$i\mathcal{O}$ via Functional Encryption: Techniques and Challenges from LWE

Shweta Agrawal
IIT Madras

Laying Out a Plan

Yesterday:

1. Why FE implies iO

2. Survey of JLS20: Identify and leverage a surprising, beautiful synergy between three different assumptions to build FE : LWE, SXDH, LPN

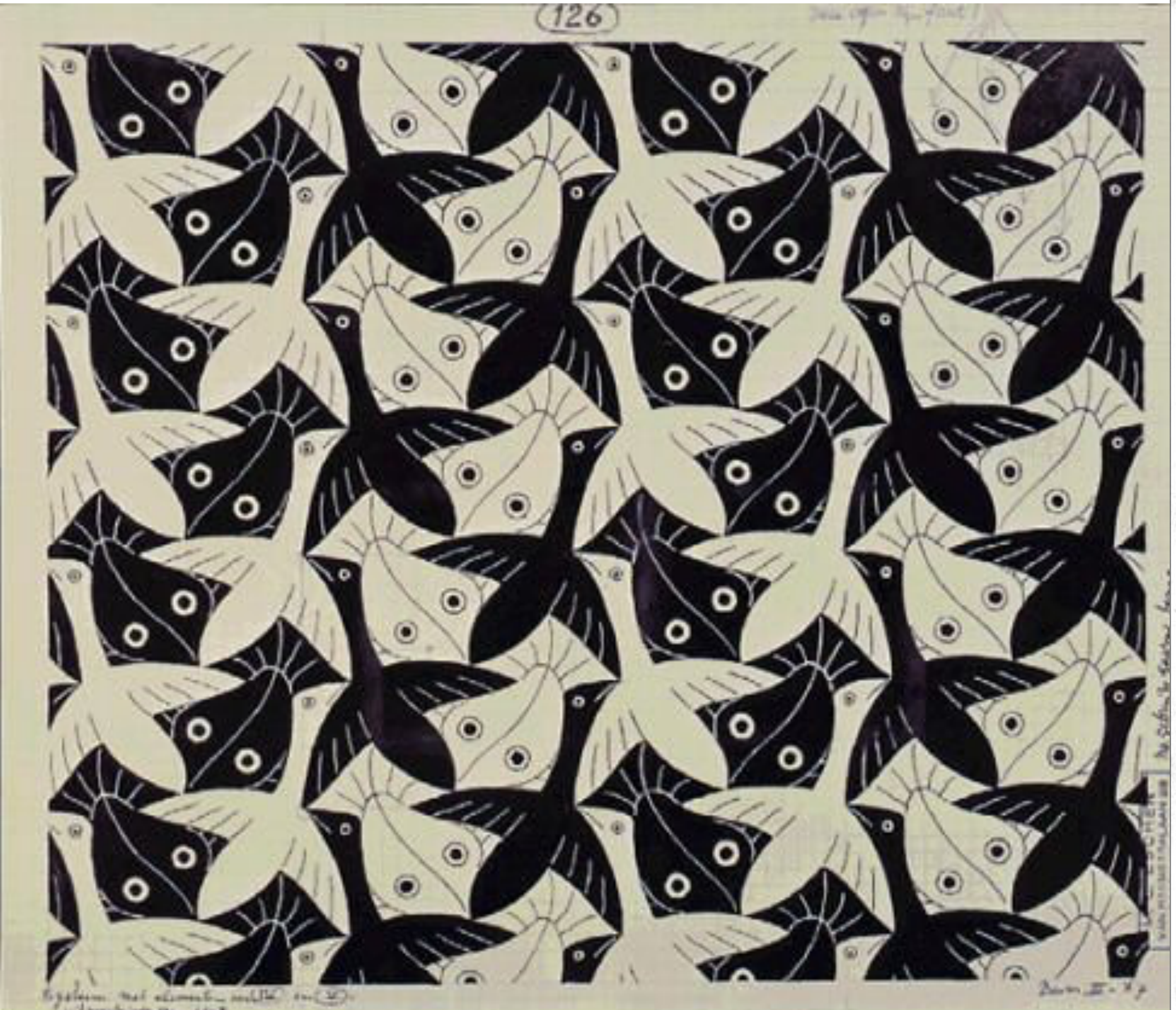
Today:

Try to build FE from LWE alone.

