The PALISADE Project

Yuriy Polyakov
polyakov@njit.edu

WAHC’2017

NJIT Cybersecurity Research

- University in metro New York City
  - Ranked highest Return-On-Investment in USA
- Cybersecurity center
  - 4 faculty, 3 scientists,~10 PhD students
  - NSA Center of Excellence
- Expertise in:
  - Encrypted Computing
  - Mobile/Android Security
  - Embedded system security
- Extensive industry and government collaboration
- We’re hiring aggressively

Yuriy Polyakov polyakov@njit.edu

Agenda

- Motivation
- Overview of the PALISADE project
- Alpha release
- Upcoming features
- How to use PALISADE for
  - Public Key Encryption (PKE)
  - Somewhat Homomorphic Encryption (SHE)
  - Proxy Re-Encryption (PRE)
- Concluding remarks

Lattice-Based Cryptography

NJIT

- Emerging cryptography field
  - Provides fully homomorphic encryption (FHE) and somewhat homomorphic encryption (SHE) capabilities
  - Constructions are resistant to quantum computing attacks (post-quantum security)
  - Implementations are based on efficient polynomial multiplications
  - Supports a wide array of advanced cryptographic protocols
    - Identity-based encryption
    - Attribute-based encryption
    - Functional encryption
    - Special-purpose black box obfuscators
    - Indistinguishability obfuscation

Practicability Challenges of Lattice Crypto

- Complexity
  - Complex algebraic constructions
  - Sophisticated parameter selection (up to a dozen parameters may be needed)
  - Non-intuitive security representation (root Hermite factor)
- Evolving security assumptions
  - Cryptoanalysis efforts change the landscape of theoretically secure schemes (NTRU and DSPR assumptions were “broken” in Feb 2016)
- Encoding challenges for encrypted computing
  - How to efficiently perform operations on non-trivial data types, such as rationals, complex numbers, etc.

Goals of the PALISADE project

- Rapid-prototyping software library for lattice crypto
  - Lattice crypto uses few computational primitives
  - New protocols mix-and-match these primitives
- Develop common crypto APIs that application developers can utilize
  - “Hide” complex details of lattice constructions/parameterization from application developers (for stable schemes)
- Provide a modular structure to mix&match components
  - Ex: System-optimized arithmetic and lattice “backends”/plugins.
  - GPU, FPGA, OpenMP, etc…
- Good software engineering
  - Standards-based design and style
  - Unit tests and benchmarking environment
  - Documentation and sample code

PALISADE Methodology

- We've been working on PALISADE since 2014
  - Next generation of PROCEED SIPHER project
- Project-based development
  - Built using external research project funds.
  - Development is focused on specific areas.
  - Existing project-oriented user and contributor community.
- Prepping for public release
  - Filling in gaps unaddressed by prior projects
  - Smoothing out documentation
  - We're looking for testers!

Yuriy Polyakov polyakov@njit.edu

Motivating Lattice Crypto Projects

- Conjunction obfuscation
  - Gaussian sampling for lattice trapdoors
  - Directed encoding
- Attribute-based encryption
- Public key encryption (PKE) schemes
  - Proxy Re-Encryption and Homomorphic Encryption
- "Systems" activities
  - Multi-precision math, Double-CRT, arbitrary cyclotomics
  - GPU acceleration, OpenMP parallelism
- Applications
  - Publish-Subscribe / Information brokering
  - Circuit execution

