# CSE208: Advanced Cryptography

Daniele Micciancio

UCSD

Fall 2020

Section 1

# Introduction

# CSE208: Advanced Cryptography

- Graduate Level Advanced Cryptography
- Prerequisites:
  - CSE207 or equivalent
  - Solid theoretical background, cryptographic definitions, etc.
  - Some programming

# CSE208: Advanced Cryptography

- Graduate Level Advanced Cryptography
- Prerequisites:
  - CSE207 or equivalent
  - Solid theoretical background, cryptographic definitions, etc.
  - Some programming
- Past topics: Zero Knowledge, Functional Encryption, Secure Computation, etc.
- Not required: CSE206A (Lattice Algorithms)

# CSE208: Advanced Cryptography

- Graduate Level Advanced Cryptography
- Prerequisites:
  - CSE207 or equivalent
  - Solid theoretical background, cryptographic definitions, etc.
  - Some programming
- Past topics: Zero Knowledge, Functional Encryption, Secure Computation, etc.
- Not required: CSE206A (Lattice Algorithms)
- Reading:
  - no textbook
  - mostly research papers
  - see course webpage, canvas, etc.

# Fall 2020 Topic

- Fully Homomorphic Encryption:
  - Encryption schemes that supports the evaluation of arbitrary programs on encrypted inputs
- Applications:
  - secure outsourced computing
  - building block for MPC and more

# Brief History of Homomorphic Encryption

- 1978: Rivest, Adleman & Dertouzos posed the problem
- 2009: Gentry 2009 proposed the first candidate solution
- 2010-2020: Work towards more efficient solutions based on standard complexity assumptions (Brakerski, Vaikuntanathan, Gentry, Halevi, Smart, . . . )

# Software libraries

- IBM HElib (Halevi & Shoup)
- Microsoft SEAL
- NJIT/Duality PALISADE (Rohloff, Cousins & Polyakov)
- Functional Lattice Cryptography LoL (Crockett & Peikert)
- Fastest FHE of the West FHEW (Ducas & Micciancio)
- FHE over the Torus TFHE (Chillotti, Gama, Georgieva & Izabachene)
- Approximate FHE HEAAN (Cheon, Kim, Kim & Song)

# Software libraries

- IBM HElib (Halevi & Shoup)
- Microsoft SEAL
- NJIT/Duality PALISADE (Rohloff, Cousins & Polyakov)
- Functional Lattice Cryptography LoL (Crockett & Peikert)
- Fastest FHE of the West FHEW (Ducas & Micciancio)
- FHE over the Torus TFHE (Chillotti, Gama, Georgieva & Izabachene)
- Approximate FHE HEAAN (Cheon, Kim, Kim & Song)

- In the News:
  - February 21, 2019: Microsoft SEAL open source homomorphic encryption library gets even better for .NET developers!
  - June 4, 2020: IBM releases FHE toolkit for MacOS and iOS; Linux and Android Coming Soon

# Homework and Evaluation

- Homework assignments:
  - 3 assignments, due within one week from assignment date
  - Cover theoretical/mathematical topics

# Homework and Evaluation

- Homework assignments:
  - 3 assignments, due within one week from assignment date
  - Cover theoretical/mathematical topics
- Small Project:
  - Goal: Try to use one of the many HE libraries
  - Not much coding, but you will have to write and compile a few lines of code
  - Evaluated primarily based on written report

# Administrivia:

- Course webpage: http://cseweb.ucsd.edu/classes/fa20/
  - general course information
  - pointers to papers and other reading material
- Canvas:
  - recording of lectures
  - homework distribution/collection
  - grades
  - discussion board

# Course Schedule (Tentative)

Week 1: Introduction and Definition

- FHE Definition
- Gentry's Boostrapping theorem
- Homework 1 out

Week 2-4: Fundamental techniques based on general lattices

- LWE encryption
- Linear Homomorphic computations
- Key Switching and Proxy re-encryption
- Nested encryption and homomorphic multiplication
- Ciphertext Tensoring and homomorphic multiplication
- Homomorphic Decryption and Bootstrapping algorithms
- Homework 2 out

Week 5: Algebraic Number Theory

- I really hope you like math!
- Homework 3 out

Week 6-10: Efficient FHE from Ring LWE

- Message packing techniques
- Linear transformations on structured matrices
- Other FHE schemes: GHS, BFV,FHEW, AP13, TFHE, CKKS . . .