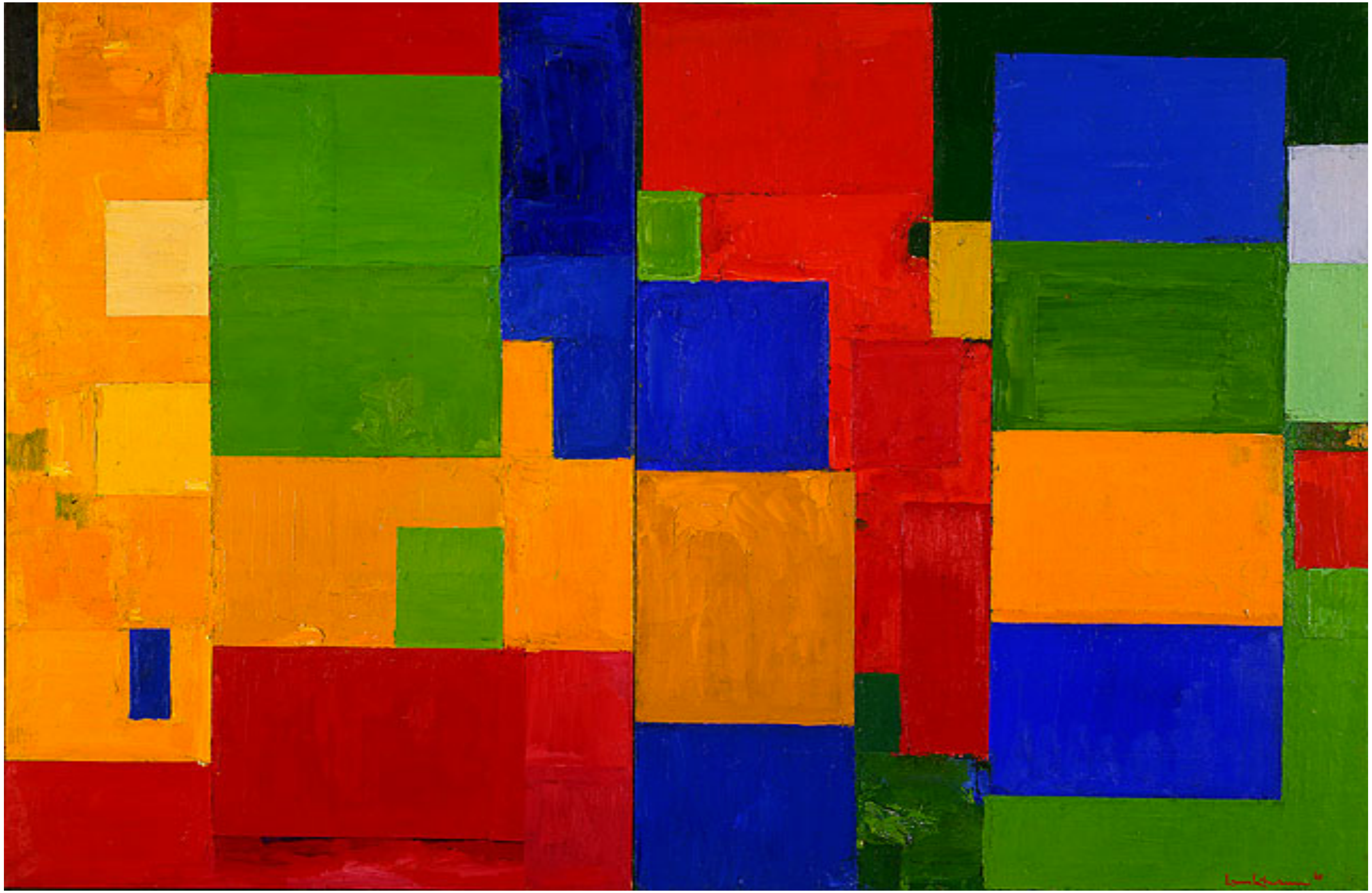
Identify technical challenges, discuss how pairings overcome these

Why?

