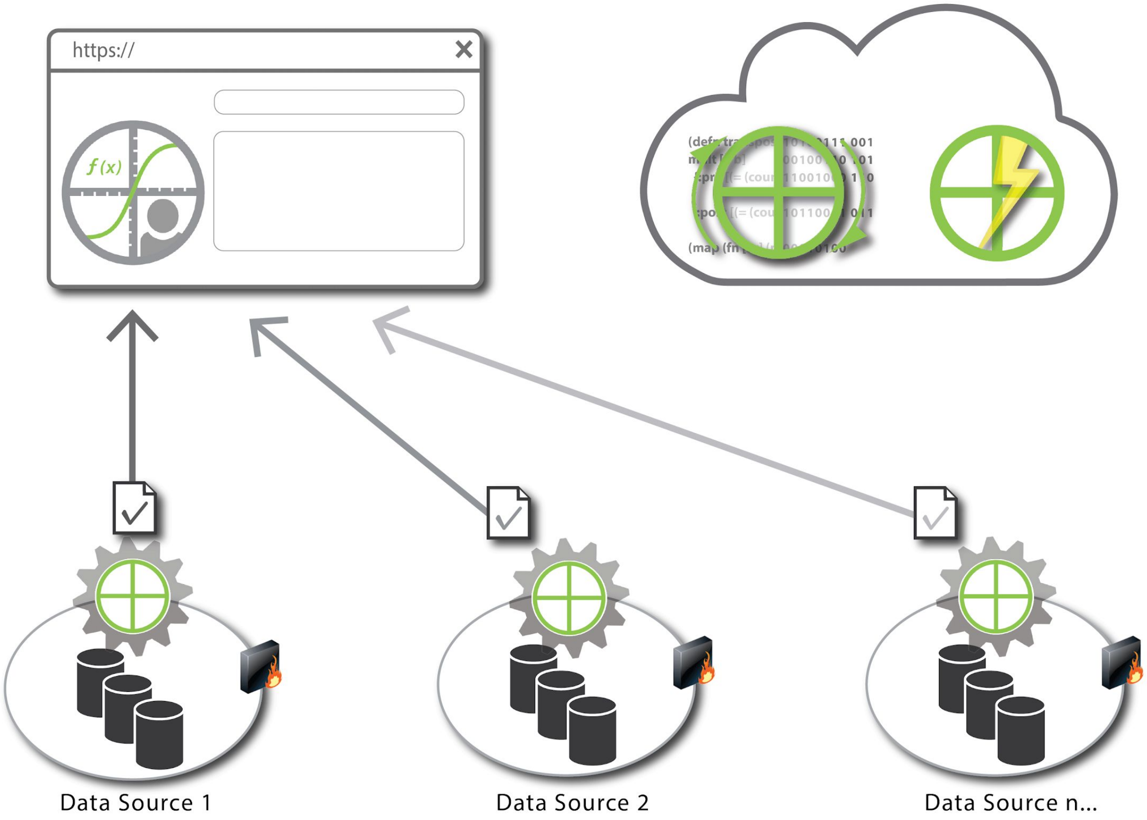
Inpher-hosted XOR Service
(never exposed to data)



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)



Partial results sent to Analyst Platform to construct final output

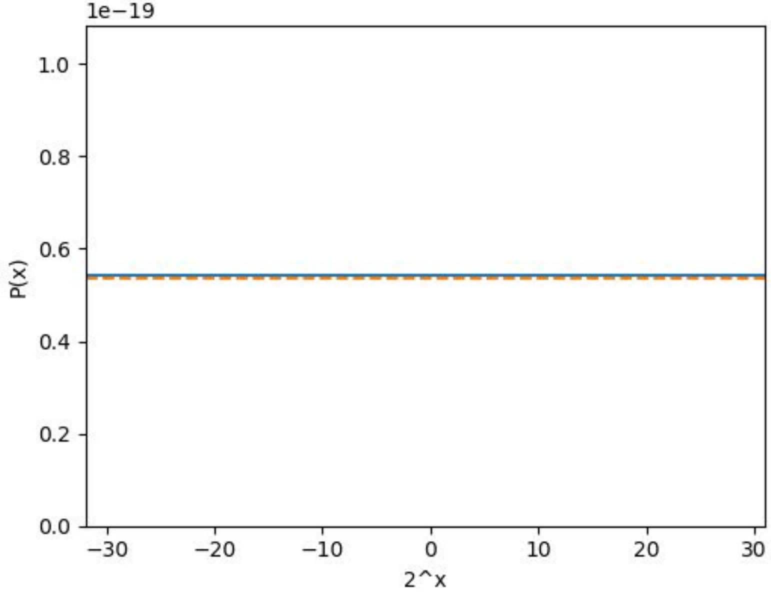
From library to full-blown compiler

- Computations **distributed** in reality
    - Must target a runtime that distributes computation and deploys
- Require **static analysis** of programs
    - Compute **statistical distributions** of intermediate values
    - **Optimize** for memory usage and communication

