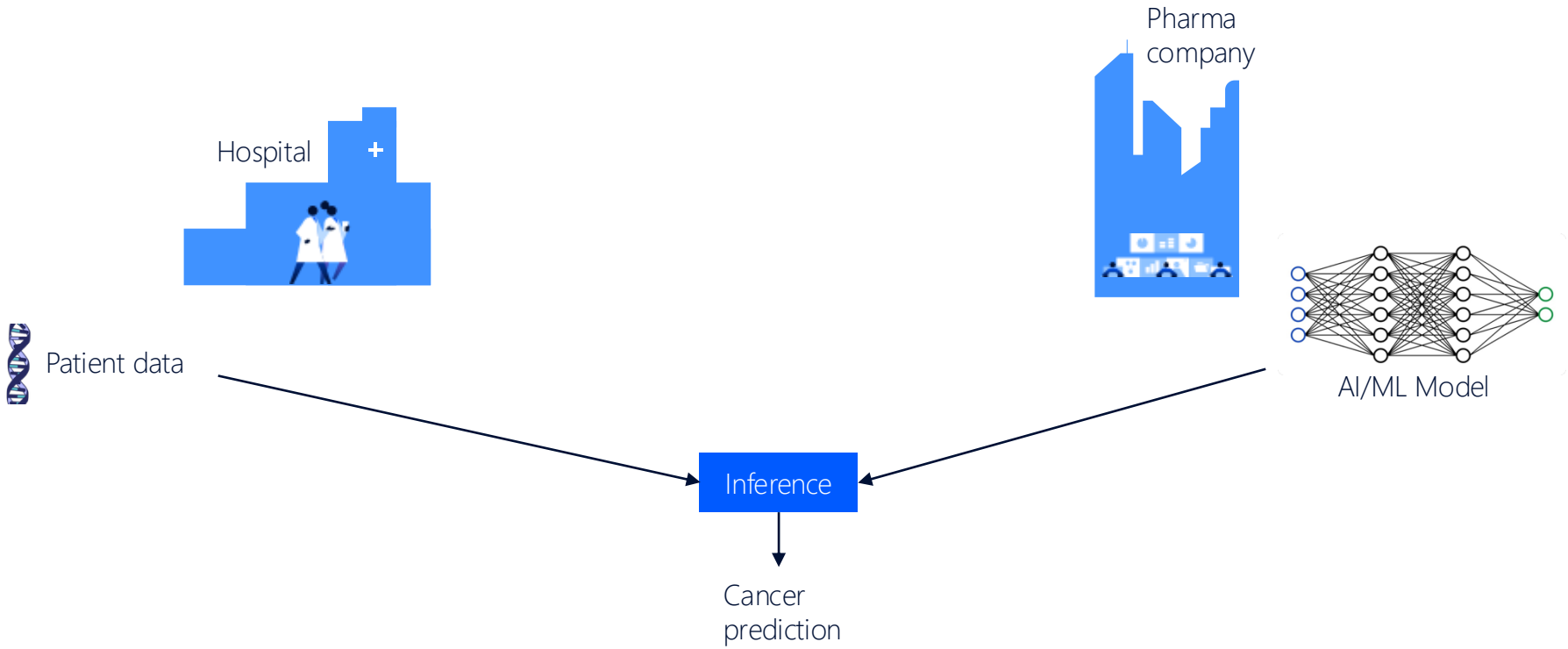
FHE: From Theory to Practice

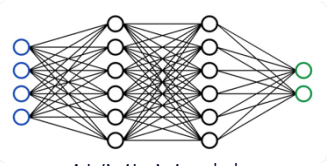
Emad Heydari Beni

Nokia Bell Labs

COSIC - KU Leuven


Hospital
Pharma company
Patient data
AI/ML Model
Inference
Cancer prediction

Patient data



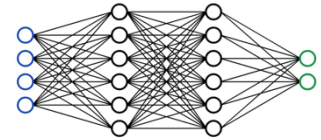
AI/ML Model

Inference

Cancer
prediction

Patient data is sensitive!

Patient data



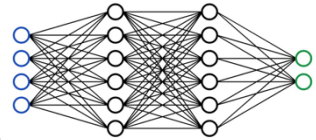
AI/ML Model

Inference

Cancer
prediction

Proprietary ML model
Business sensitive

Patient data



AI/ML Model

Scramble

Desensitized patient data

Inference

Cancer
prediction

Prone to mistakes
impact on prediction quality

Homomorphic Encryption

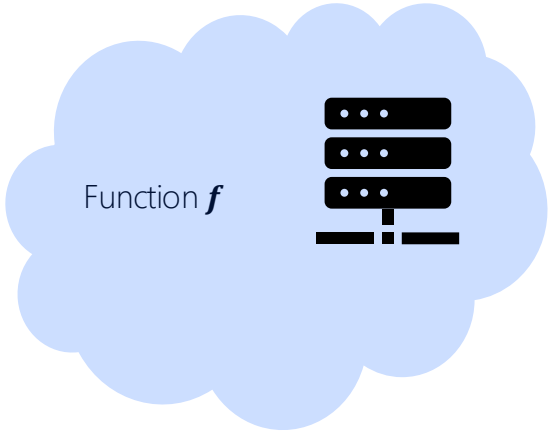
Hospital

Value x

Enc (x)

y

$Dec\,(y) = f(x)$



$x_1, \ldots, x_n$ → $f(x_1, \ldots, x_n)$

Encrypt ↓

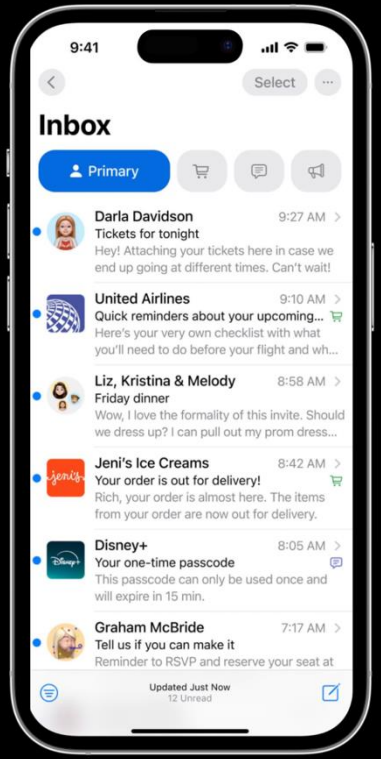
↑ Decrypt

$Enc(x_1), \ldots, Enc(x_n)$ → $Eval_{evk}\,(f, Enc(x_1), \ldots, Enc(x_n))$

Homomorphic Encryption across Apple features

Caller ID

Enhanced Visual Search



Business Services in Mail

Microsoft Edge: Password Monitor (built on FHE)

Edge

pwd

Magic protocol
(OPRF)

Secret key

Microsoft

Breached DBs

Hash(pwd)

h

C ⟵ Homomorphic.**Enc** (h)

C

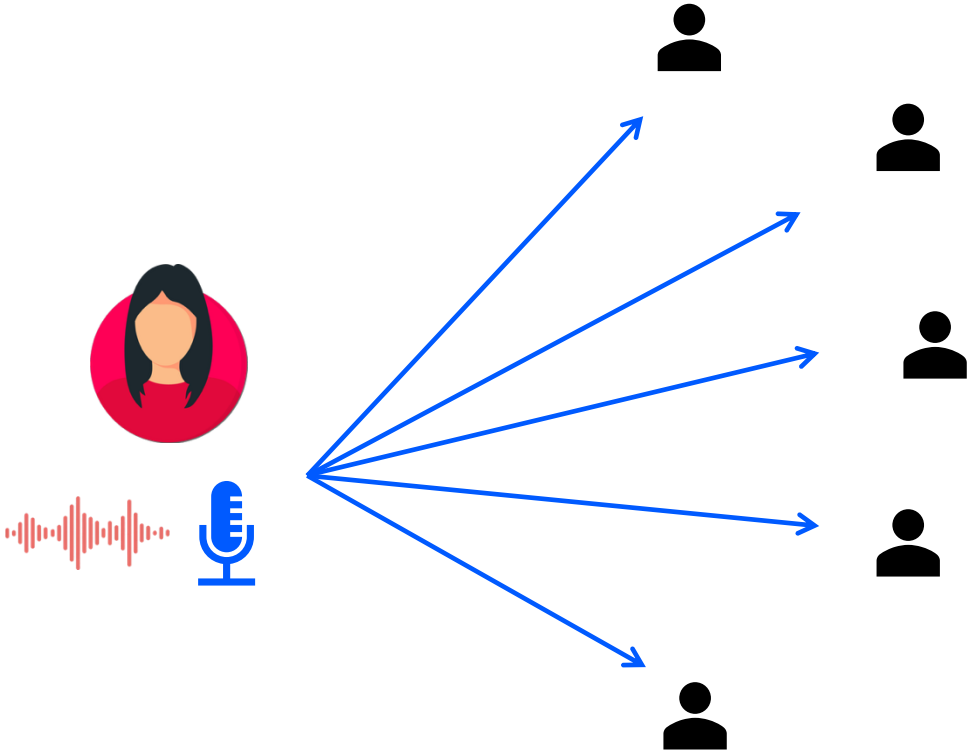
Homomorphicly:

C′ ⟵ computeMatch (C , Breached DBs)

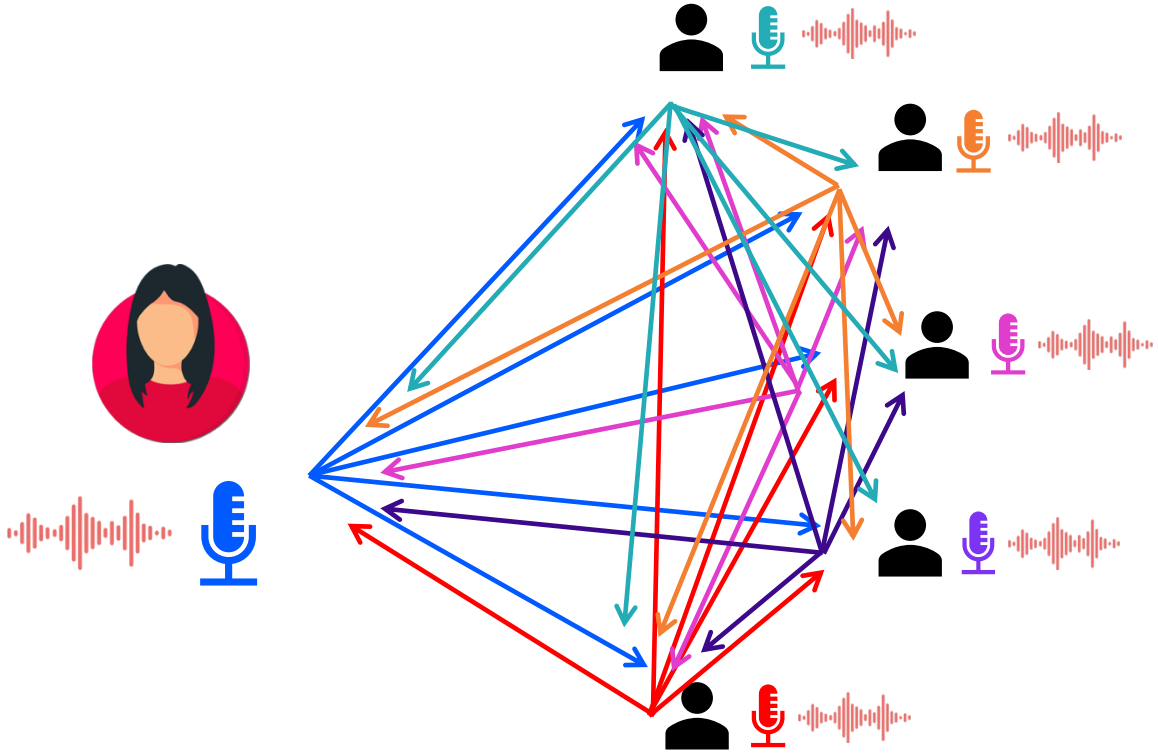
C′ = Enc (Result)

Result ⟵ Homomorphic.**Dec** (C′)