M C Escher



Hans Hoffman



Functional Encryption
Encryption with Partial Decryption Keys

(mpk, msk) ← Setup(1^n)

Encrypt (mpk, x):

Keygen(msk, F):

Decrypt (sk_F, ct):

y = F(x)

Security:

Adversary possessing keys for multiple circuits F_i cannot distinguish Enc(x_0) from Enc(x_1) unless $F_i(x_0) \neq F_i(x_1)$

Functional Encryption [SW05,BSW11]

FE → $i\mathcal{O}$ [AJ15, BV15,Lin16,LV16,AS16 → Yael's talk]

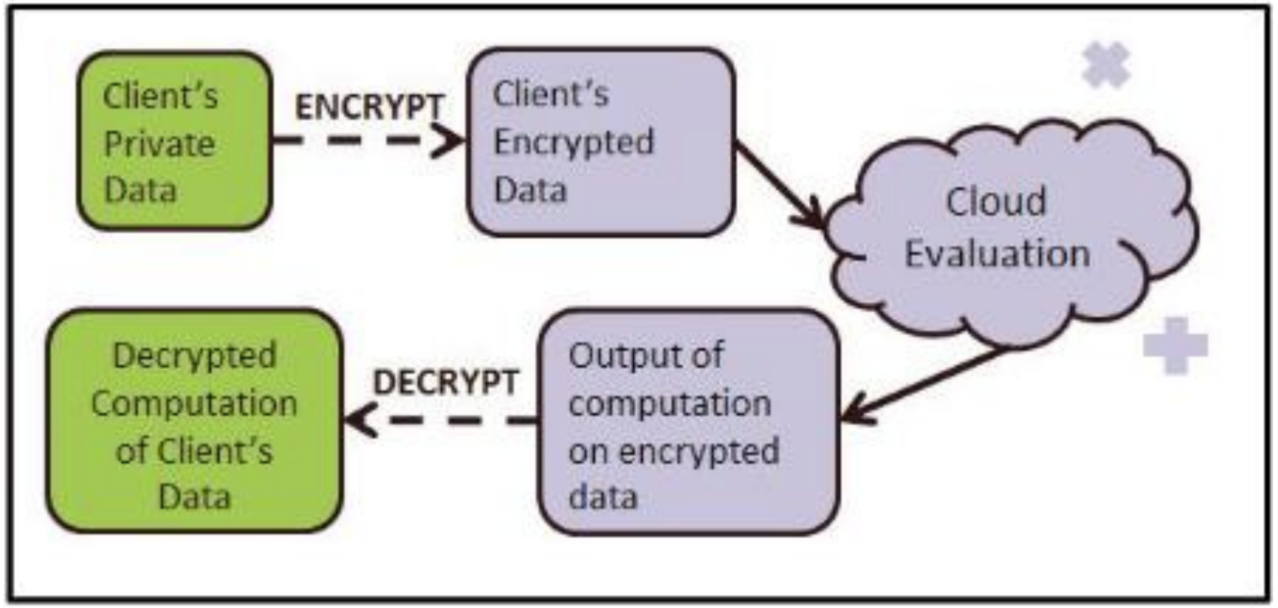
The following FE suffices for $i\mathcal{O}$:

- Single key for a function with long output $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$
- $|CT|$ is sublinear in output length m
- Supporting function class NC^0

How to build it?

Natural Idea: Use LWE

-- Recall: LWE *only* assumption yielding FHE



Expressive Functionality: Supports arbitrary circuits

Compact ciphertext, independent of circuit size

Perfect: Encrypted computation with All or Nothing Decryption

* : up to minor variations

LWE…Leakage on Partial Decryption (FE)

- Using LWE, can support all polynomial sized circuits for FE
- But only for restricted security games
  - Adversary sees limited number of queries [GVW12, GKPVZ13, AR17], restricted types of queries [GVW15], combination of these [A17]
- Attacks against scheme when adversary violates security game [A17]

Causes of Attack and Ways to Overcome them?

Challenge: Leaky LWE Keys

In Most LWE Based FE Constructions

Learning With Errors → Ciphertext

Distinguish “noisy inner products” from uniform

A , s A + e

versus

A , Unif

In Most LWE Based FE Constructions

SIS Problem ➔ Secret Key

Given matrix A, find "short" integer z such that

$A\ z = 0 \bmod q$

$$A \; z = 0 \quad \bmod q$$

Many short vectors form a trapdoor for A

Can be used to break LWE with matrix A

Decryption works

CT $\quad s\,A + e$

when matrices match

SK $\quad A\,z = 0$

We need: Partial Decryption Capability

Encrypt (mpk, x):

LWE encodings of x

$c_1 = \boxed{A_1, x_1}$ …….. $c_n = \boxed{A_n, x_n}$

$c_0 = \boxed{A, 0}$

We need: Partial Decryption Capability

BGG+14 showed homomorphic evaluation algorithms $eval_{ct}$ and $eval_{pk}$ such that:

Encrypt (mpk, x):

