PALISADE python wrapper

introduction, multiparty, and how to use

Yeonsang Shin
yeonsang.shin@desilo.ai

Part 1

Introduction

PALISADE Lattice Cryptography Library
User Manual (v1.11.3)

Yuriy Polyakov[1,2], Kurt Rohloff[1,2], Gerard W. Ryan[2], and Dave Cousins[1,2]

[1]Duality Technologies, Newark, NJ, 07102, USA.
{ypolyakov,krohloff,dcousins}@dualitytech.com
[2]Cybersecurity Research Center, New Jersey Institute of Technology (NJIT),
Newark, NJ, 07102, USA. {polyakov,rohloff,dcousins}@njit.edu

May 28, 2021

Abstract

This document is the manual for the PALISADE lattice cryptography library. This manual provides an introduction to the library by describing the library architecture and cataloging its capabilities. We focus on the PALISADE library's ability to support homomorphic encryption capabilities to evaluate arithmetic operations on data while

+ pybind11

pybind11 — Seamless operability between C++11 and Python

PyPalisade

Python wrapper library for PALISADE Homomorphic Encry

Basic Info : Why PALISADE?

BGV, BFV, CKKS

BinFHE (TFHE)

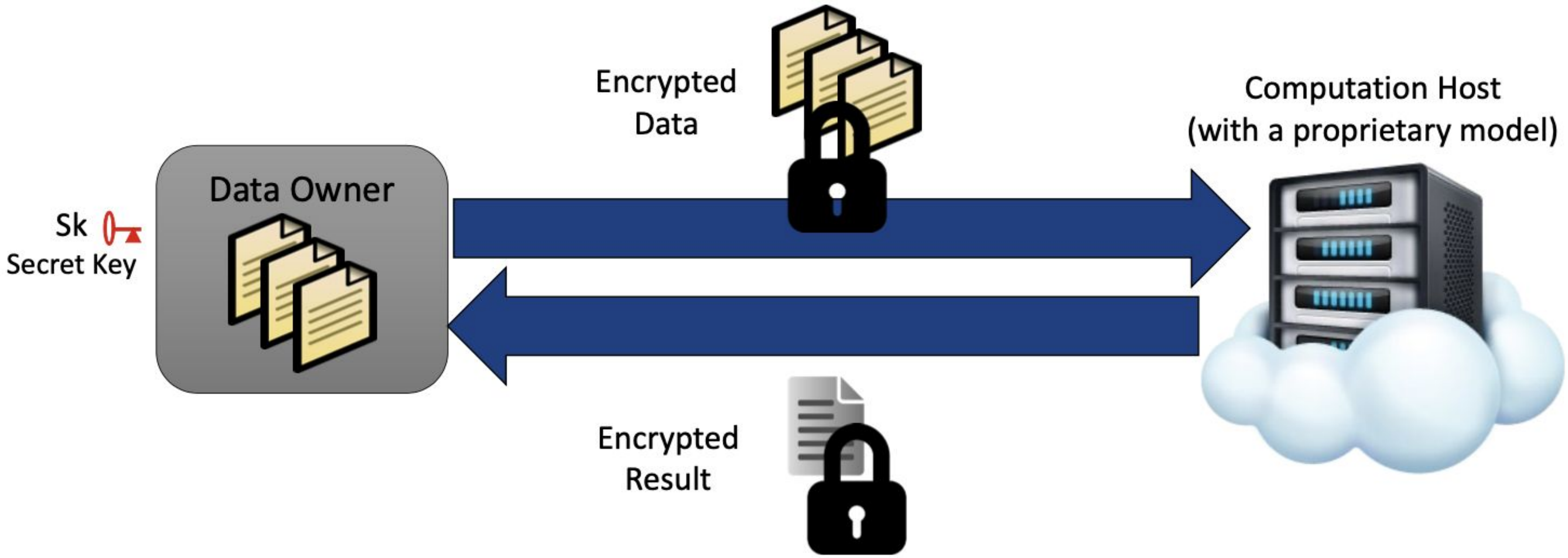
Serialization

Multiparty: Threshold HE

Part 2

Basic Info : Multiparty HE

SINGLE-KEY HE WORKFLOW



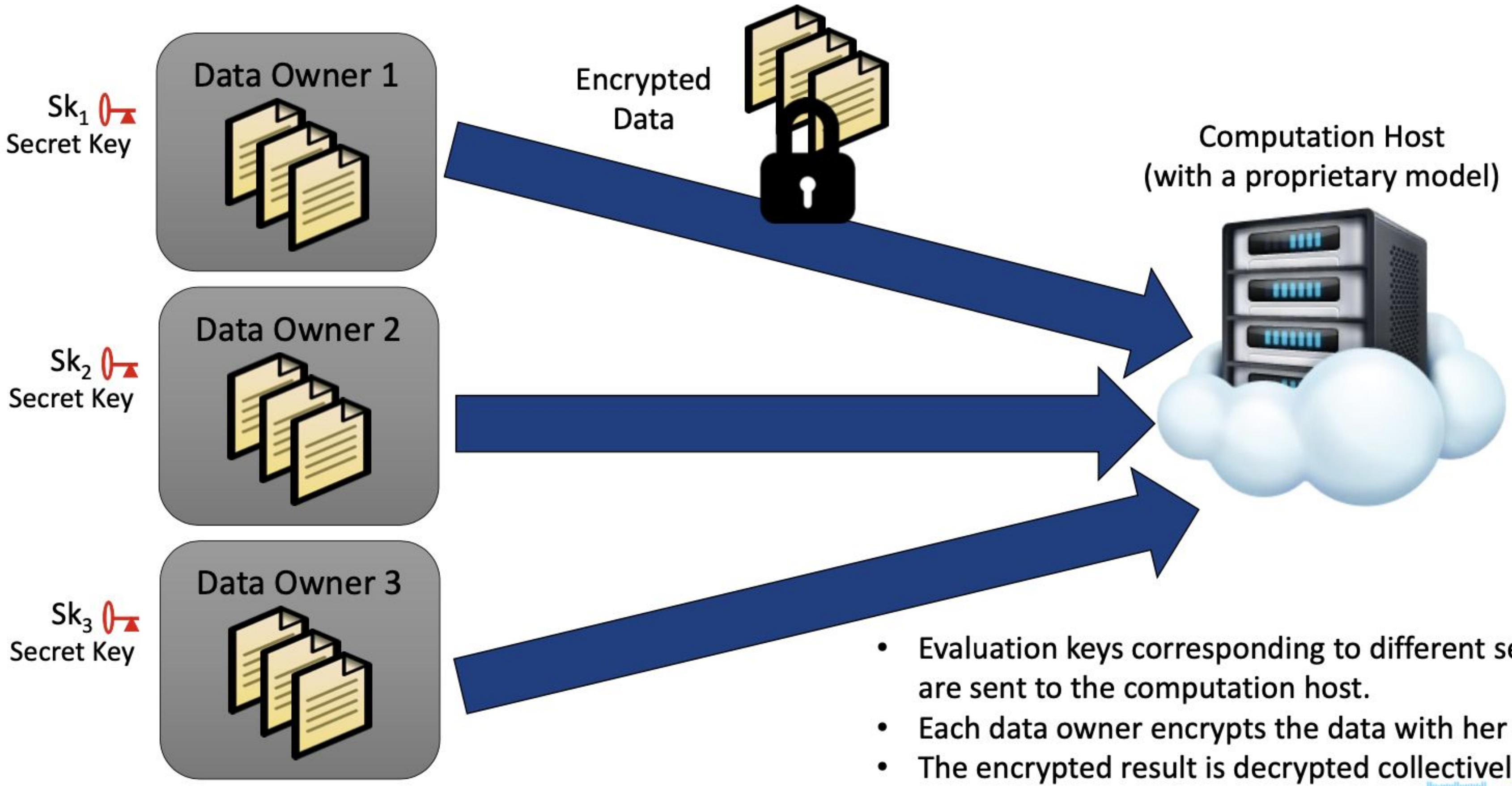
How can this model be extended to multiple data owners that do not want to share a secret key or data?
What if the model needs to be encrypted by model provider and sent to the computation host? What key should the model provider use for encryption?