Audio Mixing

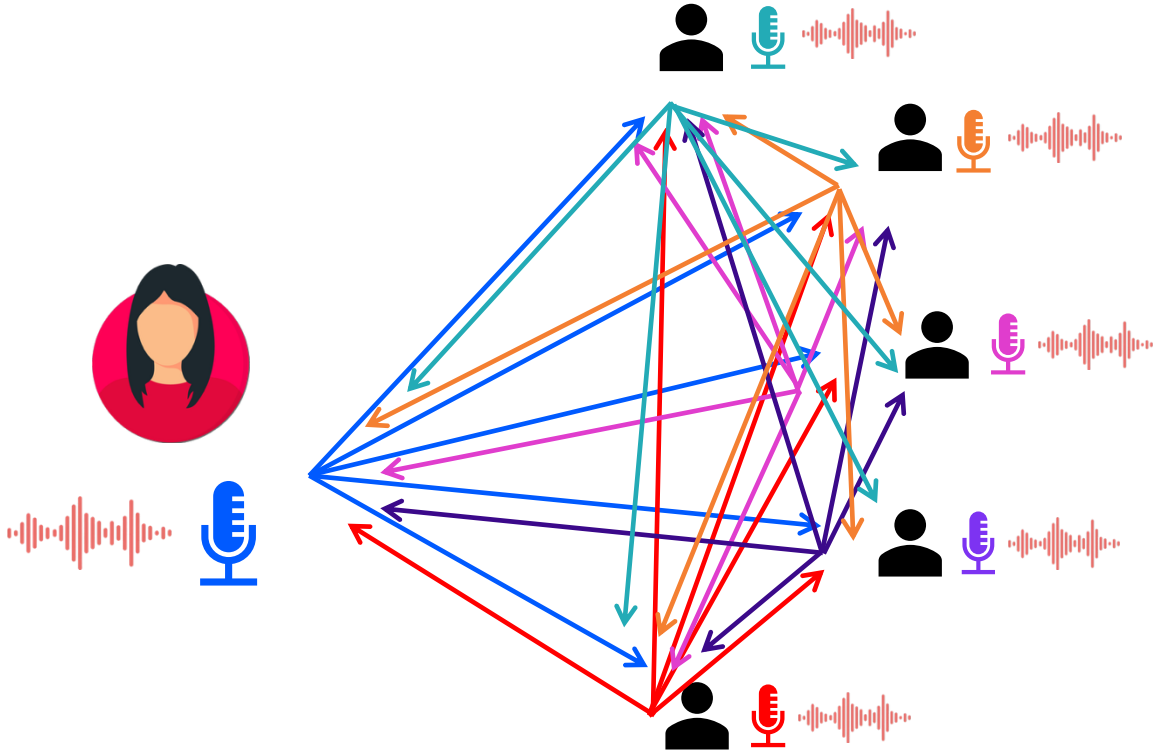


Audio Mixing



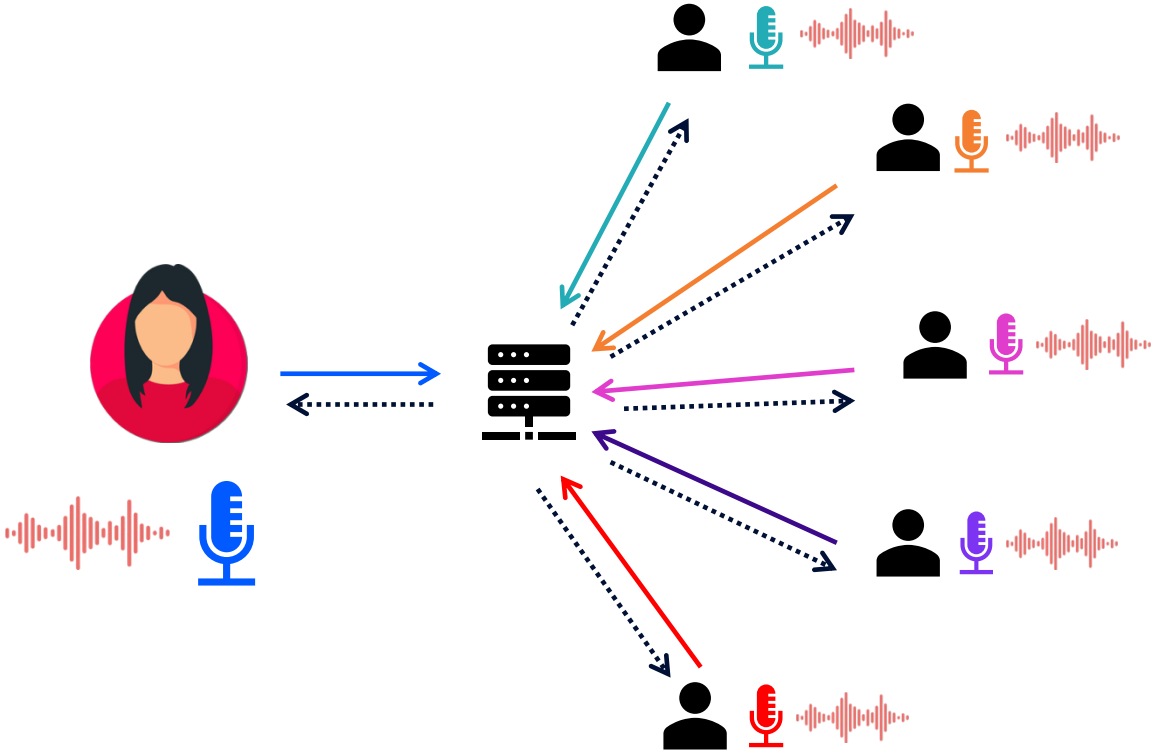
Traffic complexity grows $\boldsymbol{O(n^2)}$

Audio Mixing



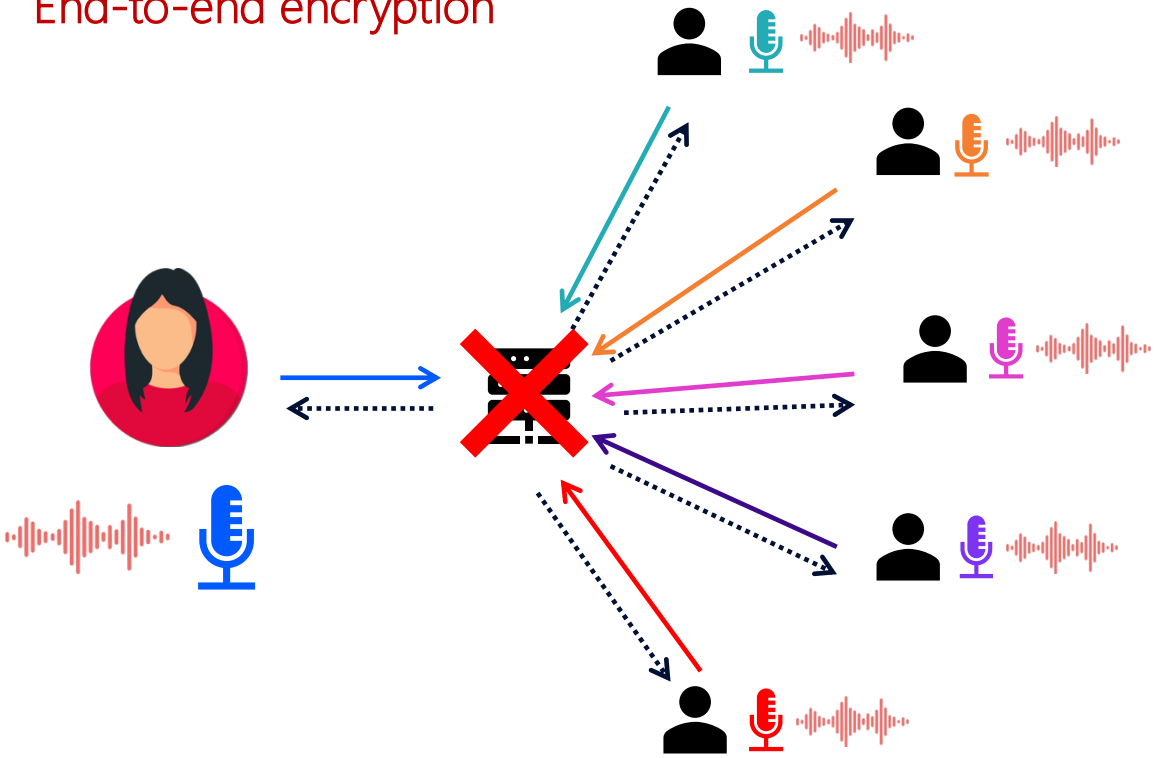
Traffic complexity grows $\boldsymbol{O(n^2)}$