LWE encodings of x

$c_1 = \boxed{A_1, x_1}$ …….. $c_n = \boxed{A_n, x_n}$

$c_0 = \boxed{A, 0}$

1. Compute $A_f = Eval_{pk}(A_1 \ldots A_n, f)$

1. Compute $ct^* = Eval_{ct}(c_1 \ldots c_n, f)$

$ct^* = \boxed{[A|A_f], f(x)}$

We need: Partial Decryption Capability

BGG+14 showed homomorphic evaluation algorithms eval_{ct} and eval_{pk} such that:

Encrypt (mpk, x):

LWE encodings of x

$c_1 = \boxed{A_1, x_1}$ …….. $c_n = \boxed{A_n, x_n}$

$c_0 = \boxed{A, 0}$

Keygen(msk, f):

1. Compute $A_f = \text{Eval}_{pk}(A_1 \ldots A_n, f)$
2. Compute short vector z such that

$$\left[A \mid A_f \right] \, z = 0$$

Decrypt (sk_f, ct) ➔ f(x)

1. Compute $ct^* = \text{Eval}_{ct}(c_1 \ldots c_n, f)$

$ct^* = \boxed{[A|A_f]\,,\, f(x)}$

We need: Partial Decryption Capability

BGG+14 showed homomorphic evaluation algorithms eval_{ct} and eval_{pk} such that:

Encrypt (mpk, x):

LWE encodings of x

$c_1 = \boxed{A_1, x_1}$ …….. $c_n = \boxed{A_n, x_n}$

$c_0 = \boxed{A, 0}$

Keygen(msk, f):

1. Compute $A_f = \text{Eval}_{pk}(A_1 \ldots A_n, f)$

2. Compute short vector z such that

$$\begin{bmatrix} A \mid A_f \end{bmatrix} \begin{bmatrix} z \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}$$

Decrypt (sk_f, ct) → f(x)

1. Compute $ct^* = \text{Eval}_{ct}(c_1 \ldots c_n, f)$

$ct^* = \boxed{[A|A_f]\,,\, f(x)}$

Matrices in ct* and key match, can recover f(x) !

Catch: x is not hidden

We need: Partial Decryption Capability

GVW15 showed how to hide x in restricted security game

Encrypt (mpk, x): Use FHE to encrypt x_i
denote by $\hat{x}_i$

$c_1 = \boxed{A_1, \hat{x}_1}$ .... $c_n = \boxed{A_n, \hat{x}_n}$

$c_0 = \boxed{A, 0}$ $c_{sk} = \boxed{\text{FHE.sk}}$

Keygen(msk, f): Let f' = FHE.Dec ∘ f

We need: Partial Decryption Capability

GVW15 showed how to hide x in restricted security game

Encrypt (mpk, x): Use FHE to encrypt x_i
denote by $\hat{x}_i$

$c_1 = \boxed{A_1, \hat{x}_1}$ .... $c_n = \boxed{A_n, \hat{x}_n}$

$c_0 = \boxed{A, 0}$ $c_{sk} = \boxed{\text{FHE.sk}}$

Keygen(msk, f): Let f' = FHE.Dec ∘ f

1. Compute $A_{f'} = \text{Eval}_{PK}(A_1 \ldots A_n, f')$
2. Compute short vector z such that

$$\left\{ A \mid A_{f'} \right\} \boxed{z} = \boxed{0}$$

Decrypt (sk_f, ct) ➔ f(x)

1. Compute $ct^* = \text{Eval}_{ct}(c_1 \ldots c_n, f')$

$ct^* = \boxed{[A \mid A_{f'}],\ f(x)}$

OK to reveal $\hat{x}_i$
Need work to argue that FHE.sk is hidden

Can be done in <u>restricted security game</u>, where Adv may not request any keys such that f(x) =1

Attacks Outside Game[A17]

- Request keys for linearly dependent vectors
- Combine keys to get short vectors, hence trapdoor in certain lattice A*
- Manipulate challenge CT to get LWE sample with matrix B*
- A* and B* only match for keys where f(x)=1
- Lessons: *Inherent vulnerability* for “attribute hiding” scheme with this structure of keys

How do pairings help [GJLS20]?

Can build FE for quadratic functions from pairings [Lin16,BCFG17,G20,Wee20]

(mpk, msk) ← Setup(1^n)

Encrypt (mpk, x = (x_1....x_n)):

ct

Keygen(msk, C = (c_{11}....c_{nn})):

Decrypt (sk_C, ct) outputs

$$\sum_{i,j} c_{ij} x_{ij}$$

No restrictions in the security game!

