# Implementing Asynchronous Multi-Party Computation

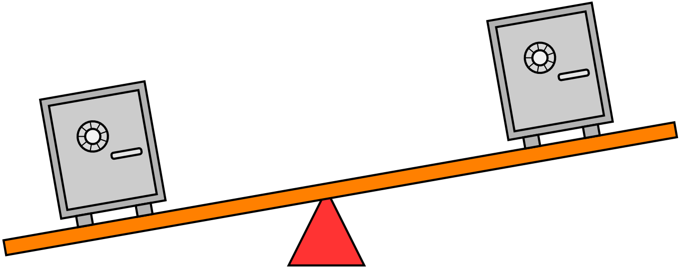

Martin Geisler

BRICS
Department of Computer Science
University of Aarhus


February 21st, 2008

Part I

## Secure Integer Comparison



# Secure Integer Comparison

- Given integers $a$ and $b$, securely compute $a > b$.

# Secure Integer Comparison

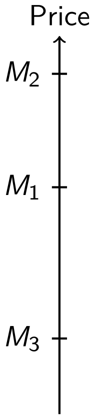
- Given integers $a$ and $b$, securely compute $a > b$.
- Many variations:
  - $a$, $b$ can be private, public or secret shared.
  - Same for the result.
  - We can have two or more players.

# Auctions

- Traditional auction:
  - Bidders must be on-line.
  - Bidding continues until a deadline is reached.

# Auctions

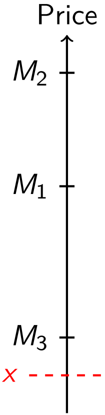
- Traditional auction:
  - Bidders must be on-line.
  - Bidding continues until a deadline is reached.
- Maximum bid auction:




- $P_i$ may submit a maximum bid $M_i$.

# Auctions

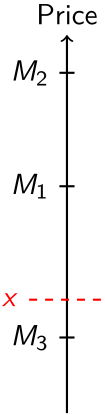
- Traditional auction:
  - Bidders must be on-line.
  - Bidding continues until a deadline is reached.
- Maximum bid auction:




- $P_i$ may submit a maximum bid $M_i$.
- A public current price $x$ is incremented until only one $M_i > x$.

# Auctions

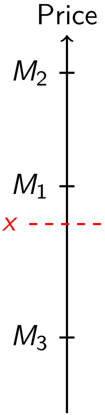
- Traditional auction:
  - Bidders must be on-line.
  - Bidding continues until a deadline is reached.
- Maximum bid auction:




- $P_i$ may submit a maximum bid $M_i$.
- A public current price $x$ is incremented until only one $M_i > x$.

# Auctions

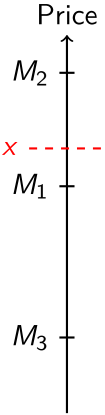
- Traditional auction:
  - Bidders must be on-line.
  - Bidding continues until a deadline is reached.
- Maximum bid auction:




- $P_i$ may submit a maximum bid $M_i$.
- A public current price $x$ is incremented until only one $M_i > x$.

# Auctions

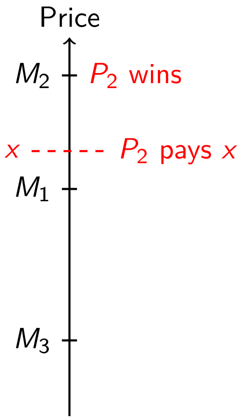
- Traditional auction:
  - Bidders must be on-line.
  - Bidding continues until a deadline is reached.
- Maximum bid auction:




- $P_i$ may submit a maximum bid $M_i$.
- A public current price $x$ is incremented until only one $M_i > x$.

# Auctions

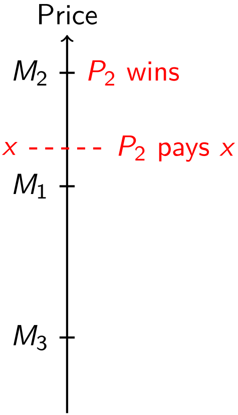
- Traditional auction:
  - Bidders must be on-line.
  - Bidding continues until a deadline is reached.
- Maximum bid auction:




- $P_i$ may submit a maximum bid $M_i$.
- A public current price $x$ is incremented until only one $M_i > x$.

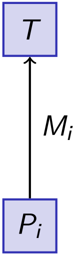
# Auctions

- Traditional auction:
  - Bidders must be on-line.
  - Bidding continues until a deadline is reached.
- Maximum bid auction:



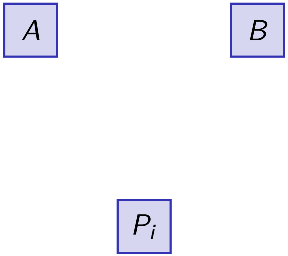

- $P_i$ may submit a maximum bid $M_i$.
- A public current price $x$ is incremented until only one $M_i > x$.
- Problem: Auction house knows $M_i$ and is a trusted third party.

# Removing Trust in the Auction House



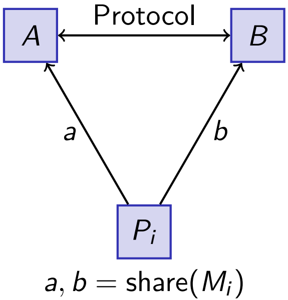

- Want to remove trusted party $T$.

# Removing Trust in the Auction House




- Want to remove trusted party $T$.
- Split $T$ into parties $A$ and $B$.