Section 2

# Defining FHE

# Public Key Encryption

```
PKE(Gen,Enc,Dec)
  Gen: () → (pk,sk)
  Enc: (pk,m) → c
  Dec: (sk,c) → m
```

# Correctness of PKE

For every $\texttt{(sk,pk)} \leftarrow \texttt{Gen()}$ and $\texttt{m} \leftarrow \texttt{[M]}$, $\texttt{r} \leftarrow \texttt{[R]}$:

```
Dec(sk,Enc(pk,m;r)) = m
```

# Chosen Plaintext Attack (CPA) security

- Ciphertext Indistinguishability under Chosen Plaintext Attack
- Experiment:

```
INDCPAgame(b:{0,1})
    (sk,pk) ← Gen()
    A(pk) → (m_0,m_1)
    b' ← A(Enc(pk,m_b))
    return b':{0,1}
```

# Chosen Plaintext Attack (CPA) security

- Ciphertext Indistinguishability under Chosen Plaintext Attack
- Experiment:

```
INDCPAgame(b:{0,1})
    (sk,pk) ← Gen()
    A(pk) → (m_0,m_1)
    b' ← A(Enc(pk,m_b))
    return b':{0,1}
```

**Definition**

`Adv(A) = |Pr(Game(0)=1) - Pr(Game(1)=1)|`

**Definition**

An encryption scheme `(Gen,Enc,Dec)` is **IND-CPA** secure if any polynomial time A has advantage `Adv(A)~ 0`

# Significance of CPA security

- Adversary can choose messages $m_0$, $m_1$
  - No assumption about input distribution
  - Adversary may have partial information about messages
  - Adversary may influence the choice of messages
- Ciphertext $\mathtt{c} = \mathtt{Enc}(\mathtt{pk}, m_b)$ is computed honestly
  - Adversary cannot tamper with ciphertexts
- Adversary models a passive attacker

# Definition of CCA security

**Definition**

An encryption scheme (Gen,Enc,Dec) is **IND-CCA** secure if any polynomial time A has advantage Adv(A) ≈ 0 in the following game.

```
Game(b:{0,1})
    (sk,pk) ← Gen()
    A[D](pk) → (m0,m1)
    c ← Enc(pk,mb)
    b' ← A[D'](c)
    return b':{0,1}
```

- A[D] is an adversary with oracle access to

  D(x) = Dec(sk,x)

- A[D'] uses a modified oracle (next slide)

# IND-CCA1 vs IND-CCA2

There are two variants of CCA security, depending on the type of oracle given to the adversary after receiving the challenge ciphertext:

- **IND-CCA1** security: No decryption oracle after receiving the challenge

```
D'(x) = Nil
```

- **IND-CCA2** security: decrypt any ciphertext, except the challenge `c`

```
D'(x) =
    if (x =? c)
        then Nil
        else Dec(sk,x)
```

# Significance of CCA security

- Goal: model active attacks, where adversary can tamper with ciphertexts
- Standard notion for regular encryption schemes
- IND-CCA2 theoretically equivalent to *non-malleable* encryption
  - Any attempt to modify a ciphertext should be detected

# Significance of CCA security

- Goal: model active attacks, where adversary can tamper with ciphertexts
- Standard notion for regular encryption schemes
- IND-CCA2 theoretically equivalent to *non-malleable* encryption
  - Any attempt to modify a ciphertext should be detected
- Seems incompatible with homomorphic encryption
  - Ability to modify ciphertexts can be a useful feature
  - Homomorphic encryption is *perfectly malleable*
- We will not consider CCA security

# Homomorphic Encryption: first attempt

- Assume `f: M` $\rightarrow$ `M`

  `f(Enc(pk,m)) = Enc(pk,f(m))`

  `Eval(pk,f, Enc(pk,m)) = Enc(pk,f(m))`

# Homomorphic Encryption: second attempt

```
Dec(sk,Eval(pk,f, Enc(pk,m))) =  f(m)
```

# Multi-input functions

- Many inputs are encrypted independently

$$c_1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1)$$
$$\ldots$$
$$c_k \leftarrow \mathsf{Enc}(\mathsf{pk}, m_k)$$