How do pairings help [GJLS20]?

- Compute ct* = $[A|A_{f'}]$, f(x) as before using evaluation algorithm
- Looking more closely at structure of ct*:

$$ct^* = [A|A_f]^T s + f(x) + noise$$

- Encryptor knows (s, noise) and can provide Linear FE ciphertext for vector (s, noise)
- Key generator knows $[A|\ A_f]$ and can provide Linear FE key for vector ($[A|\ A_f]^T$, 1)
- Decryption recovers inner product ($[A|\ A_f]^T s$ + noise, which can be subtracted from ct* to recover f(x) (upto rounding).

Using Pairing based FE to implement Quadratic (hence Linear) FE prevents the leakage created by LWE secret keys

Doing Without Pairings?

- Linear FE exists from LWE [ABDP15, ALS16] but does not suffice : same key structure
- There are other approaches [A19,AP20], but all suffer from *unsimulatable* key structure –
  - No known attacks but do not admit proof

Challenge: Construct LWE based FE with more secure keys

Difficulty in Reduction for FE.
Does not show up in Functional Encodings

Challenge: How to compute smudging noise

Degree Flattening

Given: LWE encoding of input x (encoding may vary).

Want: to compute a “deep” (say NC_1) circuit f on x, to obtain an encoding of f(x)

Can represent deep computation f as equivalent function f’ such that f’ has public computation of high degree and private computation of low degree

Deep, public computation done publicly, shallow private computation, done using Linear Functional Encryption

Linear Functional Enc [ABDP15,ALS16]

Can build FE for quadratic functions from pairings [Lin16,BCFG17,G20,Wee20]

(mpk, msk) ← Setup(1^n)

Encrypt (mpk, $x = (x_1 \ldots . x_n)$):

ct

Keygen(msk, $y = (y_1 \ldots y_n)$):

Decrypt (sk_y, ct) outputs

$$\sum_{i \in [n]} x_i y_i$$

No restrictions in the security game

More than n key requests
➔ MSK leaked

Symmetric key FHE for Quadratic Polynomials [BV11a]

s: secret key

Encrypt (s, x_1, x_2):

Sample u_1, u_2 randomly in ring. Sample err_1, err_2.

Compute :

$$c_1 = u_1 s + err_1 + x_1$$

$$c_2 = u_2 s + err_2 + x_2$$

Evaluate (c_1, c_2, $f = x_1 x_2$):

Want: Use c_1, c_2 to compute product ciphertext c_{12} that encrypts $x_1 x_2$

FHE Evaluation

We may write:

$$x_1 \approx c_1 - u_1 s$$

$$x_2 \approx c_2 - u_2 s$$

$$\therefore x_1 x_2 \approx c_1 c_2 - (c_1 u_2 + c_2 u_1) s + u_1 u_2 s^2$$

Let $c^{mult} = (c_1 c_2, \quad c_1 u_2 + c_2 u_1, \quad u_1 u_2)$

Decryption $x_1 x_2 \approx \; < (c_1 c_2,\ (c_1 u_2 + c_2 u_1),\ u_1 u_2)\ ;\ (1, -s, s^2) >$

Quadratic Functional Enc [AR17]

- Recall FHE decryption equation:

$$x_1x_2 \approx c_1c_2 - (c_1u_2 + c_2u_1)s + u_1u_2s^2$$

- What if we group the t… differently?

$$\therefore x_1x_2 \approx c_1c_2 - \ldots) + \ldots$$

Known to encryptor

Known to Key Generator

Decryption

$$x_1x_2 \approx c_1c_2 + \langle (c_1s,\ c_2s\ ,\ s^2)\ ;\ (-u_2, -u_1, u_1u_2) \rangle$$

Quadratic Functional Enc [AR17]

Encryptor provides $c_1, \ldots, c_n$ as well as Linear FE encryption of vector

$$(c_1 s,\ c_2 s\ , \ldots . c_n s\ , s^2)$$

Key Generator provides Linear FE key for vector

$$(-u_2, -u_1, 0 \ldots . 0,\ u_1 u_2)$$

Deep Computation is on public encodings

Computing $c_1 c_2$ herself, decryptor can recover :

$$x_1 x_2 \approx c_1 c_2 - u_2(c_1 s) - u_1(c_2 s) + u_1 u_2 (s^2)$$

Key Dependent Computation is Linear

Key Insight: Quadratic terms are $c_i c_j$ which are public
Only 2n ciphertexts instead of n^2

Can be generalized to NC_1 [AR17]

Compactness Vs Leakage

- Supports NC_0 with sublinear ciphertexts
- Last slide: Degree reduction to linear
  - Adversary sees exact linear equations in secrets
  - Too much leakage!
- AJLMS19: Degree reduction to quadratic
  - Adversary sees quadratic equations in secrets
  - May be secure (aka MQ assumption for some distribution)

Degree reduction to Linear Too Much! Quadratic FE from LWE?