PALISADE

Part 2

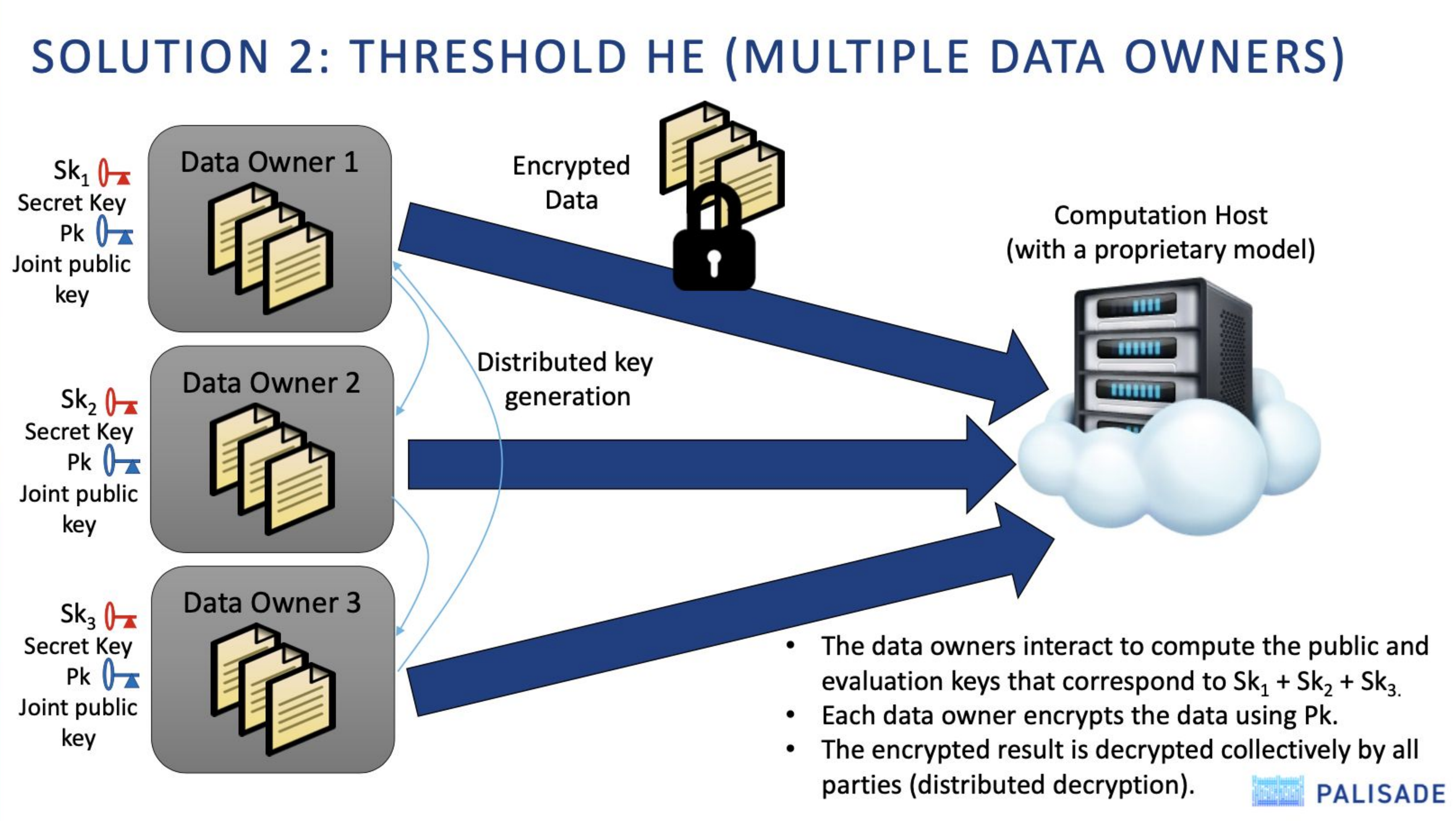
Basic Info : Multiparty HE

SOLUTION 1: MULTIKEY HE (MULTIPLE DATA OWNERS)



- Evaluation keys corresponding to different secret keys are sent to the computation host.
- Each data owner encrypts the data with her own key.
- The encrypted result is decrypted collectively by all parties (distributed decryption).