# Multi-input functions

- Many inputs are encrypted independently

```
c_1 ← Enc(pk, m_1)
...
c_k ← Enc(pk, m_k)
```

- $k$-ary function `f: (m_1, ..., m_k) → m`

```
Eval(pk, f, c_1, ..., c_k))
     = Enc(pk, f(m_1, ..., m_k)) ???

Dec(sk, Eval(pk, f, c_1, ...c_k))
     = f(m_1, ..., m_k)
```

# Multi-key Homomorphic encryption

- Assume multiple users: $P_1, P_2, \ldots$
- Each user has a key (pair): $P_i : (pk_i, sk_i)$
- Data is encrypted and sent to different users

$$\begin{aligned} c_1 &\leftarrow \mathsf{Enc}(pk_1, m_1) \\ &\cdots \\ c_t &\leftarrow \mathsf{Enc}(pk_t, m_t) \end{aligned}$$

- Users pool data together to perform a joint computation on $c_1, \ldots, c_t$

# Multi-key Homomorphic encryption

- Assume multiple users: $P_1, P_2, ...$
- Each user has a key (pair): $P_i : (pk_i, sk_i)$
- Data is encrypted and sent to different users

$c_1 \leftarrow \texttt{Enc}(pk_1, m_1)$
$...$
$c_t \leftarrow \texttt{Enc}(pk_t, m_t)$

- Users pool data together to perform a joint computation on $c_1, ..., c_t$

- Final result is an encryption of $f(m_1, ..., m_t)$ under what key?

```
Eval(???, f, c_1, ..., c_t))
    ~ Enc(???, f(m_1, ..., m_t))
```

# Restricting Homomorphic Encryption

- FHE is a useful and challenging problem already in the single key setting

# Restricting Homomorphic Encryption

- FHE is a useful and challenging problem already in the single key setting
- In order to appropach the problem we will further restrict it by parametrizing by a set of allowed computations/functions `Func = {f: ....}` where each `f: (M,...,M)`$\rightarrow$`M` may take a different number of arguments

# Restricting Homomorphic Encryption

- FHE is a useful and challenging problem already in the single key setting
- In order to appropach the problem we will further restrict it by parametrizing by a set of allowed computations/functions `Func = {f: ....}` where each `f: (M,...,M)`$\rightarrow$ `M` may take a different number of arguments
- More generally, one many consider functions `f:`$(M_1, \ldots, M_k)$ $\rightarrow$`M` taking inputs from different sets (types), e.g., `ifThenElse: (Bool,Int,Int)`$\rightarrow$`Int`

# Examples and Function Composition

- $(M, +, 0)$: abelian group, e.g., "fixed size" integers (modulo N)
- Addition: $f(x_1, ...x_t) = x_1 + ... + x_t$
- Scalar multiplication: $g_a(x) = a \cdot x$
- Linear combinations: $h(x_1, ...x_t) = \sum_i 2^{i-1} x_i$

# Examples and Function Composition

- $(M, +, 0)$: abelian group, e.g., "fixed size" integers (modulo N)
- Addition: $f(x_1, \ldots x_t) = x_1 + \ldots + x_t$
- Scalar multiplication: $g_a(x) = a \cdot x$
- Linear combinations: $h(x_1, \ldots x_t) = \sum_i 2^{i-1} x_i$

- 1-hop, n-hop, multi-hop: can functions $f$ be composed?

$$h(x_1, \ldots, x_t) = f(g_1(x_1), \ldots, g_{2^t-1}(x_t))$$

# Correctness of Function Composition

- Let $x, y, z \in M$ be messages and $f, g : M \to M$ two functions such that $y = f(x)$ and $z = g(y) = (g \circ f)(x)$
- Assume (`Gen`,`Enc`,`Dec`,`Eval`) can evaluate $f$ and $g$ correctly:

```
Dec(sk,Eval(pk,f, Enc(pk,x))) = f(x)
Dec(sk,Eval(pk,g, Enc(pk,y))) = g(y)
```

# Correctness of Function Composition

- Let $x, y, z \in M$ be messages and $f, g : M \to M$ two functions such that $y = f(x)$ and $z = g(y) = (g \circ f)(x)$
- Assume (`Gen`,`Enc`,`Dec`,`Eval`) can evaluate $f$ and $g$ correctly:

```
Dec(sk,Eval(pk,f, Enc(pk,x))) = f(x)
Dec(sk,Eval(pk,g, Enc(pk,y))) = g(y)
```