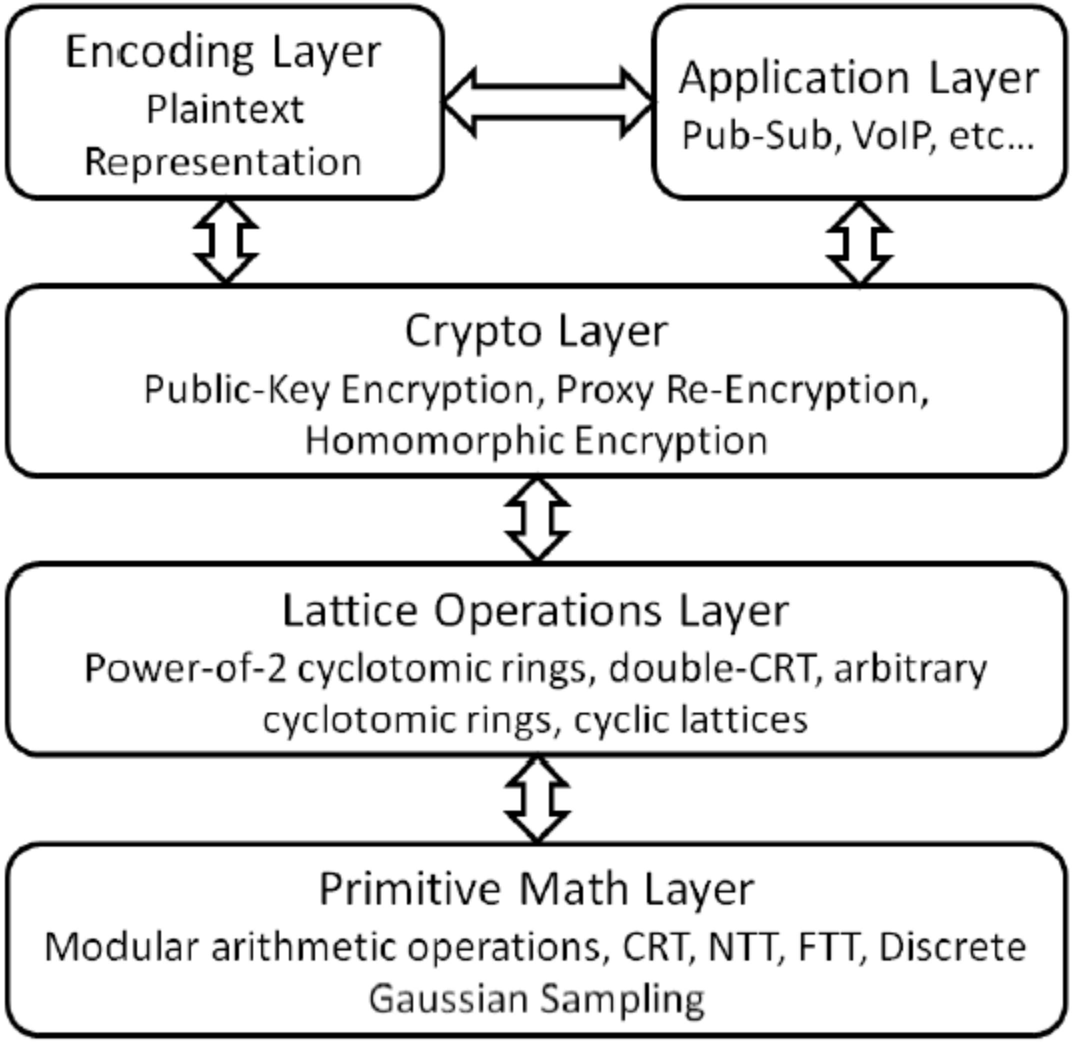
PALISADE Partners and Sponsors

- Partners
  - Academia
    - MIT, UCSD, WPI
  - Industry
    - Raytheon BBN, Vencore Labs / ACS, Lucent Government Systems, Galois
- Sponsors
  - DARPA
  - NSA Center of Excellence
  - Lucent IRAD funds
  - Simons and Sloan foundations

