PALISADE – Lattice-Based Homomorphic Encryption Library

Yuriy Polyakov[1,2], Kurt Rohloff[1,2]

[1]Duality Technologies, [2]New Jersey Institute of Technology

Introduction

PALISADE is a lattice cryptography library developed since 2014. The development of PALISADE has been funded by DARPA, IARPA, Sloan Foundation, and NSA.

The key contributors and implementation partners include:
- Academia: NJIT, MIT, UCSD, WPI, Waseda Univ., National University of Singapore, Sabanci Univ.
- Industry: Duality Technologies, CACI/LGS Innovations, Galois, Raytheon BBN Technologies, Perspecta / Vencore Labs, Two Six Labs, Duality Technologies
- US Navy / NIWC

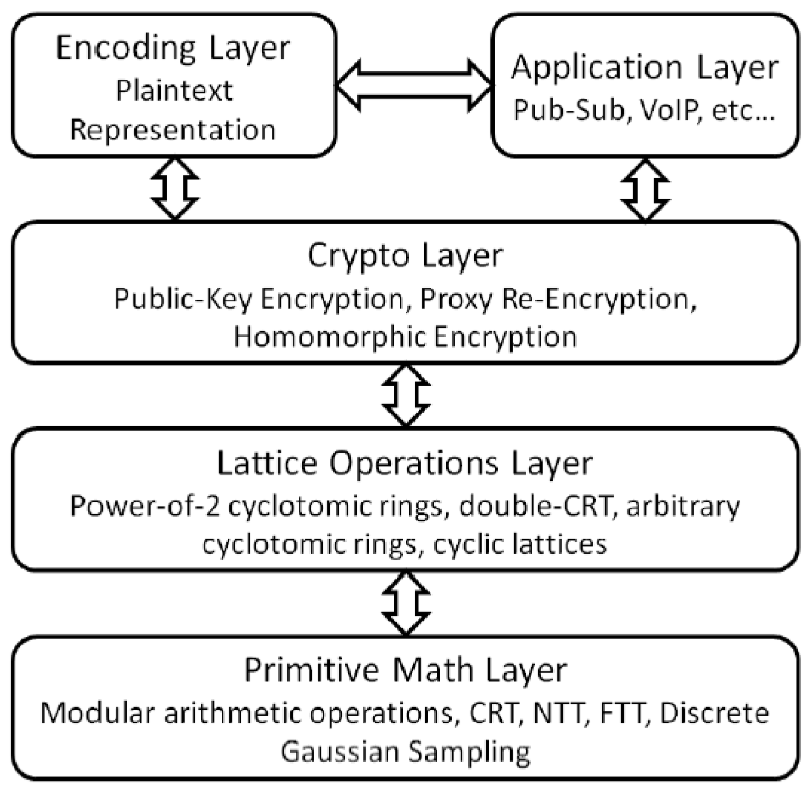
PALISADE is a cross-platform C++11 library supporting Linux, Windows, and macOS. The supported compilers are g++ and clang++.
PALISADE is publicly available under the BSD 2-clause license.

PALISADE v1.5 includes efficient implementations of the following lattice cryptography capabilities:
- Homomorphic Encryption (HE): Brakerski/Fan-Vercauteren (3 variants) [1-3], Brakerski-Gentry-Vaikuntanathan [4], and Stehle-Steinfeld [5] schemes
- Proxy Re-Encryption for all HE schemes
- Digital Signature [6]
- Identity-Based Encryption [6]
- Ciphertext-Policy Attribute-Based Encryption [7]

PALISADE implements efficient Residue Number System (RNS) algorithms to achieve best performance results, e.g., PALISADE was used as the library for a winning genome-wide association studies solution at iDASH'18 [8].

Modular Architecture and Focus on Usability

- Extendible and adaptable library for lattice cryptography; new schemes can be added or existing capabilities can be combined.
- Modular structure to mix and match components, e.g., multiple math backends are supported.
- Semi-automated lattice parameter selection.
- A common API is used for all HE schemes.
- Industrial software engineering practices are employed, including run-time polymorphism, Google unit test and benchmark frameworks, and others.



Key Concepts/Classes

Cryptocontext wrapper class:
- A wrapper that encapsulates the scheme, crypto parameters, encoding parameters, and keys.
- Provides the same API for all HE schemes.

Ciphertext class:
- Stores the ciphertext polynomials.

Plaintext class:
- Stores the plaintext data (both raw and encoded).
- Supports multiple encodings in a polymorphic manner, including PackedEncoding, IntegerEncodering, CoefPackedEncoding, etc.

