CINGULATA

An open-source toolchain for compiling and running programs over FHE

|Sergiu Carpov>/\sqrt{2}+|Renaud Sirdey>/\sqrt{2}
CEA LIST
(based on work with additional people)

May 2018
The FHE dream

• Can Charlie do something useful for Alice using both Alice and Bob data but without revealing them (the data) to him (Charlie)?

\[
\begin{align*}
[x]_{sk} &= \text{FHE} \text{ of } x \\
[y]_{sk'} &= \text{FHE} \text{ of } y \\
[e]_{sk} &= \text{FHE} \text{ of } x \\
[\text{result}]_{sk'} &= \text{FHE} \text{ of } \text{result}
\end{align*}
\]
FHE in a nutshell

• On top of allowing to encrypt and decrypt data, an FHE scheme allows to perform (any) calculations in the encrypted domain.
  – Without access to either intermediate or final calculations results by the computer.

• Although the first generation of systems were too costly, practicality has now been achieved for a first round of (lightweight-enough) applications.

Cryptosystem API:
• \( \text{enc}_{pk} : \mathbb{Z}_2 \rightarrow \Omega \).
• \( \text{dec}_{sk} : \Omega \rightarrow \mathbb{Z}_2 \).
• \( \text{add}_{pk} : \Omega \times \Omega \rightarrow \Omega \).
• \( \text{mul}_{pk} : \Omega \times \Omega \rightarrow \Omega \).

where \( \Omega \) is a large cardinality set e.g. \( \mathbb{Z}_q \).n.
Key properties: for all \( m_1 \in \mathbb{Z}_2 \) and all \( m_2 \in \mathbb{Z}_2 \)
• \( \text{dec}_{sk}(\text{add}_{pk}(\text{enc}_{pk}(m_1), \text{enc}_{pk}(m_2))) = m_1 \oplus m_2 \).
• \( \text{dec}_{sk}(\text{mul}_{pk}(\text{enc}_{pk}(m_1), \text{enc}_{pk}(m_2))) = m_1 \otimes m_2 \).

(and these properties hold long enough...)
The quest for universality

• So we can execute boolean circuits.
  – I.e. directed graphs \( G = (V, A) \) where vertices are inputs, outputs or operators (XOR, AND).

• Boolean circuits = static control structure programs = oblivious Turing machines.

• Oblivious Turing machines are Turing-complete.

• Hence we can compute everything computable!

• And, b. t. w., it is also possible to homomorphically execute an encrypted Turing machine.

• Hence, we can in principle ensure algorithm privacy.
FHE performances?

• Somewhat FHE:
  – Multiplications have an increasing cost with the multiplicative depth.
  – Additions are « free ».
  – Large (depth-dep.) overheads.

• Bootstrapped FHE (TFHE):
  – Both multiplications and additions have the same cost, independently of the depth.
  – Smaller (depth-independent) overheads (8 ko/bit).

<table>
<thead>
<tr>
<th>depth</th>
<th>kb/bits</th>
<th>ms/AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>83</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>148</td>
<td>66</td>
</tr>
<tr>
<td>10</td>
<td>282</td>
<td>150</td>
</tr>
<tr>
<td>15</td>
<td>601</td>
<td>376</td>
</tr>
<tr>
<td>20</td>
<td>1039</td>
<td>680</td>
</tr>
</tbody>
</table>

- HomNOT(c): instantaneous
- HomAND(c₁, c₂): 13 ms
- HomXOR(c₁, c₂): 13 ms
- HomMUX(c₁, c₂, c₃): 26 ms
The « strange » FHE computer

• No ifs (unless regularized by conditionnal assignment).
• No data dependant loop termination (need upper bounds and fixed-points).
• Array dereferencing/assignment in $O(n)$ (vs $O(1)$).
• Algorithms always realize (at least) their worst-case complexity!
  – Complexity of dichotomic search?
• Can handle only a priori (multiplicative) bounded-depth programs (w/o bootstrapping).
Example: bubble sorting

- Regularization of the inner if-then-else using a cond. assignment operator.

