iDASH - Secure Genome Analysis Competition Using ObliVM

Xiao Shaun Wang, Chang Liu, Kartik Nayak, Yan Huang and Elaine Shi

University of Maryland, College Park
Indiana University, Bloomington
Programming Framework for Secure Computation
ObliVM

Programming Framework for Secure Computation

Ease-of-use: easy for non-specialist programmers to use
Programming Framework for Secure Computation

**Ease-of-use:** easy for non-specialist programmers to use

**Efficiency:** compiles programs to small circuits
Real-life: **Programs**

- CPU
- `mem[x]`

Modern cryptography: **Circuits**

- AND
- XOR
- OR
- ...
Programming Framework for Secure Computation

Ease-of-use: easy for non-specialist programmers to use

Efficiency: compiles programs to *small* circuits

Formal Security: type system is being formalized

http://oblivm.com
Compute MAF

- Compute minor allele frequencies

Alice

AA AC AA

$f^A_{\text{Alice}} = 5$, $f^C_{\text{Alice}} = 1$

Bob

AA AC CC

$f^A_{\text{Bob}} = 3$, $f^C_{\text{Bob}} = 3$

Cleartext

Secure
Compute MAF

- Compute minor allele frequencies

**Alice**

<table>
<thead>
<tr>
<th>Allele</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>$f^{Alice}_{A} = 5$</td>
</tr>
<tr>
<td>AC</td>
<td>$f^{Alice}_{C} = 1$</td>
</tr>
<tr>
<td>AA</td>
<td></td>
</tr>
</tbody>
</table>

**Bob**

<table>
<thead>
<tr>
<th>Allele</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>$f^{Bob}_{A} = 3$</td>
</tr>
<tr>
<td>AC</td>
<td>$f^{Bob}_{C} = 3$</td>
</tr>
<tr>
<td>CC</td>
<td></td>
</tr>
</tbody>
</table>

Compute $min(f^{Alice}_{A} + f^{Bob}_{A}, f^{Alice}_{C} + f^{Bob}_{C})$
**Compute MAF**

- Compute minor allele frequencies

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA AC AA</td>
<td>AA AC CC</td>
</tr>
<tr>
<td>(f^A_{Alice} = 5), (f^C_{Alice} = 1)</td>
<td>(f^A_{Bob} = 3), (f^C_{Bob} = 3)</td>
</tr>
</tbody>
</table>

Compute \(\min(f^A_{Alice} + f^A_{Bob}, f^C_{Alice} + f^C_{Bob})\)

Secure Computation: \(MAF = \min(f^A_{Alice} + f^A_{Bob}, f^C_{Alice} + f^C_{Bob})\)
Code in ObliVM-lang: Compute MAF

```c
struct Task1aAutomated @m@n{}
void Task1aAutomated @m@n.funct(int @m[public n] alice_data,
                                    int @m[public n] bob_data,
                                    int @m[public n] ret,
                                    public int @m total_instances) {
    int @m total = total_instances;
    int @m half = total_instances / 2;
    for (public int32 i = 0; i < n; i = i + 1) {
        ret[i] = alice_data[i] + bob_data[i];
        if (ret[i] > half)
            ret[i] = total - ret[i];
    }
}
```
**Problem Statement: Compute $\chi^2$ statistic**

- **Task 1b: Computing $\chi^2$ statistic**

  Alice
  - Case: AA AC AA
  - Control: AA CA CA

  Bob
  - Case: AA AC CC
  - Control: CA AC CC

Cleartext
Secure
Problem Statement: Compute $\chi^2$ statistic

- Task 1b: Computing $\chi^2$ statistic

Alice
Case: AA AC AA
Control: AA CA CA

Bob
Case: AA AC CC
Control: CA AC CC

$a, b$: allele counts for case group
$c, d$: allele counts for control group
(similar to Task 1A)
**Problem Statement:** Compute $\chi^2$ statistic