PALISADE License

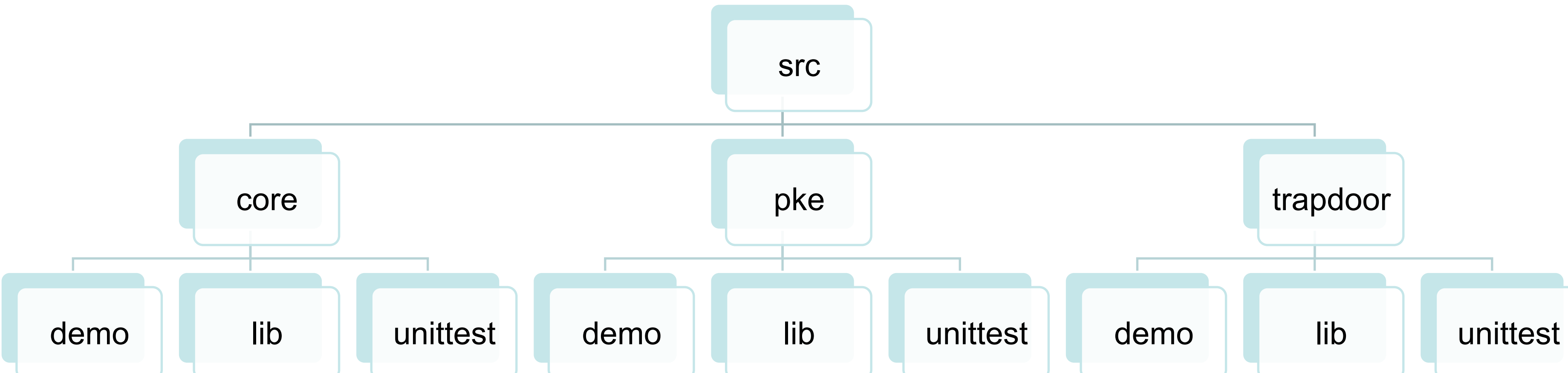
- BSD 2-Clause License
  - Fully approved by our sponsors
  - Generic open-source license
    - Very similar to MIT license
    - We want the license to be industry-friendly
  - We're avoiding GPL on purpose

PALISADE Library: Modular Design



Yuriy Polyakov polyakov@njit.edu

PALISADE Library: Structure



Yuriy Polyakov polyakov@njit.edu

PALISADE Library: Current Capabilities (Alpha release)

- Core module
  - Lattice operations layer
    - Power-of-two cyclotomic rings
    - Double-CRT representation
  - Primitive math layer
    - Number theoretic transform (NTT)/Fermat theoretic transform (FTT)
    - Discrete Fourier Transform (DFT)
    - Gaussian integer sampling
    - Matrices
    - Number theory operations (primality testing, primitive root of unity, etc.)
    - Modular arithmetic backends (only CPU at this time)
      - Multiprecision with static bitwidths for integers
      - Multiprecision with dynamic bitwidths for integers
      - Native 64-bit

PALISADE Library: Current Capabilities (Alpha release)

- PKE module
  - Homomorphic PKE schemes
    - Lopez-Alt – Tromer – Vaikuntanathan (LTV)
    - Stehle – Steinfeld (StSt)
    - Brakerski – Vaikuntanathan (BV) / Brakerski – Gentry – Vaikuntanathan (BGV)
    - Fan – Vercauteren (FV)
  - Proxy re-encryption schemes
    - NTRU-ABD-PRE
    - BV-PRE
  - Encodings
    - Integer array
    - Byte array
    - Packed (batch) encoding

PALISADE Library: Functionality matrix (Alpha release)

Function/Scheme	LTV	StSt	BV/BGV	FV
PKE	X	X	X	X
EvalAdd	X	X	X	X
EvalSub	X	X	X	X
EvalNegate	X	X	X	X
EvalMult	X	Up to two products	X	X
EvalAtIndex (EvalAutomorphism)	X	X		
EvalLinRegression				X
PRE	X	Up to two hops	X	
ModSwitch (Leveled)	X		X	
RingSwitch (Leveled)	X			

Yuriy Polyakov polyakov@njit.edu

PALISADE Library: Technical Specifications

- Supported compilers
  - GCC
    - Linux/Unix
    - Mac OS
    - MingGW
  - Visual C++ (Visual Studio 2015+)
- Multi-threaded support
  - OpenMP 2+ (runs in any GCC environment)
- Availability
  - git.njit.edu (email requests should be sent to rohloff@njit.edu or polyakov@njit.edu)

PALISADE Library: Upcoming features

- Core module
  - Lattice operations layer
    - Arbitrary cyclotomic rings
- Trapdoor module
  - Trapdoor generation & preimage sampling
  - Trapdoor-based protocols
    - GPV digital signature
    - Key-Policy Attribute-Based Encryption
    - Conjunction Obfuscator