Code Sample

```
// Sample Program: Step 1 - Set CryptoContext

//Set the main parameters
int plaintextModulus = 65537;
double sigma = 3.2;
SecurityLevel securityLevel = HEStd_128_classic;
uint32_t depth = 2;

//Instantiate the crypto context
CryptoContext<DCRTPoly> cryptoContext =
    CryptoContextFactory<DCRTPoly>::genCryptoContextBFVrns(
        plaintextModulus, securityLevel, sigma, 0, depth, 0, OPTIMIZED);

//Enable features that you wish to use
cryptoContext->Enable(ENCRYPTION);
cryptoContext->Enable(SHE);

//Sample Program: Step 2 - Key Generation

// Initialize Public Key Containers
LPKeyPair<DCRTPoly> keyPair;

// Generate a public/private key pair
keyPair = cryptoContext->KeyGen();

// Generate the relinearization key
cryptoContext->EvalMultKeyGen(keyPair.secretKey);

//Sample Program: Step 3 - Encryption

// First plaintext vector is encoded
std::vector<int64_t> vectorOfInts1 = {1,2,3,4,5,6,7,8,9,10,11,12};
Plaintext plaintext1 = cryptoContext->MakePackedPlaintext(vectorOfInts1);
// Second plaintext vector is encoded
std::vector<int64_t> vectorOfInts2 = {3,2,1,4,5,6,7,8,9,10,11,12};
Plaintext plaintext2 = cryptoContext->MakePackedPlaintext(vectorOfInts2);
// Third plaintext vector is encoded
std::vector<int64_t> vectorOfInts3 = {1,2,5,2,5,6,7,8,9,10,11,12};
Plaintext plaintext3 = cryptoContext->MakePackedPlaintext(vectorOfInts3);

// The encoded vectors are encrypted
auto ciphertext1 = cryptoContext->Encrypt(keyPair.publicKey, plaintext1);
auto ciphertext2 = cryptoContext->Encrypt(keyPair.publicKey, plaintext2);
auto ciphertext3 = cryptoContext->Encrypt(keyPair.publicKey, plaintext3);
```

Code Sample (Cont'd)

```
//Sample Program: Step 4 - Evaluation

// Homomorphic additions
auto ciphertextAdd12 = cryptoContext->EvalAdd(ciphertext1,ciphertext2);
auto ciphertextAddResult = cryptoContext->EvalAdd(ciphertextAdd12,ciphertext3);

// Homomorphic multiplications
auto ciphertextMul12 = cryptoContext->EvalMult(ciphertext1,ciphertext2);
auto ciphertextMultResult = cryptoContext->EvalMult(ciphertextMul12,ciphertext3);

//Sample Program: Step 5 - Decryption

// Decrypt the result of additions
Plaintext plaintextAddResult;
cryptoContext->Decrypt(keyPair.secretKey, ciphertextAddResult, &plaintextAddResult);

// Decrypt the result of multiplications
Plaintext plaintextMultResult;
cryptoContext->Decrypt(keyPair.secretKey, ciphertextMultResult, &plaintextMultResult);

// Output results
cout << plaintextAddResult << endl;
cout << plaintextMultResult << endl;
```

PALISADE and HE Standard

PALISADE implements the security settings recommended in Section 1.2 of the HomomorphicEncryption.org security standard.

PALISADE implements the BGV and BFV schemes described in Section 1.1.3 of the security standard.

One of the key goals of the PALISADE team is to develop a usable HE interface that a non-expert application developer can use to build applications with HE capabilities.

The PALISADE team has also been actively participating in IARPA's RAMPARTS and HECTOR programs building a framework that enables the development of a broad spectrum of secure distributed applications using homomorphic encryption.

Upcoming Releases

The PALISADE team is dedicated to improving the application developer experience when building applications with HE capabilities, and the next two releases are primarily focused on usability improvements.

Release v1.6 (early Q4 2019) includes the following improvements:
- Simplified cross-platform build process using CMake
- Much faster and simpler-to-use serialization
- Reduced footprint
- And Many More!

Release v1.7 (late Q4 2019) includes a more usable and efficient RNS variant of the Cheon-Kim-Kim-Song (CKKS) scheme.
The same API will be used for CKKS as for the other schemes.

Contact

Yuriy Polyakov
Duality Technologies & NJIT
Email: ypolyakov@duality.cloud
Website: https://duality.cloud/

Kurt Rohloff
Duality Technologies & NJIT
Email: krohloff@duality.cloud
Website: https://duality.cloud/

References

1. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive, 2012:144, 2012.
2. Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In International Conference on Selected Areas in Cryptography, pages 423–442. Springer, 2016.
3. Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved rns variant of the BFV homomorphic encryption scheme. In CT-RSA, pages 83-105, 2019.
4. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT), 6(3):13, 2014.
5. Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In KennethG. Paterson, editor, Advances in Cryptology – EUROCRYPT 2011, volume 6632 of Lecture Notes in Computer Science, pages 27–47. Springer Berlin Heidelberg, 2011.
6. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In STOC, pages 197–206, 2008.
7. Jiang Zhang, Zhenfeng Zhang, and Aijun Ge. Ciphertext policy attribute-based encryption from lattices. In Heung Youl Youm and Yoojae Won, editors, 7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012, pages 16–17. ACM, 2012.
8. Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, Kurt Rohloff, and Vinod Vaikuntanathan. Optimized Homomorphic Encryption Solution for Secure Genome-Wide Association Studies. IACR Cryptology ePrint Archive, Report, 2019:223, 2019.