**Question**

*Does it follow that*

```
ctX ← Enc(pk,x)
ctY ← Eval(pk,f,ctX)
ctZ ← Eval(pk,g,ctY)
```

$\texttt{Dec(sk,ctZ)} \stackrel{?}{=} \texttt{z}$

# Formalizing Restricted Composition

- Restrict scheme to a set $\mathcal{F}$ of strongly typed functions:

$$f : M_1 \times \ldots M_k \to M_0$$

- `Enc`,`Dec`,`Eval` are given type information

# Formalizing Restricted Composition

- Restrict scheme to a set $\mathcal{F}$ of strongly typed functions:

$$f : M_1 \times \ldots M_k \to M_0$$

- `Enc`,`Dec`,`Eval` are given type information
- We can use types to bound computation depth:
  - Start from $f : M \to M$
  - Define $M_i = M$ for $i = 1, ..., n$
  - Define $f_i : M_i \to M_{i+1}$, where $f_i(x) = f(x)$
- $\mathcal{F} = \{f\}$ allows arbitrary composition
- $\mathcal{F} = \{f_0\}$: no composition
- $\mathcal{F} = \{f_0, f_1, ..., f_n\}$: bounded depth composition

# (Multi-hop) Correctness Game

- State: (initially empty) list $L$ of message-ciphertext pairs

```
CorrectFHEgame() = (sk,pk) ← Gen()
                   L ← []
                   A[E,F](pk)
                   (m,c) ← last(L)
                   return (Dec(sk,c) ≠ m)

E(m) = c ← Enc(pk,m)
       L ← L;(m,c)
       return c
F(f,I) = (ms,cs) ← unzip L[I]
         m ← f(ms)
         c ← Eval(pk,f,cs)
         L ← L;(m,c)
         return c
```

## Terminology

Reading papers, you will find references to

- Fully Homomorphic Encryption
- Somewhat Homomorphic Encryption
- Leveled Fully Homomorphic Encryption
- etc.

# Terminology

Reading papers, you will find references to

- Fully Homomorphic Encryption
- Somewhat Homomorphic Encryption
- Leveled Fully Homomorphic Encryption
- etc.

We will use FHE as a catchall term

- Definition is parametrized by a set of functions $\mathcal{F}$
- Functions in $\mathcal{F}$ can be composed only if their types match
- $\mathcal{F}$ is closed under composition
- Can use "phantom" types to limit composition

We will rarely define $\mathcal{F}$ formally, but it is a useful exercise

# Security of Homomorphic Encryption

```
INDCPAgame(b:{0,1})
    (sk,pk) ← Gen()
    A(pk) → (m_0,m_1)
    return A(Enc(pk,m_b)): {0,1}
```

**Remark**

*The IND-CPA security definition depends only on* `Gen` *and* `Enc`*, but not on* `Dec` *(or* `Eval`*)*

**Question**

*Can the IND-CPA security definition be applied as it is to FHE schemes* (`Gen`,`Enc`,`Dec`,`Eval`)*?*

# A trivial FHE scheme

Consider the following FHE scheme:

- Let (`Gen`,`Enc`,`Dec`) be IND-CPA secure
- Define `TrivialFHE = (Gen,Enc',Dec',Eval)`

```
Enc'(pk,m) = (Enc(pk,m),[])
Dec'(sk,(ct,[])) = Dec(sk,ct)
Dec'(sk,(ct,[f;fs])) = f(Dec'(sk,(ct,fs)))
Eval(pk,f,(ct,[fs])) = (ct,[f;fs])
```

**Question**

- *Is* `TrivialFHE` *a correct FHE scheme?*
- *Is* `TrivialFHE` *a secure FHE scheme?*
- *What makes the above scheme "trivial"?*

# Compactness

- The `TrivialFHE` scheme is both correct and secure
- The problem with `TrivialFHE` is that it is not efficient:
    - Computation is performed by `Dec`, not `Eval`!

