Towards Practical Secure Computation
Handling sensitive data on untrusted machines

Workshop on Trustworthy Clouds, Egham

Stefan Katzenbeisser
Security Engineering Group
Technische Universität Darmstadt & CASED
skatzenbeisser@acm.org

Joint work with:
Martin Franz (Deutsche Bank), Andreas Holzer (TU Wien), Helmut Veith (TU Wien)
Handling private data: Does it work?

Restaurant chain customers' credit card data stolen

By Bruce Mohl, Globe Staff | October 1, 2008

Not Your Average Joe's, a Massachusetts-based restaurant chain, yesterday that thieves have stolen credit card information from its customers.

The Dartmouth-based chain estimated less than 3,500 of the 350,000 customers it served in August and September had their credit card information stolen. The 14-restaurant chain said it is working with the US Secret Service and major credit card companies to determine how the data theft occurred and precisely how many customers were affected.

Today, the chain plans to post on its website a notice to customers about the security breach.

Diana Pisciotta, a spokeswoman for Not Your Average Joe's, said the chain decided to check its credit card statements with the US Secret Service and major credit card companies about any suspicious charges.

"We're doing this out of an abundance of caution and forthright with our customers," she said.

Stolen computer contained info from 88,000 patients at Staten Island hospital

by Staten Island Advance
Wednesday April 30, 2008, 4:37 PM

Computer equipment stolen from an administrative office in Clifton in December contained personal information from 88,000 patients that have been treated at Staten Island University Hospital.

After four months with no arrests, hospital administrators are just now beginning the process of sending out letters to patients whose names, Social Security and health insurance numbers were contained in computer files on a desktop computer and a backup hard drive stolen Dec. 29 from the hospital’s finance office at 1 Edgewater Plaza.

"The hospital is in the process of issuing a hospital statement, released by spokesmen, said.

Personal data of 600,000 on lost laptop

A junior Royal Navy officer is facing a court martial after a laptop containing the personal data of 600,000 people, including serving personnel and thousands of people who have shown an interest in a military career, was stolen from his car.

The loss of the laptop was considered to be so serious that Des Browne, the Defence Secretary, will make a statement to the Commons early next week.
Computing with critical data: The traditional approach

Data secrecy relies on several technical and administrative approaches:

- Legal requirements
- Policies
- Audits
- Training
- Technical means (access control, network security, intrusion detection)
- Physical security
Cryptography (1)

Encryption

Decryption

In tabula literarum canonica sunt rectae latinarum literarum in parum per mutationem alphabeta, quot in eaper toorum sunt monogontes, et ubiles quattuor uigitur, quae facientur, ut est multiplica, paulo efficium minus qua est.
Doesn’t cryptography solve the problem?

- Encrypted data „look random“
- ... but usually need to be decrypted before use.
- How can confidentiality be guaranteed during use of data?
- How can this be done on potentially compromised hosts?
- Protection of outsourced data?
Secure Computation

- Strike a balance between data availability and privacy

- **Paradigm**: keep data encrypted, **compute with encrypted data**

- Follows the principle of **Privacy By Design**: Cryptographic protocols precisely limit amount of information available

- Cryptographic tools are available!
  - Homomorphic encryption
  - Yao’s Garbled circuits
  - Customized protocols (private set intersection, ...)

Example:
Private Face Detection

yes / no
Bob
Example: Private Processing of Genome Data

Physician

Genom O

[Prob[O | λ]]

Bioinformatics Institute

HMM

\[
\begin{align*}
\alpha_1(i) &= \pi_i b_i(o_1) \\
\alpha_{t+1}(i) &= b_i(o_{t+1}) \sum_{j=1}^{N} a_{i,j} \alpha_t(j) \\
\text{Prob}[O | \lambda] &= \sum_{i=1}^{N} \alpha_T(i)
\end{align*}
\]
Cryptographic Tools: 
Secure Multi-Party Computation (1)

- $N$ parties have secret inputs $x_1, x_2, \ldots, x_N$

- **Goal**: jointly compute $f(x_1, x_2, \ldots, x_N)$ so that the private inputs of the parties are not revealed (in addition to what leaked through the function and the shared result)

- **Special case**: **Secure Two-Party Computation** for $N = 2$

- Yao (1982): Every computable function can be computed securely in the two-party case
Cryptographic Tools: Secure Multi-Party Computation (2)

- Security notions for secure multiparty computations are non-trivial

- At first sight, security may depend on the function

- Some functions may *inevitably* leak information, since result of computation is known to all parties

  \[ f(x, y) = y \]

- SMP hides only those parts of the input *that cannot be derived* from the shared function value

- Different solutions:
  - Yao’s Garbled Circuits
  - Homomorphic encryption
  - Secret sharing
Millionaire’s problem

- Two millionaires want to compare their wealth

- Both are “rich” ...
  ... both want to know who is “richer” ...
  ... but do not want to disclose wealth to each other

- Simple instance of function to be computed in a secret way:

  \[ f(x, y) := \begin{cases} 
  1 & x \geq y \\
  0 & \text{otherwise} 
  \end{cases} \]

