# OpenFHE-"avances"

Compilar sin OpenMP - single thread.

```
29.01%  ckks  libOPENFHEpke.so.0.9.4   [.] intnat::ChineseRemainderTransformFTTNat<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::ForwardTransformToBi
 9.90%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::NativeVectorT
 8.84%  ckks  libOPENFHEcore.so.0.9.4  [.] blake2b_compress
 8.68%  ckks  libOPENFHEcore.so.0.9.4  [.] lbcrypto::DCRTPolyImpl<bigintdyn::mubintvec<bigintdyn::ubint<unsigned int> > >::ApproxSwitchCRTBasis
 6.96%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMulEq
 4.90%  ckks  libOPENFHEpke.so.0.9.4   [.] intnat::NumberTheoreticTransformNat<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::InverseTransformFromBitR
 4.31%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModAddEq
 2.71%  ckks  libOPENFHEcore.so.0.9.4  [.] lbcrypto::DiscreteUniformGeneratorImpl<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::GenerateInteger
 2.36%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMul
 1.98%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModSubEq
 1.95%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::SwitchModulus
 1.52%  ckks  libc.so.6                [.] 0x000000000015430a
 1.52%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMulEq
 1.12%  ckks  libOPENFHEpke.so.0.9.4   [.] lbcrypto::PseudoRandomNumberGenerator::GetPRNG
 1.03%  ckks  libOPENFHEpke.so.0.9.4   [.] lbcrypto::FHECKKSRNS::FitToNativeVector
 0.90%  ckks  [kernel.vmlinux]         [k] _raw_spin_lock
 0.72%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::operator=
 0.50%  ckks  libOPENFHEcore.so.0.9.4  [.] blake2b_init_param
 0.43%  ckks  [kernel.vmlinux]         [k] sync_regs
```

# OpenFHE-”avances”

Compilar sin OpenMP - single thread.

```
29.01%  ckks     libOPENFHEpke.so.0.9.4   [.] intnat::ChineseRemainderTransformFTTNat<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::ForwardTransformToBi
 9.90%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::NativeVectorT
 8.84%  ckks     libOPENFHEcore.so.0.9.4  [.] blake2b_compress
 8.68%  ckks     libOPENFHEcore.so.0.9.4  [.] lbcrypto::DCRTPolyImpl<bigintdyn::mubintvec<bigintdyn::ubint<unsigned int> > >::ApproxSwitchCRTBasis
 6.96%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMulEq
 4.90%  ckks     libOPENFHEpke.so.0.9.4   [.] intnat::NumberTheoreticTransformNat<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::InverseTransformFromBitR
 4.31%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModAddEq
 2.71%  ckks     libOPENFHEcore.so.0.9.4  [.] lbcrypto::DiscreteUniformGeneratorImpl<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::GenerateInteger
 2.36%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMul
 1.98%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModSubEq
 1.95%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::SwitchModulus
 1.52%  ckks     libc.so.6                [.] 0x000000000015430a
 1.52%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMulEq
 1.12%  ckks     libOPENFHEpke.so.0.9.4   [.] lbcrypto::PseudoRandomNumberGenerator::GetPRNG
 1.03%  ckks     libOPENFHEpke.so.0.9.4   [.] lbcrypto::FHECKKSRNS::FitToNativeVector
 0.90%  ckks     [kernel.vmlinux]         [k] _raw_spin_lock
 0.72%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::operator=
 0.50%  ckks     libOPENFHEcore.so.0.9.4  [.] blake2b_init_param
 0.43%  ckks     [kernel.vmlinux]         [k] sync_regs
```

Compilar ejemplo sin cmake complejo.

Visualizador de perf: hotspot y flamegraph (sin probar todavía)

# OpenFHE-"avances"

Compilar sin OpenMP - single thread.

```
29.01%  ckks     libOPENFHEpke.so.0.9.4   [.] intnat::ChineseRemainderTransformFTTNat<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::ForwardTransformToBi
 9.90%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::NativeVectorT
 8.84%  ckks     libOPENFHEcore.so.0.9.4  [.] blake2b_compress
 8.68%  ckks     libOPENFHEcore.so.0.9.4  [.] lbcrypto::DCRTPolyImpl<bigintdyn::mubintvec<bigintdyn::ubint<unsigned int> > >::ApproxSwitchCRTBasis
 6.96%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMulEq
 4.90%  ckks     libOPENFHEpke.so.0.9.4   [.] intnat::NumberTheoreticTransformNat<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::InverseTransformFromBitR
 4.31%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModAddEq
 2.71%  ckks     libOPENFHEcore.so.0.9.4  [.] lbcrypto::DiscreteUniformGeneratorImpl<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::GenerateInteger
 2.36%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMul
 1.98%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModSubEq
 1.95%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::SwitchModulus
 1.52%  ckks     libc.so.6                [.] 0x000000000015430a
 1.52%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMulEq
 1.12%  ckks     libOPENFHEpke.so.0.9.4   [.] lbcrypto::PseudoRandomNumberGenerator::GetPRNG
 1.03%  ckks     libOPENFHEpke.so.0.9.4   [.] lbcrypto::FHECKKSRNS::FitToNativeVector
 0.90%  ckks     [kernel.vmlinux]         [k] _raw_spin_lock
 0.72%  ckks     libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::operator=
 0.50%  ckks     libOPENFHEcore.so.0.9.4  [.] blake2b_init_param
 0.43%  ckks     [kernel.vmlinux]         [k] sync_regs
```

