Virtual Ideal Functionality Framework

High-Level Design Overview

Martin Geisler

BRICS
Department of Computer Science
University of Aarhus

September 20th, 2007

VIFF Features

- API for writing secure multi-party computations
  - Easy to use: a + b + c instead of add(add(a, b), c)
  - Simple if you already know Python
  - A custom language could be added
- Efficient asynchronous design
  - Automatic parallelism
  - Single-threaded (no locks!)
- Simple architecture
  - Small code base (2,500 lines)
  - Few layers of abstraction...
  - ...but hopefully sufficiently many

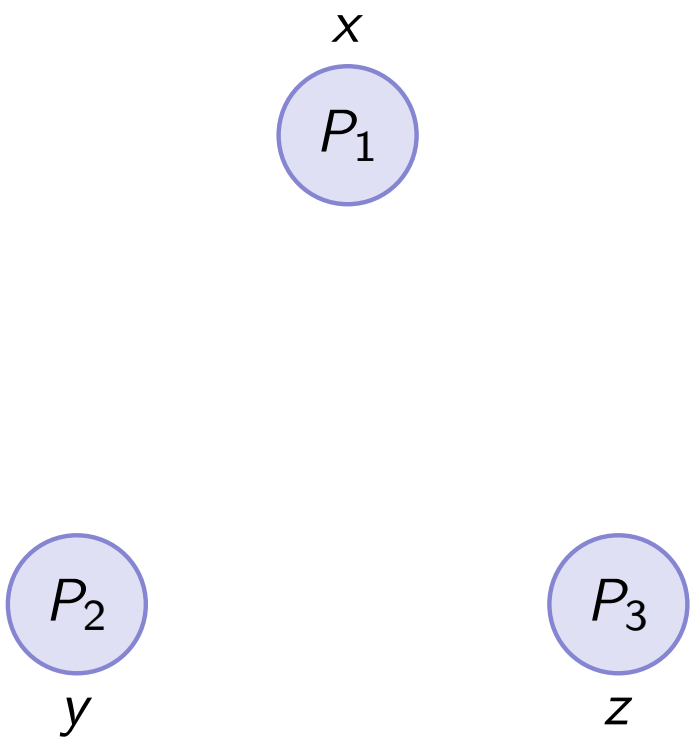
VIFF Features

- API for writing secure multi-party computations
  - Easy to use: a + b + c instead of add(add(a, b), c)
  - Simple if you already know Python
  - A custom language could be added
- Efficient asynchronous design
  - Automatic parallelism
  - Single-threaded (no locks!)
- Simple architecture
  - Small code base (2,500 lines)
  - Few layers of abstraction...
  - ...but hopefully sufficiently many

VIFF Features

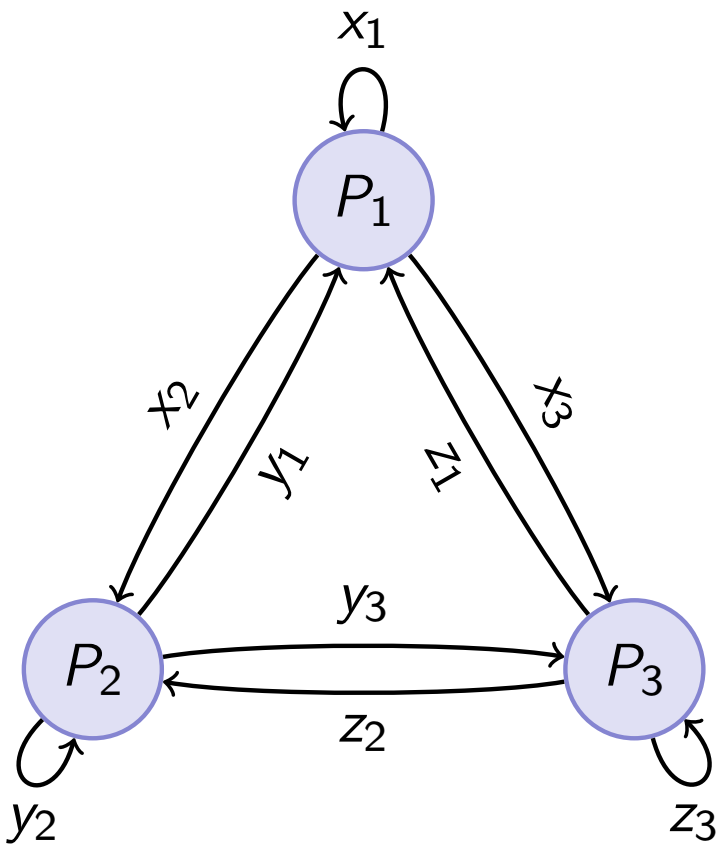
- API for writing secure multi-party computations
  - Easy to use: a + b + c instead of add(add(a, b), c)
  - Simple if you already know Python
  - A custom language could be added
- Efficient asynchronous design
  - Automatic parallelism
  - Single-threaded (no locks!)
- Simple architecture
  - Small code base (2,500 lines)
  - Few layers of abstraction...
  - ...but hopefully sufficiently many