Basic Info : Multiparty HE



Basic Info : Multiparty HE

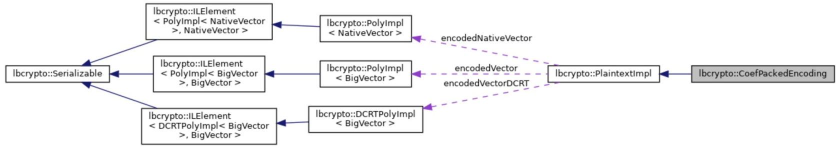
COMPARISON OF MULTIKEY AND THRESHOLD HE

Parameter	Multikey HE	Threshold HE
Key generation	Non-interactive (asynchronous)	Interactive (synchronous)
Number of parties	Supports a variable number of parties, bounding only the number of parties involved in a specific computation	The number of parties is fixed
Decryption	Interactive (all parties compute partial decryptions and merge them)	Interactive (all parties compute partial decryptions and merge them)
Computation runtime	Grows quadratically (asymptotically; slightly better in practice) with the number of parties [CDKS19]	Roughly the same as in single-key HE
Evaluation and ciphertext size	Linear in the number of parties [CDKS19]	Roughly the same as in single-key HE

Part 3

How to Use

Collaboration diagram for lbcrypto::CoefPackedEncoding:



PALISADE C++

Personal C++ Classes

PALISADE Python

templates
virtual functions
abstract classes
smart pointers

Part 3

How to Use

Run setup.py (Install library)

```
# From folder pypalisade (cd ../..)
CFLAGS="-fopenmp" python3 setup.py build_ext -i
```

Examples

- 1_CKKS_EvalAdd.py
- 2_BGVrns_EvalAddMany.py
- 3_BFVrns_EvalMultMany.py
- 4_BinFHE_EvalBinGate.py
- 5_Serialization.py
- 6_Serialization_no_sk.py
- 7_Serialization_BinFHE.py
- 8_Multiparty_two_parties_with_serializati...
- 9_Multiparty_four_parties.py

Pypalisade Manual

This is a python wrapper for PALISADE (1.11.3)

Contents

Namespace pypalisade

- Class Ciphertext
- Class Plaintext
- Class LWECiphertext
- Class CryptoContext
- Class BinFHEContext
- Class PublicKey
- Class SecretKey
- Class KeyPair
  - Constructor
  - GetPublicKeyFromPair
  - GetSecretKeyFromPair

#	Type	Name	Default
1	usint	**multDepth**	-
2	usint	**plaintextModulus**	-
3	SecurityLevel	**stdLevel**	HEStd_128_classic
4	float	**stdDev**	3.19
5	usint	**maxDepth**	2
6	MODE	**mode**	OPTIMIZED
7	KeySwitchTechnique	**ksTech**	HYBRID
8	usint	**ringDim**	0
9	uint32_t	**numLargeDigits**	0
10	usint	**firstModSize**	0
11	usint	**dcrtBits**	0
12	usint	**relinWindow**	0
13	usint	**batchSize**	0
14	ModSwitchMethod	**msMethod**	AUTO

1. **multDepth** : Depth of multiplications
2. **plaintextModulus** : plaintext modulus
3. **stdLevel** : standard security level we want the scheme to satisfy
4. **stdDev** : sigma - distribution parameter for error distribution
5. **maxDepth** : maximum power of secret key for which relin key is generated
6. **mode** : RLWE (gaussian distribution) or OPTIMIZED (ternary distribution)
7. **ksTech** : key switching technique to use (HYBRID, GHS, BV)
8. **ringDim** : the ring dimension (if not specified selected automatically based on stdLevel)
9. **numLargeDigits** : the number of big digits to use in HYBRID key switching
10. **firstModSize** : the bit-length of the first modulus
11. **dcrtBits** : the size of the moduli in bits
12. **relinWindow** : the relinearization windows (used in BV key switching, use 0 for RNS decompositi
13. **batchSize** : the number of slots being used in the ciphertext
14. **msMethod** : mod switching Method (AUTO, MANUAL)

