General Purpose Frameworks for Secure Multi-party Computation

Marcella Hastings  Brett Hemenway  Daniel Noble  Steve Zdancewic

University of Pennsylvania
Secure multi-party computation (MPC)

MPC allows a group of mutually distrustful parties to compute a function on their joint inputs without revealing anything beyond the output.

Example: Danish sugar beet auction [BCD+08]

Parties: beet farmers, government buyer, research university

Inputs: Beet prices, yields

Outputs: Market clearing price
Secure multi-party computation (MPC)

MPC allows a group of mutually distrustful parties to compute a function on their joint inputs without revealing anything beyond the output.

Example: Danish sugar beet auction [BCD+08]

**Parties:** beet farmers, government buyer, research university

**Inputs:** Beet prices, yields

**Outputs:** Market clearing price
Beyond Beets: MPC in practice

Blind auction
[BCD+08]
Beyond Beets: MPC in practice

Blind auction
[BCD+08]

Fraud detection
[BJSV16]
Beyond Beets: MPC in practice

- Blind auction [BCD+08]
- Fraud detection [BJSV16]
- Parameter computation [BGM17]
Beyond Beets: MPC in practice

- Blind auction [BCD+08]
- Fraud detection [BJSV16]
- Parameter computation [BGM17]
- Financial statistics [BLV17]
Beyond Beets: MPC in practice

Blind auction [BCD+08]

Fraud detection [BJSV16]

Parameter computation [BGM17]

Financial statistics [BLV17]

Government applications

Private companies
Motivating end-to-end frameworks for MPC

Custom one-off solutions are unsustainable
Motivating end-to-end frameworks for MPC

Custom one-off solutions are unsustainable

Protocols assumed impractical until Fairplay [MNPS04]
Motivating end-to-end frameworks for MPC

Custom one-off solutions are unsustainable

Protocols assumed impractical until Fairplay [MNPS04]

Performance improvements rapidly advanced state-of-the-art

- OT extension [IKNP03]
- Free XOR gates [KS08]
- Half-gates [ZRE15]
- AES-NI
Modern General-Purpose Frameworks

Who are frameworks designed for?
What types of cryptographic settings do they use?
Are they suitable for use in large-scale applications?
Modern General-Purpose Frameworks

Who are frameworks designed for?
What types of cryptographic settings do they use?
Are they suitable for use in large-scale applications?
Contributions
General purpose frameworks for secure multi-party computation [HHNZ19]

Survey

Surveyed 9 frameworks and 2 circuit compilers
Recorded protocol, feature, implementation details
Evaluated usability criteria
Contributions
General purpose frameworks for secure multi-party computation [HHNZ19]

Survey
Surveyed 9 frameworks and 2 circuit compilers
Recorded protocol, feature, implementation details
Evaluated usability criteria

Open-source framework repository
Three sample programs in every framework
Docker instances with complete build environments
Documentation on compilation and execution

github.com/mpc-sok/frameworks
Findings

Most frameworks are in good shape!

- Diverse set of threat models and protocols
- Expressive high-level languages
- Accessible, open-source, and compilable
Findings

Most frameworks are in good shape!

Diverse set of threat models and protocols
Expressive high-level languages
Accessible, open-source, and compilable

Room for improvement

Engineering limitations
Barriers to usability
Frameworks and protocol families

- Garbled circuit:
  - EMP-toolkit
  - Obliv-C
  - ObliVM
  - TinyGarble

- Multi-party circuit based:
  - ABY

- Hybrid:
  - SCALE-MAMBA
  - Sharemind
  - PICCO

- Wysteria
Frameworks and protocol families

- Garbled circuit
  - EMP-toolkit
  - Obliv-C
  - ObliVM
  - TinyGarble

- Multi-party circuit based
  - ABY

- Hybrid
  - SCALE-MAMBA
  - Sharemind
  - PICCO

- Wysteria
Garbled circuit protocols
Introduced by [Yao82, Yao86]

Functions represented as Boolean circuits
Typically semi-honest, 2-party
Constant-round communication, volume $\propto$ circuit size
Frameworks and protocol families

- Garbled circuit: EMP-toolkit, Obliv-C, ObliVM, TinyGarble
- Hybrid: SCALE-MAMBA, Sharemind, PICCO
- Multi-party circuit based: ABY, Wysteria
Multi-party circuit-based protocols
Introduced by [GMW87, BGW88, CCD88]

Functions represented as Boolean or arithmetic circuits
Data represented as linear secret shares
Various threat models and protocol types
(information-theoretic or cryptographic)
Rounds, volume of communication $\propto$ multiplication gates
Frameworks and protocol families