- Static control structure hence systematic worst case complexity.
  - A price to pay for not leaking any information.

```c
template<typename integer>
void bsort(integer * const arr, const int n)
{
    assert(n>0);

    for(int i=0; i<n-1; i++)
    {
        for(int j=1; j<n-i; j++)
        {
            integer swap=arr[j-1]>arr[j];
            integer t=select(swap, arr[j-1], arr[j]);
            arr[j-1]=select(swap, arr[j], arr[j-1]);
            arr[j]=t;
        }
    }
}
```

Where `select(c,a,b) = c?a:b`. 
Array dereferencing and assignment

- With cleartext indices:
  - Straightforward.
- With encrypted indices:
  - Dereferencing: \[ t[i] \equiv \sum_{j=1}^{n} \chi(i, j) \times t[j] \]
    with \( \chi(i,j)=1 \) if \( i=j \), 0 otherwise.
    • I.e., it’s just an \( == \) operator.
  - Assignment (\( t[i]:=v \)):
    \[ t[j] := \chi(i, j) \times v \ominus (1 - \chi(i, j)) \times t[j], \forall j \]
    for \( j=1 \) to \( n \).
  - Hence array assignment and dereferencing are in \( O(n) \) (sic!).
    • But it’s the intuitive price for index privacy.
The CINGULATA compiler & RTE...

- A compiler infrastructure for high-level cryptocomputing-ready programming, taking C++ code as input.

- Boolean circuit optimization (ABC-based), parallel code generation and « cryptoexecution » (OpenMP-based) runtime environment.

- Optimized prototypes of the most efficient FHE systems known so far.
  - Also, with support of open source libs such as HELIB and (coming soon) TFHE.

ASIACCS’15, PST’17
The CINGULULATA compilation process

C++ -> Bin -> Initial boolean circuit gen. -> FHE-specific optimizations

Input program analysis

FHE keys generation

FHE parameter calculations

Mult. depth calculation

BLIF

Bin

XML

TXT

Outputs décryption

Inputs encryption

Boolean circuit execution

FHE parameter calculations

Outputs décryption

C++

Bin

Initial boolean circuit gen.

FHE-specific optimizations

FHE parameter calculations

Mult. depth calculation

BLIF

Bin

Initial boolean circuit gen. -> FHE-specific optimizations

C++ -> Bin

FHE parameter calculations -> BLIF

FHE parameter calculations -> Inputs encryption

FHE keys generation
FHE-specific optimization modules

• Specifically targeted towards sFHE.
  - « multiplicative depth busting ».
  - As a secondary objective, multiplication count decreasing.

• Based on the iterative application of local circuit rewriting operators.

• Interesting results on some real-world algorithms.
  - E.g. multiplicative depth automatically downed from ~70 (prohibitively high) to ~20 (practically achievable) on RLE.
Practical achievements

• Medical diagnostic (various flavors, various complexities) – between 3 secs and 2 mins.

• Face authentication <4 secs RTD.

• Genome-based diagnostic < 10 mins.

• Energy-consumption profile classification < 1 secs.

• And a few others…

IEEE Cloud’16, IEEE CloudCom’16, ICISSP’17, SMARTGREENS’18.
IDASH 2017

- Secure genome analysis competition.
- Learn a logistic regression model over HE encrypted data.
- Batched gradient descent algorithm:
  - Straightforward implementation in Cingulata.
  - TFHE cryptosystem.
- 2\textsuperscript{nd} place by AUC score.
History and next steps.

  – Developed mostly during CRYPTOCOMP.

• Next steps (non exhaustive):
  – More backends:
    • TFHE (gate-bootstraping flavor).
    • SEAL.
    • TFHE (LUT flavor).
  – More optimization modules.
  – Longer term: integration of VC techniques in the compilation process.
Cingulata (pronounced "tchingulata") is a compiler toolchain and RTE for running C++ programs over encrypted data by means of fully homomorphic encryption techniques.

Go to [https://github.com/CEA-LIST/Cingulata](https://github.com/CEA-LIST/Cingulata) and happily cryptocompute ever after!
BACKUPS
The CINGULATA compilation process

1. Input program analysis.
2. Initial boolean circuit generation.
5. FHE parameters generation.
6. FHE keys generation.
7. Input encryption.
8. Boolean circuit execution.