Encrypt-Everything-Everywhere
A C++ framework for computing with encrypted operands

Eduardo Chielle, Oleg Mazonka, and Michail Maniatakos
{eduardo.chielle, om22, michail.maniatakos}@nyu.edu

جامعة نيويورك أبوظبي
NYU | ABU DHABI

MoMA LAB

Introduction

In this work-in-progress blueprint, we describe E³, a framework which allows compilation of C++ programs with user-annotated private variables. The framework automatically instantiates everything required to manipulate the annotated variables. The vision of this work is to enable non-crypto-savvy programmers to use encryption in their programs seamlessly, and to serve as a baseline for performance optimizations at different levels of the computational stack (e.g., algorithms, hardware support, etc.).

Terminology

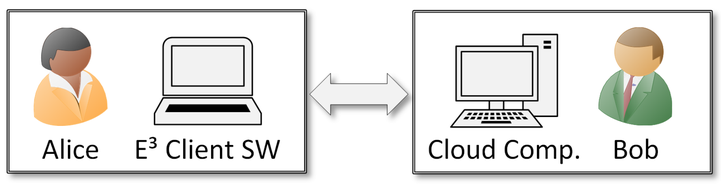
User (Alice): is a programmer who wants to run her program with encrypted data

Client: is a trusted computer in which the E³ framework software operates, building the user's program

Server (Bob): is a computer on which the user's program runs. The server is untrusted

Overview

In simple words, the problem we target is the following: Alice has a program and data, and wants to run the program on Bob's computer. Alice annotates variables which should not be exposed to Bob. Alice wants the source-code of the program to be oblivious to the underlying encryption, i.e. she wants to be able to run the same program, if needed, without encryption or additional libraries. Alice does not want to learn cryptography.



A simple example

Programming support for the E³ computation paradigm is provided through a set of software libraries and helper tools. These libraries provide new data types that can be readily used by programmers without any knowledge about cryptography. The end user (Alice) employs our developed E³ client software to compile a program with encrypted data and send it to Bob for execution.

// Data-oblivious Fibonacci program

```
#include <iostream>
#include <iostream>
#include <algorithm>
#include "circle.h"
int main(){
    int max_iter = 30;
    SecureInt input = 7_E;
    SecureInt i = 0_E, result = 0_E;
    SecureInt a = 1_E, b = 1_E;
    while( ++i, max_iter-- )
    {
        result += (i == input) * a;
        std::swap(a,b);
        a += b;
    }
    std::cout << result << '\n';
}
```

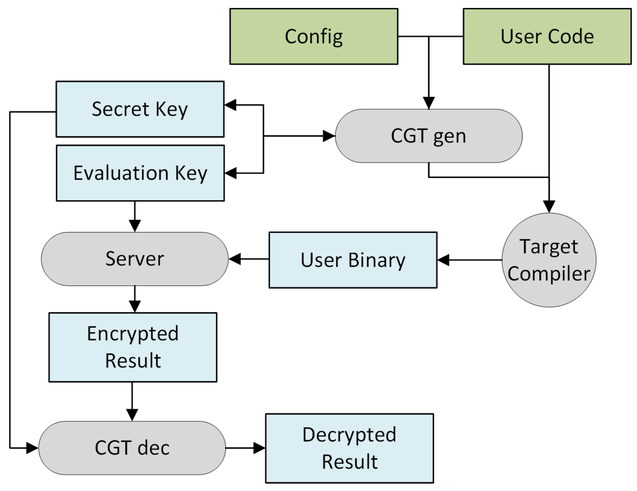
// An outline example of `circle.h`

```
class SecureInt{ ... };
constexpr SecureInt operator""_E
   (unsigned long long int x){ ... }
```

User's perspective

Alice is expected to perform a set of steps in order to run her privacy-preserving program with the E³ framework:

- modify the configuration file
- include E³ header into the program
- change the types of sensitive data
- decorate constants used for secure types
- build keys and E³ files using CGT (Class Generation Tool)
- compile the program
- send the executable to run on Bob's server
- decrypt the received output



Configuration file

The configuration file specifies the details of the secure classes to be generated as well as the details of the encryptions. A simple example of such file is the following:

```
source_dir = alice_code
password = hello_world
Secure : circuit
{
  postfix = E
  encryption = tfhe
}
```