Compilar ejemplo sin cmake complejo.

Visualizador de perf: hotspot y flamegraph (sin probar todavía)

Entender "mejor" CKKS, codificación y decodificación.

# OpenFHE-"avances"

Compilar sin OpenMP - single thread.

```
29.01%  ckks  libOPENFHEpke.so.0.9.4   [.] intnat::ChineseRemainderTransformFTTNat<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::ForwardTransformToBi
 9.90%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::NativeVectorT
 8.84%  ckks  libOPENFHEcore.so.0.9.4  [.] blake2b_compress
 8.68%  ckks  libOPENFHEcore.so.0.9.4  [.] lbcrypto::DCRTPolyImpl<bigintdyn::mubintvec<bigintdyn::ubint<unsigned int> > >::ApproxSwitchCRTBasis
 6.96%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMulEq
 4.90%  ckks  libOPENFHEpke.so.0.9.4   [.] intnat::NumberTheoreticTransformNat<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::InverseTransformFromBitR
 4.31%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModAddEq
 2.71%  ckks  libOPENFHEcore.so.0.9.4  [.] lbcrypto::DiscreteUniformGeneratorImpl<intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> > >::GenerateInteger
 2.36%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMul
 1.98%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModSubEq
 1.95%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::SwitchModulus
 1.52%  ckks  libc.so.6                [.] 0x000000000015430a
 1.52%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::ModMulEq
 1.12%  ckks  libOPENFHEpke.so.0.9.4   [.] lbcrypto::PseudoRandomNumberGenerator::GetPRNG
 1.03%  ckks  libOPENFHEpke.so.0.9.4   [.] lbcrypto::FHECKKSRNS::FitToNativeVector
 0.90%  ckks  [kernel.vmlinux]         [k] _raw_spin_lock
 0.72%  ckks  libOPENFHEcore.so.0.9.4  [.] intnat::NativeVectorT<intnat::NativeIntegerT<unsigned long> >::operator=
 0.50%  ckks  libOPENFHEcore.so.0.9.4  [.] blake2b_init_param
 0.43%  ckks  [kernel.vmlinux]         [k] sync_regs
```

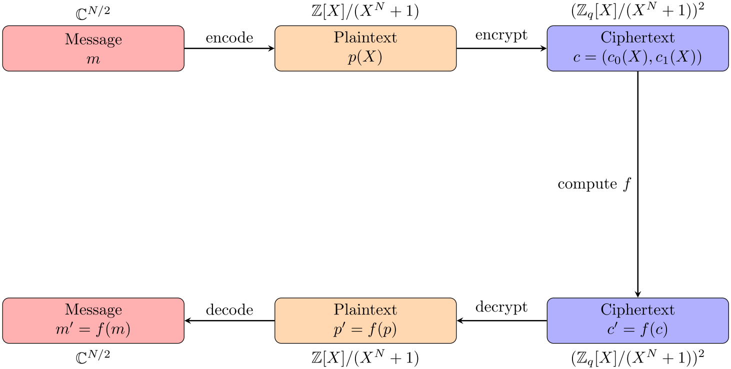
Compilar ejemplo sin cmake complejo.

Visualizador de perf: hotspot y flamegraph (sin probar todavía)

Entender "mejor" CKKS, codificación y decodificación.

(Avances de curso Onur y programación en C no los considero.)

# CKKS


$\mathbb{C}^{N/2}$
$\mathbb{Z}[X]/(X^N+1)$
$(\mathbb{Z}_q[X]/(X^N+1))^2$
Message
$m$
encode
Plaintext
$p(X)$
encrypt
Ciphertext
$c=(c_0(X),c_1(X))$
compute $f$
Message
$m'=f(m)$
decode
Plaintext
$p'=f(p)$
decrypt
Ciphertext
$c'=f(c)$
$\mathbb{C}^{N/2}$
$\mathbb{Z}[X]/(X^N+1)$
$(\mathbb{Z}_q[X]/(X^N+1))^2$

# CKKS encoder/decoder

CKKS usa las estructuras de anillos de polinomios enteros.