- Communist Millionaire’s Problem: both parties want to check whether they are equally rich:

  \[ f(x, y) := \begin{cases} 
  1 & x = y \\
  0 & \text{otherwise} 
  \end{cases} \]
Attacker Models:

- **Semi-Honest Attacker**
  - “Honest But Curious”
  - Conforms to the protocol specification

- **Malicious Attacker**
  - May deviate from the protocol by computing values dishonestly
  - Much more difficult to cope with
  - May require Zero-Knowledge Proofs to show validity of computation
IBM Discovers Encryption Scheme That Could Improve Cloud Security, Spam Filtering

A researcher at IBM reports having developed a fully homomorphic encryption scheme that allows data to be manipulated without being exposed. Researcher Craig Gentry’s discovery could prove to be important in securing cloud computing environments and fighting encrypted spam.

An IBM researcher has uncovered a way to analyze data while it is still encrypted, in what could be a boon for both spam-filtering applications and cloud computing environments.

The challenge of manipulating data without exposing it has bugged cryptographers for decades. But in a breakthrough, IBM researcher and Stanford University Ph.D. candidate Craig Gentry has developed a “fully homomorphic encryption” scheme that keeps data protected.
Secure Two-Party Computation
Yao’s Garbled Circuits (1)

“Garbled” circuit construction:
1. Represent function to be computed as Boolean circuit
Secure Two-Party Computation
Yao’s Garbled Circuits (2)

“Garbled” circuit construction:
1. Represent function to be computed as Boolean circuit
2. Choose 2 keys for each “wire” in the circuit
3. For each gate in the circuit, construct a “garbled” version (encrypted & permuted)
4. Output wires from output gates are not garbled

\[
\begin{array}{ccc}
\wedge & x = 1 & x = 0 \\
y = 1 & 1 & 0 \\
y = 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
 & x = 1 & x = 0 \\
y = 1 & E(k_1^1, E(k_2^1, k_3^1)) & E(k_1^1, E(k_2^0, k_3^0)) \\
y = 0 & E(k_1^0, E(k_2^1, k_3^0)) & E(k_1^0, E(k_2^0, k_3^0)) \\
\end{array}
\]
Secure Two-Party Computation
Yao’s Garbled Circuits (3)

5. Party A generates garbled circuits and gives it to B, including the “input keys” corresponding to its own input.

6. Party B runs (together with A) “Oblivious Transfer” to obtain keys corresponding to B’s input.

7. Party B evaluates the circuit, announces the result.

\[
\begin{array}{c|c|c}
\wedge & x = 1 & x = 0 \\
y = 1 & 1 & 0 \\
y = 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
 & x = 1 & x = 0 \\
y = 1 & E(k_1^1, E(k_2^1, k_3^1)) & E(k_1^1, E(k_2^0, k_3^0)) \\
y = 0 & E(k_1^0, E(k_2^1, k_3^0)) & E(k_1^0, E(k_2^0, k_3^0)) \\
\end{array}
\]
Nice approach, but is it ready for practice?

- Cryptographic frameworks are ready, but tedious to use

- Lack of a good tool chain that a programmer can use

- Research prototypes are available:
  - Fairplay, FairplayMP, Sharemind, Tasty
  - Fast GC frameworks (implementation support for Java)

- We need “usable” compilers that help a programmer implement secure computation!
void millionaires() {
    int INPUT_A_mila;
    int INPUT_B_milb;
    int OUTPUT_res;

    if (INPUT_A_mila > INPUT_B_milb)
        OUTPUT_res = 1;
    else
        OUTPUT_res = 0;
}
Central idea: transforms ANSI C code to circuit useable for secure computation using Yao’s Garbled Circuits

ANSI C Program

Preprocessing
  Loop unrolling
  Constant propagation
  Array handling

Circuit synthesis
  Core CBMC engine
    „Placeholder“ gates

Circuit optimization
  Instantiation of placeholder gates

Netlist of circuit

http://forsyte.at/software/cbmc-gc/
Basis for CBMC-GC:
Bit-precise Model Checker CBMC

- Checks violations of assertions in ANSI C programs

Bounded Model Checker:
- “Bit-precise” transformation of C program into a Boolean formula
- Boolean formula consists of program model and negated assertions
- SAT solver checks for solution indicating a violated assertion

We use in CBMC-GC the transformation of an ANSI C program to a Boolean formula

http://www.cprover.org
CBMC-GC
Preprocessing

- Loop unrolling to
  - remove all for/while loops
  - create a flat program

- Bounds for loops need to be determined
  - constant propagation
  - manual specification if necessary

- Array handling requires MUX circuits
  - optimizations for performance
CBMC-GC
Circuit Synthesis

- Use core CBMC engine to generate Boolean circuit from unrolled program
  → code “evolution” of memory
  → yields “sparse” formula

- Rewriting formula as circuit

- Placeholder gates: highl-level representation of certain basic operations
  → to be optimized at later stage
CBMC-GC
Circuit Optimization