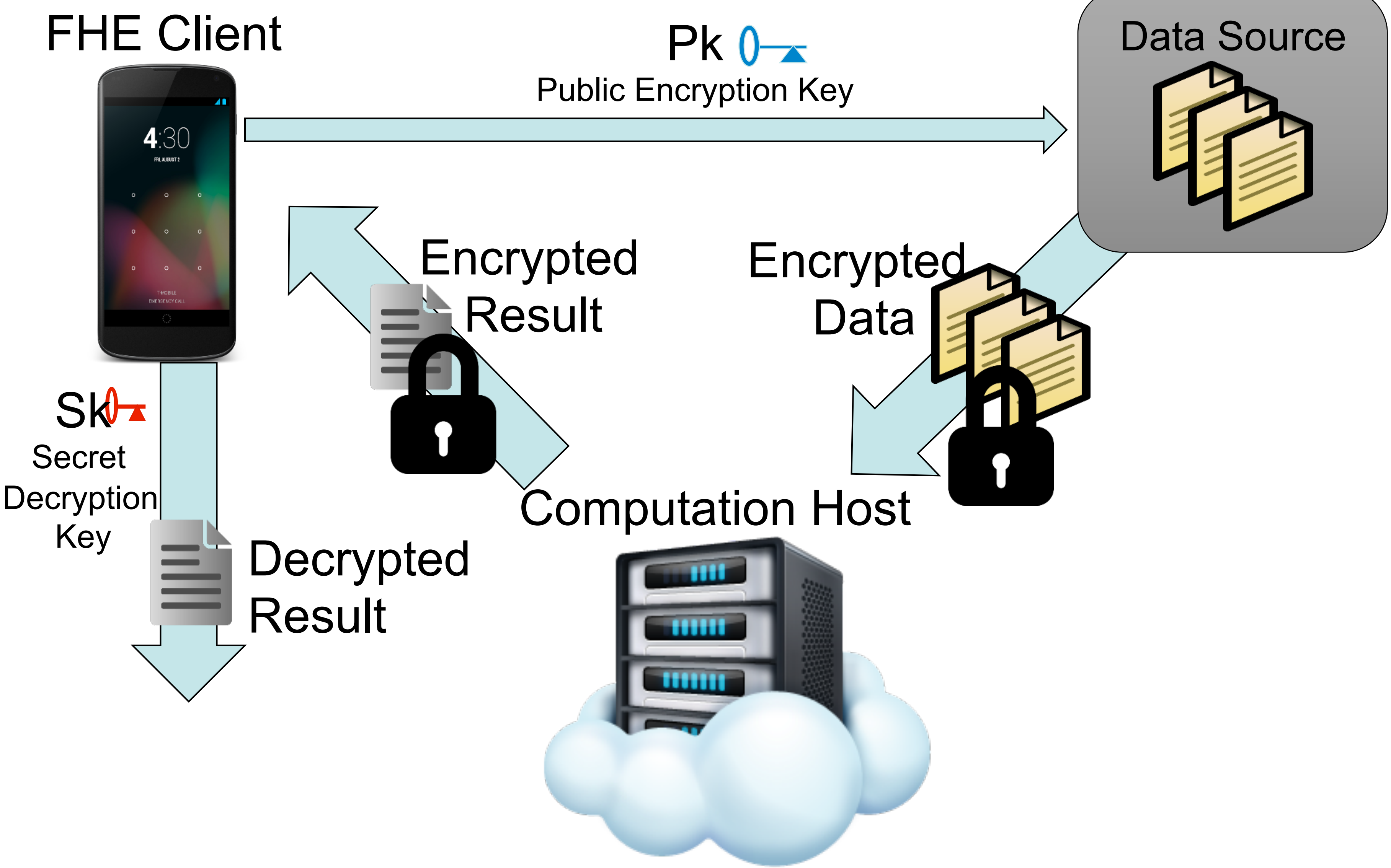
How To: PKE (based on FV)

```
int relWindow = 1;
int plaintextModulus = 64;
double sigma = SIGMA;
double alpha = 9;
double rootHermiteFactor = 1.006;

//Set crypto parametes
CryptoContext<ILVector2n> cc = CryptoContextFactory<ILVector2n>::genCryptoContextFV(
        plaintextModulus, 0, "0", "0",
        relWindow, sigma, "0",
        OPTIMIZED, "0", "0", 0, alpha, rootHermiteFactor);

//Turn on features
cc.Enable(ENCRYPTION);

//Run automated parameter generation
cc.GetEncryptionAlgorithm()->ParamsGen(cc.GetCryptoParameters(), 0, 1);

// Initialize the public key containers.
LPKeyPair<ILVector2n> kp;

////////////////////////////////////////////////////////////
//Perform the key generation operation.
////////////////////////////////////////////////////////////

kp = cc.KeyGen();

if (!kp.good()) {
    std::cout << "Key generation failed!" << std::endl;
    exit(1);
}
```