Masking numbers: some intuition

Masking integers

- Security: looking at two masked values, can I infer anything about their relationship? If so, how many computations?
- Uniform distribution possible over $\mathbb{Z}_n$
  - **information-theoretic security**



Masking floating point numbers

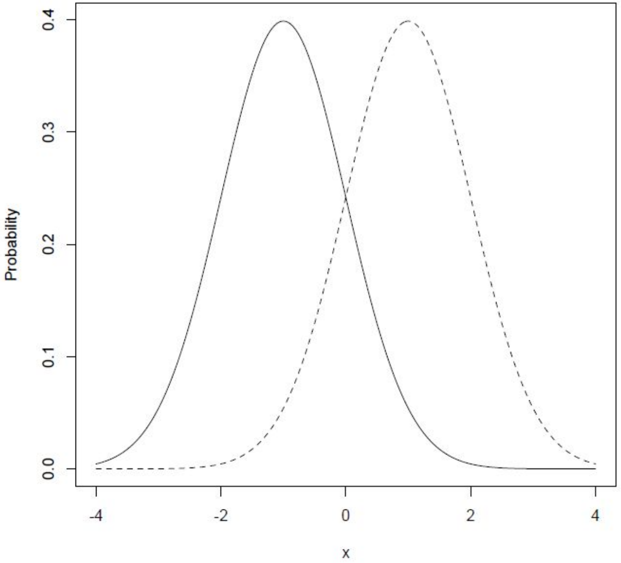


Figure: Masking $x = \pm 1$, $\sigma = 1$

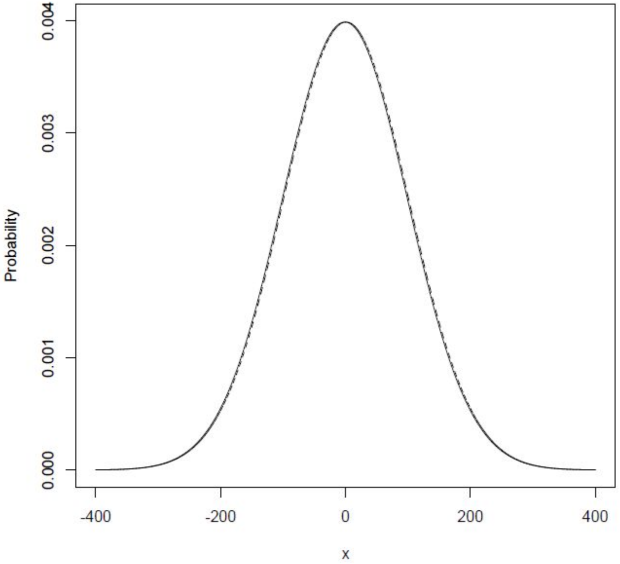


Figure: Masking $x = \pm 1$, $\sigma = 100$

Masking floating point numbers

- **Computational security**: an attacker needs to work **a lot** to distinguish two masked values

Masking floating point numbers

- **Computational security**: an attacker needs to work **a lot** to distinguish two masked values
- Masking and multiplying can blow up quickly

Masking floating point numbers

- **Computational security**: an attacker needs to work **a lot** to distinguish two masked values
- Masking and multiplying can blow up quickly
- **Fixed-point representation** as a solution:
  - Use different number representations at the back
  - Helps avoiding format explosion
- Needs to be **analysed statically**