Google FHE Transpiler

introduction, examples, and some experiments

Yeonsang Shin
yeonsang.shin@desilo.ai

Part 1

Introduction

A General Purpose Transpiler for Fully Homomorphic Encryption

Shruthi Gorantala, Rob Springer, Sean Purser-Haskell, William Lam, Royce Wilson, Asra Ali, Eric P. Astor, Itai Zukerman, Sam Ruth, Christoph Dibak, Phillipp Schoppmann, Sasha Kulankhina, Alain Forget, David Marn, Cameron Tew, Rafael Misoczki, Bernat Guillen, Xinyu Ye, Dennis Kraft, Damien Desfontaines, Aishe Krishnamurthy, Miguel Guevara, Irippuge Milinda Perera, Yurii Sushko, and Bryant Gipson

fhe-open-source@google.com

June 14, 2021

Abstract

Fully homomorphic encryption (FHE) is an encryption scheme which enables computation on encrypted data without revealing the underlying data. While there have been many advances in the field of FHE, developing programs using FHE still requires expertise in cryptography. In this white paper, we present a fully homomorphic encryption transpiler that allows developers to convert high-level code (e.g., C++) that works on unencrypted data into high-level code that operates on encrypted data. Thus, our transpiler makes transformations possible on encrypted data.

Our transpiler builds on Google's open-source XLS SDK [1] and uses an off-the-shelf FHE library, TFHE [2], to perform low-level FHE operations. The transpiler design is modular, which means the underlying FHE library as well as the high-level input and output languages can vary. This modularity will help accelerate FHE research by providing an easy way to compare arbitrary programs in different FHE schemes side-by-side. We hope this lays the groundwork for eventual easy adoption of FHE by software developers. As a proof-of-concept, we are releasing an experimental transpiler [3] as open-source software.

Introduction

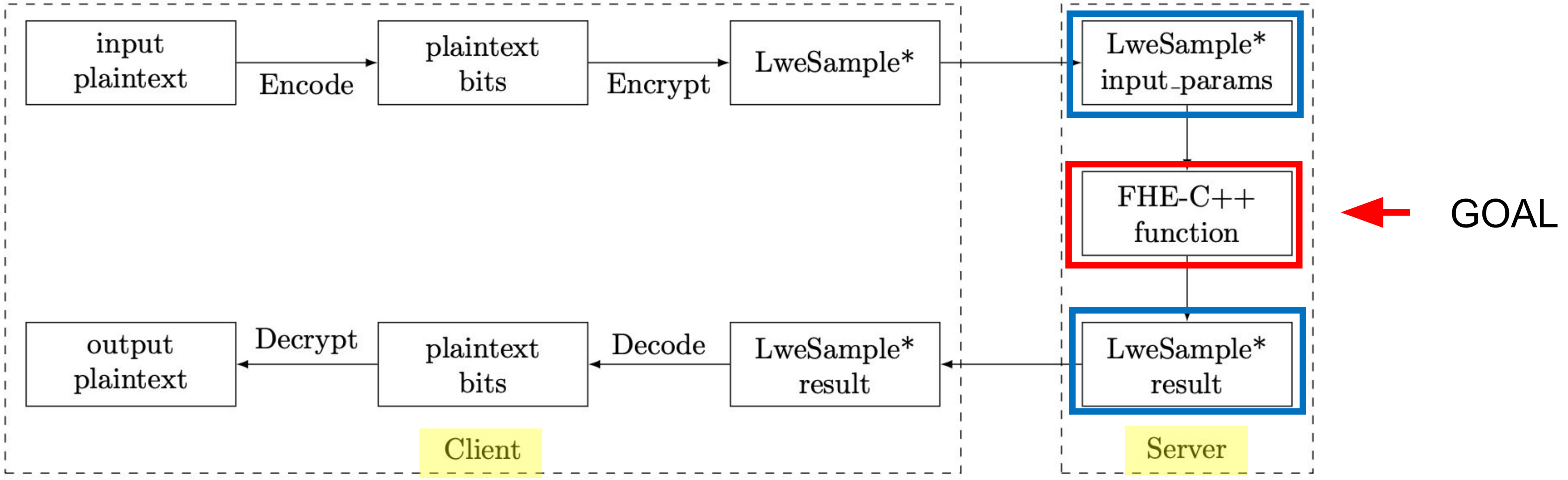


Figure 4: Client server interaction.