Example MPC Calculation



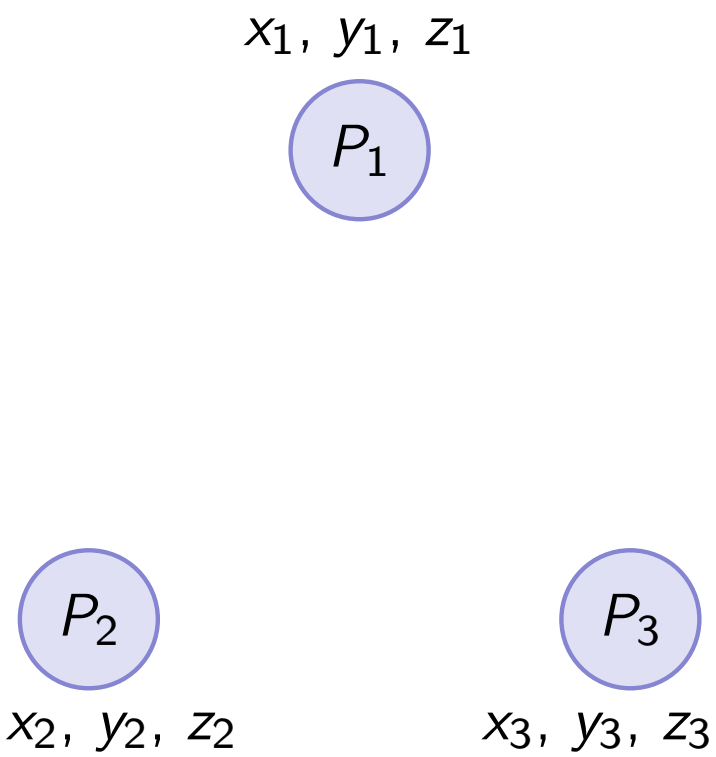
- Players P_1, P_2, and P_3
- Sharing x, y, and z
- Compute $r = (x + y) \cdot z$
- Opens the result r

Example MPC Calculation



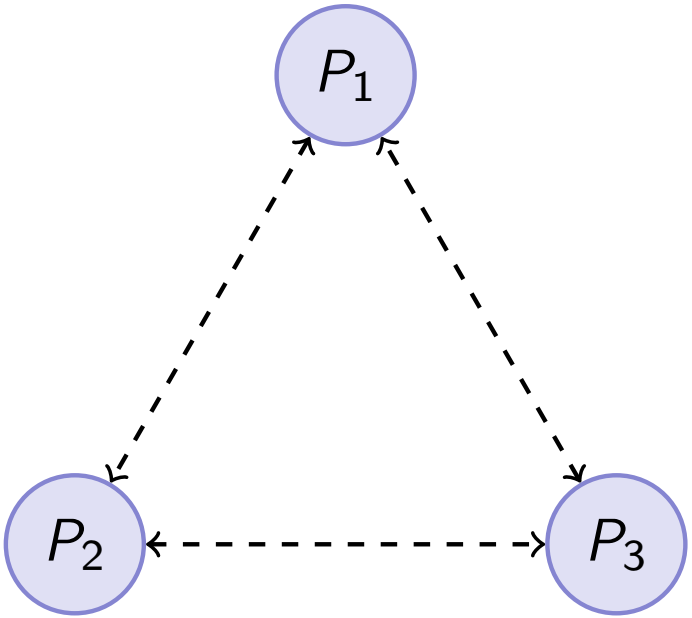
- Players P_1, P_2, and P_3
- Sharing x, y, and z
- Compute $r = (x + y) \cdot z$
- Opens the result r