How To: PKE (based on FV)

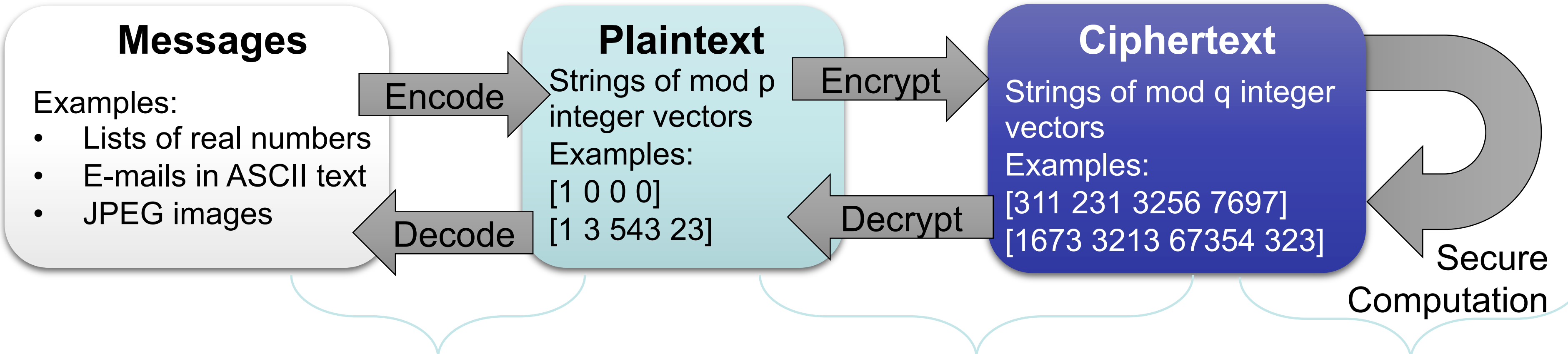
```
std::vector<uint32_t> vectorOfInts1 = { 1,0,3,1,0,1,2,1 };
IntPlaintextEncoding plaintext1(vectorOfInts1);

////////////////////////////////////////////////////////////////
//Encryption
////////////////////////////////////////////////////////////////

vector<shared_ptr<Ciphertext<ILVector2n>>> ciphertext1;

ciphertext1 = cc.Encrypt(kp.publicKey, plaintext1, false);


////////////////////////////////////////////////////////////////
//Decryption
////////////////////////////////////////////////////////////////

DecryptResult result = cc.Decrypt(kp.secretKey, ciphertext1, &plaintext1, true);
```

Fully Homomorphic Encryption



Yuriy Polyakov polyakov@njit.edu

Using FHE for Computation



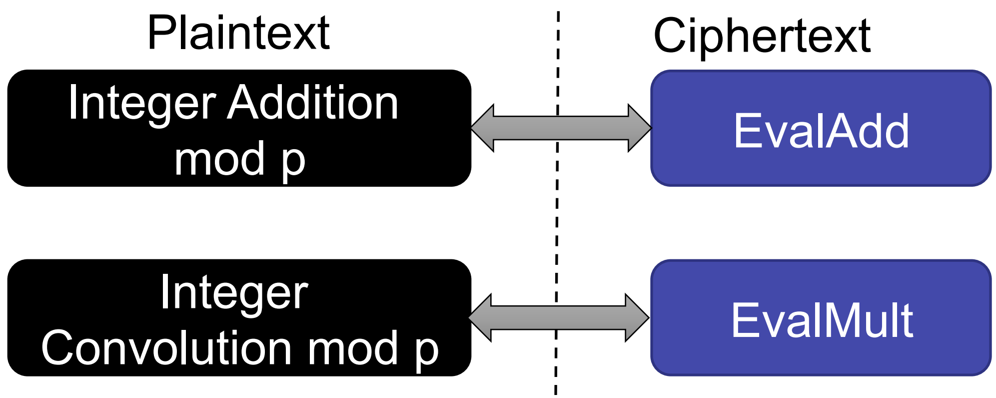
- Message-Plaintext encodings determined by translation of program into EvalAdd, EvalMult operations.
- Coding is an open research topic and drastically impacts effective runtime.