Garbled circuit
- EMP-toolkit
- Obliv-C
- ObliVM
- TinyGarble

Multi-party circuit based
- ABY
- Wysteria

Hybrid
- SCALE-MAMBA
- Sharemind
- PICCO
Frameworks and protocol families

Garbled circuit
- EMP-toolkit
- Obliv-C
- ObliVM
- TinyGarble

Hybrid
- SCALE-MAMBA
- Sharemind
- PICCO

Multi-party circuit based
- ABY
- Wysteria
Hybrid protocols

Integrates optimized subprotocols for common functions
  Bitwise operators in arithmetic settings
  Matrix operations

Seamless front-end experience (no explicit protocol selection)
Currently: One-to-one mapping from operations to protocols
Hybrid protocols

Integrates optimized subprotocols for common functions
  Bitwise operators in arithmetic settings
  Matrix operations
Seamless front-end experience (no explicit protocol selection)
Currently: One-to-one mapping from operations to protocols
Frameworks and protocol families

Hybrid

Garbled circuit

EMP-toolkit
Obliv-C
ObliVM
TinyGarble

SCALE-MAMBA
Sharemind
PICCO

Multi-party circuit based

ABY

Wysteria
Frameworks and protocol families (2019)

- Garbled circuit
  - EMP-toolkit
  - Obliv-C
  - ObliVM
  - TinyGarble

- Multi-party circuit based
  - ABY
  - HyCC
  - ABY³

- MP-SPDZ
  - Wysteria

- Hybrid
  - SCALE-MAMBA
  - Sharemind
  - PICCO
  - EzPC
  - JIFF

- FRESCO
Design decisions

**Architecture:** system structure and data representation

**Circuit model:** representing data-independent paradigm

**Language accessibility:** cryptographic abstraction level
Circuits are a data-independent representation.

Branching programs are flattened in this model.

Non-expert users might not recognize this performance disparity.
Data independence: Private conditionals

Should branching programs reveal atypical performance?

Obliv-C: traditional paradigm

```c
obliv int result;
obliv if (a >= b) {
    result = a * a;
} else {
    result = b;
}
```
Data independence: Private conditionals

Should branching programs reveal atypical performance?

Obliv-C: traditional paradigm

```c
obliv int result;
obliv if (a >= b) {
    result = a * a;
} else {
    result = b;
}
```

EMP-toolkit: explicit branch selection

```c
Bit a_bigger = a.geq(b);
Integer result = b.select(a_bigger, a * a);
```
Data independence: Private conditionals
Should branching programs reveal atypical performance?

Obliv-C: traditional paradigm

```c
obliv int result;
obliv if (a >= b) {
    result = a * a;
} else {
    result = b;
}
```

EMP-toolkit: explicit branch selection

```c
Bit a_bigger = a.geq(b);
Integer result = b.select(a_bigger, a * a);
```

Recommendation
Depends on your users, but data independence is a good paradigm
Design decisions: Cryptographic abstraction level
Should the user have control over the underlying cryptographic representation?

Frigate: standard (C-style) abstraction

```c
int result = 0;
for (int i = 0; i < LEN; i++) {
    result = result + (A.data[i] * B.data[i]);
}
```
Design decisions: Cryptographic abstraction level
Should the user have control over the underlying cryptographic representation?

Frigate: standard (C-style) abstraction

```c
int result = 0;
for (int i=0; i<LEN; i++) {
    result = result + (A.data[i] * B.data[i]);
}
```

PICCO: custom primitive, high level abstraction

```c
int result = A @ B;
```
Design decisions: Cryptographic abstraction level
Should the user have control over the underlying cryptographic representation?

**ABY: Low-level access**

```c
share *A, *B;
A = circ->PutMULGate(A, B);
A = circ->PutSplitterGate(A);
for (uint32_t i = 1; i < LEN; i++) {
    A->set_wire_id(0, circ->PutADDGate(A->get_wire_id(0),
                                       A->get_wire_id(i)));
}
A->set_bitlength(1);
share *result = circ->PutOUTGate(A, ALL);
```
Software engineering

Complicated, non-trivial build systems

Set up certificate authority or PKI
Compile specific OpenSSL version from source
No dependency lists, manual search for compile errors
Estimated time: 1-2 weeks per framework
Software engineering

Complicated, non-trivial build systems

- Set up certificate authority or PKI
- Compile specific OpenSSL version from source
- No dependency lists, manual search for compile errors
- Estimated time: 1-2 weeks per framework

Significant software projects

- Cryptographic protocols
- Distributed communication
- Interfacing with other systems
Language documentation: How do I write secure code?
Code samples: What does a working example look like?
Code documentation: How does this example work?
Online support: Where can I ask questions?
Open-source: Can I run this without complex licensing?

Half the frameworks have no more than 3 of these 😊
Limited language documentation is frustrating

CBMC-GC:

```c
int mpc_main(int alice, int bob) {
    return alice * bob;
}
```

$ make
[...]
Uncaught exception: Unknown literal: 33. Did you forget to return a value or assign a value to a OUTPUT variable?
Limited language documentation is frustrating

**CBMC-GC**: Arguments must be called `INPUT_<var>`

```c
int mpc_main(int INPUT_alice, int INPUT_bob) {
    return INPUT_alice * INPUT_bob;
}
```

$ make

[...]