Audio Mixing

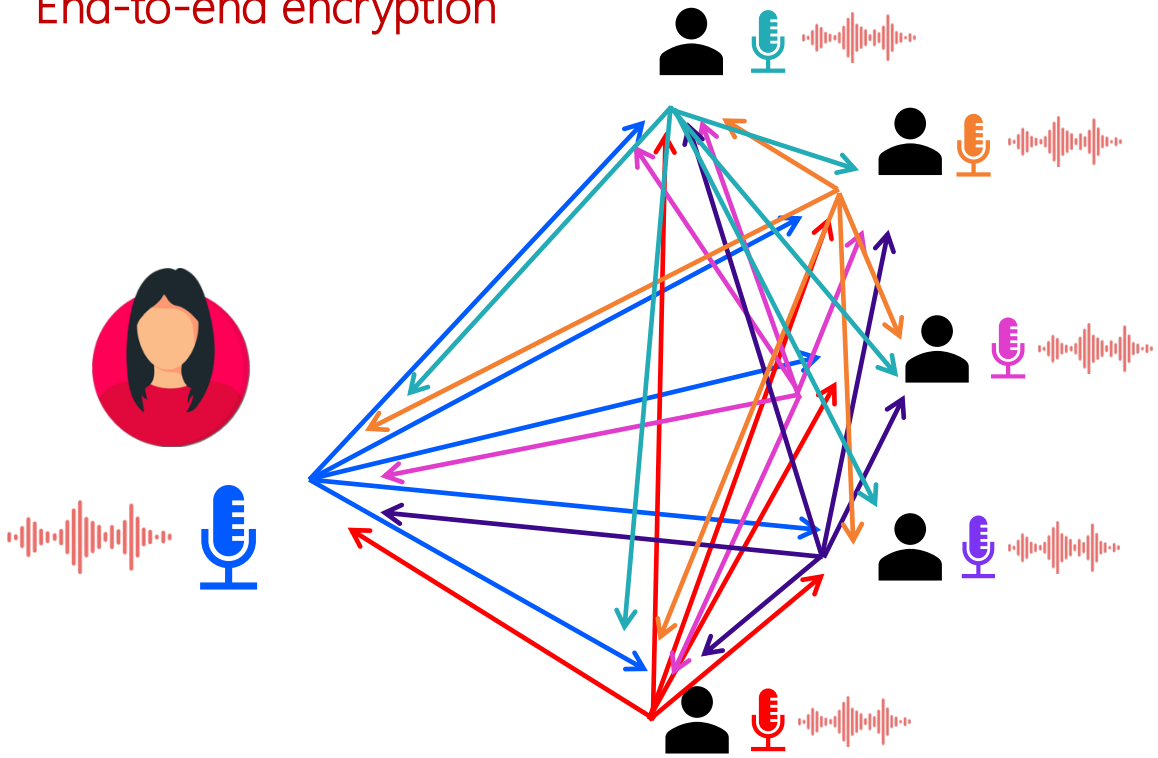


Audio Mixing

End-to-end encryption



Audio Mixing
End-to-end encryption

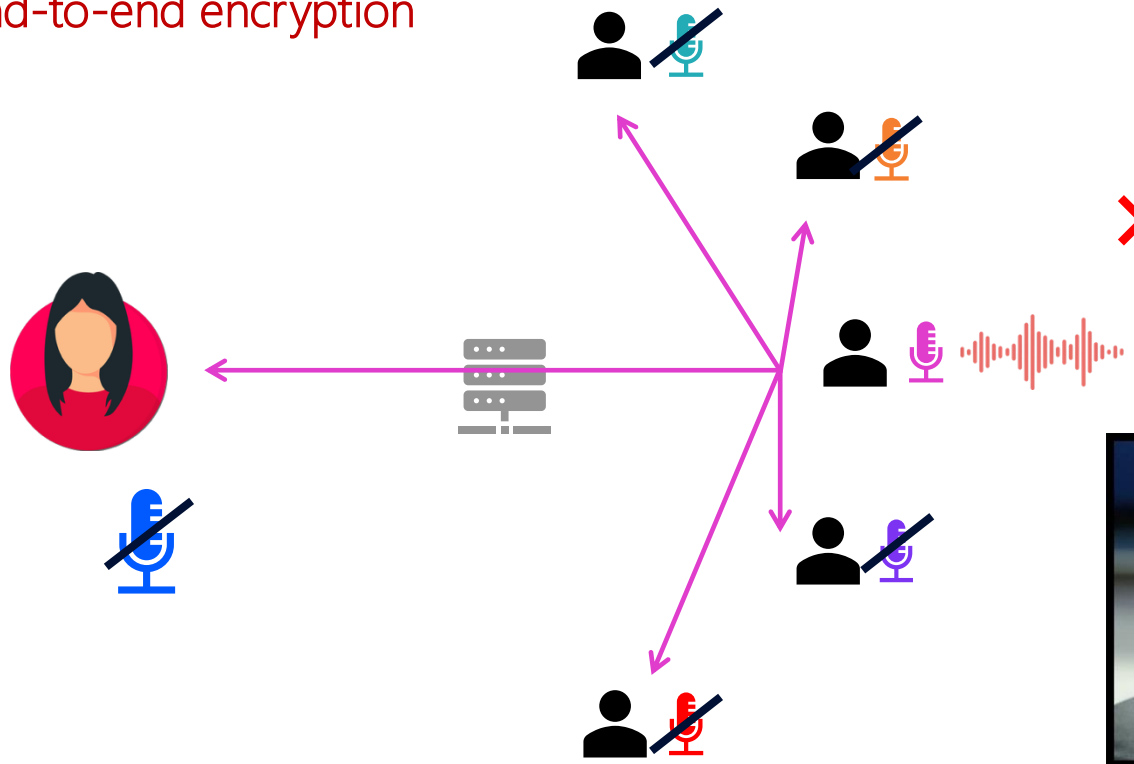


Traffic complexity grows $\boldsymbol{O(n^2)}$

Audio synchronization complexity

Limitations on audio Enhancement techniques

Audio Mixing
End-to-end encryption

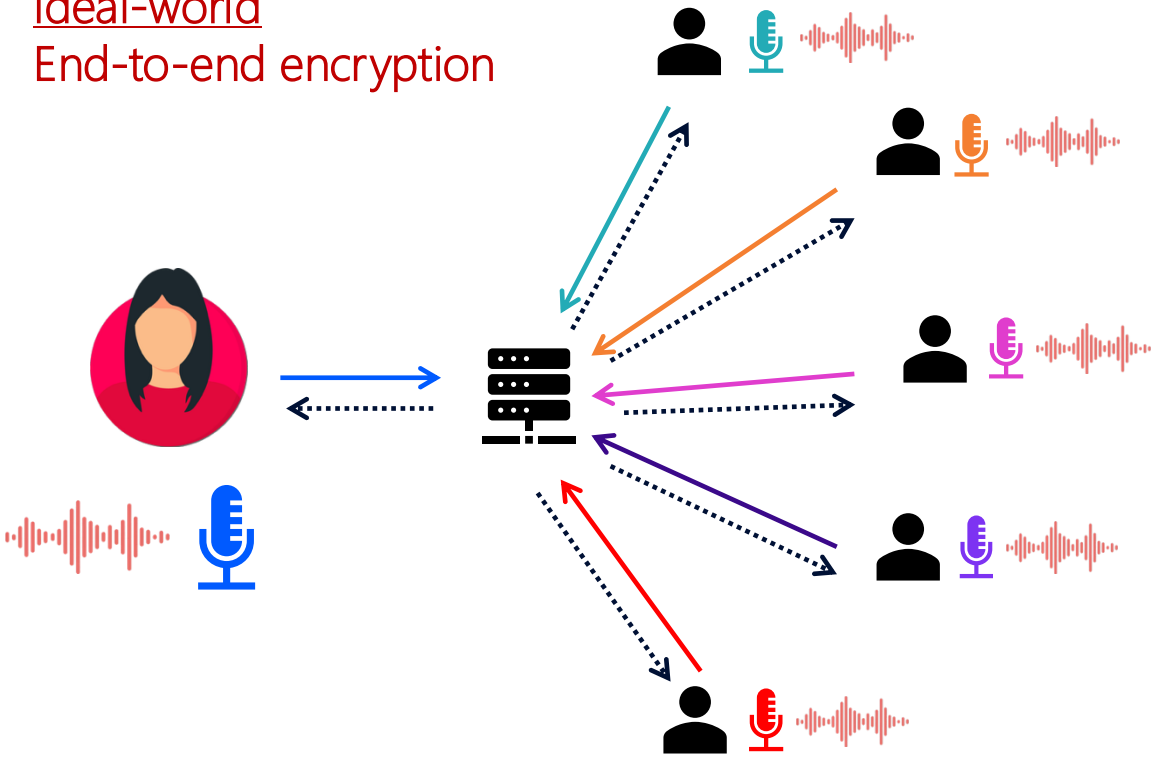


✕ Speaker anonymity

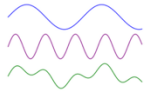
Audio Mixing

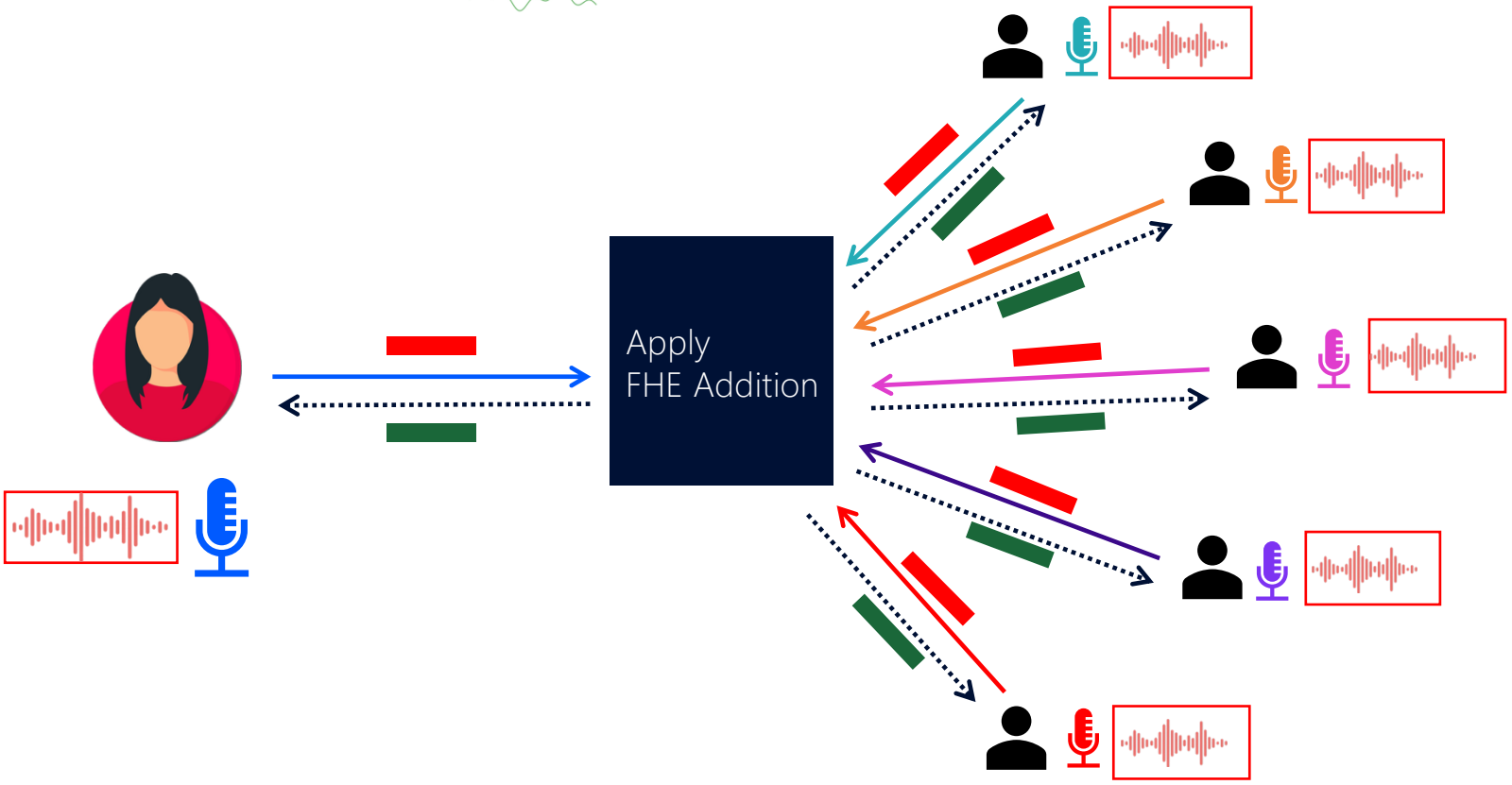
Ideal-world

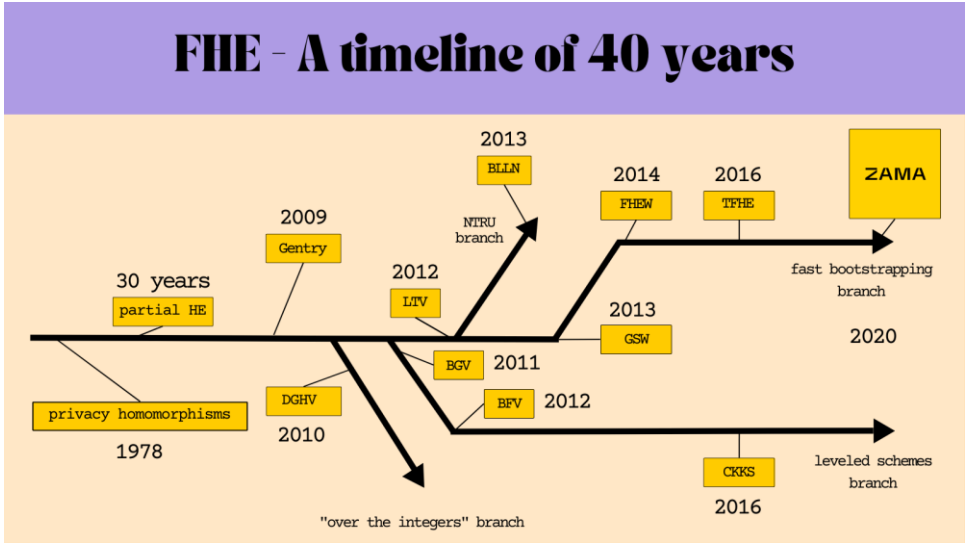
End-to-end encryption



Blind audio mixing







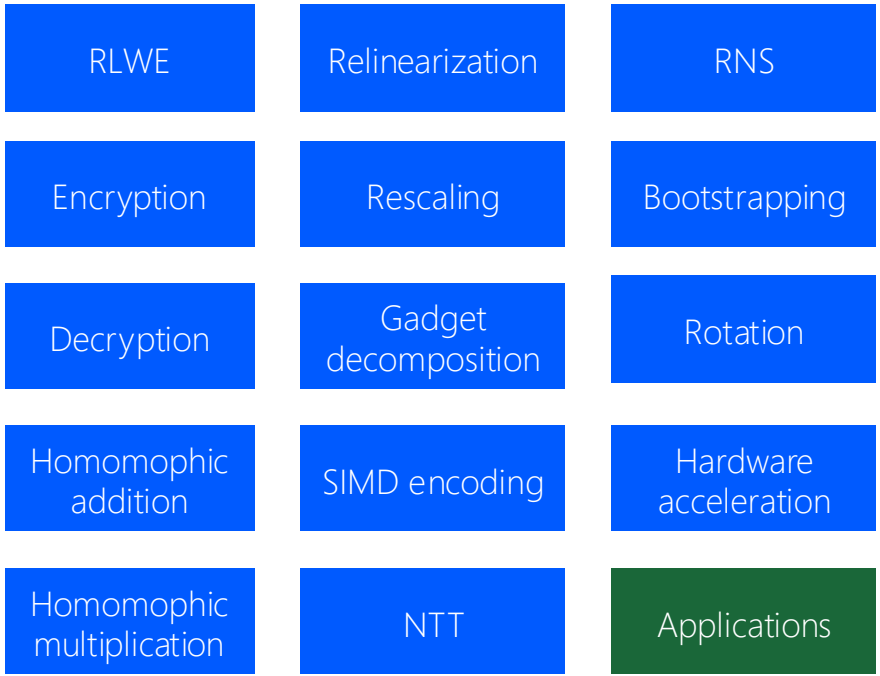
Source: Zama [Marc Joye]

- 1978 - Rivest, Adleman, and Dertouzos "On Data Banks and Privacy Homomorphisms"
- 1984 – 2008 Pallier cryptosystem, El-Gamal cryptosystem, Goldwasser-Micali, Damgard-Jurik, Boneh-Goh-Nissim, and Gentry-Galevi-Vaikuntanathan
- 2009 – Craig Gentry's PhD

[IZ'21] categorizes FHE schemes:

1. Bit-wise operations
   FHEW, TFHE
2. Word-wise operations (Integers) with SIMD
   BGV, BFV
3. Approximate numbers operations (Fixed-point arithmetics)
   CKKS/HEAAN

This talk



How does it work?

