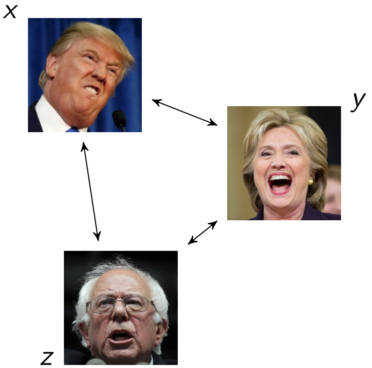
The Oblivous Machine
or: How to Put the C into MPC

Marcel Keller Peter Scholl

University of Bristol

9 June 2016

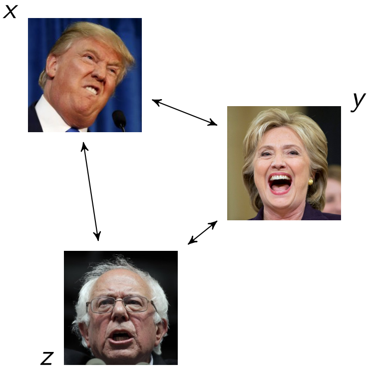
Secure Multiparty Computation



Wanted: $f(x, y, z)$

- Computation on secret inputs
- Replace trusted third party

Secure Multiparty Computation



Wanted: $f(x, y, z)$

- Computation on secret inputs
- Replace trusted third party
- How to formulate f?
  - Start with circuit
- Central questions in MPC
  - How many trusted parties?
  - What deviation?

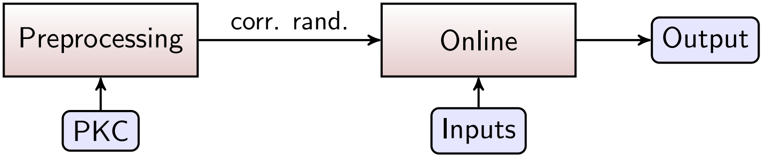
Multiparty Computation in This Talk

Security model

How many parties are how corrupted? In this work:

- Malicious adversary: Corrupted parties deviate from protocol.
- Dishonest majority of corrupted parties
  - Impossible without computational assumptions (PK crypto)
  - Shamir secret sharing does not help
  - No guaranteed termination

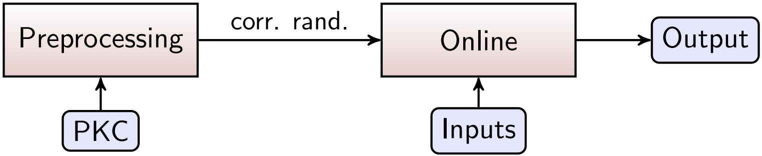
Malicious Offline-Online MPC Protocols



Advantages

- No secret inputs on the line when using crypto
  $\Rightarrow$ No one gets hurt if protocol aborts!
- Online computation might have many rounds,
  but preprocessing is constant-round.

Malicious Offline-Online MPC Protocols



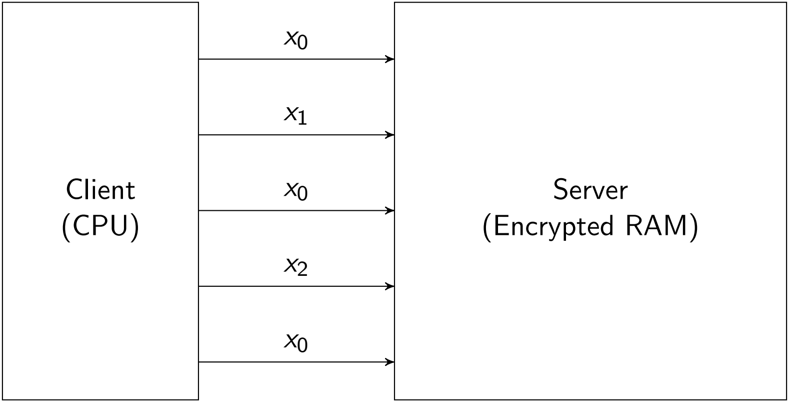
Suitable public-key crypto

- Somewhat homomorphic encryption (SPDZ)
- Oblivious transfer (TinyOT, MASCOT)

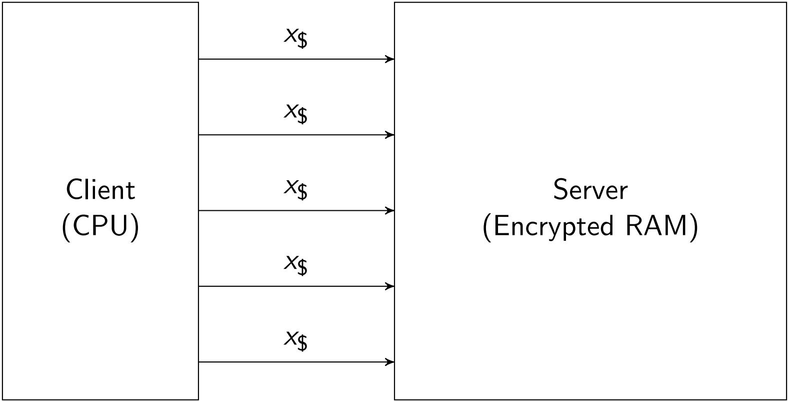
First Step — Oblivious Data Structures

- Generally
  - Secret pointers
  - Secret type of access if needed
- Oblivious array / dictionary
  - Secret index / key
  - Secret whether reading or writing
- Oblivious priority queue
  - Secret priority and value
  - Secret whether decreasing priority or inserting

