Privacy-Preserving Frequent Itemset Mining With the SecreC Language

Roman Jagomägis

Theory days, Elva 2010

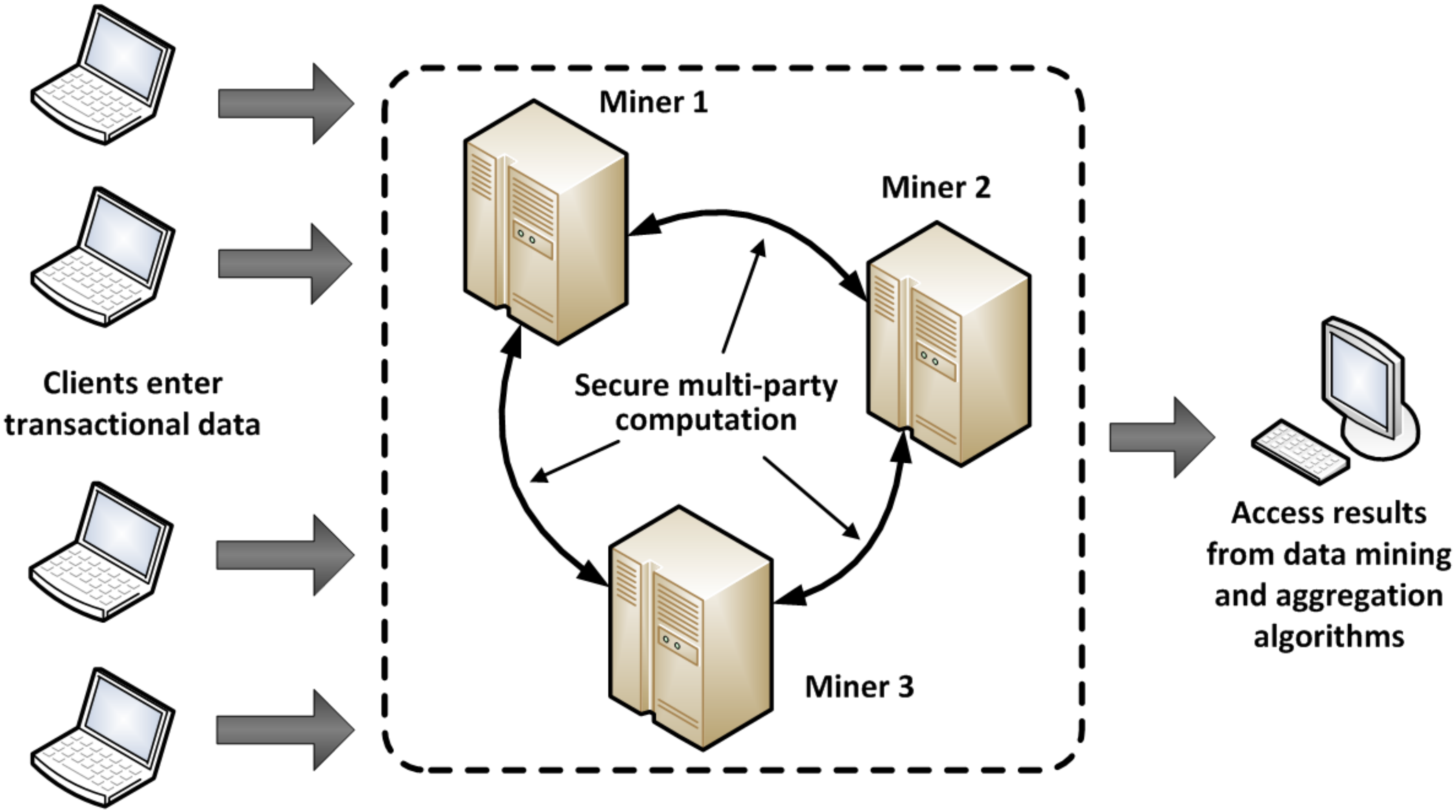
PROBLEM STATEMENT

- It is possible to gain significant added value by combining and analyzing confidential information
- Serious security issues arise
- Cryptography researchers have proposed several technical solutions to deal with the problem
- We want to implement Frequent Itemset Mining algorithms with provable security guarantees

THE SECURE COMPUTATION PLATFORM

- We will implement our solution on the SHAREMIND secure multi-party computation platform
- It can sequentially and in parallel execute operations on private and public data
- Consists of 3 parties (miners) that process the data
- Uses the additive secret scheme in the ring $\mathbb{Z}_{2^{32}}$
  - $s_1 + s_2 + \cdots + s_n = s \bmod 2^{32}$
- Proven to be secure in the honest-but-curious security model