Ring Polynomials

A practical intro

$[a]_q$: the remainder of a modulu q $\quad [18]_7 = 4$

Ring $\mathbb{Z}[x]/(f)$: polynomials *modulo some polynomial* f

Example:

$f = x^4 + 1$

$p(x) = x^5 \bmod f = x \,.\, (x^4 + 1) - x$

$= -x \bmod f$

Trick:

Anything $(mod\ f)$

Replace x^4 with -1

Replace x^5 with $-x$

Adding two polynomials

$a(x) = x^3 + x^2 + 7$

$b(x) = x^2 + 11x$

$a(x) + b(x) \bmod f = x^3 + 2x^2 + 11x + 7 \bmod f$

Multiplying two polynomials

$a(x) = x^3 + x^2 + 7$

$b(x) = x^2 + 11x$

$a(x) \,.\, b(x) = (x^3 + x^2 + 7) \,.\, (x^2 + 11x) \bmod f$

$= x^5 + 11x^4 + x^4 + 11x^3 + 7x^2 + 77x \ \bmod f$

$= -x - 11 \ - 1 + 11x^3 + 7x^2 + 77x \ \bmod f$

$= 11x^3 + 7x^2 + 76x - 12 \ \bmod f$

Ring Polynomials

A practical intro

$[a]_q$: the remainder of a modulu q $\quad [18]_7 = 4$

Ring $\mathbb{Z}[x]/(f)$: polynomials *modulo some polynomial* f

Ring $\mathbb{Z}_q[x]/(f)$: polynomials *modulo some polynomial* f and coefficients of these polynomials are reduced modulo q

Example:

$f = x^4 + 1$

$p(x) = x^5 \bmod f = x \,.\, (x^4 + 1) - x$

$= -x \bmod f$

Ring $\mathbb{Z}_5[x]/(x^4 + 1)$

Trick:

Anything $(mod\ f)$

Replace x^4 with -1

Replace x^5 with $-x$

Adding two polynomials

$a(x) = x^3 + x^2 + 7$

$b(x) = x^2 + 11x$

$[a(x) + b(x)]_5 = x^3 + 2x^2 + 11x + 7 \bmod f$

$= x^3 + 2x^2 + x + 2 \bmod f$

Multilying two polynomials

$a(x) = x^3 + x^2 + 7$

$b(x) = x^2 + 11x$

$[a(x) \,.\, b(x)]_5 = (x^3 + x^2 + 7) \,.\, (x^2 + 11x) \bmod f$

$= x^5 + 11x^4 + x^4 + 11x^3 + 7x^2 + 77x \bmod f$

$= -x - 11 - 1 + 11x^3 + 7x^2 + 77x \bmod f$

$= 11x^3 + 7x^2 + 76x - 12 \bmod f$

$= x^3 + 2x^2 + x + 3 \bmod f$

Learning With Error (Regev ‘05)

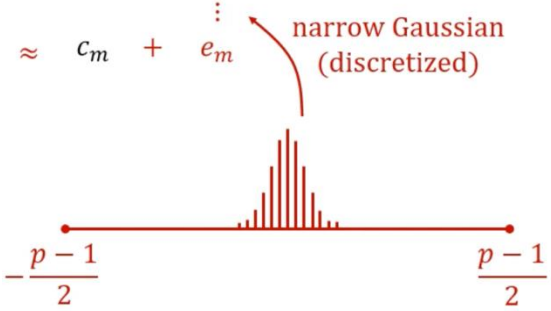
System of linear equations over $\mathbb{F}_p$:

$$\begin{cases} a_{11}s_1 + a_{12}s_2 + \dots + a_{1n}s_n = c_1 \\ \vdots \\ a_{m1}s_1 + a_{m2}s_2 + \dots + a_{mn}s_n = c_m \end{cases}$$

Learning With Error (Regev ‘05)

System of approximate linear equations over $\mathbb{F}_p$:

$$\begin{cases} a_{11}s_1 + a_{12}s_2 + \dots + a_{1n}s_n \approx c_1 + e_1 \\ \vdots \\ a_{m1}s_1 + a_{m2}s_2 + \dots + a_{mn}s_n \approx c_m + e_m \end{cases}$$



Learning With Error (Regev '05)

System of approximate linear equations over $\mathbb{F}_p$:

$$\begin{cases} a_{11}s_1 + a_{12}s_2 + \dots + a_{1n}s_n \approx c_1 + e_1 =: b_1 \\ \vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots \\ a_{m1}s_1 + a_{m2}s_2 + \dots + a_{mn}s_n \approx c_m + e_m =: b_m \end{cases}$$

Learning With Error (Regev '05)

System of approximate linear equations over $\mathbb{F}_p$:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} + \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

$$\begin{cases} a_{11}s_1 + a_{12}s_2 + \cdots + a_{1n}s_n \approx b_1 \\ \vdots \\ a_{m1}s_1 + a_{m2}s_2 + \cdots + a_{mn}s_n \approx b_m \end{cases}$$

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} \approx \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

Goal: find $S_1, S_2, \ldots, S_n$ (requires $m > n$ or extra assumptions on the s_i's).

How to solve? Gaussian elimination?

Errors heap up and become indistinguishable from uniform

Learning With Error (Regev '05)

Applications:

- Key exchange, signatures, homomorphic encryption, …

Disadvantage:

- Large key size

need $\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}, \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$ to hide $\begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix}$

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} + \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

RLWE: Ring Learning With Error (Lyubashevsky, Peikert, Regev '12)

- Reduce key size: use **structured** matrices, such as circulant matrices

$$\begin{pmatrix} a_1 & a_n & \dots & a_2 \\ a_2 & a_1 & \dots & a_3 \\ \vdots & \vdots & \ddots & \vdots \\ a_n & a_{n-1} & \dots & a_1 \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} + \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Sufficient to store first column

Matrix of multiplication by $a_1 + a_2 x + \dots + a_n x^{n-1}$

In the ring $\mathbb{F}_p[x]/(x^n - 1)$

Learning With Error (Regev ‘05)

Applications:

- Key exchange, signatures, homomorphic encryption, …

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} + \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Disadvantage:

- Large key size

need $\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}, \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$ to hide $\begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix}$

RLWE: Ring Learning With Error (Lyubashevsky, Peikert, Regev ‘12)

- Reduce key size: use **structured** matrices, such as circulant matrices

$$(a_1 + a_2x + \dots + a_nx^{n-1}) \cdot (s_1 + s_2x + \dots + s_nx^{n-1}) + (e_1 + e_2x + \dots + e_nx^{n-1}) = b_1 + b_2x + \dots + b_nx^{n-1}$$

$a(x)$ s(x) e(x) b(x)

Ring Learning With Error (RLWE, LPR ‘12)

$$a(x) \,.\, s(x) \;+\; e(x) = b(x)$$

Ring $\mathbb{Z}_p[x]/(x^n + 1)$
With $n = 2^k$

Imagine Alice secretly chooses a polynomial: $\boldsymbol{s(x)}$
which is her secret key!

Everyone knows a random polynomial: $\boldsymbol{a(x)}$

1. She generates a tiny random error $\mathbf{e(x)}$
2. She computes: $\boldsymbol{a(x).s(x) + e(x) = b(x)}$

The public sees (a, b)
The public cannot figure out her secret $\boldsymbol{s(x)}$

Brakerski / Fan-Vercauteren

Notations and parameters

$\boldsymbol{t}$: plaintext modulus
$\boldsymbol{q}$: ciphertext modulus
$\boldsymbol{\Delta} = \lfloor q/t \rfloor$

$\boldsymbol{R_t} = \mathbb{Z}_t[x]/(x^n + 1)$: plaintexts
$\boldsymbol{R_q} = \mathbb{Z}_q[x]/(x^n + 1)$: ciphertexts
$\boldsymbol{R_2}$: polynomials with binary coeff.
$\boldsymbol{\mathcal{X}}$: error distribution

KeyGen

a ← random polynomial from $\boldsymbol{R_q}$

e ← random polynomial from $\boldsymbol{\mathcal{X}}$

Secret Key $\boldsymbol{s}$ ← random poly. $\boldsymbol{R_2}$

Public key $(\boldsymbol{PK_0}, \boldsymbol{PK_1})$

$$\boldsymbol{PK_0} = [-(a.\boldsymbol{s} + e)]_q$$
$$\boldsymbol{PK_1} = a$$

Encrypt (PK, m)

e_1, e_2 ← random polynomial from $\boldsymbol{\mathcal{X}}$

u ← random polynomial from $\boldsymbol{R_2}$

Ciphertext $(\boldsymbol{C_0}, \boldsymbol{C_1})$

$$\boldsymbol{C_0} = [PK_0.u + e_1 + \Delta m]_q$$
$$\boldsymbol{C_1} = [PK_1.u + e_2]_q$$

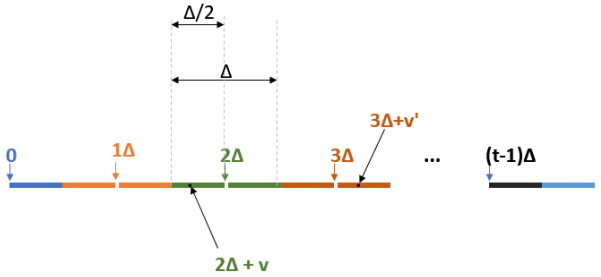
Decrypt (SK, C)

$$m = \left[\left\lfloor \frac{t.[C_0 + C_1.\boldsymbol{s}]_q}{q} \right\rceil\right]_t$$

Let's dig in...