**Definition**

A FHE scheme is **compact** if the size of ciphertext `ct = Eval(pk,f,Enc(pk,m))` is independent of `Size(f)`

- Weaker forms of compactness:
    - Ciphertext size may grow logarithmic with `Size(f)`
    - Ciphertext size may depend on `Depth(f)`

# Function Privacy

```
f0(x,y) = x + y
f1(x,y) = y + x

Game[A](b: {0,1})
  (sk,pk) ← Gen()
  ctX ← Enc(pk,x)
  ctY ← Enc(pk,y)
  ct ← Eval(pk,fb,ctX,ctY)
  return A(ct)
```

**Question**

*Assume* (Gen,Enc,Dec,Eval) *is a secure FHE scheme. Can an efficient adversary* A *recover the bit* b = Game[A](b) *?*

# Passive Attacks to FHE

```
Game[A](b: {0,1})
  (sk,pk) ← Gen()
  State ← []
  b' ← A[E,D,F](pk)
  return b'
```

Adversary has access to three stateful oracles:

- Encryption oracle: $\mathtt{E}(m_0, m_1)$
- Function Evaluation oracle: $\mathtt{F}(f_0, f_1, \mathtt{I})$
- Decryption oracle: `D(i)`
- Joint State: List of message-message-ciphertext triplets $(m_0, m_1, \mathtt{ct})$

# Passive Attack (oracles)

```
E(m0,m1) = ct ←Enc(pk,mb)
           State ← (State;(m0,m1,ct))
           return ct

F(f0,f1,I) = (ms0,ms1,cts) ←unzip State[I]
             ct ← Eval(pk,fb,cts)
             m0 ←f0(ms0)
             m1 ←f1(ms1)
             State ← State;(m0,m1,ct)
             return ct

D(i): (m0,m1,ct) ←State[i]
      if (m0 ≡m1)
        then return Dec(sk,ct)
        else return Nil
```

# Passive Attack with/without function privacy

- The game we just described guarantees function privacy
- A similar definition without function privacy can be obtained by requiring `f0≡f1` in the function evaluation queries

```
F'(f,I): (ms_0,ms_1,cts) ←unzip State[I]
         ct ← Eval(pk,f,cts)
         m_0 = f(ms_0)
         m_1 = f(ms_1)
         State ← (State;(m_0,m_1,ct))
         return ct
```

# Example: Circuit Privacy

- Assume messages are single bits `m: {0,1}`
- Let `FHE=(Gen,Enc,Dec,Eval)` a function private FHE scheme supporting `NAND(x,y)= not (x && y)`
- `EvalC(pk,C,...)`: evaluates boolean circuit `C:` $\{0,1\}^n$ $\rightarrow$`{0,1}` one gate at a time using `Eval(pk,NAND,...)`
- Let $C_0, C_1$ : NAND circuits with the same number of inputs and NAND gates
- `(sk,ps)` $\leftarrow$ `Gen()`
- Let $xs_0, xs_1$ be input bits such that $C_0(xs_0) = C_1(xs_1)$

**Question**

*Are the following two distributions indistinguishable?*

(pk , EvalC(pk , $C_0$ , Enc(pk , $xs_0$)))
(pk , EvalC(pk , $C_1$ , Enc(pk , $xs_1$)))

Section 3

# Bootstrapping

# Bootstrapping

- For simplicity: fix message space to $\{0, 1\}$
- `HE=(Gen,Enc,Dec,Eval)`
  - Homomorphic functions: `Func = {` **nand** `}`
  - Supports only bounded computations: `Depth(C)< D`

# Bootstrapping

- For simplicity: fix message space to $\{0, 1\}$
- `HE=(Gen,Enc,Dec,Eval)`
  - Homomorphic functions: `Func = {` **nand** `}`
  - Supports only bounded computations: `Depth(C)< D`

**Question**

*Can we use* `HE` *to build a FHE scheme supporting arbitrary circuits/functions?*

- The process of building FHE from HE is called "bootstrapping"

# Decryption as a boolean function

- Everything is a sequence of bits
  - Secret key `sk`: $\{0,1\}^k$
  - Ciphertext `ct`: $\{0,1\}^l$
- `Dec(sk,ct): {0,1}`

# Decryption as a boolean function

- Everything is a sequence of bits
    - Secret key `sk`: $\{0,1\}^k$
    - Ciphertext `ct`: $\{0,1\}^l$
- `Dec(sk,ct): {0,1}`
- Usually we think of `Dec` as a function
    - described by secret key `sk`
    - mapping ciphertext `ct` to message bit `Dec(sk,ct): {0,1}`