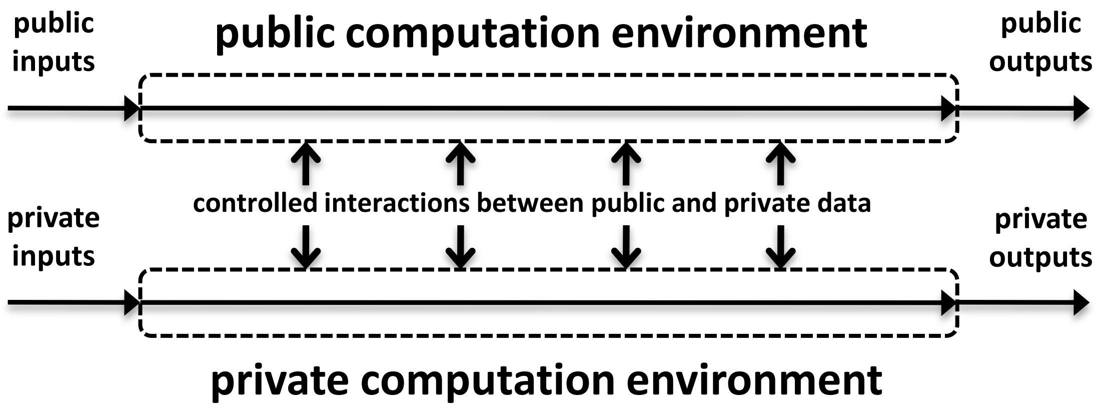
The Sharemind deployment model



THE SECREC LANGUAGE

- Syntactically based on C, but
  - Omits several features (e.g. pointers)
  - Adds some new ones (e.g. vectorized operations)

- Separation of public and private data



- Explicit declassification

Writing privacy-preserving algorithms

- Declassify the private data as little as possible
- The control flow is public and must not be affected by private data
- Oblivious selection still allows to hide the selected branch from the observer by evaluating both branches
  - if (a) x = y; else x = z;   vs   x = a*y + (1-a)*z
- Use aggregation techniques to maximize the entropy of the output results (e.g. sum)
- Take a reasonable amount of data
  - Better contribution to the uncertainty of the final result
  - Better statistical results
- Parallelize operations smartly for better execution-times

What is Frequent Itemset Mining?

items / products

	Tea	Beer	Honey	Diapers
C	1	1	1	0
B	0	1	0	1
C	1	0	1	0
A	1	1	0	0
B	0	1	1	1

customers

transactions

- What is the behavior of the customers in terms of purchased products?
- What kind of products are frequently bought together?

What is Frequent Itemset Mining?

	Tea	Beer	Honey	Diapers
C	1	1	1	0
B	0	1	0	1
C	1	0	1	0
A	1	1	0	0
B	0	1	1	1

Let $\mathcal{A} = (a_1, \dots, a_m)$ be a list of all attributes.

The transaction $\mathcal{T}$ is then a subset of $\mathcal{A}$.

Thus, $\mathcal{D}^{n \times m} = \begin{matrix} \mathcal{T}_1 \\ \vdots \\ \mathcal{T}_n \end{matrix}$, so that $\mathcal{D}[i,j] = 1$ iff $a_j \in \mathcal{T}_i$.

$\text{support}(\mathcal{X})$ – number of transactions that contain all items of $\mathcal{X}$.

Frequent itemsets: $\text{support}(\mathcal{X}) \geq t$

$\text{cover}(\mathcal{X})$ – the set of transaction identifiers that contain the itemset $\mathcal{X}$.

Frequent itemset mining & privacy

	Tea	Beer	Honey	Diapers
C	1	1	1	0
B	0	1	0	1
C	1	0	1	0
A	1	1	0	0
B	0	1	1	1

- Transactions are associated with the customers
  - One can find out and exploit habits of individuals
- Stripping the associations does not protect the privacy enough
  - Having extra knowledge when analysing the transactions makes it possible to distinguish who is who
- We are thus motivated to use secure multi-party computation systems

FREQUENT ITEMSET MINING IN MPC

	Tea	Beer	Honey	Diapers
C	1	1	1	0
B	0	1	0	1
C	1	0	1	0
A	1	1	0	0
B	0	1	1	1

In privacy-preserving computations covers can be represented as index vectors $\boldsymbol{x}$ such that:

$x_i = 1$ if $\mathcal{X} \in \mathcal{T}_i$ and otherwise $x_i = 0$.