Way Forward?

- Don’t have quadratic FE from LWE
- Previously: multivariate quadratic equations may hide secrets
- But… noisy linear equations can also hide secrets

[A19,AJLMS19]:
Suffices to construct FE for linear functions plus noise

FE for linear functions plus noise

Noisy Linear Functional Encryption [A19]

- Recall Linear FE : Enc(x), Keygen(y), Decrypt to get <x,y>.
- Noisy Linear FE : Enc(x), Keygen(y), Decrypt to get <x,y> plus noise
- Special Case via Degree 2.5 FE we saw yesterday

- Where does noise come from?
- What security properties does it need to satisfy?

Noise must satisfy only mild statistical properties

A key observation: Computing a noise term may be easier as exact value not important

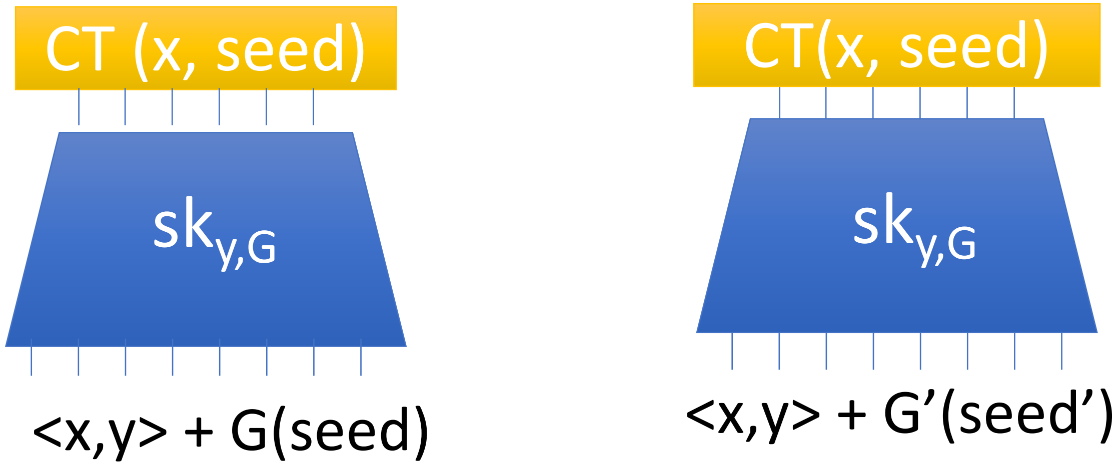
A key Observation: Old grandma advice!

If you cannot have
what you want, you
must learn to want
what you can have

A key Observation:
Relax requirement on correctness

If you cannot compute what you can use, you must learn to use what you can compute

Noisy Linear Functional Encryption [A19]



- Only <x,y> needs to be correct! G(seed) is allowed some corruption
- So far: Assume polynomial is PRG and insist on computing it exactly
- Here: Compute whatever can be computed and check if it can satisfy PRG like properties

Noisy Linear Functional Encryption [A19]

- Let’s try to build it
- From LWE alone, we don’t know how to
- Extend LWE based Linear FE of ALS16 to Noisy Linear FE using new hardness conjectures on lattices.

Let’s see how…

Recap: Regev Public Key Encryption

Recall: Finding short $\vec{e}$ such that $<\vec{a};\vec{e}>=u$ is hard

- SK : $\vec{e}$ PK : $\vec{a},u$

Pseudorandom
By R-LWE

- Encrypt (PK, x) :

$$\vec{c_0} = \vec{a}\cdot s + 2\cdot e\vec{r}r_1$$

$$c_1 = u\cdot s + 2\cdot err_2 + x$$

Small only if
e is small

- Decrypt (SK) :

$$\begin{aligned} c_1 - \langle\vec{e};\vec{c_0}\rangle &= u\cdot s + 2\cdot err_2 + x - u\cdot s - \langle\vec{e}; e\vec{r}r_1\rangle \\ &= x + 2\cdot err \\ &= x \mod 2 \end{aligned}$$