Introduction

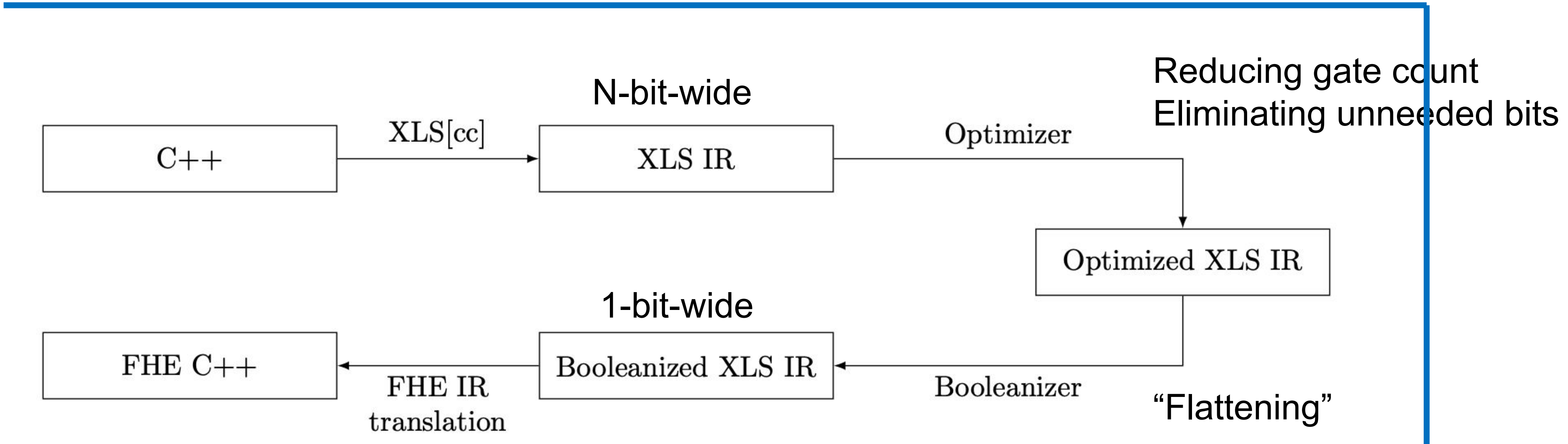


Figure 2: High-level overview of the transpiler invocation process.