# Decryption as a boolean function

- Everything is a sequence of bits
  - Secret key `sk`: $\{0,1\}^k$
  - Ciphertext `ct`: $\{0,1\}^l$
- `Dec(sk,ct): {0,1}`
- Usually we think of `Dec` as a function
  - described by secret key `sk`
  - mapping ciphertext `ct` to message bit `Dec(sk,ct): {0,1}`
- But we can also think of `Dec` as a function
  - described by ciphertext `ct`
  - mapping secret key `sk` to message bit `Dec(sk,ct): {0,1}`

# Homomorphic Decryption

- Fix a ciphertext `c`
- Define $f_c : sk \mapsto Dec(sk, c)$
- Assume $Size(f_c) < S$, $Depth(f_c) < D$
- Let `bk[1..k] = Enc(pk,sk[1..k])`

**Question**

*What is the result of the following computation?*

`EvalC(pk,`$f_c$`,bk[1..k])`

# Proxy Re-encryption

- Primary key: `(pk,sk)`
- Secondary key: `(pk1,sk1)`
- Re-encryption key: `rk = Enc(pk1,sk[1..k])`
- Input ciphertext `c = Enc(pk,m)`
- Decryption function $f_c$`(sk) = Dec(sk,c)`

**Question**

*What is the result of the following computation?*

`EvalC(pk1,`$f_c$`,rk)`

# Decrypt and compute (unary)

- Homomorphic Encryption `(Gen,Enc,Dec,Eval)`
- Assume `Func = {` $f_c$ `| c: CipherText }` where

$f_c$`(sk) = not (Dec(sk,c))`

# Decrypt and compute (unary)

- Homomorphic Encryption (`Gen`,`Enc`,`Dec`,`Eval`)
- Assume `Func = {` $f_c$ `| c: CipherText }` where

$f_c$`(sk) = not (Dec(sk,c))`

```
(pk,sk) ← Gen()
ek  = Enc(pk,sk)
c   = Enc(pk,m)
```

**Question**

*What is the result of the following computation?*

`EvalC(pk,`$f_c$`,ek)`

# Decrypt and compute (binary)

- Homomorphic Encryption (`Gen`,`Enc`,`Dec`,`Eval`)
- Assume `Func = {` $f_{c,c'}$ `| c,c': CipherText }` where

$f_{c,c'}$`(sk) = Dec(sk,c)` **`nand`** `Dec(sk,c')`

# Decrypt and compute (binary)

- Homomorphic Encryption (Gen,Enc,Dec,Eval)
- Assume `Func = {` $f_{c,c'}$ `| c,c': CipherText }` where

$f_{c,c'}$`(sk) = Dec(sk,c)` **nand** `Dec(sk,c')`

```
(pk,sk) ← Gen()
ek  ← Enc(pk,sk)
c   ← Enc(pk,m)
c'  ← Enc(pk,m')
```

**Question**

*What is the result of the following computation?*

`EvalC(pk,`$f_{c,c'}$`,ek)`

# Bootstrapping

- Given (1-hop) `(Gen,Enc,Dec,Eval)` supporting functions

  $f_{c,c'}$`(sk) = Dec(sk,c)` **`nand`** `Dec(sk,c')`

# Bootstrapping

- Given (1-hop) (`Gen`,`Enc`,`Dec`,`Eval`) supporting functions

  $f_{c,c'}$`(sk) = Dec(sk,c)` **`nand`** `Dec(sk,c')`

- Define (multi-hop) FHE scheme with `Func = {` **nand** `}`

```
Gen'() = (sk,pk) ← Gen()
  ek ← Enc(pk,sk)
  return (sk,(pk,ek))

Enc'((pk,ek),m) = Enc(pk,m)

Eval'((pk,ek),nand,c,c')
  = EvalC(pk,f_{c,c'},ek)
```

# Correctness

Let `(Gen',Enc',Dec,Eval')` be the new encryption scheme

**Theorem**

*If* `Dec(sk,c)= m` *and* `Dec(sk,c')= m'`*, then*

`Dec(sk,Eval'((pk,ek),`**`nand`**`,c,c') = m `**`nand`**` m'`

# Correctness

Let `(Gen',Enc',Dec,Eval')` be the new encryption scheme

**Theorem**

*If* `Dec(sk,c)= m` *and* `Dec(sk,c')= m'`, *then*

```
Dec(sk,Eval'((pk,ek),nand,c,c') = m nand m'
```

Strong correctness property:

```
Dec(sk,Eval'((pk,ek),nand,c,c')
    = Dec(sk,c) nand Dec(sk,c')
```

for **any** ciphertexts `c,c'` !