Notation $ct(s) \coloneqq C_0 + C_1.\boldsymbol{s}$

$$\begin{aligned}[ct(s)]_q &= [(PK_0.u + e_1 + \Delta m) + (PK_1.u + e_2).\boldsymbol{s}]_q \\ &= [-(a.\boldsymbol{s} + e).u + e_1 + \Delta m + a.u.\boldsymbol{s} + e_2.\boldsymbol{s}]_q \\ &= \Delta m - e.u + e_1 + e_2.\boldsymbol{s} \\ &= \Delta m + v\end{aligned}$$



Remark: $\|v\| < \Delta/2$

Brakerski / Fan-Vercauteren

Notations and parameters

$\boldsymbol{t}$: plaintext modulus
$\boldsymbol{q}$: ciphertext modulus
$\boldsymbol{\Delta} = \lfloor q/t \rfloor$

$\boldsymbol{R_t} = \mathbb{Z}_t[x]/(x^n + 1)$: plaintexts
$\boldsymbol{R_q} = \mathbb{Z}_q[x]/(x^n + 1)$: ciphertexts
$\boldsymbol{R_2}$: polynomials with binary coeff.
$\boldsymbol{\mathcal{X}}$: error distribution

KeyGen

Secret Key $\boldsymbol{s} \leftarrow$ random poly. $\boldsymbol{R_2}$

$\boldsymbol{PK_0} = [-(a.\boldsymbol{s} + e)]_q$

$\boldsymbol{PK_1} = a$

Encrypt (PK, m)

$\boldsymbol{C_0} = [PK_0.u + e_1 + \Delta m]_q$

$\boldsymbol{C_1} = [PK_1.u + e_2]_q$

Decrypt (SK, C)

$$m = \left\lfloor \left[\frac{t.[C_0 + C_1.\boldsymbol{s}]_q}{q} \right] \right\rfloor_t$$

$ct(s) \coloneqq C_0 + C_1.\boldsymbol{s}$

$[ct(s)]_q = \Delta m + v$

$$ct = Enc(PK, m) = (C_0, C_1)$$
$$ct' = Enc(PK, m') = (C_0', C_1')$$

EvalAdd $(ct, ct') \rightarrow c_{add}$ // $\mathrm{Enc}(m_1 + m_2)$

$$c_{add} = (C_0 + C_0', C_1 + C_1')$$

$$= \begin{pmatrix} [PK_0.u + e_1 + \Delta m + PK_0.u' + e_1' + \Delta m']_q, \\ [PK_1.u + e_2 + PK_1.u' + e_2']_q \end{pmatrix}$$

$$= \begin{pmatrix} [PK_0\,(u + u') + (e_1 + e_1') + \Delta(\boldsymbol{m} + \boldsymbol{m}')]_q, \\ [PK_1(u + u') + (e_2 + e_2')]_q \end{pmatrix}$$

$$= Enc(PK, \boldsymbol{m} + \boldsymbol{m}')$$

Brakerski / Fan-Vercauteren

Notations and parameters

$\boldsymbol{t}$: plaintext modulus
$\boldsymbol{q}$: ciphertext modulus
$\boldsymbol{\Delta} = \lfloor q/t \rfloor$

$\boldsymbol{R_t} = \mathbb{Z}_t[x]/(x^n + 1)$: plaintexts
$\boldsymbol{R_q} = \mathbb{Z}_q[x]/(x^n + 1)$: ciphertexts
$\boldsymbol{R_2}$: polynomials with binary coeff.
$\boldsymbol{\mathcal{X}}$: error distribution

Scale-and-round

Relinearization

Gadget Decomposition

KeyGen

Secret Key $\boldsymbol{s} \leftarrow$ random poly. $\boldsymbol{R_2}$

$\boldsymbol{PK_0} = [-(a.\boldsymbol{s} + e)]_q$

$\boldsymbol{PK_1} = a$

Encrypt (PK, m)

$\boldsymbol{C_0} = [PK_0.u + e_1 + \Delta m]_q$

$\boldsymbol{C_1} = [PK_1.u + e_2]_q$

Decrypt (SK, C)

$$m = \left[\left\lfloor \frac{t.[C_0 + C_1.\boldsymbol{s}]_q}{q} \right\rceil\right]_t$$

$ct(s) \coloneqq C_0 + C_1.\boldsymbol{s}$

$[ct(s)]_q = \Delta m + v$

$$ct = Enc(PK, m) = (C_0, C_1)$$
$$ct' = Enc(PK, m') = (D_0, D_1)$$

EvalMult $(ct, ct') \rightarrow c_{mult}$

$[ct(s)]_q = \Delta m + v$
$[ct'(s)]_q = \Delta m' + v'$

$$ct.ct'(s) = (C_0 + C_1 s)(D_0 + D_1 s)$$
$$ct.ct'(s) = C_0 D_0 + (C_0 D_1 + C_1 D_0)\boldsymbol{s} + C_1 D_1 \boldsymbol{s^2}$$

Ciphertext with 3 components:

$$(ct_{mult})_0 = M_0 = C_0 D_0$$
$$(ct_{mult})_1 = M_1 = C_0 D_1 + C_1 D_0$$
$$(ct_{mult})_2 = M_2 = C_1 D_1$$

$$ct_{mult}(s) = M_0 + M_1 s + M_2 s^2$$

$$[ct(s).ct'(s)]_q = (\Delta m + v)(\Delta m' + v')$$
$$= \boldsymbol{\Delta^2} m.m' + \Delta(m.v' + m'.v) + v.v$$
$$= \boldsymbol{\Delta^2} m.m' + \text{noise}$$

The scale is wrong!

Brakerski / Fan-Vercauteren

Notations and parameters

$\boldsymbol{t}$: plaintext modulus
$\boldsymbol{q}$: ciphertext modulus
$\boldsymbol{\Delta} = \lfloor q/t \rfloor$

$\boldsymbol{R_t} = \mathbb{Z}_t[x]/(x^n + 1)$: plaintexts
$\boldsymbol{R_q} = \mathbb{Z}_q[x]/(x^n + 1)$: ciphertexts
$\boldsymbol{R_2}$: polynomials with binary coeff.
$\boldsymbol{\mathcal{X}}$: error distribution

Scale-and-round

Relinearization

Gadget Decomposition

KeyGen

Secret Key $\boldsymbol{s} \leftarrow$ random poly. $\boldsymbol{R_2}$

$\boldsymbol{PK_0} = [-(a.\boldsymbol{s} + e)]_q$

$\boldsymbol{PK_1} = a$

Encrypt (PK, m)

$\boldsymbol{C_0} = [PK_0.u + e_1 + \Delta m]_q$

$\boldsymbol{C_1} = [PK_1.u + e_2]_q$

Decrypt (SK, C)

$$m = \left[\left\lfloor \frac{t.[C_0 + C_1.\boldsymbol{s}]_q}{q} \right\rceil\right]_t$$

$ct(s) \coloneqq C_0 + C_1.\boldsymbol{s}$

$[ct(s)]_q = \Delta m + v$

$$ct = Enc(PK, m) = (C_0, C_1)$$
$$ct' = Enc(PK, m') = (D_0, D_1)$$

EvalMult $(ct, ct') \rightarrow c_{mult}$

$$\Delta \approx \frac{q}{t}$$

$$M'_0 + M'_1 s + M'_2 s^2 \approx \frac{q}{t}(M_0 + M_1 s + M_2 s^2)$$

$$\approx \Delta m.m' + (m.v' + m'.v) + \frac{q}{t} v.v'$$

Multiplication noise!

$$[ct(s).ct'(s)]_q = (\Delta m + v)(\Delta m' + v')$$

$$= \boldsymbol{\Delta^2} m.m' + \Delta(m.v' + m'.v) + v.v'$$

$$= \boldsymbol{\Delta^2} m.m' + \text{noise}$$

The scale is wrong!

- the old noises interact with the plaintexts
- the noises multiply each other
- scale-and-round introduces additional rounding error
- relinearization will introduce key-switching noise

Actual Scale-and-round:

$$\left[round\left(\frac{t}{q} M_i\right)\right]_q \quad for\ i \in \{0,1,2\}$$

Brakerski / Fan-Vercauteren

Notations and parameters

$\boldsymbol{t}$: plaintext modulus
$\boldsymbol{q}$: ciphertext modulus
$\boldsymbol{\Delta} = \lfloor q/t \rfloor$

$\boldsymbol{R_t} = \mathbb{Z}_t[x]/(x^n + 1)$: plaintexts
$\boldsymbol{R_q} = \mathbb{Z}_q[x]/(x^n + 1)$: ciphertexts
$\boldsymbol{R_2}$: polynomials with binary coeff.
$\boldsymbol{\mathcal{X}}$: error distribution

Scale-and-round

Relinearization

Gadget Decomposition

KeyGen

Secret Key $\boldsymbol{s} \leftarrow$ random poly. $\boldsymbol{R_2}$

$\boldsymbol{PK_0} = [-(a.\boldsymbol{s} + e)]_q$

$\boldsymbol{PK_1} = a$

Encrypt (PK, m)

$\boldsymbol{C_0} = [PK_0.u + e_1 + \Delta m]_q$

$\boldsymbol{C_1} = [PK_1.u + e_2]_q$

Decrypt (SK, C)

$$m = \left[\left\lfloor \frac{t.[C_0 + C_1.\boldsymbol{s}]_q}{q} \right\rceil\right]_t$$

$ct(s) \coloneqq C_0 + C_1.\boldsymbol{s}$
$[ct(s)]_q = \Delta m + v$

$ct = Enc(PK, m) = (\mathrm{C}_0, \mathrm{C}_1)$
$ct' = Enc(PK, m') = (\mathrm{D}_0, \mathrm{D}_1)$

EvalMult $(ct, ct') \rightarrow c_{mult}$

$M'_0 + M'_1 s + M'_2 s^2$ We have to get rid of s^2

Reliniarization key $\mathrm{rlk} = (\mathrm{a}, \mathrm{b})$

$\mathrm{b} = -\mathrm{as} + \mathrm{s}^2 + \mathrm{e} \pmod{\mathrm{q}}$

$b + as = s^2 + e$
$b + as \approx s^2$

Multiply both sides by M'_2:

$M'_2 b + M'_2 as \approx M'_2 s^2$

So instead of keeping $M's^2$, we replace it with:

$M'_2 b + M'_2 as$

That gives a new 2-component ciphertext:

$\widetilde{M}_0 = M'_0 + M'_2 b$
$\widetilde{M}_1 = M'_1 + M'_2 a$

To check its correctness, let's decrypt the new ciphertext:

$(M'_0 + M'_2 b) + (M'_1 + M'_2 a)\mathrm{s}$
$= M'_0 + M'_1 s + M'_2 (b + as)$

Since: $b + as = s^2 + e$

$M'_0 + M'_1 s + M'_2 (s^2 + e)$
$= M'_0 + M'_1 s + M'_2 s^2 + M'_2 e$

So we recovered the original 3-component decryption value, plus some extra noise
Which is called **reliniearization noise!**

Brakerski / Fan-Vercauteren

Notations and parameters

t: plaintext modulus
q: ciphertext modulus
$\Delta = \lfloor q/t \rfloor$

$R_t = \mathbb{Z}_t[x]/(x^n + 1)$: plaintexts
$R_q = \mathbb{Z}_q[x]/(x^n + 1)$: ciphertexts
R_2: polynomials with binary coeff.
χ: error distribution

Scale-and-round

Relinearization

Gadget Decomposition

KeyGen

Secret Key $s \leftarrow$ random poly. R_2

$PK_0 = [-(a.s + e)]_q$

$PK_1 = a$

Encrypt (PK, m)

$C_0 = [PK_0.u + e_1 + \Delta m]_q$

$C_1 = [PK_1.u + e_2]_q$

Decrypt (SK, C)

$$m = \left[\left\lfloor \frac{t.[C_0 + C_1.s]_q}{q} \right\rceil\right]_t$$

$ct(s) := C_0 + C_1.s$

$[ct(s)]_q = \Delta m + v$

$ct = Enc(PK, m) = (C_0, C_1)$

$ct' = Enc(PK, m') = (D_0, D_1)$

We have a problem here !!

EvalMult $(ct, ct') \rightarrow c_{mult}$

$M'_0 + M'_1 s + M'_2 s^2$ We have to get rid of s^2

Reliniarization key $\mathrm{rlk} = (a, b)$

$b = -as + s^2 + e \pmod q$

$b + as = s^2 + e$

$b + as \approx s^2$

Multiply both sides by M'_2:

$M'_2 b + M'_2 as \approx M'_2 s^2$

So instead of keeping $M's^2$, we replace it w

$M'_2 b + M'_2 as$

That gives a new 2-component ciphertext:

$\tilde{M}_0 = M'_0 + M'_2 b$
$\tilde{M}_1 = M'_1 + M'_2 a$

M'_2 can have large coefficients modulo q.

If we multiply M'_2 directly by the key noise e, the extra noise:

$M'_2 e$

can become huge!

To check its correctness, let's decrypt the new ciphertext:

$(M'_1 + M'_2 a)s$

$b + as)$

$^2 + e$

$+ e)$

$^2 + M'_2 e$

original 3-component

some extra noise

arization noise!