Introduction

Modular in three ways

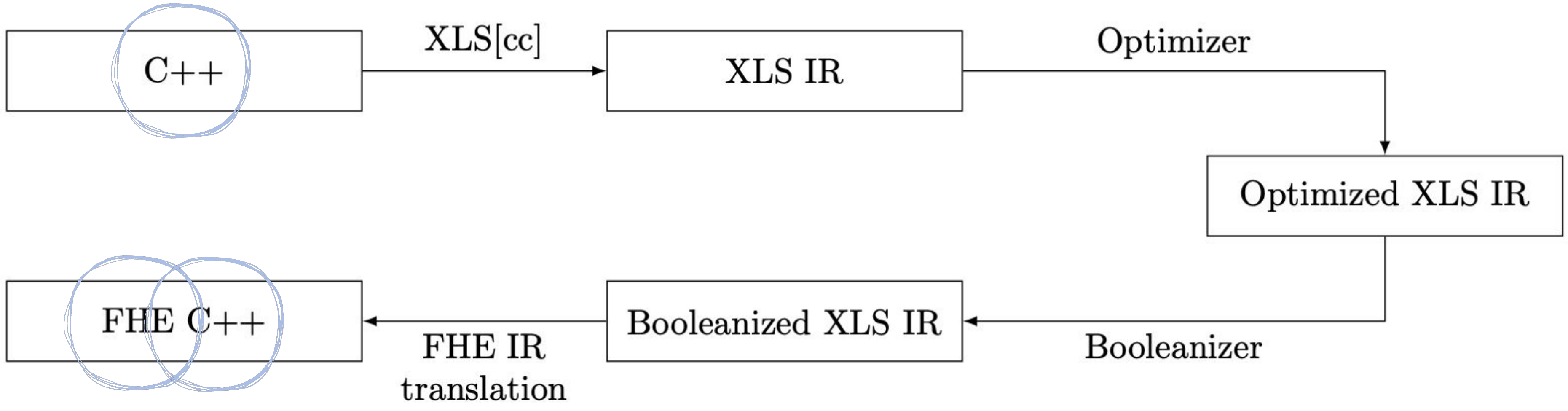


Figure 2: High-level overview of the transpiler invocation process.

Introduction

TFHE

TFHE: Fast Fully Homomorphic Encryption over the Torus*

Ilaria Chillotti[1], Nicolas Gama[3,2], Mariya Georgieva[4,3], and Malika Izabachène[5]

[1] imec-COSIC, KU Leuven,
Kasteelpark Arenberg 10, Bus 2452, B-3001 Leuven-Heverlee, Belgium
ilaria.chillotti@kuleuven.be
[2] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, 78035 Versailles, France
[3] Inpher, Lausanne, Switzerland
nicolas@inpher.io, mariya@inpher.io
[4] EPFL, Route Cantonal, CH-1015 Lausanne, Switzerland
[5] CEA, LIST, Point Courrier 172, 91191 Gif-sur-Yvette Cedex, France
malika.izabachene@cea.fr

Abstract. This work describes a fast fully homomorphic encryption scheme over the torus (TFHE), that revisits, generalizes and improves the fully homomorphic encryption (FHE) based on GSW and its ring variants. The simplest FHE schemes consist in bootstrapped binary gates.

- **Gate Operations**
  (AND, OR, NOT, …)
- **Unlimited computations**
  without noise management
  (Bootstrapping after every op)

Basic Info : How To Use

- Add **#pragma** at original C++ codes
  - #pragma hls_top, #pragma hls_unroll yes

- **BUILD** files recognized by Bazel
  - **Transpiler Type**:
    TFHE (Single-Core), Interpreted TFHE (Multi-Core), bool (FHE X, for debugging)
  - **num_opt_passes**:
    number of optimization passes to run on XLS IR

Basic Info : Restrictions

“Data-Independent Code”

- Variable-length loop, array X
- Recursion X
- Pointers X
- Branch-and-bound optimizations X
- Early returns: not useful

Examples

- calculator
- fibonacci
- hangman
- pir
- rock_paper_scissor
- simple_sum
- string_cap
- string_cap_char
- string_reverse
- structs

Total Time

by abseil::Now( )

CPU Time

by clock( )

Examples

6. Simple_Sum

두 input의 sum을 반환한다 (예시: 4052 + 913)