Example MPC Calculation



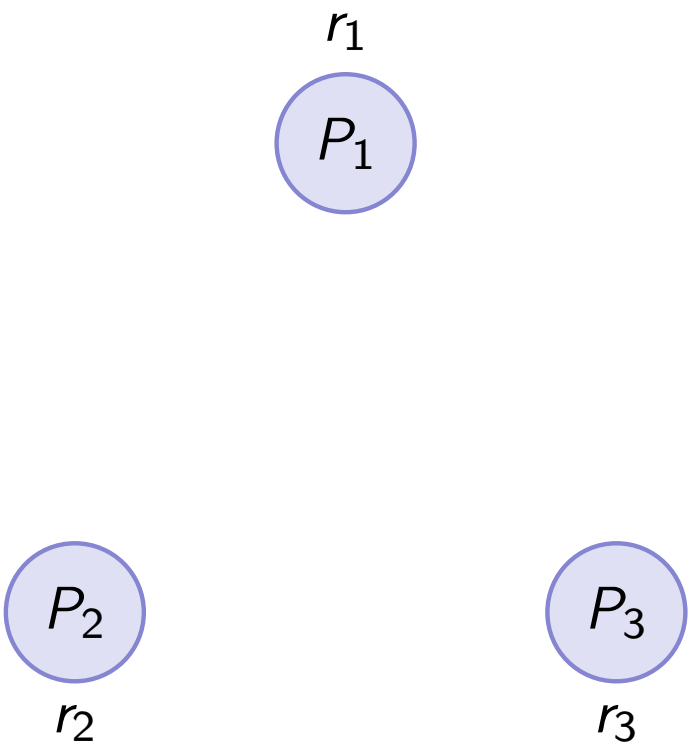
- Players P_1, P_2, and P_3
- Sharing x, y, and z
- Compute $r = (x + y) \cdot z$
- Opens the result r

Example MPC Calculation



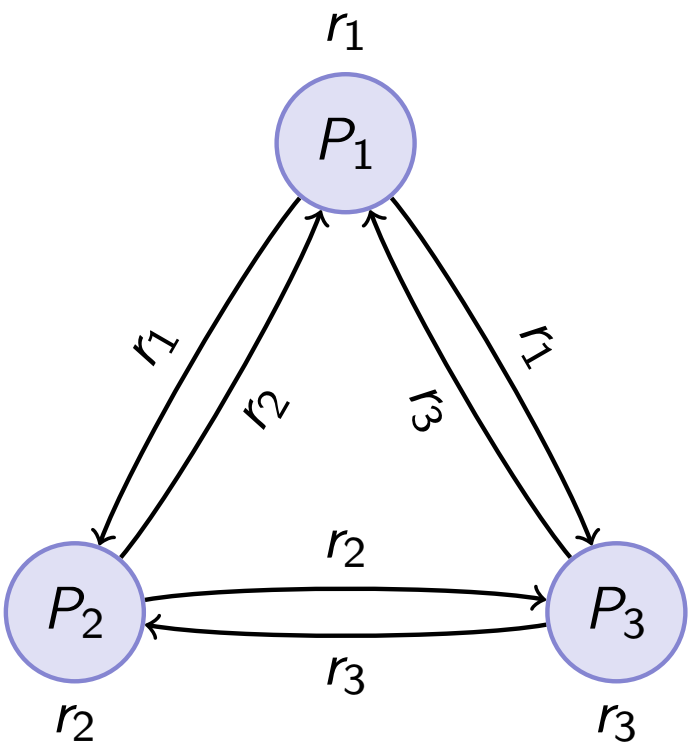
- Players P_1, P_2, and P_3
- Sharing x, y, and z
- Compute $r = (x + y) \cdot z$
- Opens the result r

Example MPC Calculation



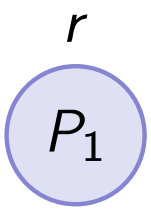
- Players P_1, P_2, and P_3
- Sharing x, y, and z
- Compute $r = (x + y) \cdot z$
- Opens the result r

Example MPC Calculation

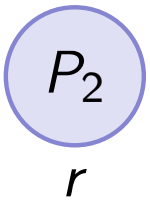


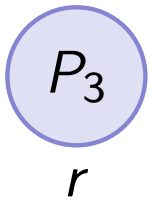
- Players P_1, P_2, and P_3
- Sharing x, y, and z
- Compute $r = (x + y) \cdot z$
- Opens the result r

Example MPC Calculation



- Players P_1, P_2, and P_3
- Sharing x, y, and z
- Compute $r = (x + y) \cdot z$
- Opens the result r





Example VIFF Program

```
import sys
from viff.field import GF
from viff.config import load_config
from viff.runtime import Runtime
from viff.util import dprint
```

- Import standard and VIFF modules

```
Z31 = GF(31)
my_id, conf = load_config(sys.argv[1])
my_input = Z31(int(sys.argv[2]))
```

- Load command line config

```
rt = Runtime(conf, my_id, 1)
x, y, z = rt.shamir_share(my_input)
result = (x + y) * z
```

- Create Runtime, do calculation

```
opened_result = rt.open(result)
dprint("Result: %s", opened_result)
rt.wait_for(opened_result)
```