Linear Functional Encryption [ALS16]

MSK: $\vec{e}_1, \ldots \vec{e}_\ell$ (short)

PK: $\vec{a},\ \vec{u} = (u_1, \ldots, u_\ell)$

where $\langle \vec{a};\ \vec{e}_i \rangle = u_i \in R_q$

Linear Functional Encryption [ALS16]

MSK: $\vec{e}_1, \ldots \vec{e}_\ell$ (short)

PK: $\vec{a},\ \vec{u} = (u_1, \ldots, u_\ell)$

where $\langle \vec{a};\ \vec{e}_i \rangle = u_i \in R_q$

Enc(PK, x):

$$\vec{c}_0 = \vec{a} \cdot s + 2 \cdot er\vec{r}_0$$

$$\vec{c}_1 = \vec{u} \cdot s + 2 \cdot er\vec{r}_1 + \vec{x}$$

Linear Functional Encryption [ALS16]

MSK: $\vec{e}_1, \ldots \vec{e}_\ell$ (short)

PK: $\vec{a},\ \vec{u} = (u_1, \ldots, u_\ell)$

where $\langle \vec{a};\ \vec{e}_i \rangle = u_i \in R_q$

Enc(PK, x):

$$\vec{c}_0 = \vec{a} \cdot s + 2 \cdot e\vec{rr}_0$$

$$\vec{c}_1 = \vec{u} \cdot s + 2 \cdot e\vec{rr}_1 + \vec{x}$$

KeyGen(MSK, y):

$$\sum_{i \in [\ell]} y_i\ \vec{e}_i$$

Linear Functional Encryption [ALS16]

MSK: $\vec{e}_1, \ldots \vec{e}_\ell$ (short)

PK: $\vec{a},\ \vec{u} = (u_1, \ldots, u_\ell)$

where $\langle \vec{a};\ \vec{e}_i \rangle = u_i \in R_q$

Enc(PK, x):

$$\vec{c}_0 = \vec{a} \cdot s + 2 \cdot e\vec{rr}_0$$

$$\vec{c}_1 = \vec{u} \cdot s + 2 \cdot e\vec{rr}_1 + \vec{x}$$

KeyGen(MSK, y):

$$\sum_{i \in [\ell]} y_i\ \vec{e}_i$$

Decrypt:

$$\Big(\sum_{i \in [\ell]} y_i\ \vec{e}_i\Big)^\top \cdot \vec{c}_0 = \Big(\sum_{i \in [\ell]} y_i\ \vec{u}_i\Big) \cdot s + 2 \cdot err$$

$$- \quad \vec{y}^T \vec{c}_1 = \Big(\sum_{i \in [\ell]} y_i\ u_i\Big) \cdot s + 2 \cdot err + \langle \vec{x};\ \vec{y} \rangle$$

$$= \quad \langle \vec{x},\ \vec{y} \rangle + 2 \cdot err$$

Note that ..

- Decryption reveals $\langle \vec{x},\ \vec{y} \rangle + 2 \cdot err$: inner product + noise
- Isn’t this noisy linear FE already?

Noise not pseudorandom

Noise is learnt fully after sufficient key requests!

Adding Noise to Linear FE

- Starting point idea: Linear FE computes $\langle \vec{x},\ \vec{y} \rangle$ where $\vec{x}, \vec{y} \in R^{\ell}$
- Add dummy co-ordinate $x[\ell+1] = \text{noise}, \quad y[\ell+1] = 1$
- Now output $\langle \vec{x},\ \vec{y} \rangle + \text{noise}$
- Repeat m times, once for each output bit

Satisfies security, violates succinctness
CT size grows with m

Can we compress encodings of noise ?

- Polynomial for computing noise must be degree at least 3 [LV18, BBKK18]

- Recall: Do not have FE for even degree 2 polynomials from LWE

- Is approximate computation easier?

Is approximate computation easier?
Or, Enter NTRU

Let $R = Z[x]/\langle x^n + 1\rangle$, $p_1 < p_2$ primes, $R_{p_1} = R/(p_1 \cdot R)$, $R_{p_2} = R/(p_2 \cdot R)$

$$\boxed{\text{Want to compute } d = h \cdot s + p_1 \cdot err + noise}$$

“noise” is message!