Brakerski / Fan-Vercauteren

Notations and parameters

$\boldsymbol{t}$: plaintext modulus
$\boldsymbol{q}$: ciphertext modulus
$\boldsymbol{\Delta} = \lfloor q/t \rfloor$

$\boldsymbol{R_t} = \mathbb{Z}_t[x]/(x^n + 1)$: plaintexts
$\boldsymbol{R_q} = \mathbb{Z}_q[x]/(x^n + 1)$: ciphertexts
$\boldsymbol{R_2}$: polynomials with binary coeff.
$\boldsymbol{\mathcal{X}}$: error distribution

Scale-and-round

Relinearization

Gadget Decomposition

KeyGen

Secret Key $\boldsymbol{s} \leftarrow$ random poly. $\boldsymbol{R_2}$

$\boldsymbol{PK_0} = [-(a.\boldsymbol{s} + e)]_q$

$\boldsymbol{PK_1} = a$

Encrypt (PK, m)

$\boldsymbol{C_0} = [PK_0.u + e_1 + \Delta m]_q$

$\boldsymbol{C_1} = [PK_1.u + e_2]_q$

Decrypt (SK, C)

$$m = \left[\left\lfloor \frac{t.[C_0 + C_1.\boldsymbol{s}]_q}{q} \right\rceil\right]_t$$

$ct(s) := C_0 + C_1.\boldsymbol{s}$
$[ct(s)]_q = \Delta m + v$

$$ct = Enc(PK, m) = (C_0, C_1)$$
$$ct' = Enc(PK, m') = (D_0, D_1)$$

EvalMult $(ct, ct') \rightarrow c_{mult}$

$M'_0 + M'_1 s + M'_2 s^2$ We have to get rid of s^2

Every coefficient of M'_2 is decomposed in base B

$$M'_2 = M'_2{}^{(0)} + BM'_2{}^{(1)} + B^2 M'_2{}^{(2)} + \cdots + B^l M'_2{}^{(l)}$$

Reliniarization key $\mathrm{rlk}_i = (a_i, b_i)$

$b_i = -a_i s + B^i s^2 + e_i \pmod q$

$b_i + a_i s = B^i s^2 + e_i$
$b_i + a_i s \approx B^i s^2$

Multiply both sides

$M'_2{}^{(i)} b_i + M'_2{}^{(i)} a_i s = M'_2{}^{(i)} B^i s^2 + M'_2{}^{(i)} e_i$
$M'_2{}^{(i)} b_i + M'_2{}^{(i)} a_i s \approx M'_2{}^{(i)} B^i s^2$

That gives a new 2-component ciphertext:

$$\tilde{M}_0 = M'_0 + \sum_i M'_2{}^{(i)} b_i$$
$$\tilde{M}_1 = M'_1 + \sum_i M'_2{}^{(i)} a_i$$

Example: $B = 2$, so this is bit-decomposition

$M'_2 = M'_2{}^{(0)} + 2M'_2{}^{(1)} + 2^2 M'_2{}^{(2)} + \cdots + 2^l M'_2{}^{(l)}$
$M'_2{}^{(i)}$: very small

Brakerski / Fan-Vercauteren

Notations and parameters

$\boldsymbol{t}$: plaintext modulus
$\boldsymbol{q}$: ciphertext modulus
$\boldsymbol{\Delta} = \lfloor q/t \rfloor$

$\boldsymbol{R_t} = \mathbb{Z}_t[\boldsymbol{x}]/(\boldsymbol{x^n + 1})$: plaintexts
$\boldsymbol{R_q} = \mathbb{Z}_q[\boldsymbol{x}]/(\boldsymbol{x^n + 1})$: ciphertexts
$\boldsymbol{R_2}$: polynomials with binary coeff.
$\boldsymbol{\mathcal{X}}$: error distribution

Scale-and-round

Relinearization

Gadget Decomposition

KeyGen

Secret Key $\boldsymbol{s} \leftarrow$ random poly. $\boldsymbol{R_2}$

$\boldsymbol{PK_0} = [-(a.\boldsymbol{s} + e)]_q$

$\boldsymbol{PK_1} = a$

Encrypt (PK, m)

$\boldsymbol{C_0} = [PK_0.u + e_1 + \Delta m]_q$

$\boldsymbol{C_1} = [PK_1.u + e_2]_q$

Decrypt (SK, C)

$$m = \left[\left\lfloor \frac{t.[C_0 + C_1.\boldsymbol{s}]_q}{q} \right\rceil\right]_t$$

$ct(s) \coloneqq C_0 + C_1.\boldsymbol{s}$
$[ct(s)]_q = \Delta m + v$

$$ct = Enc(PK, m) = (\mathrm{C}_0, \mathrm{C}_1)$$
$$ct' = Enc(PK, m') = (\mathrm{D}_0, \mathrm{D}_1)$$

EvalMult $(ct, ct') \rightarrow c_{mult}$

$M'_0 + M'_1 s + M'_2 s^2$ We have to get rid of s^2

Every coefficient of M'_2 is decomposed in base B

$$M'_2 = M'_2{}^{(0)} + BM'_2{}^{(1)} + B^2 M'_2{}^{(2)} + \cdots + B^l M'_2{}^{(l)}$$

Reliniarization key $\mathrm{rlk}_i = (\mathrm{a}_i, \mathrm{b}_i)$

$$b_i = -a_i \mathrm{s} + B^i s^2 + e_i \pmod{\mathrm{q}}$$

$$b_i + a_i \mathrm{s} = B^i s^2 + e_i$$
$$b_i + a_i \mathrm{s} \approx B^i s^2$$

That gives a new 2-component ciphertext:

$$\widetilde{M}_0 = M'_0 + \sum_i M'_2{}^{(i)} b_i$$
$$\widetilde{M}_1 = M'_1 + \sum_i M'_2{}^{(i)} a_i$$

Example: $B = 2$, so this is bit-decomposition

$$M'_2 = M'_2{}^{(0)} + 2M'_2{}^{(1)} + 2^2 M'_2{}^{(2)} + \cdots + 2^l M'_2{}^{(l)}$$
$M'_2{}^{(i)}$: very small

Check correctness via decryption:

$$\left(M'_0 + \sum_i M'_2{}^{(i)} b_i\right) + \left(M'_1 + \sum_i M'_2{}^{(i)} a_i\right) s$$
$$= M'_0 + M'_1 s + \sum_i M'_2{}^{(i)} (b_i + a_i s)$$
$$= M'_0 + M'_1 s + \textstyle\sum_i M'_2{}^{(i)} \left(B^i s^2 + e_i\right)$$
$$= M'_0 + M'_1 s + \left(\textstyle\sum_i B^i M'_2{}^{(i)}\right) s^2 + \textstyle\sum_i M'_2{}^{(i)} e_i$$
$$= M'_0 + M'_1 s + M'_2 s^2 + \textstyle\sum_i M'_2{}^{(i)} e_i$$

Much smaller reliniearization noise!

How is it done in real world?

How do we **encode** our message *m* into the polynomial to encrypt?

Message encoding

High level workflow

Single Value Encoding

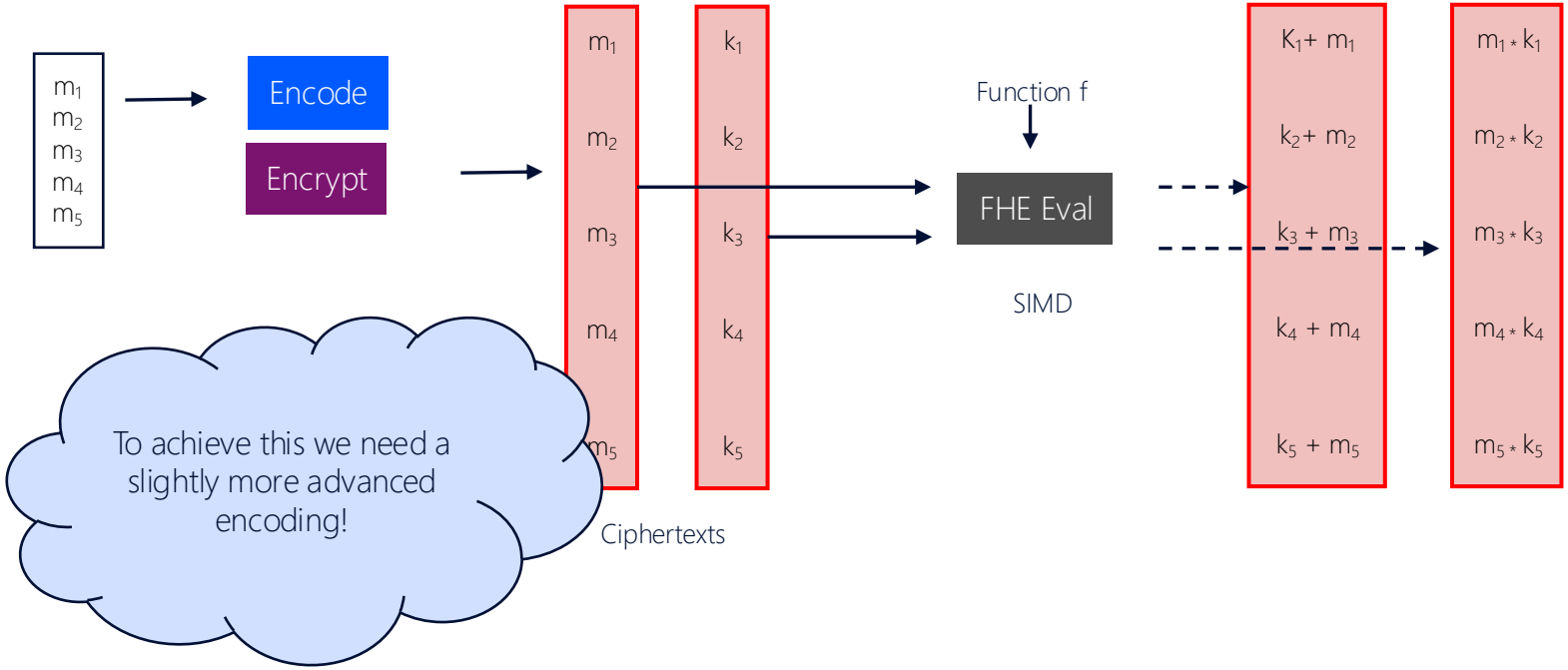
$m = b_{n-1}.2^{n-1} + b_{n-2}.2^{n-2} + \ \ldots + b_1.2^1 + b_0.2^0$, where each $b_i \in \{0,1\}$

Encoded polynomial: $\mathrm{M(X)} = b_{n-1}.X^{n-1} + b_{n-2}.X^{n-2} + \ \ldots + b_1.X^1 + b_0.2^0$

Decoding: $\mathrm{M}(X = 2) = m$

SIMD Packing in BGV/BFV

High-level goal



SIMD Packing in BGV/BFV

Basic Intuition

Prod degree gets increased!
Coeffiecients are over rings of integer!
Coefficients grow rapidly

Data vector $u = \{m_1, m_2, m_3, m_4\} = \{1, 10, 49, 142\}$

Position x	Value
0	1
1	10
2	49
3	142

$U(0) = 1$
$U(1) = 10$
$U(2) = 49$
$U(3) = 142$

↓ Interpolate

$U(x) = 1 + 2x + 3x^2 + 4x^3$

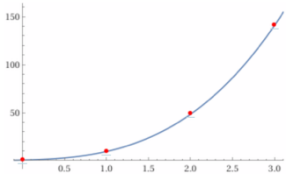
Data vector $\mathrm{V} = \{m'_1, m'_2, m'_3, m'_4\} = \{5, 2, 5, 32\}$

Position x	Value
0	5
1	2
2	5
3	32

$V(0) = 5$
$V(1) = 2$
$V(2) = 5$
$V(3) = 32$

↓ Interpolate

$V(x) = 5 - 6x^2 + 3x^3$



$Sum(x) = U(x) + V(x) = 7x^3 - 3x^2 + 2x + 6$
$Prod(x) = U(x) \times V(x) = 12x^6 - 15x^5 - 12x^4 + 11x^3 + 9x^2 + 10x + 5$

$Sum(2) = 54$
$Prod(1) = 20$

SIMD Packing in BGV/BFV

via Forward CRT

$U(x)$ resides in Ring $\mathbb{Z}_t[x]/(X^N + 1)$

- Degree less than N
- Coefficients in $\mathbb{Z}_t$

What serves as the "coordinate system" for our interpolation in this modular world?

The Ring and the Roots (the "Slots")

Choose t such that the polynomial $(X^N + 1)$ splits into N distinct linear factors!

This occurs when $t \equiv 1 \ (mod\ 2N)$

Under this condition, there exist N distinct roots (values) $\{r_0, r_q, \dots, r_{N-1}\}$

$\Longrightarrow \ x^n + 1 \equiv (x - r_0)(x - r_1) \dots (x - r_{N-1}) \quad (mod\ t)$

Math jargon: $2N$-th roots of unity => We can use them as our fixed x-coordinates!