- Plaintext-Ciphertext encryption/decryption defined by FHE scheme.

- EvalAdd and EvalMult operations on ciphertexts

Secure Programs with FHE

- Very different computation model.
- Base operations on ciphertext:



- Conditional “if” statements on encrypted data not permitted.
- Special case: EvalAdd and EvalMult correspond to bitwise XOR and AND.

How To: SHE/FHE

```
int relWindow = 1;
int plaintextModulus = 64;
double sigma = SIGMA;
double alpha = 9;
double rootHermiteFactor = 1.006;

//Set crypto parametes
CryptoContext<ILVector2n> cc = CryptoContextFactory<ILVector2n>::genCryptoContextFV(
    plaintextModulus, 0, "0", "0",
    relWindow, sigma, "0",
    OPTIMIZED, "0", "0", 0, alpha, rootHermiteFactor);

//Turn on features
cc.Enable(ENCRYPTION);
cc.Enable(SHE);

//Automated parameter generation
cc.GetEncryptionAlgorithm()->ParamsGen(cc.GetCryptoParameters(), 0, 1);

// Initialize the public key containers.
LPKeyPair<ILVector2n> kp;

////////////////////////////////////////////////////////////////
//Perform the key generation operation.
////////////////////////////////////////////////////////////////

kp = cc.KeyGen();

if (!kp.good()) {
    std::cout << "Key generation failed!" << std::endl;
    exit(1);
}
```

How To: SHE/FHE

```
std::vector<uint32_t> vectorOfInts1 = { 1,0,3,1,0,1,2,1 };
IntPlaintextEncoding plaintext1(vectorOfInts1);

std::vector<uint32_t> vectorOfInts2 = { 2,1,3,2,2,1,3,0 };
IntPlaintextEncoding plaintext2(vectorOfInts2);

////////////////////////////////////////////////////////////
//Encryption
////////////////////////////////////////////////////////////

vector<shared_ptr<Ciphertext<ILVector2n>>> ciphertext1;
vector<shared_ptr<Ciphertext<ILVector2n>>> ciphertext2;

ciphertext1 = cc.Encrypt(kp.publicKey, plaintext1, false);
ciphertext2 = cc.Encrypt(kp.publicKey, plaintext2, false);

////////////////////////////////////////////////////////////
//EvalAdd Operation
////////////////////////////////////////////////////////////

vector<shared_ptr<Ciphertext<ILVector2n>>> ciphertextAdd;

shared_ptr<Ciphertext<ILVector2n>> ciphertextTemp;

ciphertextTemp = cc.EvalAdd(ciphertext1[0], ciphertext2[0]);

ciphertextAdd.push_back(ciphertextTemp);
```