- **Task 1b:** Computing $\chi^2$ statistic

**Alice**
- Case: AA AC AA
- Control: AA CA CA

**Bob**
- Case: AA AC CC
- Control: CA AC CC

$a, b$: allele counts for case group

$c, d$: allele counts for control group

(similar to Task 1A)

$$\chi^2 = n \times \frac{(ad-bc)^2}{rsgk}$$

where $r = a + b$, $s = c + d$, $g = a + c$,

$k = b + d$, $n = r + s$
**Results: Compute $\chi^2$ statistic**

- Floating point computation
- Absolute accuracy
  - $1.11 \times 10^{-4}$ with 7763 gates
  - $5.6 \times 10^{-8}$ with 14443 gates
Code in ObliVM-lang: Compute $\chi^2$ statistic

```
struct Task1bAutomated@n{}
float32[public n] Task1bAutomated@n.func(
    float32[public n][public 2] alice_case, float32[public n][public 2] alice_control,
    float32[public n][public 2] bob_case, float32[public n][public 2] bob_control)
{
    float32[public n] ret;
    for (public int32 i = 0; i < n; i = i + 1) {
        float32 a = alice_case[i][0] + bob_case[i][0];
        float32 b = alice_case[i][1] + bob_case[i][1];
        float32 c = alice_control[i][0] + bob_control[i][0];
        float32 d = alice_control[i][1] + bob_control[i][1];
        float32 g = a + c, k = b + d;
        float32 tmp = a*d - b*c;
        tmp = tmp*tmp;
        ret[i] = tmp / (g * k);
    }
    return ret;
}
```
Building Block: Secure Set Union

Alice
\[ S^A \]
\{ a, b, c \}

Bob
\[ S^B \]
\{ b, d, e \}

Cardinality of the union of the sets i.e.
\[ |S^A \cup S^B| = 5 \]
**Building Block: Secure Set Union**

Alice

\[ S^A \]
\{ a, b, c \}

Bob

\[ S^B \]
\{ b, d, e \}

Cardinality of the union of the sets i.e.

\[ |S^A \cup S^B| = 5 \]
Building Block: Secure Set Union

Alice
\( S^A \)
\{a, b, c\}

Bob
\( S^B \)
\{b, d, e\}

Cardinality of the union of the sets i.e. \( |S^A \cup S^B| \)
\( |S^A \cup S^B| = 5 \)

Strawman solution:

\[
\text{union}(S^A, S^B)
\]

1: Sort the combined array \( S^A || S^B \) obliviously

\( O(N \log^2 N) \)
**Building Block: Secure Set Union**

Alice

\[ S^A \]
\{a, b, c\}

Bob

\[ S^B \]
\{b, d, e\}

Cardinality of the union of the sets i.e.
\[ |S^A \cup S^B| \]
\[ |S^A \cup S^B| = 5 \]

Strawman solution:

1. Sort the combined array \( S^A \| S^B \) obliviously
2. Compute cardinality in a single pass

\[ O(N \log^2 N) \]
Set Union: Oblivious Merge

\[
\text{union}(S^A, S^B)
\]

1: Local sort of \(S^A\) and \(S^B\)

Cleartext

Secure
Set Union: Oblivious Merge

union($S^A, S^B$)

1: Local sort of $S^A$ and $S^B$
2: Oblivious merge of sorted lists
**SET UNION: OBLIVIOUS MERGE**

\[ \text{union}(S^A, S^B) \]

1: Local sort of \( S^A \) and \( S^B \)
2: Oblivious merge of sorted lists
3: Compute cardinality in a single pass

\[ O(N \log N) \]
void Task2Automated\texttt{@m}@n.\texttt{obliviousMerge}(int\texttt{@m}[\texttt{public} n] key, 
\texttt{public} int32 lo, 
\texttt{public} int32 l) {
\begin{verbatim}
if (l > 1) {
    public int32 k = 1;
    while (k < l) k = k << 1;
    k = k >> 1;
    for (public int32 i = lo; i < lo + l - k; i = i + 1)
        this.compare(key, i, i + k);
    this.obliviousMerge(key, lo, k);
    this.obliviousMerge(key, lo + k, l - k);
}
\end{verbatim}
}
Set Union: Bloom Filter

- Common case: Check for existence of elements
- Our case: Approximate the cardinality of a set $S$

Elements Inserted: $S=\{x, y, z\}$

Lookup $w$
**Set Union: Bloom Filter**

- Common case: Check for existence of elements
- Our case: Approximate the cardinality of a set $S$

\[
|S|_{MLE} = \frac{\ln(1 - \frac{X}{m})}{k \ln(1 - 1/m)}
\]

where
- $X$: number of bits set,
- $m$: number of bits in the bloom filter,
- $k$: number of hash functions,
- $|S|_{MLE}$: maximum likelihood estimate of $|S|$
Set Union: Bloom Filter

\[
\text{union}(S^A, S^B) \\
1: \text{Compute bloom filters locally}
\]
Set Union: Bloom Filter

union($S^A, S^B$)

1. Compute bloom filters locally
2. In secure computation, compute bitwise OR and count number of 1's

Cleartext
Secure
**Set Union: Bloom Filter**

\[
\text{union}(S^A, S^B)
\]