# Security

- Assume `FHE = (Gen,Enc,Dec,Eval)` is IND-CPA secure
- Build new scheme `FHE'`:

```
Gen'() = (sk,pk) ← Gen()
  ek ← Enc(pk,sk)
  return (sk,(pk,ek))

Enc'((pk,ek),m) = Enc(pk,m)
```

Is `FHE'` IND-CPA secure?

# Leveled Homomorphic Encryption

- Goal: build a FHE supporting NAND circuits of depth up to L, for any given L
- Key generation procedure takes `L` as input:

# Leveled Homomorphic Encryption

- Goal: build a FHE supporting NAND circuits of depth up to L, for any given L
- Key generation procedure takes `L` as input:

```
Gen'(L) =
  for (i=0..L)
    (sk[i],pk[i]) ← Gen()
  for (i=1..L)
    ek[i] = Enc(pk[i],sk[i-1])
  sk' = sk[0..L]
  pk' = pk[0..L],ek[1..L]
  return (sk',pk')

Enc'(pk',m) = Enc(pk[0],m)
```

# FHE Today

## State of the art

We can build leveled FHE from standard LWE assumption

- Built using bootstrapping
- Inefficient, but better than nothing

## Open problem

Build (non-leveled) FHE from standard LWE

- In practice, one can apply bootstrapping with
  `ek = Enc(pk,sk)`
- Much smaller key than leveled FHE
- No known attacks to circular security
- Still, it is not known how to prove security

Section 4

## LWE

# Linear equations

- $q$: integer modulus
- $\mathbb{Z}_q$: integers modulo $q$
- $A \in \mathbb{Z}_q^{n \times m}$: matrix
- $b \in \mathbb{Z}_q^n$

**Problem**

*Given $A, b$, find $x \in \mathbb{Z}^m$ such that $Ax = b \pmod{q}$*

**Problem**

*Given $A, b$, find $x \in \{0,1\}^m$ such that $Ax = b \pmod{q}$*

**Question**

*Which problem can be efficiently solved?*

# Worst-case vs Average-case hardness

**Problem**

*Given* $A, b$*, find* $x \in \{0,1\}^m$ *such that* $Ax = b \pmod q$

- NP-hard: no polynomial time algorithm unless P=NP
- Is it hard to solve on the average?
- For what probability distribution?
  - $A \leftarrow \mathbb{Z}_q^{n\times m}$
  - $x \leftarrow \{0,1\}^m$
  - $b = Ax \pmod q$
- Is $f : (A, x) \mapsto (A, Ax \bmod q)$ is a *One-Way Function*?
- For what values of $n, m, q$?

# One-Way Functions

## Definition

$f : D \to R$ is a one-way function if for any PPT algorithm $I$

`Pr{InvertGame(I)}` $\approx$`0` where

```
InvertGame:
  x ← D
  y = f(x)
  x' ← I(y)
  return (f(x') ≟ y)
```

- $D = \mathbb{Z}_q^{n \times m} \times \{0, 1\}^m$
- $R = \mathbb{Z}_q^n$
- $f(A, x) = Ax \bmod q$
- Asymptotics: $q(m) = 2^{poly(m)}$, $n(m) = poly(m)$

# One-Way?

- $A \leftarrow \mathbb{Z}^{n \times m}$
- $x \leftarrow \{0,1\}^m$
- $f(A,x) = Ax \bmod q$

**Question**

*Is $f$ a one-way function when*

1. $q = 2^m,\ n = m$
2. $q = 2^m,\ n = m/2$
3. $q = m,\ n = m/2$
4. $q = m,\ n = \sqrt{m}$