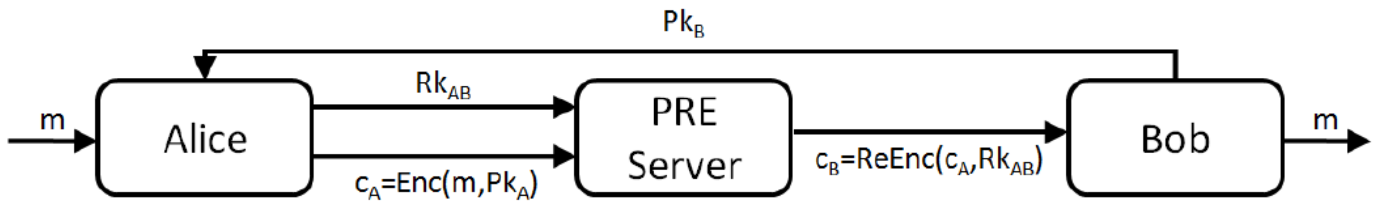
Yuriy Polyakov polyakov@njit.edu

How To: SHE/FHE

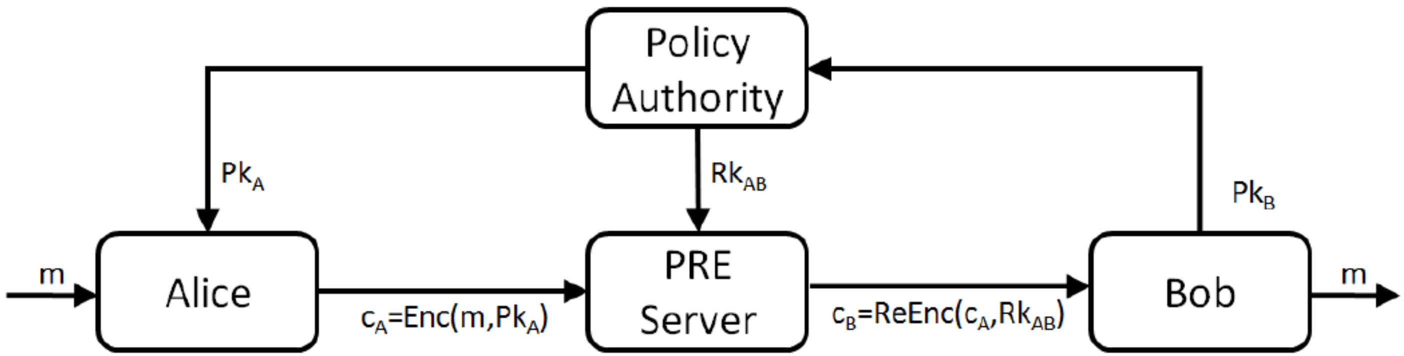
```
////////////////////////////////////////////////////////////
//Evaluation Key Generation
////////////////////////////////////////////////////////////

cc.EvalMultKeyGen(kp.secretKey);

////////////////////////////////////////////////////////////
//EvalMult Operation
////////////////////////////////////////////////////////////

vector<shared_ptr<Ciphertext<ILVector2n>>> ciphertextMult;

shared_ptr<Ciphertext<ILVector2n>> ciphertextTempMult;

ciphertextTempMult = cc.EvalMult(ciphertext1[0], ciphertext2[0]);

ciphertextMult.push_back(ciphertextTempMult);

IntPlaintextEncoding plaintextNewMult;

////////////////////////////////////////////////////////////
//Decryption after EvalMult Operation
////////////////////////////////////////////////////////////

result = cc.Decrypt(kp.secretKey, ciphertextMult, &plaintextNewMult, true);
```

Proxy Re-Encryption (PRE)