- Open and print result

Example VIFF Program

```
import sys
from viff.field import GF
from viff.config import load_config
from viff.runtime import Runtime
from viff.util import dprint
```

- Import standard and VIFF modules

```
Z31 = GF(31)
my_id, conf = load_config(sys.argv[1])
my_input = Z31(int(sys.argv[2]))
```

- Load command line config

```
rt = Runtime(conf, my_id, 1)
x, y, z = rt.shamir_share(my_input)
result = (x + y) * z
```

- Create Runtime, do calculation

```
opened_result = rt.open(result)
dprint("Result: %s", opened_result)
rt.wait_for(opened_result)
```

- Open and print result

Example VIFF Program

```
import sys
from viff.field import GF
from viff.config import load_config
from viff.runtime import Runtime
from viff.util import dprint
```

- Import standard and VIFF modules

```
Z31 = GF(31)
my_id, conf = load_config(sys.argv[1])
my_input = Z31(int(sys.argv[2]))
```

- Load command line config

```
rt = Runtime(conf, my_id, 1)
x, y, z = rt.shamir_share(my_input)
result = (x + y) * z
```

- Create Runtime, do calculation

```
opened_result = rt.open(result)
dprint("Result: %s", opened_result)
rt.wait_for(opened_result)
```

- Open and print result

Example VIFF Program

```
import sys
from viff.field import GF
from viff.config import load_config
from viff.runtime import Runtime
from viff.util import dprint
```

```
Z31 = GF(31)
my_id, conf = load_config(sys.argv[1])
my_input = Z31(int(sys.argv[2]))
```

```
rt = Runtime(conf, my_id, 1)
x, y, z = rt.shamir_share(my_input)
result = (x + y) * z
```

```
opened_result = rt.open(result)
dprint("Result: %s", opened_result)
rt.wait_for(opened_result)
```

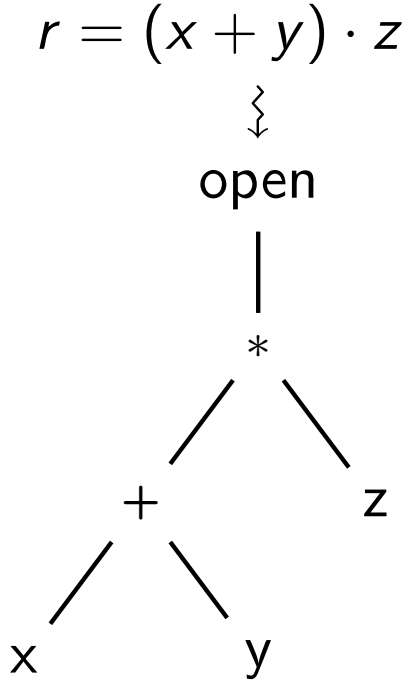
- Import standard and VIFF modules
- Load command line config
- Create Runtime, do calculation
- Open and print result

Example Configuration File

```
[Player 1]
  host = camel09
  port = 7100
[Player 2]
  host = camel17
  port = 7200
[Player 3]
  host = camel19
  port = 7300
  [[prss_keys]]
    1 3 = 0x47D5B9B367DAFC9267EDF518AD5B5F396299L
    2 3 = 0x7EBC55E5CF1D014D081EA428B5F35FD12C64L
  [[prss_dealer_keys]]
    [[[Dealer 1]]]
      1 3 = 0x903039893A06800A0FA7175CED8CC17B873BL
      2 3 = 0x11C9135423719339758 9A1D1C6455F87CB79L
    # More PRSS keys...

# End of config
```

Asynchronous Design



- Entire tree is scheduled as once
- Operations wait on their operands
- Results are sent when ready
- Result is a form of "greedy scheduling"
- Implicit synchronization, no rounds

Greedy Scheduling

Advantages:

- At least as efficient as round-based scheduling
- No cost when adding primitives (modular design)
- Automatic parallelism

Disadvantages:

- Not yet proven secure...

What has been Implemented?

- Shamir sharing
- PRSS
- Opening
- Addition, multiplication, exclusive-or
- Classic SCET comparison

Assumptions

Current primitives assume:

- Fixed number of players
- Passive and static adversary
- Threshold adversary structure, $t < n/2$

Conclusion

- VIFF offers a light-weight design for doing MPC
- Asynchronous design gives automatic parallelism

Installation instructions, source code and more:

- `http://viff.dk/`

Conclusion

- VIFF offers a light-weight design for doing MPC
- Asynchronous design gives automatic parallelism

Installation instructions, source code and more:

- `http://viff.dk/`

Questions?