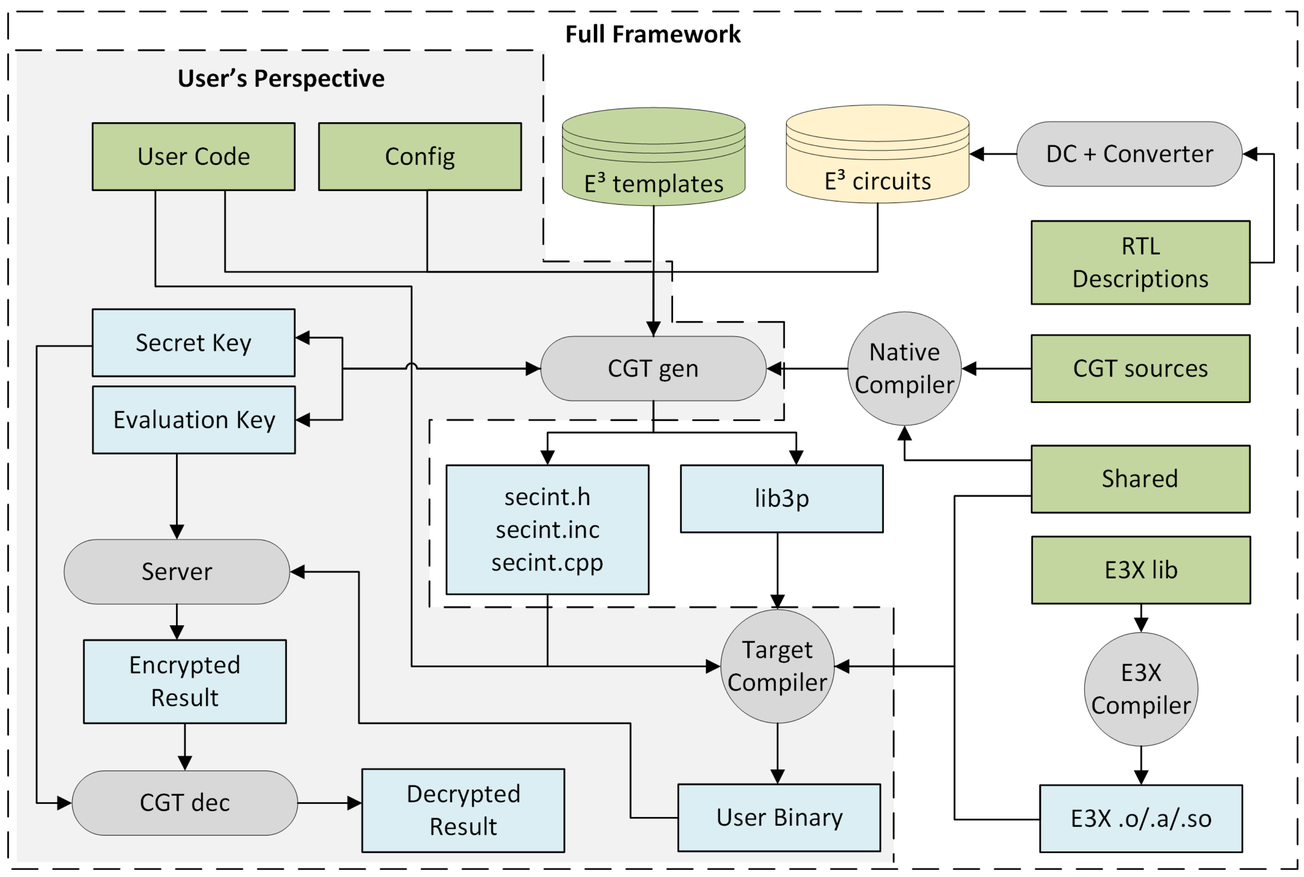
Architecture

The developed client software is managing the process transparently to Alice. To better understand the framework flow, we can walk through the example backwards: The decrypted result is obtained by running CGT on the encrypted result. In this case, CGT reads the secret key from a file. The encrypted result is obtained by running the user's binary on the server. The evaluation key is also required by the binary to run.

The user's binary is compiled by the target compiler using 1) user's code; 2) generated classes (`secint`); 3) shared code; and 4) encryption engine library (lib3p).

The generated classes are produced by CGT, which reads the configuration file, the user's code, E³ templates, and E³ circuits (if necessary). CGT also generates the adapter for the encryption library engine. CGT generates keys if they are not present.

E³ templates are not C++ templates; they are a specific code written by hand which is used as a prototype for the generated classes. E³ circuits are functions which are generated from RTL circuits by Design Compiler (DC) and converted into C++. CGT is built from its sources which are split into two parts: the private part, and the part shared with the user's code. The shared sources are utility functions and the evaluation key classes.



Final remarks

The E³ framework presented is the first step towards a generic practical tool to advance research in privacy-preserving computation. Currently, the framework is being actively developed. We have already implemented the overall structure with a mock-up and encryption libraries – SEAL, TFHE, FHEW, and Helib, as well as in-house faster homomorphic encryptions PIL and BDD-based.

August 2019