The Lattigo lattice-based cryptographic library

Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Pierre Hubaux

Motivation

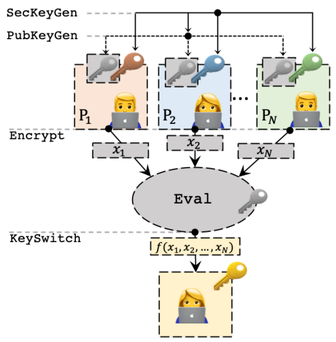
- The tremendous performance improvement of homomorphic encryption makes it now usable
- Its greatest potential is not in privacy-preserving cloud outsourcing, but for **Secure Multiparty Computation** (SMC)
- This requires a lattice-based crypto library that:
  - is easy to deploy in modern software stacks
  - can support quick academic prototyping

Lattigo at a glance

- ✓ Pure Go (1.12)
- ✓ Optimized math layer
- ✓ Encrypted integer-arithmetic (BFV[1])
- ✓ Encrypted complex/float arithmetic (CKKS[2])
- ✓ Automatic Chebyshev approximation
- ✓ Distributed/Threshold key & SMC support

SMC Framework

- Secret shared keys / collective encryption
- Collective key-switching/re-encrpytion



Upcoming

- Bootstrapping for CKKS
- Post-quantum key exchange & signatures
- Zero-knowledge proofs
- Network layer implementation of SMC

School of Computer and Communication Science

LATTIGO

The **Lattigo** library unleashes the potential of **lattice-based cryptography** in **secure multiparty computation** for **modern software** stacks.

> **go get** Lattigo !

https://github.com/lca1/lattigo

Library overview

`lattigo/dbfv`
- Collective key generation
- Relinearization key generation
- Collective key-switching

`lattigo/dckks`
- Collective key generation
- Relin/bootstraping key gen.
- Collective key-switching

`lattigo/bfv`
- Encrypted integer arithmetic
- Full-RNS operations

`lattigo/ckks`
- Encrypted complex arithmetic
- Full-RNS operations
- Automatic Chebyshev approx.

`lattigo/ring`
- **NTT**-polynomial and **RNS**-coefficients representations & arithmetic
- Efficient 64b modular arithmetic
  - **Short Barret** reduction for 64b integers
  - **Barret** reduction for arbitrary 64b×64b product
  - **Montgomery** reduction
- Efficient sampling in
  - **Gaussian** distribution
  - **RNS** and **Montgomery** forms

Benchmarks

Processor: Intel Xeon Gold 6132 @ 2.6 GHz **Memory**: 256 GB
Go version: Go1.12.7 linux/amd **Baseline**: SEAL 3.2.0[3]

Operation timing [ms] (baseline ratio)

`lattigo/bfv`

	8192-4		16384-8		32768-16	
Encrypt	13.3	(4.6)	31.1	(2.9)	86.3	(2.0)
Decrypt	2.3	(1.4)	9.4	(1.4)	39.7	(1.5)
Add	0.3	(5.9)	0.7	(3.2)	2.7	(1.9)
Multiply	47.4	(2.6)	200.6	(2.6)	947.1	(2.9)
Relin	7.0	(1.8)	44.2	(1.7)	331.4	(1.9)
RotateCols	6.9	(1.1)	44.8	(1.3)	335.8	(1.7)
RotateRows	6.9	(2.3)	45.0	(2.7)	335.9	(3.3)

`lattigo/ckks`

	8192-4		16384-8		32768-16	
Encode	3.0	(0.6)	11.9	(0.8)	47.9	(1.0)
Decode	8.3	(1.1)	26.8	(0.8)	96.8	(0.7)
Encrypt	13.2	(4.8)	32.2	(3.2)	88.8	(2.4)
Decrypt	0.4	(2.0)	1.3	(1.4)	4.8	(1.2)
Add	0.3	(6.3)	0.7	(3.0)	2.4	(1.6)
Multiply	0.6	(0.6)	2.2	(0.5)	8.9	(0.4)
Relin	5.0	(1.5)	36.8	(1.6)	302.7	(1.9)
Rescale	1.6	(0.6)	6.8	(0.6)	28.2	(0.6)
Conjugate	5.7	(0.9)	40.4	(1.2)	319.8	(1.6)
RotateCols	5.8	(1.9)	40.6	(2.4)	317.5	(3.1)

Ring size [#coeff – #moduli]

0 1 2 3 4 5 6 7 8 9 10

Baseline ratio scale

[1] Fan, Junfeng, and Frederik Vercauteren. “Somewhat Practical Fully Homomorphic Encryption.” IACR Cryptology ePrint Archive 2012 (2012): 144.
[2] Cheon, Jung Hee, et al. “Homomorphic encryption for arithmetic of approximate numbers.” International Conference on the Theory and Application of Cryptology and Information Security. Springer, Cham, 2017.
[3] Microsoft SEAL (3.2). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA., 2019