Sea $z \in \mathbb{C}^{N/2}$, codificarlo en un polinomio $m(X) \in \mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$

Con modulo $N$, múltiplo de 2.

# CKKS encoder/decoder

CKKS usa las estructuras de anillos de polinomios enteros.

Sea $z \in \mathbb{C}^{N/2}$, codificarlo en un polinomio $m(X) \in \mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$

Con modulo $N$, múltiplo de 2.

Se define $M = 2N$.

Se usa el **encaje canónico** (`canonical embeddign`) $\sigma$: nos da isomorfismo, entre vectores y polinomios.

# CKKS encoder/decoder

CKKS usa las estructuras de anillos de polinomios enteros.

Sea $z \in \mathbb{C}^{N/2}$, codificarlo en un polinomio $m(X) \in \mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$

Con modulo $N$, múltiplo de 2.

Se define $M = 2N$.

Se usa el **encaje canónico** (`canonical embeddign`) $\sigma$: nos da isomorfismo, entre vectores y polinomios.

En particular, el m-esimo **polinomio cíclico** $\Phi_M(X) = X^N + 1$.

Sus **raíces** de la unidad: $\xi_M = (e^{2i\pi/M})^k$, $k \in \mathbb{Z} < M$ y coprimos de $M$.

# CKKS encoder/decoder simplificado

La decodificación es sencilla.

Dado $m(X)$, se evalua el polinomio en las raices.

# CKKS encoder/decoder simplificado

La decodificación es sencilla.

Dado $m(X)$, se evalua el polinomio en las raices.

Es decir $\sigma(m) = (m(\xi), m(\xi^3), ..., m(\xi^{2N-1})) = (z_1, z_2, ..., z_N)$. Más algunos detalles (en breve).

**Complejidad**: como calcular $\sigma^{-1}$. Como codificar el vector $z$.

# CKKS encoder/decoder simplificado

La decodificación es sencilla.

Dado $m(X)$, se evalua el polinomio en las raices.

Es decir $\sigma(m) = (m(\xi), m(\xi^3), ..., m(\xi^{2N-1})) = (z_1, z_2, ..., z_N)$. Más algunos detalles (en breve).

**Complejidad**: como calcular $\sigma^{-1}$. Como codificar el vector $z$.

La forma "facil": $\sigma : \mathbb{C}[X]/(X^N + 1) \to \mathbb{C}^N$

$m(X) = \sum_{j=0}^{N-1} \alpha_j X^j \to z_j = \sum_{j=0}^{N-1} \alpha_j (\xi^{2i-1})^j$

# CKKS encoder/decoder simplificado

La decodificación es sencilla.

Dado $m(X)$, se evalua el polinomio en las raices.

Es decir $\sigma(m) = (m(\xi), m(\xi^3), ..., m(\xi^{2N-1})) = (z_1, z_2, ..., z_N)$. Más algunos detalles (en breve).

**Complejidad**: como calcular $\sigma^{-1}$. Como codificar el vector $z$.

La forma "facil": $\sigma : \mathbb{C}[X]/(X^N + 1) \to \mathbb{C}^N$

$m(X) = \sum_{j=0}^{N-1} \alpha_j X^j \to z_j = \sum_{j=0}^{N-1} \alpha_j (\xi^{2i-1})^j$

Sistema lineal $A\alpha = z$ con $A$ una matriz de Vandermonde de $\xi^{2j-1}$, $\alpha$ vector de los coeficientes del polinomio.

Entonces: $\alpha = A^{-1}z \to \sigma^{-1}(z) = \sum_{j=0}^{N-1} \alpha_j X^j$

# CKKS encoder

Resolvimos esto al revés $\mathbb{C}^N \rightarrow \mathbb{C}[X]/(X^N + 1)$ queremos $\mathbb{C}^{N/2} \rightarrow \mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$.

Es decir $m(X)$ tiene que tener coeficientes enteros.

# CKKS encoder

Resolvimos esto al revés $\mathbb{C}^N \to \mathbb{C}[X]/(X^N+1)$ queremos $\mathbb{C}^{N/2} \to \mathcal{R} = \mathbb{Z}[X]/(X^N+1)$.

Es decir $m(X)$ tiene que tener coeficientes enteros.

Con esto, al evaluarlo con raíces complejas, donde la mitad de estas son complejas conjugadas de las otras, tenemos:

$\sigma(\mathcal{R}) \to z_j = \overline{z_{-j}} \to m(\xi^j) = m(\overline{\xi^{-j}})$.

# CKKS encoder

Resolvimos esto al revés $\mathbb{C}^N \to \mathbb{C}[X]/(X^N+1)$ queremos $\mathbb{C}^{N/2} \to \mathcal{R} = \mathbb{Z}[X]/(X^N+1)$.

Es decir $m(X)$ tiene que tener coeficientes enteros.