# Removing Trust in the Auction House




- Want to remove trusted party $T$.
- Split $T$ into parties $A$ and $B$.
- User $P_i$ shares $M_i$ into $a$ and $b$.
- $A$ gets $a$, $B$ gets $b$.
- $A$ and $B$ run a comparison protocol.

# Homomorphic Encryption Scheme

- Encryption:

$$E_{pk}(m, r) = g^m h^r \bmod n.$$

- Homomorphic:

$$E_{pk}(m, r) \cdot E_{pk}(m', r') \bmod n = E_{pk}(m + m' \bmod u, r + r').$$

- Check $c = E_{pk}(m, r)$ for $m = 0$:

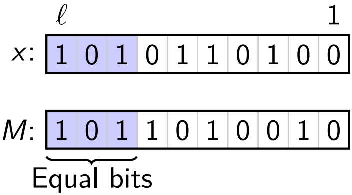
$$c^v \bmod n = (g^v)^m \bmod n.$$

# Calculating $M > x$

| | $\ell$ | | | | | | | | | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| $x$: | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| $M$: | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

- We wish to compute $M > x$ for $\ell$-bit numbers.
- $x_i$ is the $i$'th bit of $x$, $m_i$ is the $i$'th bit of $M$.

# Calculating $M > x$




- We wish to compute $M > x$ for $\ell$-bit numbers.
- $x_i$ is the $i$'th bit of $x$, $m_i$ is the $i$'th bit of $M$.

# Calculating $M > x$

| | $\ell$ | | | | | | | | | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| $x$: | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| $M$: | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| | | | | Significant bit | | | | | | |

- We wish to compute $M > x$ for $\ell$-bit numbers.
- $x_i$ is the $i$'th bit of $x$, $m_i$ is the $i$'th bit of $M$.

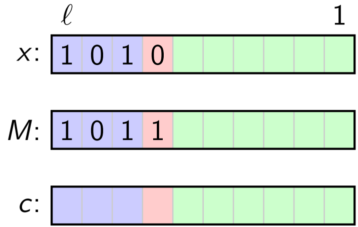
# Calculating $M > x$

$$\begin{array}{rcccccccccc}
 & \ell & & & & & & & & & 1 \\
x: & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}$$

$$M: \; 1 \; 0 \; 1 \; 1 \; \underbrace{0 \; 1 \; 0 \; 0 \; 1 \; 0}_{\text{Non-significant bits}}$$

- We wish to compute $M > x$ for $\ell$-bit numbers.
- $x_i$ is the $i$'th bit of $x$, $m_i$ is the $i$'th bit of $M$.

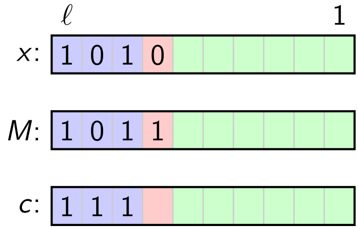
# Calculating $M > x$




- We wish to compute $M > x$ for $\ell$-bit numbers.
- $x_i$ is the $i$'th bit of $x$, $m_i$ is the $i$'th bit of $M$.
- Define the following:

$$c_i = x_i - m_i + 1 + \sum_{j=i+1}^{\ell} m_j \oplus x_j.$$

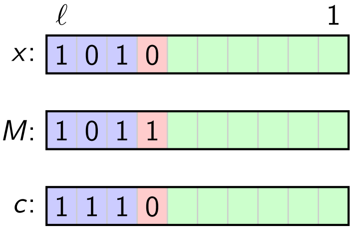
# Calculating $M > x$




- We wish to compute $M > x$ for $\ell$-bit numbers.
- $x_i$ is the $i$'th bit of $x$, $m_i$ is the $i$'th bit of $M$.
- Define the following:

$$c_i = x_i - m_i + 1 + \sum_{j=i+1}^{\ell} m_j \oplus x_j.$$

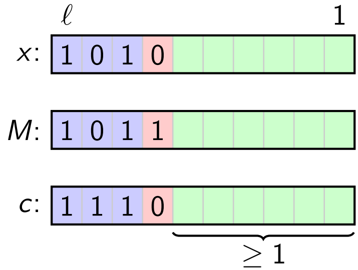
# Calculating $M > x$




- We wish to compute $M > x$ for $\ell$-bit numbers.
- $x_i$ is the $i$'th bit of $x$, $m_i$ is the $i$'th bit of $M$.
- Define the following:

$$c_i = x_i - m_i + 1 + \sum_{j=i+1}^{\ell} m_j \oplus x_j .$$

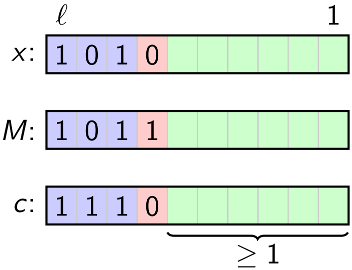
# Calculating $M > x$




- We wish to compute $M > x$ for $\ell$-bit numbers.
- $x_i$ is the $i$'th bit of $x$, $m_i$ is the $i$'th bit of $M$.
- Define the following:

$$c_i = x_i - m_i + 1 + \sum_{j=i+1}^{\ell} m_j \oplus x_j.$$

# Calculating $M > x$

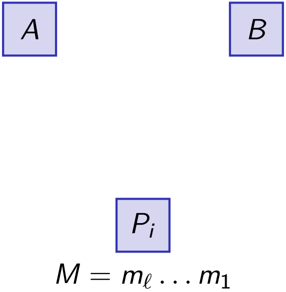



- We wish to compute $M > x$ for $\ell$-bit numbers.
- $x_i$ is the $i$'th bit of $x$, $m_i$ is the $i$'th bit of $M$.
- Define the following:

$$c_i = x_i - m_i + 1 + \sum_{j=i+1}^{\ell} m_j \oplus x_j.$$

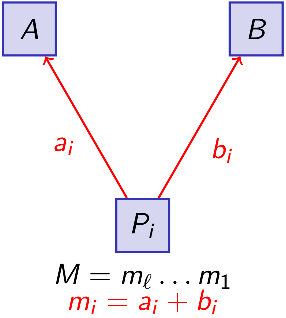
- $M > x \iff \exists i : c_i = 0.$

# Protocol for $M > x$




- $A$ and $B$ know $pk$, $A$ knows $sk$.
- Input $x$ is public, $M$ known to $P_i$.

# Protocol for $M > x$




- $A$ and $B$ know $pk$, $A$ knows $sk$.
- Input $x$ is public, $M$ known to $P_i$.
- Input $m_i$ additively secret shared.

# Protocol for $M > x$

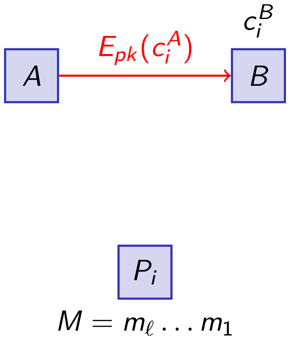
$c_i^A$ $A$ $c_i^B$ $B$

$P_i$

$M = m_\ell \dots m_1$

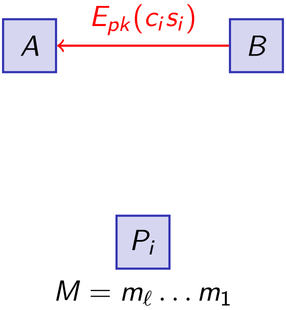
- $A$ and $B$ know $pk$, $A$ knows $sk$.
- Input $x$ is public, $M$ known to $P_i$.
- Input $m_i$ additively secret shared.
- $A$ and $B$ compute shares of $c_i$.

# Protocol for $M > x$



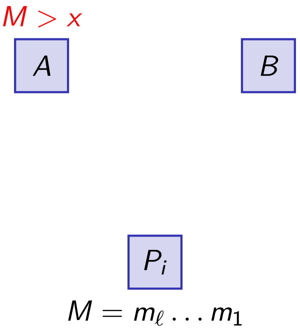

- $A$ and $B$ know $pk$, $A$ knows $sk$.
- Input $x$ is public, $M$ known to $P_i$.
- Input $m_i$ additively secret shared.
- $A$ and $B$ compute shares of $c_i$.
- $A$ sends $E_{pk}(c_i^A)$ to $B$.

# Protocol for $M > x$



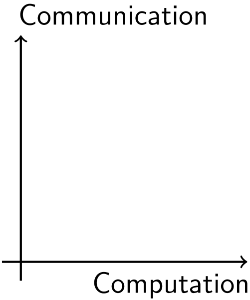

- $A$ and $B$ know $pk$, $A$ knows $sk$.
- Input $x$ is public, $M$ known to $P_i$.
- Input $m_i$ additively secret shared.
- $A$ and $B$ compute shares of $c_i$.
- $A$ sends $E_{pk}(c_i^A)$ to $B$.
- $B$ calculates $E_{pk}(c_i s_i)$ using the homomorphic property.
- $B$ sends shuffled $E_{pk}(c_i s_i)$ to $A$.

# Protocol for $M > x$




- $A$ and $B$ know $pk$, $A$ knows $sk$.
- Input $x$ is public, $M$ known to $P_i$.
- Input $m_i$ additively secret shared.
- $A$ and $B$ compute shares of $c_i$.
- $A$ sends $E_{pk}(c_i^A)$ to $B$.
- $B$ calculates $E_{pk}(c_i s_i)$ using the homomorphic property.
- $B$ sends shuffled $E_{pk}(c_i s_i)$ to $A$.
- $A$ checks if any $c_i s_i$ is zero.
- $\exists i : c_i s_i = 0 \iff M > x$.

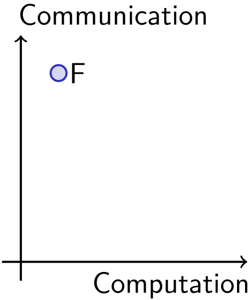
# Related Work




- Marc Fischlin's protocol:

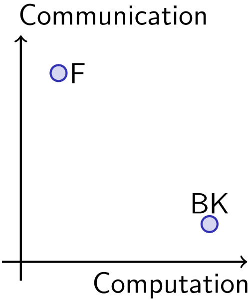
- Blake and Kolesnikov's protocol:

# Related Work



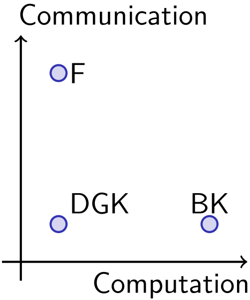

- Marc Fischlin's protocol:
  - Quadratic residuosity assumption.
  - Encoding expands by $\lambda$ factor.
- Blake and Kolesnikov's protocol:

# Related Work



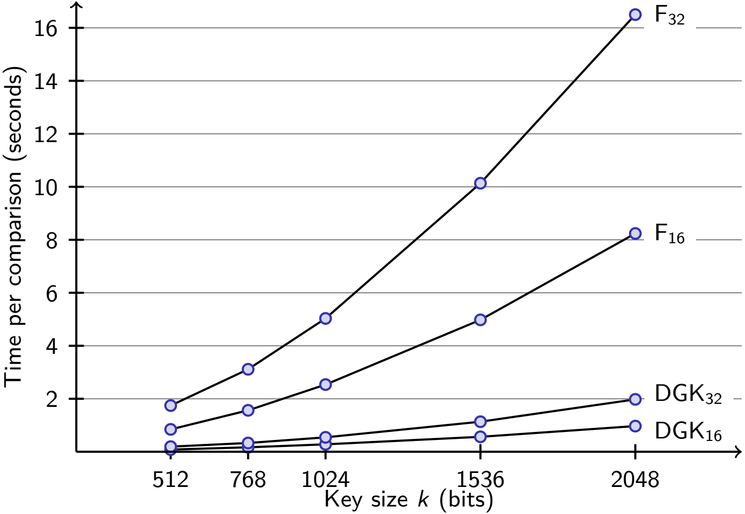

- Marc Fischlin's protocol:
  - Quadratic residuosity assumption.
  - Encoding expands by $\lambda$ factor.
- Blake and Kolesnikov's protocol:
  - Paillier encryption.
  - No expansion.

# Related Work



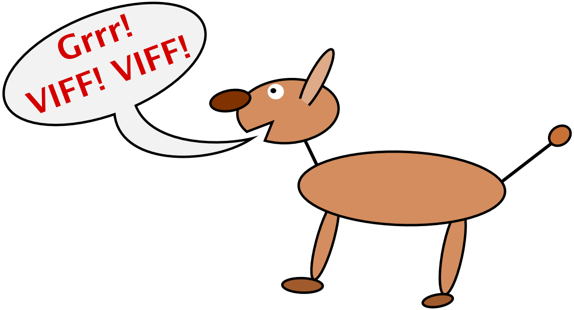

- Marc Fischlin's protocol:
  - Quadratic residuosity assumption.
  - Encoding expands by $\lambda$ factor.
- Blake and Kolesnikov's protocol:
  - Paillier encryption.
  - No expansion.
- Our protocol: Best of both worlds.

# Benchmark Results



# Part II

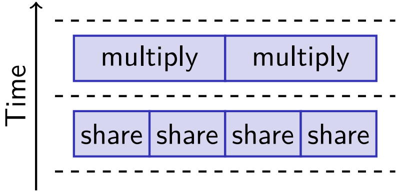
## Virtual Ideal Functionality Framework



# VIFF Overview

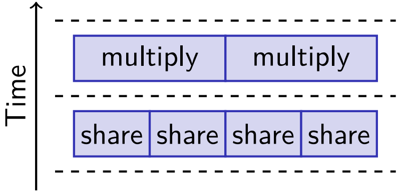
- Framework for specifying MPC.
- Provides building-blocks for larger protocols.
- Asynchronous design.
- Automatic parallel scheduling.

# Asynchronous vs. Synchronous

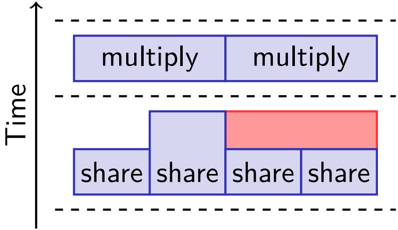



- All rounds equally fast.
- Optimal execution.

# Asynchronous vs. Synchronous



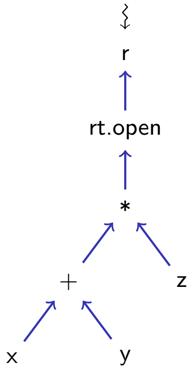

- All rounds equally fast.
- Optimal execution.




- Processing stalls.
- Wasted time!

# Asynchronous Design

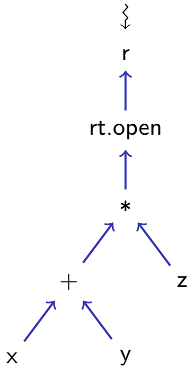
$r = rt.open((x + y) * z)$




- Entire tree is scheduled at once.
- Result is a form of “greedy scheduling”.
- Implicit synchronization, no rounds.

# Asynchronous Design

$r = \text{rt.open}((x + y) * z)$




- Entire tree is scheduled at once.
- Result is a form of "greedy scheduling".
- Implicit synchronization, no rounds.
- Advantages:
  - Automatic parallel scheduling.
  - Software scalability.

# Example: Hamming Distance

```
def xor(a, b):
    assert a.field is b.field
    if a.field is GF256:
        return a + b
    else:
        return a + b − 2 ∗ a ∗ b
```

- Straight-forward exclusive-or.
- Fast for $GF(2^8)$ elements.
- Slower for $\mathbb{Z}_p$ elements.
- (Already part of VIFF.)