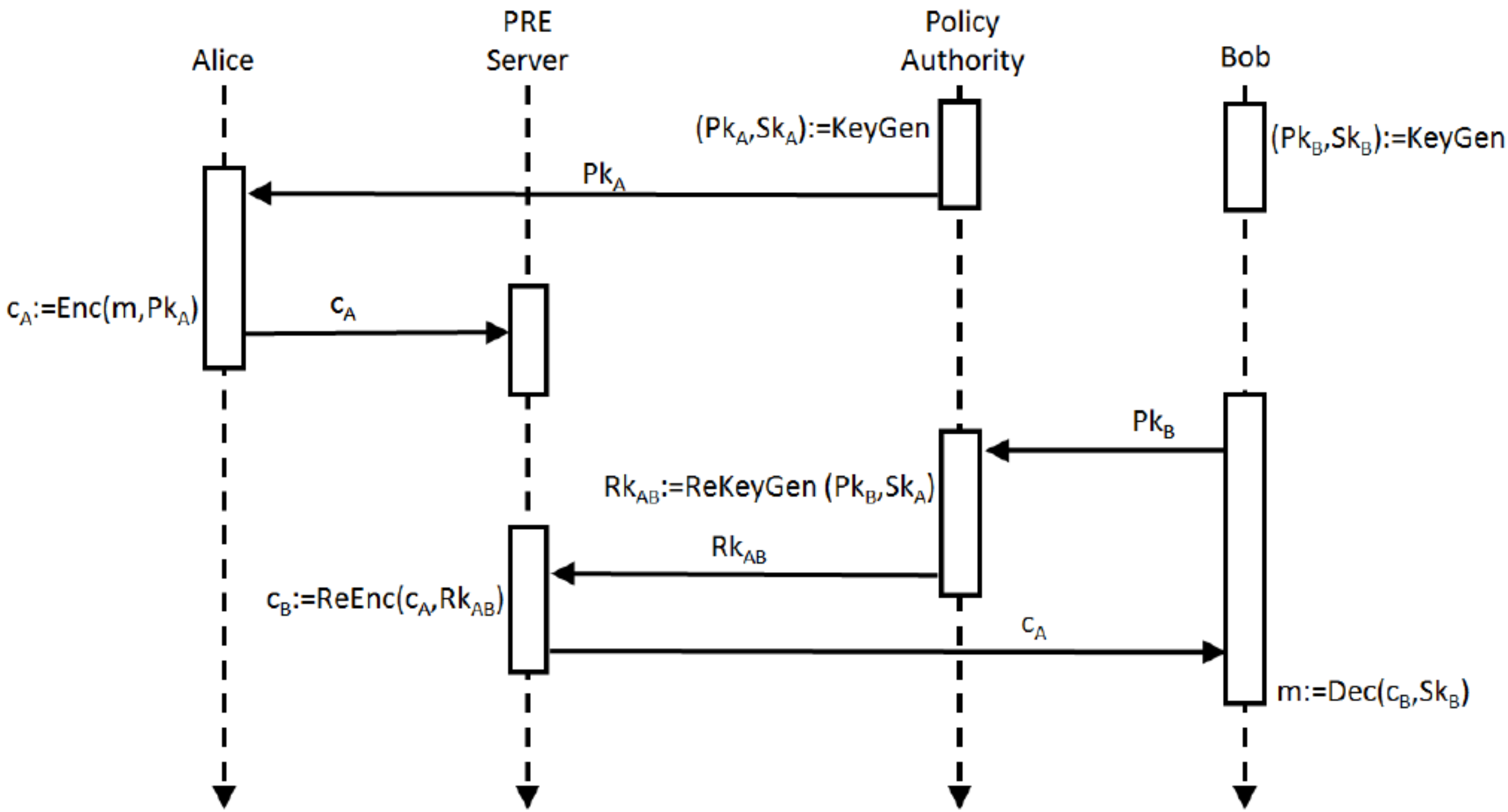
Classical non-interactive PRE model



Pub/Sub PRE model



Proxy Re-Encryption (PRE)



Yuriy Polyakov polyakov@njit.edu

How To: PRE

```
//setting parameters manually
double sigma = SIGMA;

usint m = 2048;
BigBinaryInteger q("268441601");
BigBinaryInteger rootOfUnity("16947867");

CryptoContext<ILVector2n> cc = CryptoContextFactory<ILVector2n>::genCryptoContextLTV(2, m, q.ToString(),
    rootOfUnity.ToString(), 1, sigma);

//turn on features
cc.Enable(ENCRYPTION);
cc.Enable(PRE);
cc.Enable(SHE);

////////////////////////////////////////////////////////////
//Perform the key generation operations
////////////////////////////////////////////////////////////

// Initialize the key containers.
LPKeyPair<ILVector2n> kp = cc.KeyGen();

if (!kp.good()) {
    std::cout << "Key generation failed!" << std::endl;
    exit(1);
}
```

How To: PRE

```
/////////////////////////////////////////////////////////////
//Perform the second key generation operation.
// This generates the keys used to decrypt the ciphertext after the re-encryption operation.
/////////////////////////////////////////////////////////////

LPKeyPair<Element> newKp = cc.KeyGen();

/////////////////////////////////////////////////////////////
//Perform the proxy re-encryption key generation operation.
// This generates the keys which are used to perform the key switching.
/////////////////////////////////////////////////////////////

shared_ptr<LPEvalKey<Element>> evalKey = cc.ReKeyGen( newKp.publicKey, kp.secretKey );

// Encryption
vector<shared_ptr<Ciphertext<Element>>> ciphertext = cc.Encrypt(kp.publicKey, plaintextShort, true);
BytePlaintextEncoding plaintextShortNew;

// Re-encryption
vector<shared_ptr<Ciphertext<Element>>> reCiphertext = cc.ReEncrypt(evalKey, ciphertext);

// Decryption
DecryptResult result = cc.Decrypt(newKp.secretKey, reCiphertext, &plaintextShortNew, true);
```

Request for feedback

- What capabilities (cryptographic protocols/lattice primitives) should be added?
- What applications PALISADE could be useful for?

Questions?

Yuriy Polyakov

polyakov@njit.edu

NJIT

THE EDGE IN KNOWLEDGE