1. **Compute bloom filters locally**
2. **In secure computation, compute bitwise OR and count number of 1’s**
3. **Compute estimated \(|S|\) in cleartext**

**Cleartext**

**Secure**
**Set Union: Bloom Filter**

\[
\text{union}(S^A, S^B)
\]

1. **Compute** bloom filters locally
2. In secure computation, compute bitwise OR and count number of 1’s
3. **Compute** estimated \(|S|\) in cleartext

\[O(m)\] operations, \(m\): number of bits used for bloom filter

\[m = O(N)\], number of elements inserted in the bloom filter
```c
25  int@log(n + 1) BF_circuit.countOnes@n(int@n x) {
26      if (n==1) return x;
27  int@log(n - n/2 + 1) first = this.countOnes@(n/2)(x$0^n/2$);
28  int@log(n - n/2 + 1) second = this.countOnes@(n - n/2)(x$n/2^n$);
29  Pair<bit, Int@log(n - n/2)> ret = this.add@log(n - n/2 + 1)(first, second);
30  int@log(n + 1) r = ret.right.v;
31  r$log(n+1)-1$ = ret.left.v;
32      return r;
33  }
```
**Problem Statement: Hamming Distance**

Alice and Bob maintain records of type \((\text{ref}, \text{svtype}, \text{alt})\) that differ from the reference

```java
int d = 0;
for each record of type SNP or SUB
    if ((x == null) || (y == null) || (x.ref == y.ref && x.alt != y.alt))
        d += 1;
end for
```

Solution: Hamming Distance

Alice

\[ S^A = \{(1, T, SNP), (75, G, SNP)\} \]

Bob

\[ S^B = \{(1, T, SNP), (18, A, SNP)\} \]

We need all positions that have been modified, but not modified to the same value.

\[
\text{Hamming Distance} = |S^A \cup S^B| - |S^A \cap S^B| = |
\{(75, G, SNP), (18, A, SNP)\}|
\]
Problem Statement: Edit Distance

Alice and Bob maintain records of type \((\text{ref}, \text{svtype}, \text{alt})\) that differ from the reference

- **Replacement:** Calculate like hamming distance
- **Insertion/Deletion:**
  - If one party modifies a position, add \(\text{len(alt)}\) to edit distance
  - If both parties modify a position, add \(\text{len(max(alt1, alt2))}\) to edit distance
Solution: Edit Distance

Alice
{(1, T, SNP),
(10, TCG, INS),
(75, G, SNP)}

Bob
{(1, T, SNP),
(10, CA, INS),
(18, A, SNP)}
Solution: Edit Distance

Alice
{(1, T, SNP),
(10, TCG, INS),
(75, G, SNP)}

Bob
{(1, T, SNP),
(10, CA, INS),
(18, A, SNP)}

\[ S_1^A = \{(1, 1), (10, 1), (10, 2), (10, 3), (75, 1)\} \]
\[ S_2^A = \{(1, T, 1), (10, T, 1), (10, C, 2), (10, G, 3), (75, G, 1)\} \]
**Solution: Edit Distance**

Alice
\{(1, T, SNP),
(10, TCG, INS),
(75, G, SNP)\}

Bob
\{(1, T, SNP),
(10, CA, INS),
(18, A, SNP)\}

\[S_1^A = \{(1, 1), (10, 1), (10, 2), (10, 3), (75, 1)\}\]

\[S_2^A = \{(1, T, 1), (10, T, 1), (10, C, 2), (10, G, 3), (75, G, 1)\}\]

\[d_1 = |S_1^A \cup S_1^B| = |\{(1, 1), (10, 1), (10, 2), (10, 3), (75, 1), (18, 1)\}|\]
Solution: Edit Distance

Alice
{(1, T, SNP),
(10, TCG, INS),
(75, G, SNP)}

Bob
{(1, T, SNP),
(10, CA, INS),
(18, A, SNP)}

\[ S_A^1 = \{(1, 1), (10, 1), (10, 2), (10, 3), (75, 1)\} \]
\[ S_A^2 = \{(1, T, 1), (10, T, 1), (10, C, 2), (10, G, 3), (75, G, 1)\} \]

\[ d1 = |S_A^1 \cup S_B^1| = |\{(1, 1), (10, 1), (10, 2), (10, 3), (75, 1), (18, 1)\}| \]

\[ d2 = |S_A^2 \cap S_B^2| = |\{(1, T, 1)\}| \]

Compute \[ d1 - d2 \]
Thank You!

http://oblivm.com/

kartik@cs.umd.edu