SIMD Packing in BGV/BFV

via Forward CRT

This is basically interpolation (iNTT or iCRT)

The **Slots** : the values at roots r_i

Forward CRT (Decomposition): Polynomial → Vector of values at roots

Inverse CRT (Composition): Vector of values at roots → Polynomial

Packing (Encoding) via Inverse CRT (Interpolation):

Vector $u = \{u_0, \dots, u_{n-1}\} \leftrightarrow Polynomial\ U(x)$

Position x	Value
r_0	u_0
…	…
r_{N-1}	u_{N-1}

→ $U(x)$

Interpolation
Inverse CRT
Inverse NTT

SIMD Operation on Ciphertext

Works for both Sum(x) and Prod(x)

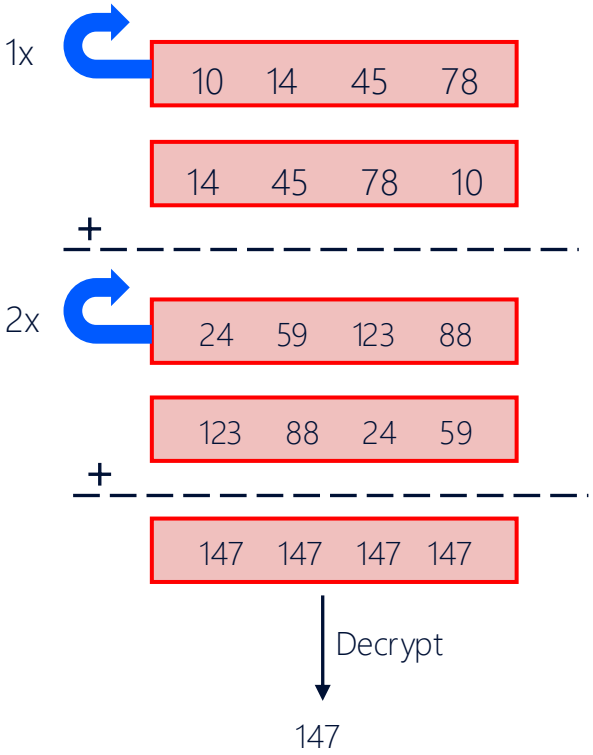
We evaluate at the roots of $X^N + 1$ because they satisfy the ring's defining equation, causing multiplication in the polynomial ring to become **slot-wise multiplication**.

If 10 numbers are SIMD-packed in a ciphertext, how can we add them up?

$$Enc(10, 14, 45, 78)$$

We want to compute: $10 + 14 + 45 + 78 = 147$

Can we rotate the SIMD slots?



SIMD Slot Rotation

BFV, BGV, CKKS

A rotation is **not** performed by moving data directly.

Instead, FHE applies a **ring automorphism** (Frobenius automorphisms).

$$X \longmapsto X^k$$

This automorphism permutes the roots of unity!

For rotation, we require **Rotation (Galois) keys**

There are many polynomial multiplications! Is it going to be slow?

Textbook multiplication of two polynomials: 166 seconds !!!

$N = 2^{16}$
$Q = 360\ bits$

Polynomial multiplication

Using NTT

$f(x)$ resides in Ring $\mathbb{Z}_t[x]/(X^N + 1)$

- Degree less than N
- Coefficients in $\mathbb{Z}_t$

Polynomial representation → Coefficients
Polynomial representation → Evaluation

Fast Fourier Transform (FFT) specialized for $\mathbb{Z}_t$ where

- t is a prime
- $t \equiv 1\ (mod\ 2N)$
- N is a power of 2 (in case of BFV/BGV)

NTT (coefficients of $f(x)$) = Evaluating the polynomial on the roots of unity in $\mathbb{Z}_t$

$A(x) = a_0 + a_1 x + \cdots + a_{N-1} x^{N-1}$
$B(x) = b_0 + b_1 x + \cdots + b_{N-1} x^{N-1}$

NTT →

Evaluations of $A = \{A(r_1), \ldots, A(r_{N-1})\}$
Evaluations of $B = \{B(r_1), \ldots, B(r_{N-1})\}$

Pointwise multiplication

Evaluations of $C = \{C(r_1), \ldots, C(r_{N-1})\}$

iNTT →

$C(x) = c_0 + c_1 x + \cdots + c_{N-1} x^{N-1}$

$C(x) = A(x).B(x)$

Textbook convolution: $O(N^2)$
NTT-based mult: $O(N \log N)$

How to deal with extremely large integers?
Is this going to be super slow?

$N = 2^{16}$
$Q = 360\ bits$

One example coefficient: 415726398472398472398472398472398472398472398472398472398472

↑
360-bit integer

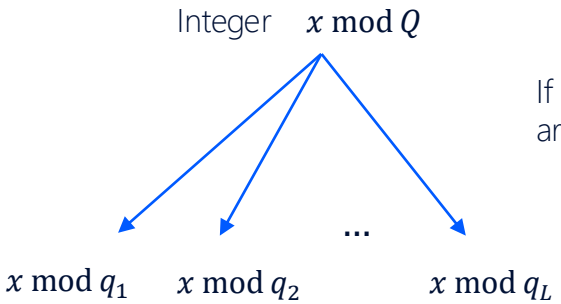
Millions or billions of coefficient multiplications !!!

➔ multi-precision integer arithmetic ➔ Not fun!

Chinese Remainder Theorem (CRT)

Goal: representing on integer with several residues

Let $Q = q_1 q_2 \dots q_L$
Where $q_1, q_2, \dots, q_L$
are **pairwise co-prime.**



If $q_1, q_2, \dots, q_L$
are **pairwise co-prime.** ➡ Uniquely determining original integer modulo Q

Example:

$Q = 13 \times 17 \times 19 = 4199$
$x = 923 \bmod Q$

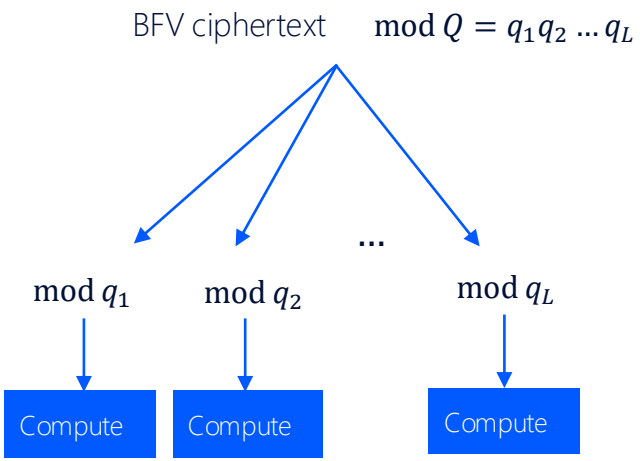
$\mod 13 \rightarrow 0$
$\mod 17 \rightarrow 5$
$\mod 19 \rightarrow 11$

➡ 923 ⟺ (0,5,11)

One large integer → many small residues

Residue Number Systems (RNS) Representation

In FHE



1. Ring Isomorphism:

$$\mathbb{Z}_Q[X]/(X^N+1) \cong \prod_{i=1}^{L-1} \mathbb{Z}_{q_i}[X]/(X^N+1)$$

2. Can use NTT
3. Small integer arithmetic.
4. Highly parallelizable!!
5. Better for hardware acceleration
6. Better cache locality

Double-CRT:
SIMD-packing + RNS

Prototype implementation **Cedarcrypt**

Source code: github.com/emad7105/cedarcrypt

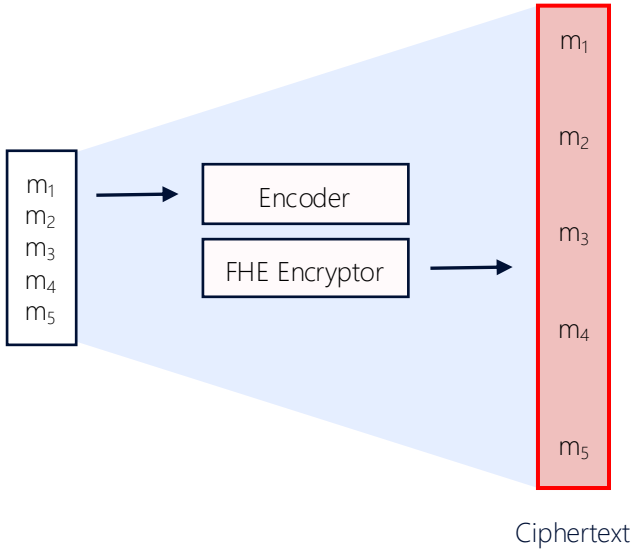
$$\boldsymbol{R_q} = \mathbb{Z}_q[\boldsymbol{x}]/(\boldsymbol{x^N} + \boldsymbol{1})$$
$$N = 65536 = 2^{16}$$
$$Q = \{60,60,60,60,60,60\}$$

Naïve multiplication (convolution)	Multiplication with **RNS + NTT**	Multiplication with RNS + NTT (Parallel)
166s	240ms	42ms

2nd & 4th Gen FHE

BFV, BGV, CKKS

More operations, larger ciphertexts



BFV Example for single message:

2nd & 4th Gen FHE

BFV, BGV, CKKS

Noise management

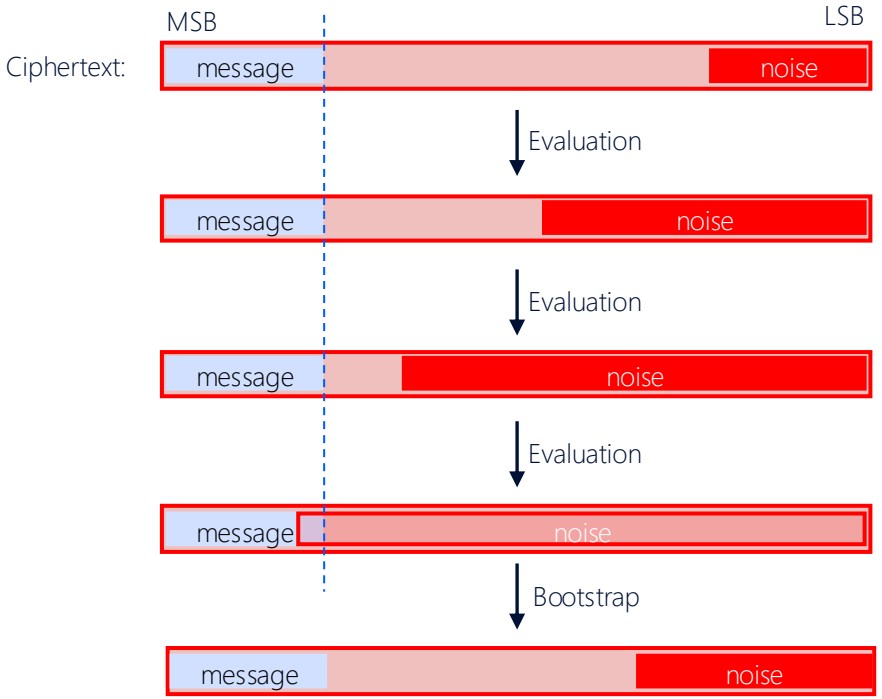
Noise budget is often defined by **multiplication depth**

Bootstrapping reduces the noise

Remarks

- Bootsrapping consumes noise budget too
- Bootsrapping is not fast

BFV Example for single message:



2nd & 4th Gen FHE

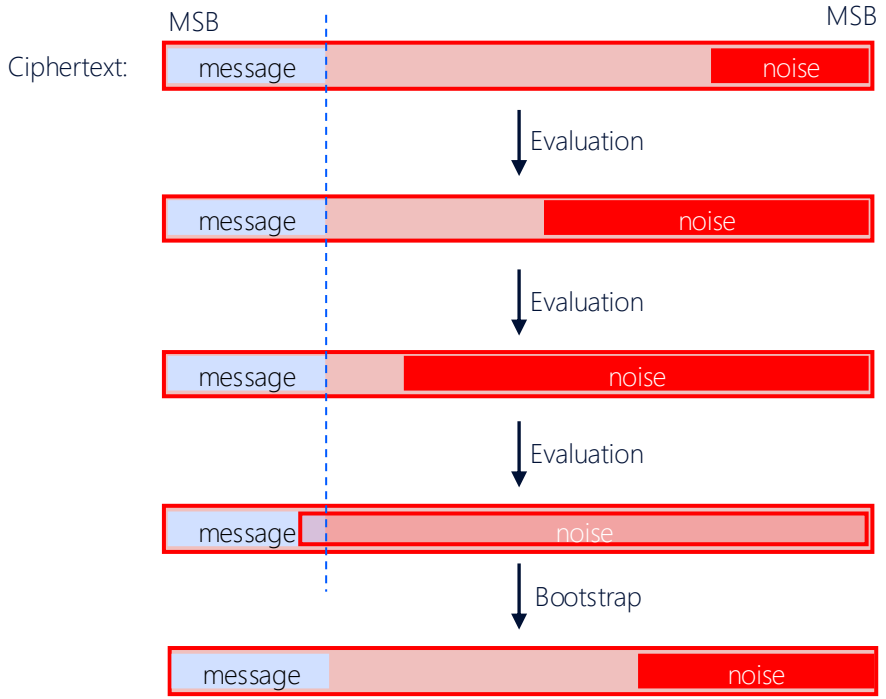
BFV, BGV, CKKS

Parameterization

Important parameters:
- Message domain size
- Ciphertext domain size
- Number of packing slots
- Required multiplication

