MPyC – Python Package for Secure Multiparty Computation

Berry Schoenmakers
Coding & Crypto group
Dept of Mathematics & Computer Science
Usability
“Love Game” for Alice and Bob

Alice’s rule:

Suppose Alice thinks “yes”

Bob’s rule:

Suppose Bob thinks “yes”

Alice and Bob make a random cut and open the deck …
Matching without embarrassments

Match!

No match

No match

No match

same up to rotation
Random cut of five-card deck
Tom Verhoeff’s crypto gadget

demonstrations.wolfram.com/ZeroKnowledgeMatchmaker
Python – widely used programming language
  • platform-independent
  • high-level, simple and readable
  • free, open-source, tons of applications
  • popular for data mining & machine learning

MPyC – open-source Python package for MPC
  • secure types to operate on secret-shared values
  • passive adversary, honest majority
  • asynchronous evaluation of underlying protocols
    – special coroutines: no explicit callbacks
  • transparent communication between parties
VIFF -> TUeVIFF -> MPyC

• VIFF – Virtual Ideal Functionality Framework [viff.dk]
  • PhD project Martin Geisler, Aarhus University
  • VIFF 1.0 released Dec. 2009 (0.1 released Oct. 2007)
  • contains many fundamental ideas
  • Marcel Keller also made ‘boost’ version

• Tomas Toft introduced VIFF at TUE as a Postdoc
• TUeVIFF – local “fork” at TU Eindhoven
  • developed with my postdocs and PhD students
    – Frank Blom, Niek Bouman, Sebastiaan de Hoogh, Mikkel Krøigaard, Meilof Veeningen, Niels de Vreede
  • also used for assignments and MSc thesis projects
MPC implementation projects @TUE

- Linear Programming, incl. fixed-point numbers  
  Sebastiaan de Hoogh
- ID3 decision trees  
  Ping Chen, Sebastiaan de Hoogh
- Shuffle / QuickSort / ORAM  
  Niels de Vreede
- DNA sequence alignment  
  Sakina Asadova, Meilof Veeningen
- Verifiable MPC – Trinocchio, Geppetri, incl. intro of inline-callbacks  
  Meilof Veeningen
- Linear algebra / ridge analysis  
  Frank Blom, Niek Bouman, Niels de Vreede
- Convolutional Neural Networks  
  Harm Campmans
MPyC main features

- \( m \)-party computation, up to \( t \) passive corruptions
  - \( m \geq 1 \) and \( 0 \leq t \leq (m-1)/2 \)
  - \( m = 1, t = 0 \): normal computation
  - \( m > 1, t = 0 \): parallel computation

- Secure types: secint, secfxp, secfld
  - overloaded operators +, -, *, /, ==, <
  - input/output methods
  - hides (pseudorandom) secret-sharing

- Asynchronous computation via MPyC coroutines
  - no explicit callbacks at all – not go to “callback hell”
  - natural control flow
**MPyC coroutines**

- Python coroutine uses `async/await` syntax, like in many other programming languages
- MPyC coroutine: special kind of Python coroutine

```python
@mpc.coroutine
async def mul(a, b) -> secint:
    a, b = await gather_shares(a, b)
    c = reshare(a * b)
    return c
```

- Call to async `mul()` returns immediately with `secint` object containing **empty placeholder**.
- Evaluation of expression \((s*t) + (u*v) * (w+x) * y * z\):
  - first creates tree with empty placeholders
  - then starts filling in these placeholders
Demos at [github.com/lschoe/mpyc](https://github.com/lschoe/mpyc)

- Secure sorting
  - from sorting networks
- Secret Santa
  - random permutation without fixed-points
- Convolutional neural network for MNIST dataset
  - recognizing hand-written digits 0,1,2,..,9

![Convolutional neural network diagram](image)
Conclusion

• **MPyC**: standard Python
• **Small footprint**: easier verification of framework
• **Balance usability and efficiency for**
  • rapid prototyping, executable specification
  • educational purposes, e.g.,
    – teaching secure/oblivious programming
    – explaining how MPC works – look inside MPyC
• **Some next steps:**
  • secure numpy arrays
  • transparent secret indexing
• See [github.com/lschoe/mpyc](http://github.com/lschoe/mpyc)
This work is part of projects that have received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 731583 (SODA) and No 780477 (PRIVILEGED)