Con esto, al evaluarlo con raíces complejas, donde la mitad de estas son complejas conjugadas de las otras, tenemos:

$\sigma(\mathcal{R}) \to z_j = \overline{z_{-j}} \to m(\xi^j) = m(\overline{\xi^{-j}})$.

Entonces usamos vectores $z \in \mathbb{C}^{N/2}$ y lo extendemos con sus conjungados (paper $\pi^{-1}$).

# CKKS encoder

Resolvimos esto al revés $\mathbb{C}^N \to \mathbb{C}[X]/(X^N+1)$ queremos $\mathbb{C}^{N/2} \to \mathcal{R} = \mathbb{Z}[X]/(X^N+1)$.

Es decir $m(X)$ tiene que tener coeficientes enteros.

Con esto, al evaluarlo con raíces complejas, donde la mitad de estas son complejas conjugadas de las otras, tenemos:

$\sigma(\mathcal{R}) \to z_j = \overline{z_{-j}} \to m(\xi^j) = m(\overline{\xi^{-j}})$.

Entonces usamos vectores $z \in \mathbb{C}^{N/2}$ y lo extendemos con sus conjungados (paper $\pi^{-1}$).

Para mantener el isomorfismo de $\sigma(\mathcal{R}) \to \mathcal{R}$ (recordemos que están en diferentes espacios ahora) tenemos que proyectar $\pi^{-1}$ a $\sigma(\mathcal{R})$ de alguna forma.

# CKKS encoder

Como $\mathcal{R}$ tiene una base $\mathbb{Z}$ ortogonal $1, X, ..., X^{N-1}$ y que $\sigma$ es un isomorfismo, entonces $\sigma(\mathcal{R})$ también tiene una.

Sea $\beta = (b_1, ..., b_N) = (\sigma(1), \sigma(X), ..., \sigma(X^{N-1}))$.

# CKKS encoder

Como $\mathcal{R}$ tiene una base $\mathbb{Z}$ ortogonal $1, X, ..., X^{N-1}$ y que $\sigma$ es un isomorfismo, entonces $\sigma(\mathcal{R})$ también tiene una.

Sea $\beta = (b_1, ..., b_N) = (\sigma(1), \sigma(X), ..., \sigma(X^{N-1}))$.

Entonces para cualquier $Z = \pi^{-1}(z) \in \mathbb{C}^N$ (rec:$z \in \mathbb{C}^{N/2}$ ) lo proyecto en $\beta$.

$$Z = \sum_{i=1}^{N} \frac{< Z, b_i >}{||b_i||^2} b_i$$

# CKKS encoder

Como $\mathcal{R}$ tiene una base $\mathbb{Z}$ ortogonal $1, X, ..., X^{N-1}$ y que $\sigma$ es un isomorfismo, entonces $\sigma(\mathcal{R})$ también tiene una.

Sea $\beta = (b_1, ..., b_N) = (\sigma(1), \sigma(X), ..., \sigma(X^{N-1}))$.

Entonces para cualquier $Z = \pi^{-1}(z) \in \mathbb{C}^N$ (rec:$z \in \mathbb{C}^{N/2}$ ) lo proyecto en $\beta$.

$$Z = \sum_{i=1}^{N} \frac{< Z, b_i >}{||b_i||^2} b_i$$

Por ultimo usamos el algoritmo de `coordinate-wise random rounding` para redondear los $Z_i$ reales en enteros. (Redondea x a $\lfloor x \rfloor$ o $\lfloor x \rfloor + 1$ dando más proba mientras x este más cerca de alguno.)

Tendremos un polinomio $m \in \sigma(\mathcal{R})$, ya que tiene coeficientes enteros en la base $\beta$.

# CKKS encoder

Con esto simplemente aplicamos $\sigma^{-1}$ que nos dará la codificación (un elemento de $\mathcal{R}$).

Como último detalle, para evitar que el redondeo elimine algunos números significativos, se multiplica a $z$ por un factor $\triangle > 0$ al principio de la codificación.

# CKKS encoder

Con esto simplemente aplicamos $\sigma^{-1}$ que nos dará la codificación (un elemento de $\mathcal{R}$).

Como último detalle, para evitar que el redondeo elimine algunos números significativos, se multiplica a $z$ por un factor $\triangle > 0$ al principio de la codificación.

Para decodificar simplemente se divide por el mismo factor.

Es decir para codificar simplemente del texto plano $m(X)$ obtenemos $z = \pi \circ \sigma(\triangle^{-1}.m)$

# Próxima reunion (propuesta)

- Cerrar algunas ideas de la codificación.
- Comparar implementación codificación OpenFHE con lo explicado hoy.

# Próxima reunion (propuesta)

- Cerrar algunas ideas de la codificación.
- Comparar implementación codificación OpenFHE con lo explicado hoy.
- Avanzar con encriptación de CKKS.
- Visualizador perf: navegar dentro del código.