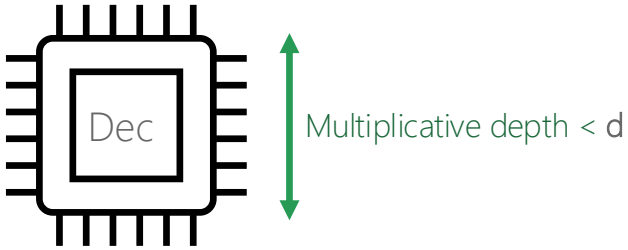
Impacts on security of the the scheme

BFV Example for single message:

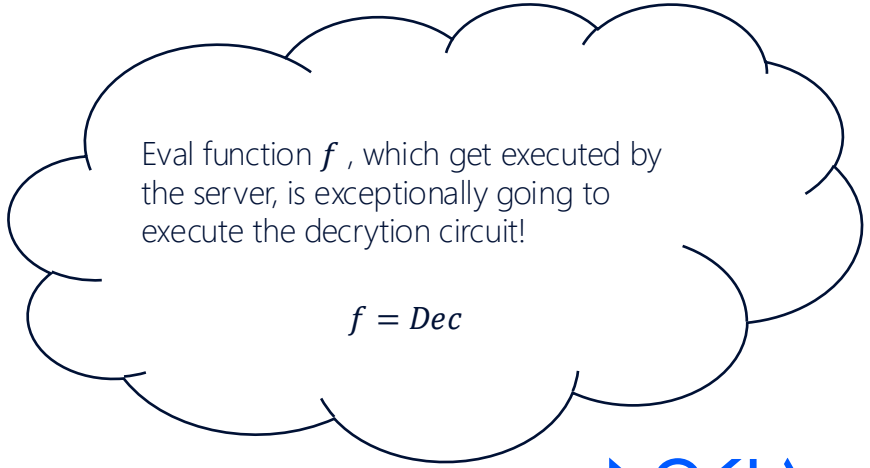


Gentry's Bootstrapping Theorem (2009)

- We require a Somewhat HEs (SHE) that supports evaluating depth-**d** circuits.
- **Core idea:**
  a SHE should be able to evaluate its own Decryption circuit, in other words:



Bootstrapable SHE => FHE

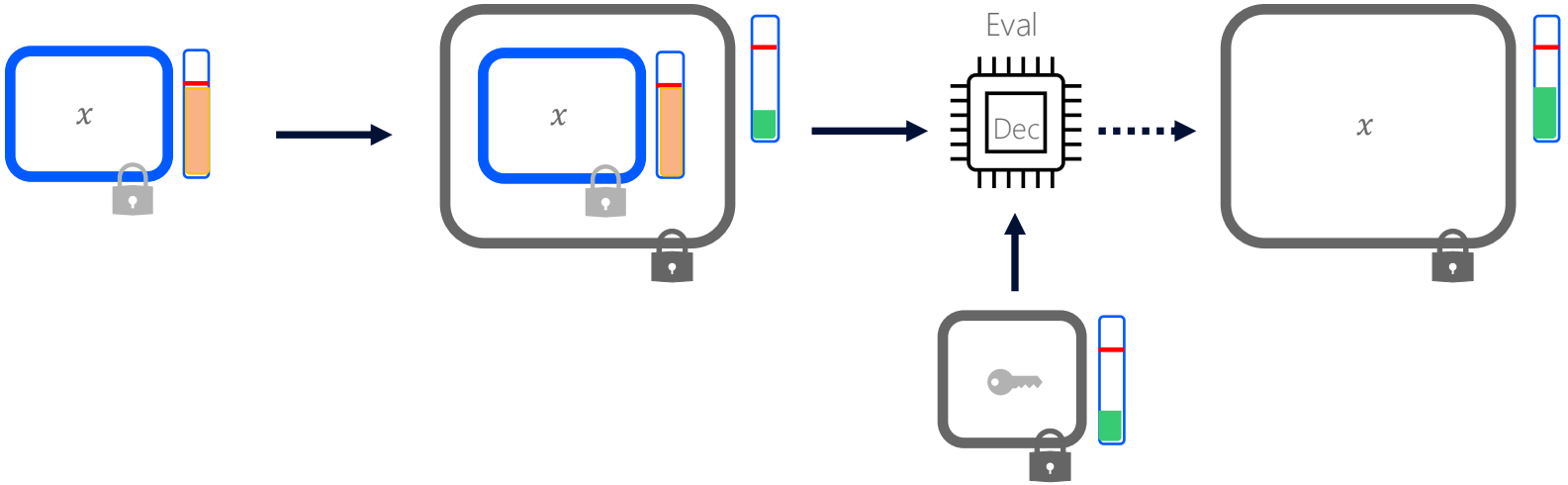


Gentry's Bootstrapping Theorem (2009)

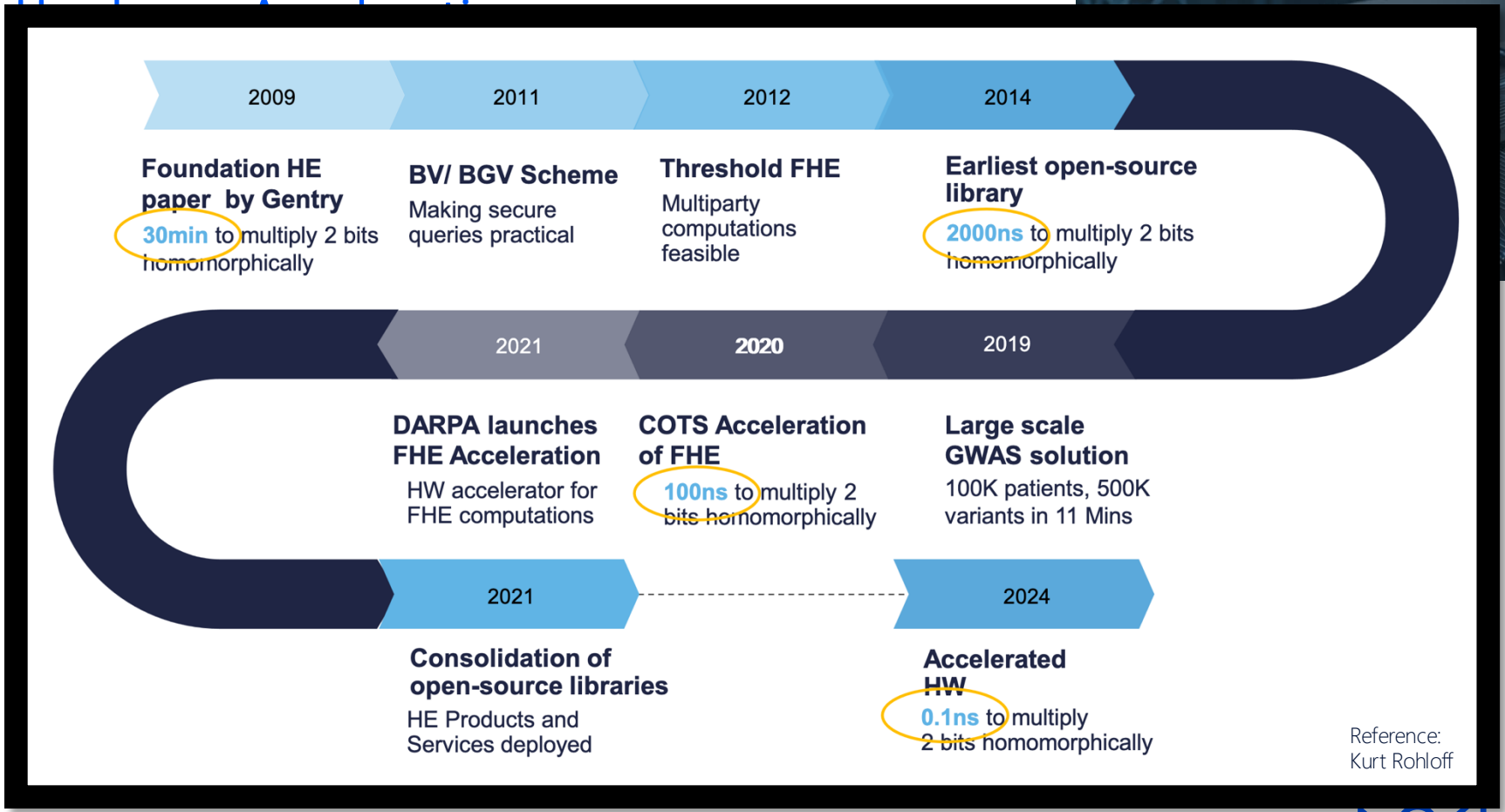
Key Switching (simplified)

We have 2 keys:

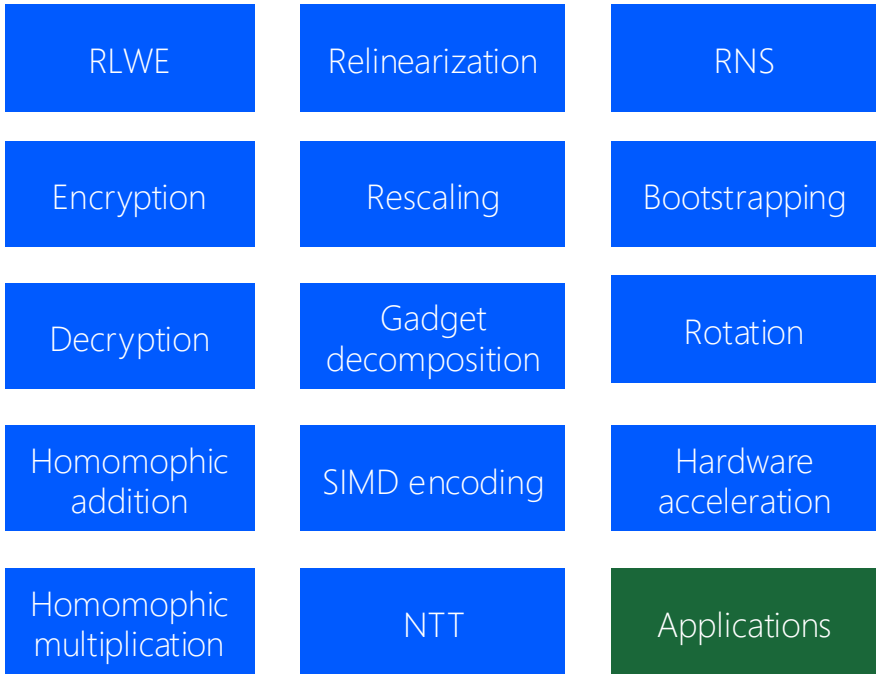
- Secret key : the main key for decryption
- Bootstrapping key and



The content of this slide is partially inspired by Zama's slide deck


2009
2011
2012
2014
Foundation HE paper by Gentry
30min to multiply 2 bits homomorphically
BV/ BGV Scheme
Making secure queries practical
Threshold FHE
Multiparty computations feasible
Earliest open-source library
2000ns to multiply 2 bits homomorphically
2021
2020
2019
DARPA launches FHE Acceleration
HW accelerator for FHE computations
COTS Acceleration of FHE
100ns to multiply 2 bits homomorphically
Large scale GWAS solution
100K patients, 500K variants in 11 Mins
2021
2024
Consolidation of open-source libraries
HE Products and Services deployed
Accelerated HW
0.1ns to multiply 2 bits homomorphically
Reference:
Kurt Rohloff

What we learnt today



FHE: From Theory to Practice

Emad Heydari Beni

@heydari_be

LinkedIn

NOKIA BELL LABS

100 years innovating