Then, given $a \in \mathcal{A}$

$$\mathrm{cover}(\{a\}) = \mathcal{D}[*, a]$$

$$\mathrm{cover}(\mathcal{X} \cup \mathcal{Y}) = \mathrm{cover}(\mathcal{X}) \odot \mathrm{cover}(\mathcal{Y})$$

$$\mathrm{supp}(\mathcal{X}) = |\mathrm{cover}(\mathcal{X})| = |\boldsymbol{x}| = x_1 + \cdots + x_n$$

T		H		TH
1		1		1
0		0		0
1	$\odot$	1	=	1
1		0		0
0		1		0

FREQUENT ITEMSET MINING STRATEGIES

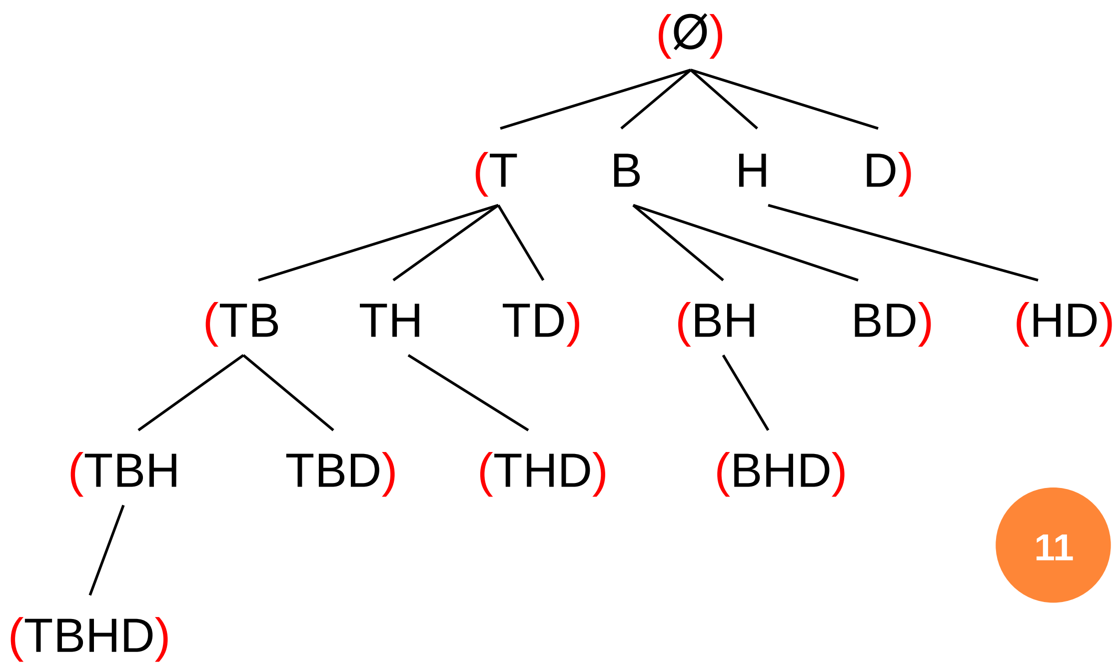
	Tea	Beer	Honey	Diapers
C	1	1	1	0
B	0	1	0	1
C	1	0	1	0
A	1	1	0	0
B	0	1	1	1

- Note, that support is an anti-monotone function

$$\mathcal{X} \subseteq \mathcal{Y} \Longrightarrow \mathrm{supp}(\mathcal{X}) \geq \mathrm{supp}(\mathcal{Y})$$

- Subsets of frequent itemsets must also be frequent

- Tree traversal problem
- Apriori – breadth-first
- Eclat – depth-first



EXECUTING APRIORI


```
void main () {
        public int[0][0] itemsets;
        dbLoad ("dataTransactions");
        itemsets = apriori (5000, 5, "mushroom");
        matPrint (itemsets);
}

public int[][] apriori( public int threshold,
                        public int setSize,
                        public string table ) {
...
}
```

DETERMINING FREQUENT COLUMNS IN DB

```
for (i = 0; i < dbColumns; i = i + 1) {
        colName = "" + (i + 1);
        z = dbGetColumn(colName, table);
        frequency = vecSum(z);
        isGood = (frequency >= threshold);
        result = declassify(isGood);

        if (result) {
                *** cache the column data for reuse ***
        }
}
```

GENERATING CANDIDATES


```
for (i = 0; i < F_size; i = i + 1) {
    for (j = i + 1; j < F_size; j = j + 1) {
        prefixEqual = true;

        for (n = 0; n < k - 1; n = n + 1) {               // check if the prefix of
            if (F[i][n] != F[j][n]) prefixEqual = false;   // two potential candidates
        }                                                  // are equal or not

        // are the two itemsets suitable for constructing a new candidate?
        if (prefixEqual && F[i][k-1] < F[j][k-1]) {
            *** verify the new candidate ***
            result = declassify(isGood);

            if (result) {
                matAppendRow(F_newcache, C_dot);

                matResize(C, k, 1);
                C = F[i][*];
                matAddRow(C);
                C[k] = F[j][k-1];
                matAppendRow(F_new, C);
            }
        }
    }
}
```