- Instantiation of placeholder gates
  - choose “most optimal” circuits
  - many XOR gates good for performance

- Output in the form of netlist

- Circuit can be executed in any framework implementing Yao’s Garbled Circuits
CBMC-GC:
Example, Yao’s Millionaires

```c
void millionaires() {
    int INPUT_A_mila;
    int INPUT_B_milb;
    int OUTPUT_res;

    if (INPUT_A_mila > INPUT_B_milb)
        OUTPUT_res = 1;
    else
        OUTPUT_res = 0;
}
```

Local variables code inputs and outputs
Computations specified as C program
#define S 8 // size of matrices

int INPUT_A_a[S][S];
int INPUT_B_b[S][S];
int OUTPUT_c[S][S];

void multiply()
{
    int i, j, k;
    for (i = 0; i < S; i++)
    {
        for (j = 0; j < S; j++)
        {
            for (k = 0; k < S; k++)
            {
                OUTPUT_c[i][j] += INPUT_A_a[i][k] * INPUT_B_b[k][j];
            }
        }
    }
}
Current Limitations

CBMC-GC inherits limits from model checker CBMC:

- **Bounded programs**: bounds for all loops must be known
  - in practice no problem, as we use terminating programs anyway
- No support for **floating point arithmetic**
- No support for **library functions** (yet)
- Limited **pointer arithmetic**
- **Integer data types** of fixed size
  - limits efficiency in secure computations
#define K 11 // length of array
#define MEDIAN 5 // position of median
int INPUT_A_a[K];
int OUTPUT_median;

void median_bubblesort() {
    int i, j, tmp, tmp1, tmp2;
    for (i = K - 1; i > 0; i--) {
        for (j = 0; j < i; j++) {
            tmp1 = INPUT_A_a[j]; tmp2 = INPUT_A_a[j + 1];
            if (tmp1 > tmp2) {
                INPUT_A_a[j] = tmp2; INPUT_A_a[j + 1] = tmp1;
            }
        }
    }
    OUTPUT_median = INPUT_A_a[MEDIAN];
}

CBMC can determine loop bounds by static analysis
CBMC-GC supports recursion
Example: Mergesort

```c
int b[K]; // temporary array for mergesort

void mergesort(int l, int r) {
    int i, j, k, m;
    if (r > l) {
        m = (r + l)/2; mergesort(l, m); mergesort(m + 1, r);
        for (i = m + 1; i > l; i--)
            b[i - 1] = INPUT_A_a[i - 1];
        for (j = m; j < r; j++)
            b[r + m - j] = INPUT_A_a[j + 1];
        for (k = l; k <= r; k++) {
            if (b[i] < b[j])
                INPUT_A_a[k] = b[i]; i++;
            else
                INPUT_A_a[k] = b[j]; j--;
        }
    }
}
```

Recursion: CBMC can determine bounds by static analysis
void millionaires() {
    int INPUT_A_mila;
    int INPUT_B_milb;
    int OUTPUT_res;

    if (INPUT_A_mila > INPUT_B_milb)
        OUTPUT_res = 1;
    else
        OUTPUT_res = 0;
}
Experimental results

We used CBMC-GC in conjunction with framework for execution of garbled circuits by Huang et al (USENIX 2011)

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Number of gates</th>
<th>Execution time, preprocessing</th>
<th>Execution time, circuit evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000 random arithmetic operations</td>
<td>2,298,441 (608,668)</td>
<td>970 ms</td>
<td>9,774 ms</td>
</tr>
<tr>
<td>8x8 matrix multiplication</td>
<td>3,257,345 (905,728)</td>
<td>680 ms</td>
<td>18,173 ms</td>
</tr>
<tr>
<td>Median, bubble sort, 31 elements</td>
<td>149,040 (45,120)</td>
<td>733 ms</td>
<td>1,644 ms</td>
</tr>
<tr>
<td>Median, merge sort, 31 elements</td>
<td>1,339,084 (436,916)</td>
<td>660 ms</td>
<td>3,790 ms</td>
</tr>
</tbody>
</table>
Future Release of CBMC-GC

Next release (under preparation) will feature:

- More precise heuristics for loop unrolling
- Optimized circuits for basic operations
- Automatic optimization engine for Boolean circuits
- Automatic bit-width selection for integers
- Test framework for circuits

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>1</td>
</tr>
<tr>
<td>Multiplication</td>
<td>(\approx 1)</td>
</tr>
<tr>
<td>Millionaire’s Problem</td>
<td>(\approx 2)</td>
</tr>
<tr>
<td>Hamming Distance</td>
<td>(\approx 3.5)</td>
</tr>
<tr>
<td>Bitwise AND</td>
<td>(\approx 2)</td>
</tr>
<tr>
<td>Bit shifts</td>
<td>(\approx 32)</td>
</tr>
</tbody>
</table>
Conclusions

- Protection of sensitive data requires **technical means**
- We should not rely (entirely) on administrative measures!
- **Secure computation can be practical already**, despite lack of efficient FHE!
- “Usable” compilers exist
- Integration in software engineering pipeline required…