An overview of the compiler

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.col

  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;
}
```

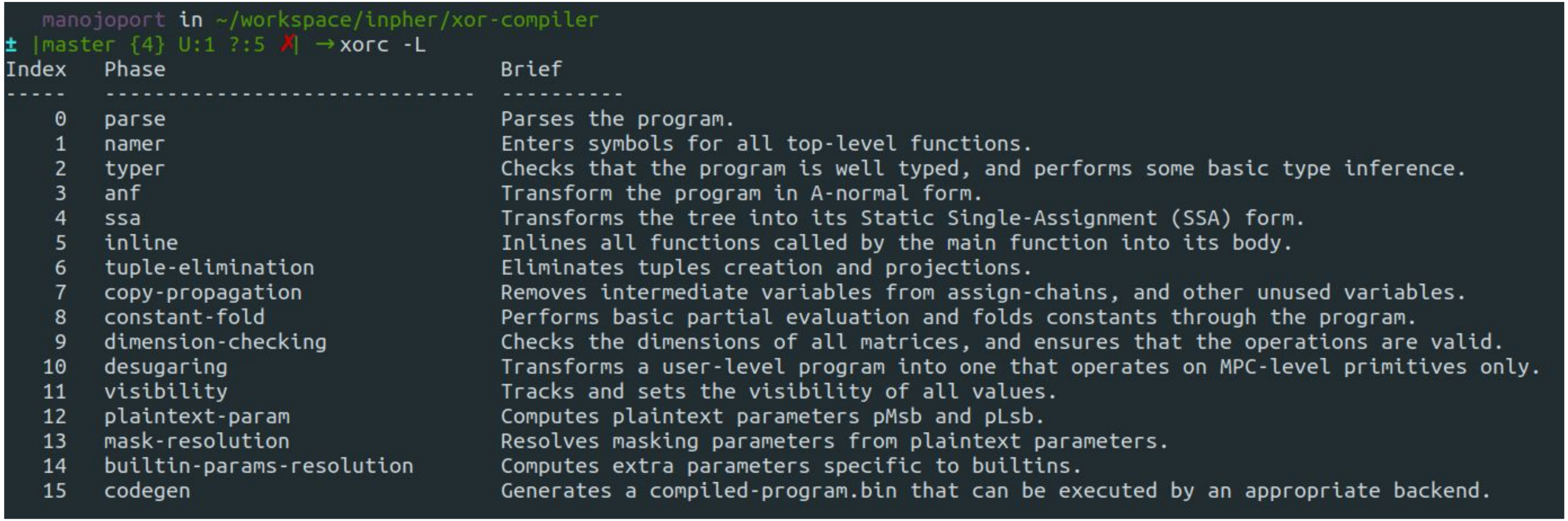
builtin

builtin

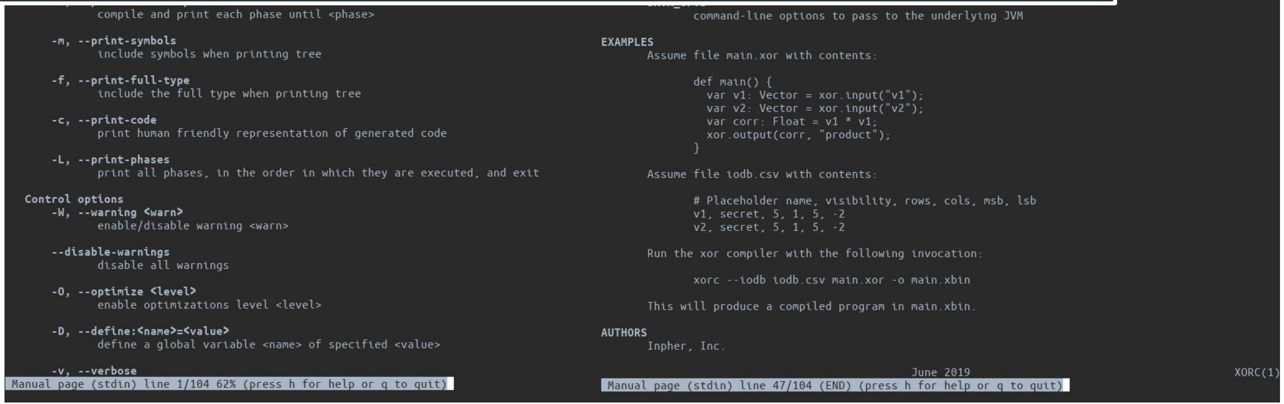
```
def linreg(y: Vector, X: Matrix): Vector {
  var A: Matrix = xor.transpose(X) * X;
  var b: Vector = xor.transpose(X) * y;
  return solve(A, b);
}