# Example: Hamming Distance

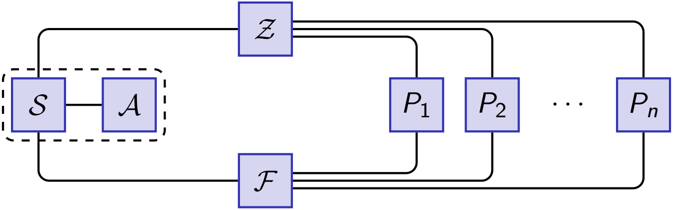
```
def xor(a, b):
    assert a.field is b.field
    if a.field is GF256:
        return a + b
    else:
        return a + b - 2 * a * b
```

- Straight-forward exclusive-or.
- Fast for $GF(2^8)$ elements.
- Slower for $\mathbb{Z}_p$ elements.
- (Already part of VIFF.)

```
def hamming(s, t):
    distance = 0
    for i in range(len(s)):
        distance += xor(s[i], t[i])
    return distance
```

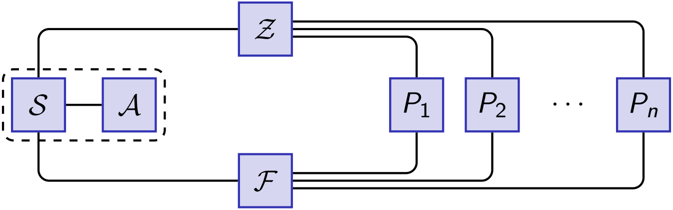
- Hamming distance.
- Exclusive-ors run in parallel!

# Asynchronous Ideal Functionality




- Reacts on input from $\mathcal{Z}$ via $P_i$.
- Inputs are tagged with a program counter.
- $\mathcal{F}$ forwards masked input to $\mathcal{S}$.
- $\mathcal{F}$ relays traffic between $\mathcal{S}$ and $P_i$.

# Asynchronous Ideal Functionality




- Reacts on input from $\mathcal{Z}$ via $P_i$.
- Inputs are tagged with a program counter.
- $\mathcal{F}$ forwards masked input to $\mathcal{S}$.
- $\mathcal{F}$ relays traffic between $\mathcal{S}$ and $P_i$.
- $\mathcal{F}$ queues replies.
- Released upon signal from $\mathcal{S}$.

# Operations

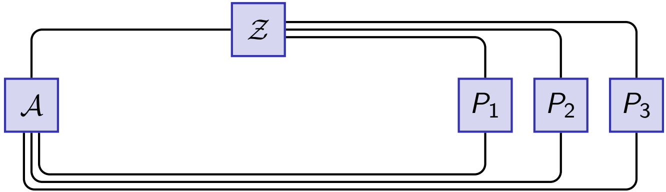
- Assignment: $\langle x := v, pc \rangle$.
- Output: $\langle \text{output}, x, P_i, pc \rangle$.
- Linear combination: $\langle x := c_1 \cdot x_1 + \cdots + c_j \cdot x_j, pc \rangle$.
- Multiplication: $\langle x := y \cdot z, pc \rangle$.
- Synchronization: $\langle \text{synchronize}, pc \rangle$.

# Operations

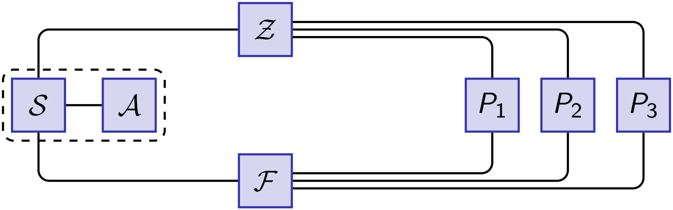
- Assignment: $\langle x := v, pc \rangle$.
- Output: $\langle \text{output}, x, P_i, pc \rangle$.
- Linear combination: $\langle x := c_1 \cdot x_1 + \cdots + c_j \cdot x_j, pc \rangle$.
- Multiplication: $\langle x := y \cdot z, pc \rangle$.
- Synchronization: $\langle \text{synchronize}, pc \rangle$.
- Direct correspondence to methods in VIFF Runtime.

# Simulating Assignment

Real World:

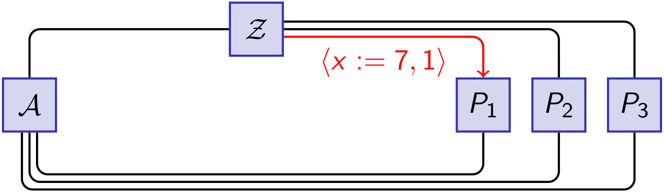



Ideal World:

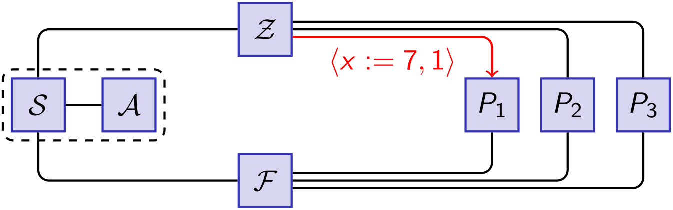


# Simulating Assignment

Real World:

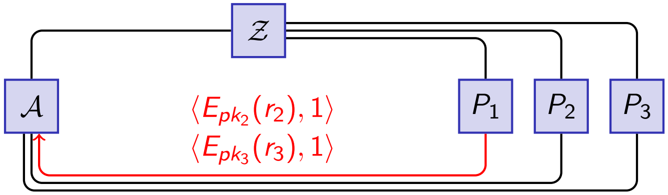



Ideal World:

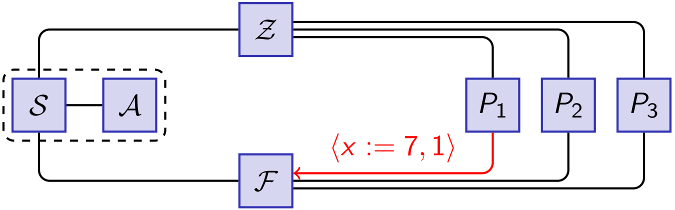


# Simulating Assignment

Real World:

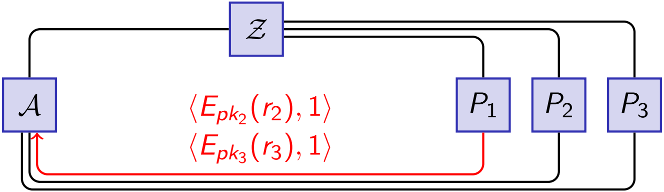



Ideal World:

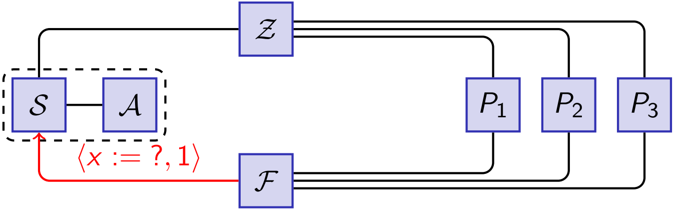


# Simulating Assignment

Real World:

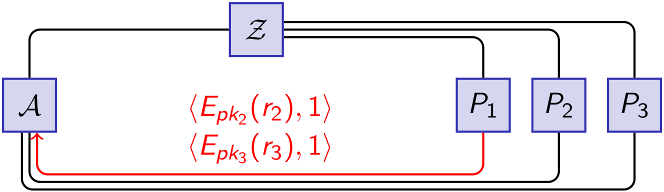



Ideal World:

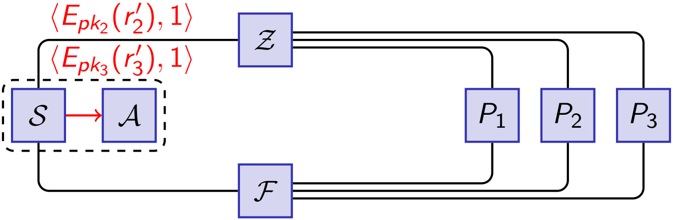


# Simulating Assignment

Real World:

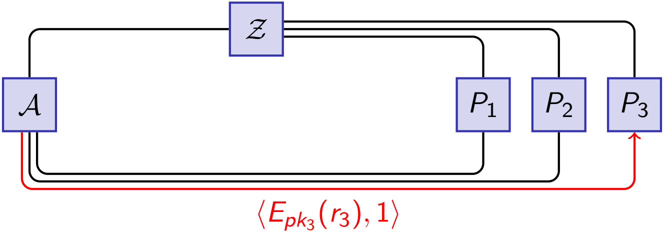



Ideal World:

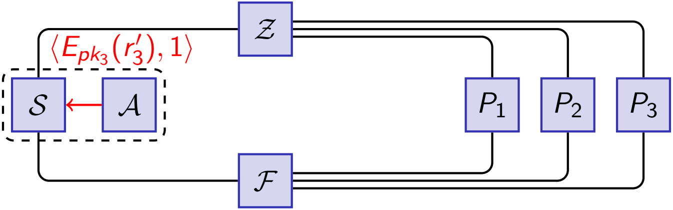


# Simulating Assignment

Real World:

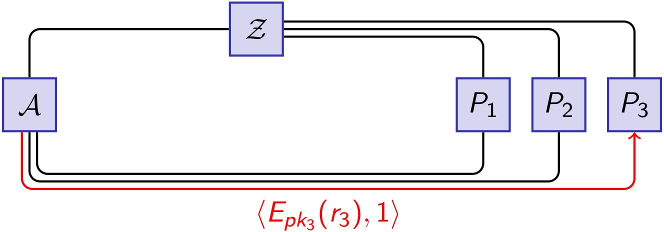



Ideal World:

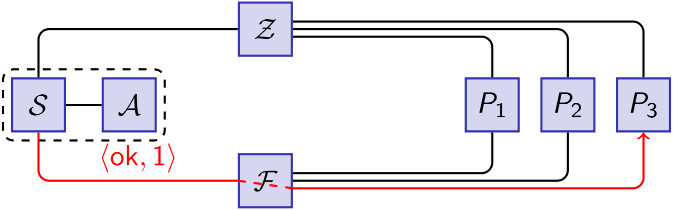


# Simulating Assignment

Real World:

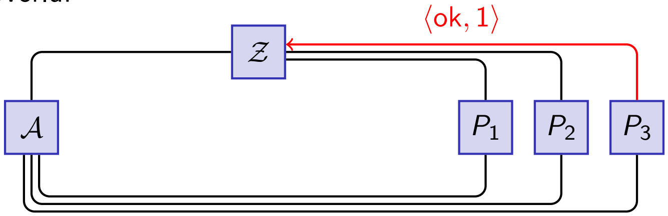



Ideal World:

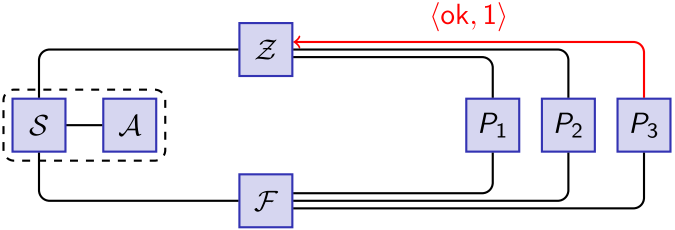


# Simulating Assignment

Real World:

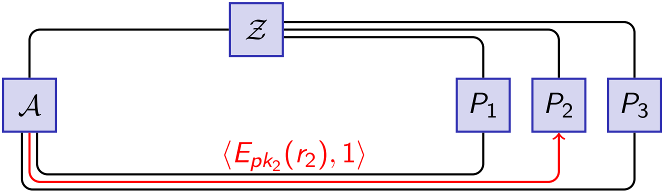



Ideal World:

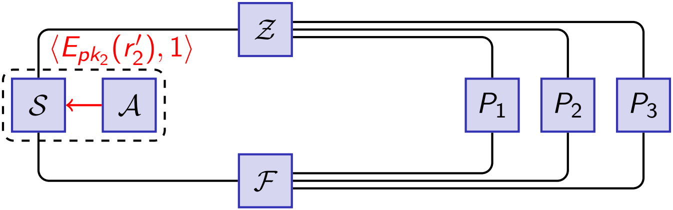


# Simulating Assignment

Real World:

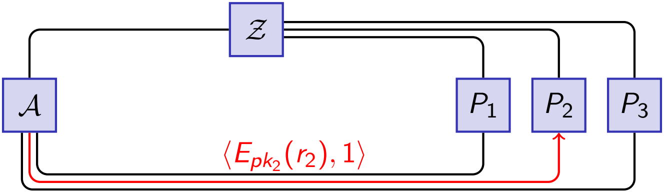



Ideal World:

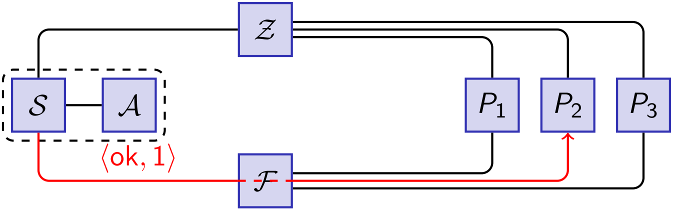


# Simulating Assignment

Real World:

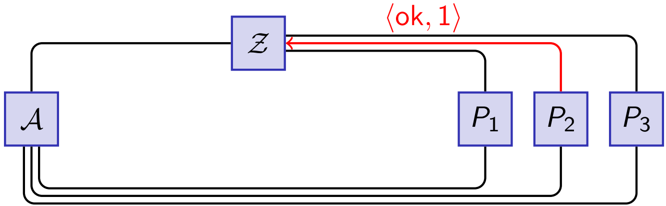



Ideal World:

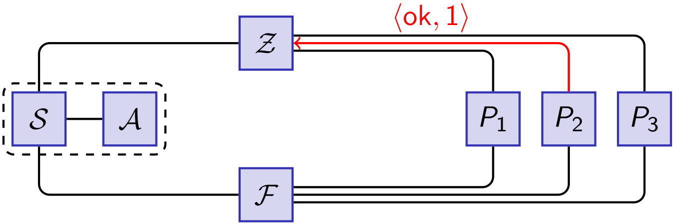


# Simulating Assignment

Real World:




Ideal World:



# Performance Results

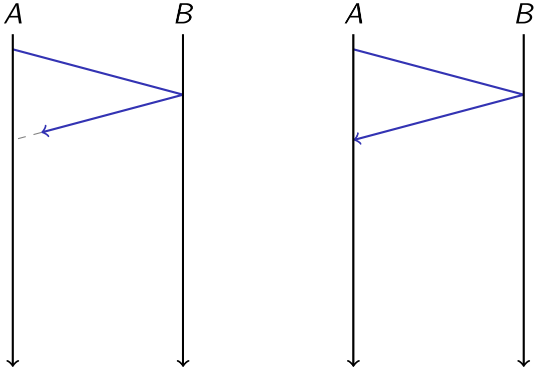
- Tested on 3 machines: USA, Norway, and Denmark.

# Performance Results

- Tested on 3 machines: USA, Norway, and Denmark.
- Tested multiplications and comparisons.

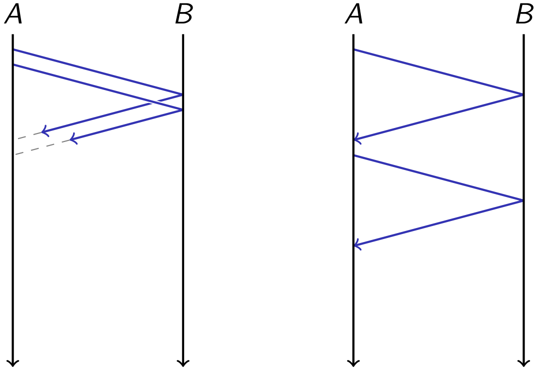
# Performance Results

- Tested on 3 machines: USA, Norway, and Denmark.
- Tested multiplications and comparisons.
- Tested parallel and serial multiplications:



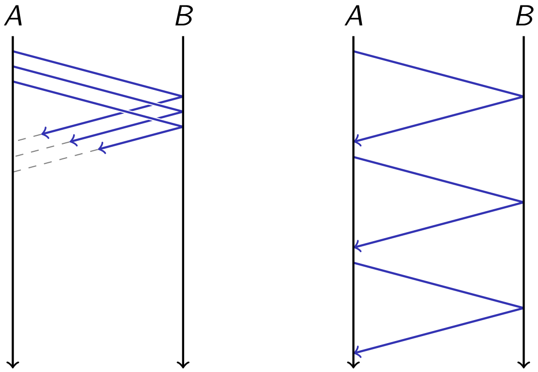
# Performance Results

- Tested on 3 machines: USA, Norway, and Denmark.
- Tested multiplications and comparisons.
- Tested parallel and serial multiplications:



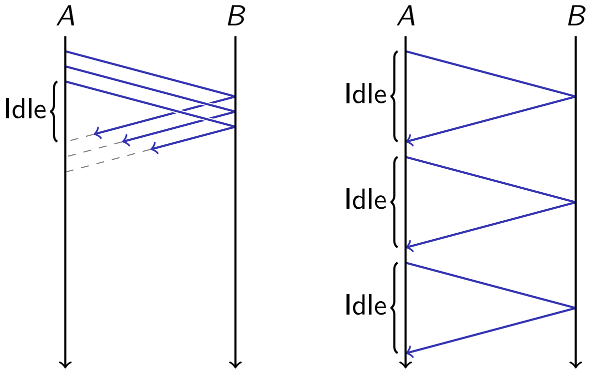
# Performance Results

- Tested on 3 machines: USA, Norway, and Denmark.
- Tested multiplications and comparisons.
- Tested parallel and serial multiplications:

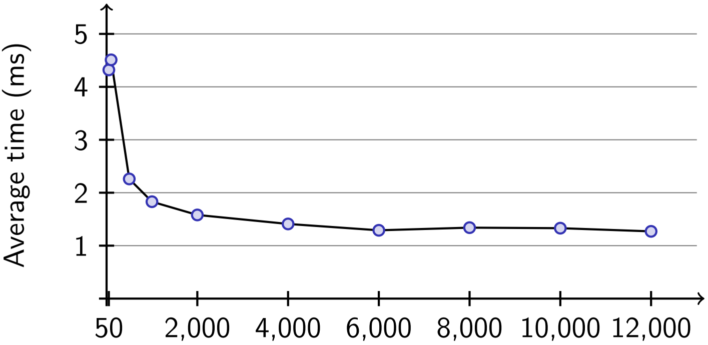


# Performance Results

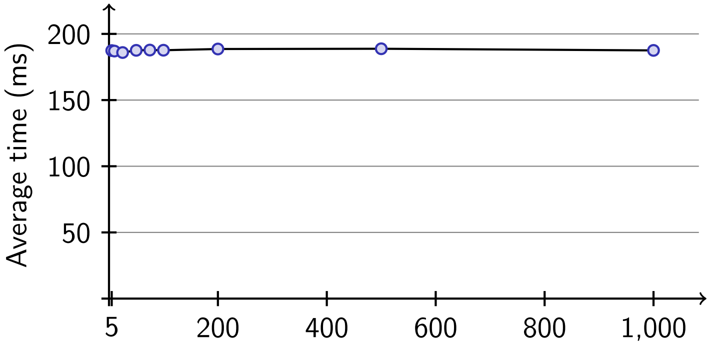
- Tested on 3 machines: USA, Norway, and Denmark.
- Tested multiplications and comparisons.
- Tested parallel and serial multiplications:



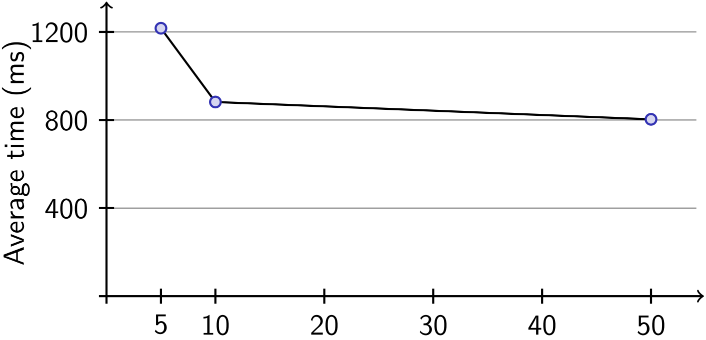
# Parallel Multiplications



# Serial Multiplications



# Parallel Comparisons



# Future Work

- Implement protocols for active security.
- Self-trust: protocols with $t = n - 1$.

# Conclusion

# Conclusion

- Comparison protocol for one public and one shared input.
  - A homomorphic encryption scheme.
  - Low communication complexity.
  - Low computational complexity.

# Conclusion

- Comparison protocol for one public and one shared input.
  - A homomorphic encryption scheme.
  - Low communication complexity.
  - Low computational complexity.
- Virtual Ideal Functionality Framework.
  - Light-weight design for doing MPC.
  - Asynchronous design gives automatic parallelism.
  - See: `http://viff.dk/`.

# Conclusion

- Comparison protocol for one public and one shared input.
  - A homomorphic encryption scheme.
  - Low communication complexity.
  - Low computational complexity.
- Virtual Ideal Functionality Framework.
  - Light-weight design for doing MPC.
  - Asynchronous design gives automatic parallelism.
  - See: `http://viff.dk/`.

**Thank you for listening!**