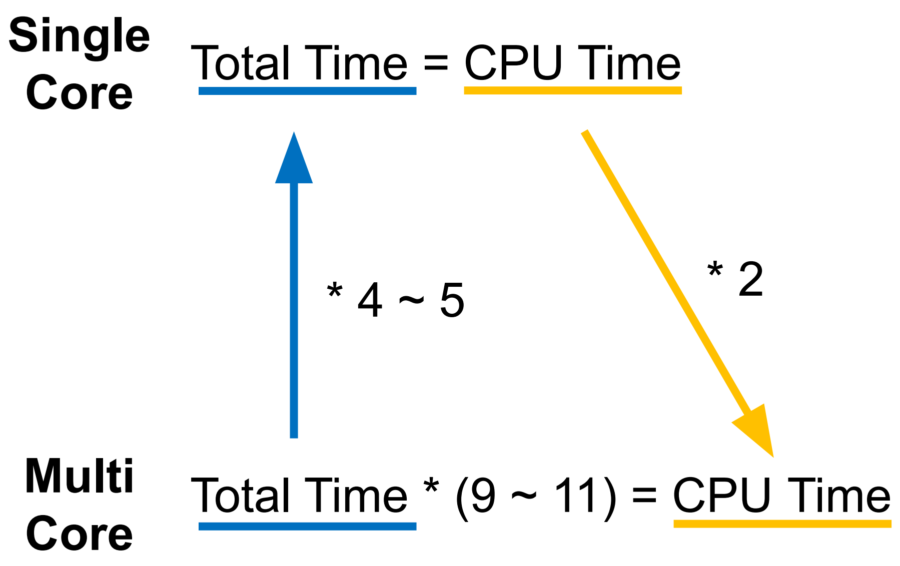
→ 덧셈만 함

▼ simple_sum_tfhe_testbench

```
inputs are 4052 and 913, sum: 4965
Encryption done
Initial state check by decryption:
4052  913
                                        Server side computation:
                                        Computation done

Computation done
Total time: 31.6786 secs
  CPU time: 31.6736 secs

Decrypted result: 4965
Decryption done
```

▼ simple_sum_interpreted_tfhe_testbench

```
inputs are 4052 and 913, sum: 4965
Encryption done
Initial state check by decryption:
4052  913
                                        Server side computation:
                                        Computation done

Computation done
Total time: 9.64436 secs
  CPU time: 56.3658 secs

Decrypted result: 4965
Decryption done
```



Examples

8. Reverse String

Reverses the input string

Name	Total Time	CPU Time
string_reverse_tfhe	1669.76	1663.39
string_reverse_interpreted_tfhe	324.441	3719.09

← 28 min..?

Examples

9. Structs

- ▶ 9-1. simple_struct
- ▶ 9-2. struct_with_array
- ▶ 9-3. return_struct
- ▶ 9-4. return_struct_with_inout
- ▶ 9-5. struct_with_struct_array
- ▶ 9-6. array_of_array_of_structs

▼ 9-2. struct_with_array

```
struct StructWithArray {
  int a[3];
  short b[4];
  int c;
}

// return a struct with every element reversed
StructWithArray NegateStructWithArray(StructWithArray input);
```

```
Initial round-trip check:
  a[0]: 0
  a[1]: 100
  a[2]: 200
  b[0]: 0
  b[1]: -100
  b[2]: -200
  b[3]: -300
  c: 1024

Starting computation.
Total time: 113.651 secs
  CPU time: 113.626 secs

Done. Result:
  a[0]: 0
  a[1]: -100
  a[2]: -200
  b[0]: 0
  b[1]: 100
  b[2]: 200
  b[3]: 300
  c: -1024
```

Desilo Analysis Functions

1. **Column Sum**
   Input: Sum Column, Apply Column
   Returns: sum of sum_column[i] * apply_column[i]

2. **Filter**
   Input: Column Array, Filter Array
   Returns: filtered column array

3. **Count**
   Input: Count Array (0 or 1)
   Returns: number of 1

Variables

- num_opt_passes
- array size

Part 4

Desilo Analysis Functions

Column Sum

Name	num_opt_passes	Array size	Total Time	CPU Time
TFHE	2	5	3112.49	2848.44
Interpreted #01	1	5	2891.4	31021.4
Interpreted #02	2	5	539.908	6036.84
Interpreted #03	3	5	542.224	6012.09
Interpreted #04	4	5	521.15	6098.69
Interpreted #05	2	10	1044.67	12312.2
Interpreted #06	2	7	736.794	8539.61
Interpreted #07	2	20	2126.44	24758

Filter

Name	num_opt_passes	Array size	Total Time	CPU Time
TFHE	2	5	3157.46	2677.3
Interpreted #01	1	5	3153.91	30868.4
Interpreted #02	2	5	583.555	5574.85
Interpreted #03	3	5	479.025	5552.96
Interpreted #04	4	5	482.143	5510.62

Count

Name	num_opt_passes	Array size	Total Time	CPU Time
TFHE	2	5	121.293	121.183
Interpreted #00	0	5	21.7771	187.898
Interpreted #01	1	5	17.375	149.08
Interpreted #02	2	5	26.0417	248.337

Desilo Analysis Functions

Conclusions

- Success!

- **Array Size ∝ Time**
  - time is required for TFHE

- **num_opt_passes ≤ 2** is enough
  But bigger is not always good…