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.
```



```
              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)
```

```
              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

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

Linear Regression - Assembly

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

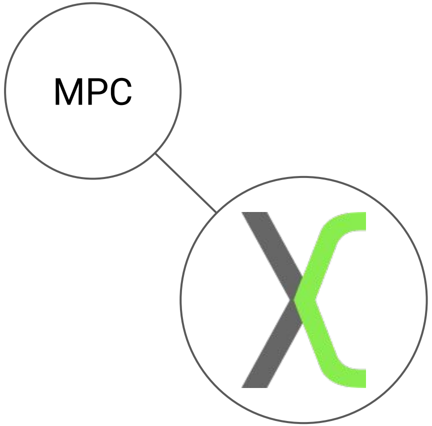
Linear Regression - Assembly

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

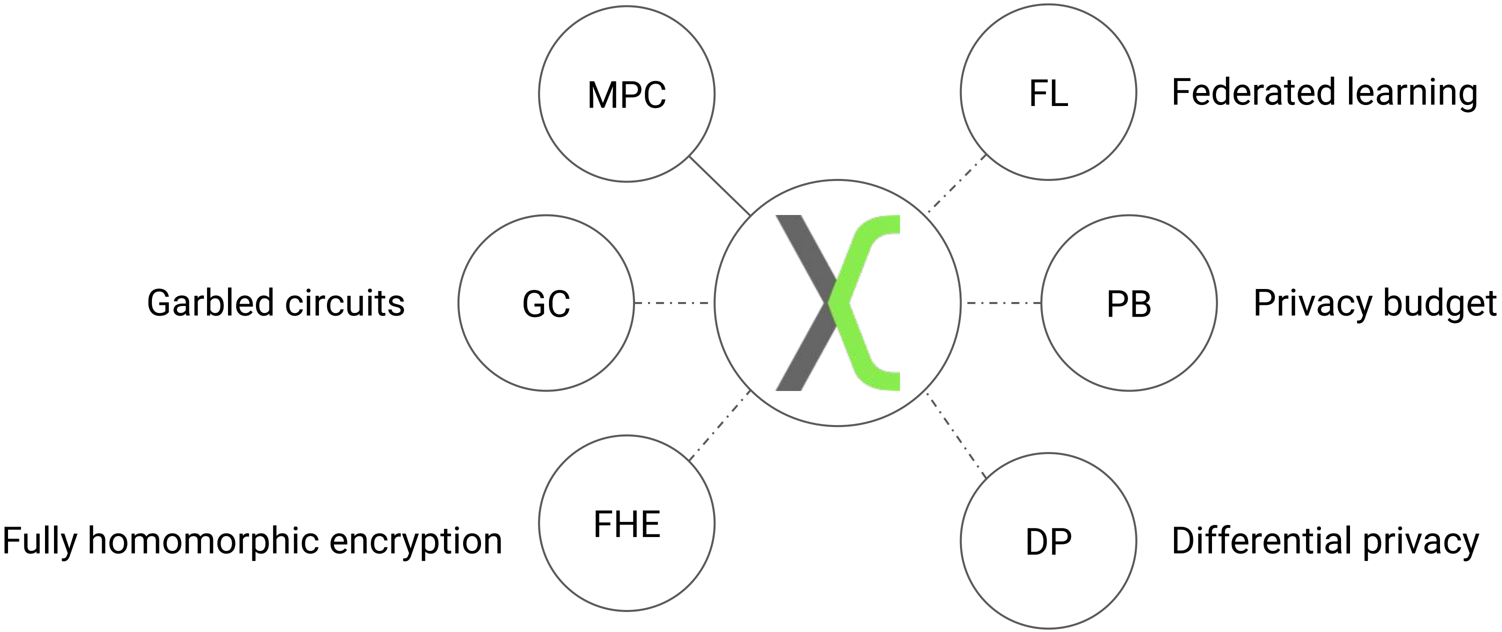
Linear Regression - Assembly

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

Compiling to preserve privacy - beyond MPC

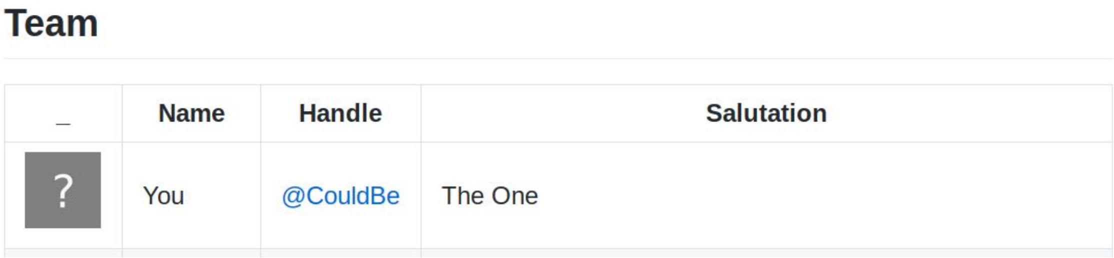


Compiling to preserve privacy - beyond MPC



Team

_	Name	Handle	Salutation
	Anton	@antonrd	Acolyte of the Pythonic Way, Builder of Builtins, Hero of the Seven Setups, and Supreme Sage of Artificial Intelligence
	Benedikt	@Picnixz	Disciple of Taz, Breaker of Changes, and Guardian of Precision
	Dimitar	@djetchev	The Omnipresent, Grand Master of Numbers, and Oracle of the Ivory Tower
	Iulian	@dragos	First of His Name, Champion of the Phases, and Elder of Scala
	Jakob	@jodersky	Second of His Name, Writer of Compilers, and Prodigal Son
	Manohar	@manojo	Ambassador to the High Council, Slayer of the Jirassic, and Chief Prophet of Linguistics



Team

_	Name	Handle	Salutation
?	You	@CouldBe	The One

Merci beaucoup!