C = F[i][*] ∪ F[j][k-1];

VERIFYING CANDIDATES

```
if (prefixEqual && F[i][k-1] < F[j][k-1]) {
        C_dot = F_cache[i][*];
        z = F_cache[j][*];
        C_dot = C_dot * z;

        frequency = vecSum(C_dot);
        isGood = (frequency >= threshold);
        result = declassify(isGood);

        if (result) {
                matAppendRow(F_newcache, C_dot);

                *** C = F[i][*] ∪ F[j][k-1]; ***
                matAppendRow(F_new, C);
        }
}
```

```
C_dot = F_cache[i][*] * F_cache[j][*];
```

11 June 2010

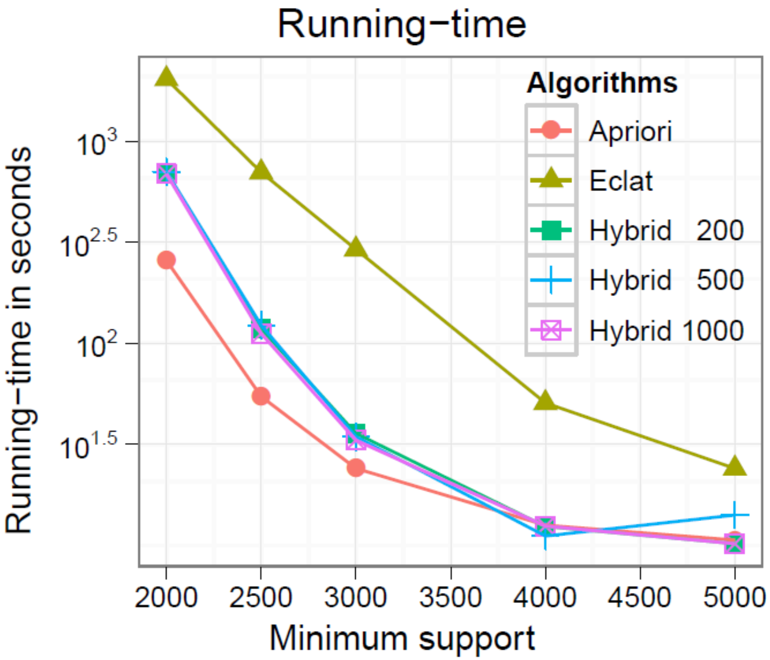
SECURITY

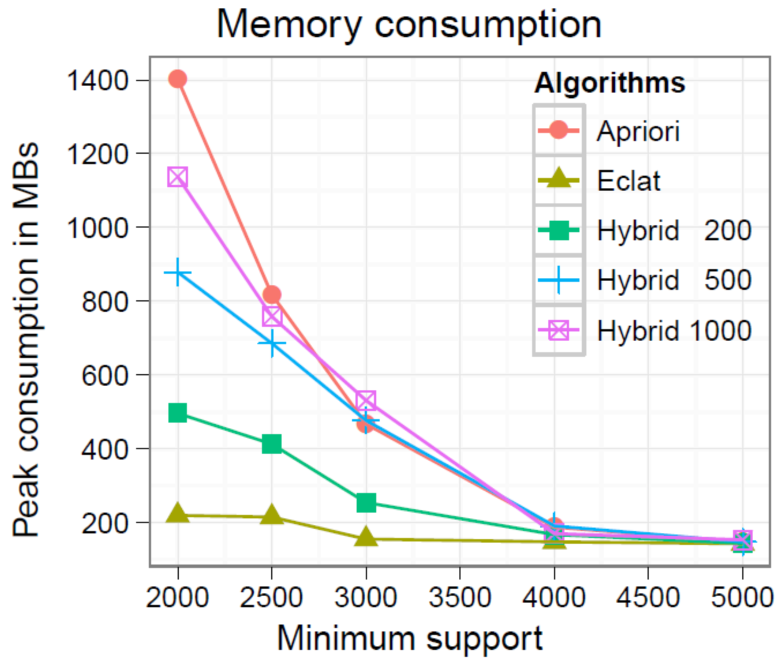
- As long as sensitive data stays in the private computation environment of SHAREMIND, it is fine.
- The only places in the code which declassify secret data, do not leak more information than needed
- Individual rows are not distinguished
- We only open answers to the question: is the itemset frequent or not?
- The final answer reveals the information about the intermediate results, so there is no sense in hiding them.

Performance

- Tested on the Mushroom dataset with 119 items, 8124 transactions and data density of 19.3%.

- High Performance Computing Center @ UT
  - Machines with 2.5GHz quad-core Intel Xeon CPUs, 32GB RAM and very fast network





Thank You!

Questions?