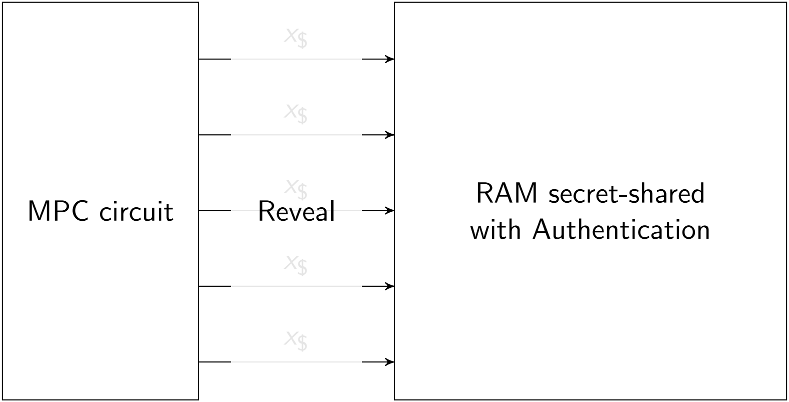
Oblivious RAM



Oblivious RAM



Oblivious RAM in MPC



Dijkstra's Algorithm in MPC

```
for each vertex do
    outer loop body
    for each neighbor do
        inner loop body
```

- Dijkstra's algorithms uses two nested loops
  - One for vertices, one for neighbors thereof
  - MPC would reveal the number of neighbors for every vertex
  - Replace by loop over all edges
  - Flag set when starting with a new vertex
- Oblivious data structures with public size
- Polylog overhead over classical algorithm

Dijkstra's Algorithm in MPC

for each edge **do**
    outer loop body
    (maybe dummy)
    inner loop body

- Dijkstra's algorithms uses two nested loops
  - One for vertices, one for neighbors thereof
  - MPC would reveal the number of neighbors for every vertex
  - Replace by loop over all edges
  - Flag set when starting with a new vertex
- Oblivious data structures with public size
- Polylog overhead over classical algorithm

Going General

Dijkstra (special case)

Obscure inner vs outer loop by doing both all the time

General case

Obscure by doing everything all the time

- Including memory accesses
- Data registers provide no value
- Memory-only machine with one register for program counter

Memory-only Machine

- Need 3 accesses for arithmetic operations like addition
- 3 is enough for any operation
- For every possible operation there is a circuit before, after, and in-between memory accesses
- Oblivous selection using instruction from program memory
- Last circuit outputs next program counter

Example

```c
int main() {
  unsigned int a[5];
  for (unsigned int i = 0; i < 5; i++)
    a[i] = i;
}
```

Example

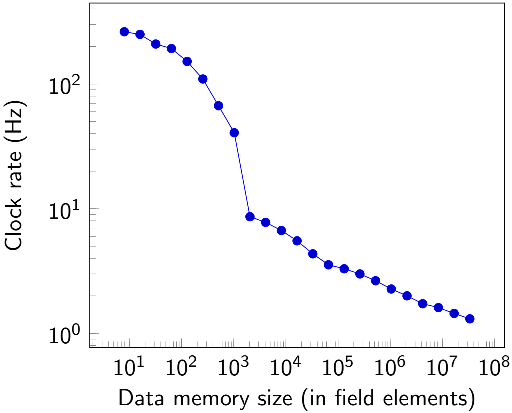
```
for.cond:
  %0 = load i64* %i, align 8
  %cmp = icmp ult i64 %0, 5
  br i1 %cmp, label %for.body, label %for.end

for.body:
  %1 = load i64* %i, align 8
  %2 = load i64* %i, align 8
  %arrayidx = getelementptr inbounds [5 x i64]* %a, i32 0, i64 %2
  store i64 %1, i64* %arrayidx, align 8
  br label %for.inc
```

Example

```
  # for.cond:
    ult_pos_const 9 5 8 # 2
    br 4 8 9 # 3
  # for.body:
    add_const 10 3 1 # 4
    store 0 8 10 # 5
  # for.inc:
    add_const 8 1 8 # 6
    jmp 2 0 0 # 7
  # for.end:
    mov 0 2 0 # 8
    jmp 10 0 0 # 9
```

Machine Speed



- 2 desktop machines
- 1 Gbps local network
- Path ORAM (CORAM too deep)

100-Party Oblivious Machine

Online

0.385 Hz

RAM: 1 million field elements (64 bit)

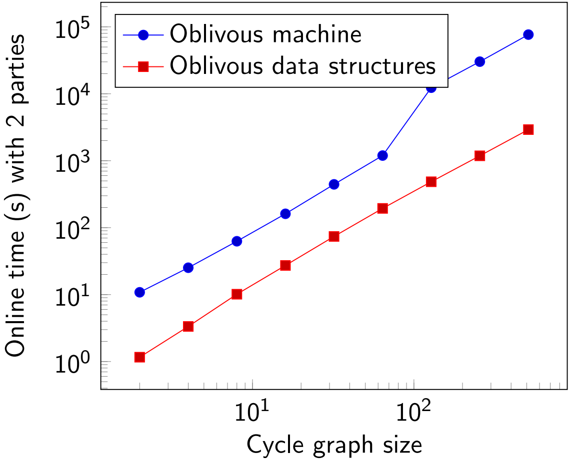
8.2¢ per clock cycle and party

c4.8xlarge

Offline

Per clock cycle	Time	Cost per party
c4.8xlarge	16 minutes	49¢
t2.small	7.7 hours	21¢

Overhead for Dijkstra's Algorithm



Comparison to Garbled Circuits for MIPS

Set intersection

Input size per party	64 inputs	256 inputs	1024 inputs
Wang et al. baseline	58.35 s	324.09 s	3068.19 s
Wang et al. optimized	2.77 s	12.96 s	108.45 s
This work (online)	6.43 s	44.12 s	1346.82 s

Bottom Line

Slow but as general as possible

- No static analysis
- Allows private function evaluation