For $i \in \{1, \ldots, w\}$, sample f_{1i}, f_{2i} and g_1, g_2 from a discrete Gaussian over ring R. Set

$$h_{1i} = \frac{f_{1i}}{g_1}, \quad h_{2j} = \frac{f_{2j}}{g_2} \in R_{p_2} \;\; \forall \;\; i, j \in [w]$$

Assume these look random.

Note difference from NTRU: Reusing denominator!

RLWE with Structured Noise

Discrete Gaussian

$$\text{Want to compute } d = h \cdot s + p_1 \cdot err + noise$$

Sample

$$e_{1i} \leftarrow \widehat{\mathcal{D}}(\Lambda_2), \quad \text{where } \Lambda_2 \triangleq g_2 \cdot R. \text{ Let } \boxed{e_{1i} = g_2 \cdot \xi_{1i}} \in \text{small},$$

$$e_{2i} \leftarrow \widehat{\mathcal{D}}(\Lambda_1), \quad \text{where } \Lambda_1 \triangleq g_1 \cdot R. \text{ Let } \boxed{e_{2i} = g_1 \cdot \xi_{2i}} \in \text{small},$$

Recall

$$\boxed{h_{1i} = \frac{f_{1i}}{g_1}, \quad h_{2j} = \frac{f_{2j}}{g_2}}$$

We have that: $h_{1i} \cdot e_{2j} = f_{1i} \cdot \xi_{2j}, \quad h_{2j} \cdot e_{1i} = f_{2j} \cdot \xi_{1i} \in$ small

RLWE with Structured Noise

$\boxed{\text{Want to compute } d = h \cdot s + p_1 \cdot err + noise}$

We showed: $h_{1i} \cdot e_{2j} = f_{1i} \cdot \xi_{2j}, \quad h_{2j} \cdot e_{1i} = f_{2j} \cdot \xi_{1i} \quad \in$ small

Compute encodings of "PRG seed" :

$$d_{1i} = \boxed{h_{1i}} \cdot t_1 + p_1 \cdot \boxed{e_{1i}} \in R_{p_2}$$
$$d_{2i} = \boxed{h_{2i}} \cdot t_2 + p_1 \cdot \boxed{e_{2i}} \in R_{p_2}$$

Multiply encodings:

$$d_{1i} \cdot d_{2j} = (h_{1i} \cdot h_{2j}) \cdot (t_2\ t_2) + p_1 \cdot \text{noise}$$

As desired!

where $\text{noise} = p_1 \cdot (f_{1i} \cdot \xi_{2j} \cdot t_1 + f_{2j} \cdot \xi_{1i} \cdot t_2 + p_1 \cdot g_1 \cdot g_2 \cdot \xi_{1i} \cdot \xi_{2j}) \in$ small

RLWE with Structured Noise

Noise lives in an ideal that "cancels" large term in RLWE sample

Extends to higher degree

"Theorem": Its easy to make noise!

Description oversimplified. Please see paper [A19]

Security

- Proof from clumsy assumption in overly weak security game
  - Adversary only gets single ciphertext
- Security based on inability to find attacks ☹ [A19,AP20]
- Hurdles in proof:
  - Compressed PK is correlated $d_{1i} \cdot d_{2j} = (h_{1i} \cdot h_{2j}) \cdot t + p_1 \cdot noise$
  - Don't know to simulate secret keys (short preimages) for correlated images

$$\langle \vec{a};\ \vec{e}_{ij} \rangle = h_{1i} \cdot h_{2j}$$

  - Interactive assumption in general
    - can be made non-interactive if Adv only gets one CT

Connection with Functional Encodings [WW20]

- Functional encodings are akin to functional encryption with *single* ciphertext
  - “Open” (counterpart of keygen) can have message x as input
- Assumption in A19 can be made non-interactive for this setting
- As is, does not achieve compression required by WW20
- Can be modified to do so (schemes can be seen as duals)
  - But leakage/correlation in noise inherent to both
  - Does not improve WW20 assumption, even for functional encodings
- But gives Functional Encryption, which is stronger

Summary: Three Nuggets for Thought

How to Strengthen LWE keys

How to generate smudging noise using only linear FE?

Can we perform approximate computation more easily?

Open Problems

- Replace pairings with some weaker structure that can be built from LWE?
- New, simpler, plausible assumptions from lattices? Chart territory between LWE and multilinear map assumptions?
- Use idea that noise computation need not be exact?
- Build post quantum FE and base applications on this?

Thank You

Images Credit:
M C Escher
Hans Hoffman
Jackson Pollock