Gates: 5648 with 1986 Non-XOR and 0 LUTs
Depth: 151 with 32 Non-XOR
Limited language documentation is frustrating

**CBMC-GC:** Arguments must be called `INPUT_<var>`

**ObliVM:**

```c
int main(int alice, int bob){
    secure int result = alice * bob;
    return result;
}
```

$ ./run-compiler 12345 multiply.lcc


Was expecting one of: ⟨ IDENTIFIER ⟩ ... "[" ... @" ... "i" ...
Limited language documentation is frustrating

**CBMC-GC:** Arguments must be called `INPUT_<var>`

**ObliVM:** *alice* and *bob* are reserved keywords

```c
int main(int aaaaaa, int bbb) {
    secure int result = aaaaaa * bbb;
    return result;
}
```

$ ./run-compiler 12345 multiply.lcc
[INFO] The program type checks
[INFO] Compiling mult3.lcc succeeds
[INFO] Compilation finishes successfully.
Limited language documentation is frustrating

**CBMC-GC:** Arguments must be called `INPUT_{<var>}`

**ObliVM:** `alice` and `bob` are reserved keywords

**Wysteria:**

```wysteria
let richer = \x:ps . \w:W \ x \ nat .

let b @ sec(x) =
  wfold x (w, 0, \accum:nat . \p:ps . \n:nat .
    if accum > n then accum
    else n )

in b

let all = { !Alice, !Bob } in
let w = (wire !Alice:10) ++ (wire !Bob:100) in
richer all w
```

$$\text{wysteria} \ -i\text{-am} \ Alice \ -gmw\text{-port} \ 9000 \ examples/tutorial.wy$$

File examples/fakemill.wy, line 1, character 16: syntax error at ‘:’
Limited language documentation is frustrating

CBMC-GC: Arguments must be called \texttt{INPUT_{<var>}}

ObliVM: \texttt{alice} and \texttt{bob} are reserved keywords

Wysteria: Language docs don’t account for parser limitations

\begin{verbatim}
let \texttt{richer = \{(x:ps\{true\}) . \{(w:W x nat) .
let \texttt{tmp @ par(x) =
let \texttt{b @ sec(x) =
let \texttt{result = wfold x [w ; 0;
\{(accum:nat) . \{(p:ps\{true\}) . \{(n:nat) .
if accum > n then accum
else n ]
in \texttt{result}
in \texttt{b}
in \texttt{wire x:tmp}
in \texttt{let all = \{ !Alice, !Bob \} in
let \texttt{w = (wire !Alice:10) ++ (wire !Bob:100) in
richer all w
\end{verbatim}

\$ \texttt{wysteria \_i-am Alice \_gmw-port 9000 examples/tutorial.wy}
done with type checking the program
Limited language documentation is frustrating

**CBMC-GC:** Arguments must be called `INPUT_<var>`

**ObliVM:** *alice* and *bob* are reserved keywords

**Wysteria:** Language docs don’t account for parser limitations

**EMP-toolkit:** ~1 comment per 600 lines of code
Documentation appreciation and recommendations

Frameworks with excellent documentation

**ABY**: 35-page language guide; only slightly out-of-date

**SCALE-MAMBA**: 100+ pages of documentation

**Sharemind**: Auto-generated language guide online
Documentation appreciation and recommendations

Frameworks with excellent documentation

ABY: 35-page language guide; only slightly out-of-date
SCALE-MAMBA: 100+ pages of documentation
Sharemind: Auto-generated language guide online

Two recommendations for maintainers

Multiple types of documentation drastically increase usability
Online resources are sustainable and reduce workload
Produces a living FAQ
Allows users to interact
Good news for usability

Documentation issues aren’t fundamental
IARPA HECTOR includes usability criteria

Recent frameworks focus on usability!*

“JIFF is built to be highly flexible with a focus on usability [...] designed so that developers need not be familiar with MPC techniques or know the details of cryptographic protocols in order to build secure applications.”

HyCC makes “highly efficient hybrid MPC [...] accessible for developers without cryptographic background.”

____________________________________________

*Claims made by authors may not be verified by the speaker.
Future directions in MPC frameworks

Continued support for multiple settings
   Extend frameworks with different threat models and protocols

Better integration of work in other disciplines
   Heavy-duty circuit compilers (TinyGarble)
   Formal guarantees about front-ends (Wysteria, ObliVM)

Maintaining the repository
   I’m continuing to add modern frameworks
   We accept pull requests!
General Purpose Frameworks for Secure Multi-party Computation

Marcella Hastings
Brett Hemenway
Daniel Noble
Steve Zdancewic

University of Pennsylvania

github.com/mpc-sok/frameworks