# Short Integer Solution (SIS) problem

- Parameters:
  - modulus $q$
  - dimensions $n < m$
  - bound $\beta$

**Problem**

*SIS: Given $A \in \mathbb{Z}_q^{n\times m}$ and $b \in \mathbb{Z}_q^n$, find $x \in \mathbb{Z}^m$ such that $Ax = b \pmod{q}$ and $\|x\| \leq \beta$*

- More generally: $x \in S \subset \mathbb{Z}^m$
- Special cases:
  - $S = \{x : \|x\| \leq \beta\}$
  - $S = \{0,1\}^m$
  - $S = \{x : \|x\|_\infty \leq \beta\}$

# Systematic Form

- Assume $n < m$ (e.g., $n = m/2$)
- Let $A = [I, A'] \in \mathbb{Z}^{n \times m}$ for some $A' \in \mathbb{Z}^{n \times (m-n)}$

**Lemma**

*If SIS is hard, then SIS' is hard*

# Learning With Errors

- SIS': $A = [I, A'] \in \mathbb{Z}^{n \times m}$ where $n < m$ (say, $n = m/2$)
- Let $x = (e, s)$
- $Ax = [I, A'](e, s) = A's + e$

**Problem**

*LWE: Given $A'$ and $b$, find small $e, s$ such that $A's + e = b$*

**Problem**

*LWE: Given $A'$ and $b$, find small $e, s$ such that $A's \approx b$*

Notice:

- $A' \in \mathbb{Z}_q^{n \times n}$
- $A's = b$ is easy to solve
- $A's \approx b$ seems hard

# LWE problem

Notation:

- secret $s \leftarrow \mathbb{Z}_q^n$, usually chosen at random
- modulus $q(n) = poly(n)$
- $A \leftarrow \mathbb{Z}_q^{m \times n}$
- error $e \leftarrow \chi^m$, usually $|e_i| \approx \sqrt{n}$
- $b = As + e \in \mathbb{Z}_q^m$

**Problem**

*Search LWE: Given A and b, find s*

- Each row of $A$ gives an approximate equation $\langle a, s \rangle \approx b$
- if $m \gg n$, then $s$ is uniquely determined
- Still, hard to find $s$

# Uniform vs Small secrets

**Lemma**

*If LWE is hard for $s \leftarrow \chi^n$, then it is hard for $s \leftarrow \mathbb{Z}_q^n$*

# Uniform vs Small secrets

**Lemma**

*If LWE is hard for $s \leftarrow \chi^n$, then it is hard for $s \leftarrow \mathbb{Z}_q^n$*

**Proof:** Assume *Adv* solves LWE with uniform $s \leftarrow \mathbb{Z}_q^n$

```
Adv'(A,b)
   s ← Z_q^n
   b' = b + As
   s' = Adv(A,b')
   return (s' - s)
```

# Decisional LWE (DLWE)

## Definition

LWE distribution:

```
LWE[q,n,m] =
  do A ← ℤ_q^{m×n}
     s ← ℤ_q^n
     e ← χ^m
     b = As+e
     return (A,b)
```

## Definition

Decisional LWE (DLWE): distinguish LWE[q,n,m] from `Uniform`($\mathbb{Z}_q^{m\times(n+1)}$)

# Decision to Search reduction

**Theorem**

*If DLWE is hard, then LWE is hard*

# Decision to Search reduction

**Theorem**

*If DLWE is hard, then LWE is hard*

**Proof:**

- Assume Adv solves LWE
- Given Adv' that solves DLWE

```
Adv'(A,b):
   s ← Adv(A,b)
   if (As ≈ b)
      then return "LWE"
      else return "random"
```

# Search vs Decision

- Is (Search) LWE harder than DLWE?

**Theorem**

*If Seach LWE is hard for any $m = poly(n)$, then DLWE is also hard for any $m = poly(n)$*

**Theorem**

*For any $m = poly(n)$, if Seach LWE is hard, then DLWE is also hard for any $m = poly(n)$*

# LWE Search to Decision reduction (easy version)

- Assume Adv can distinguish LWE from uniform
- Task: Given A,b, find s such that As $\approx$b (mod q)
- Assumption: s is unique (holds with very high probability)
- We show how to check if $s_i = \gamma$:

```
Adv(A,b):
  a ← ℤ^m
  A' = A + [0..0,a,0..0]
  b' = b + γ a
  case Adv(A',b') of
    "LWE"    : return s_i = c
    "random" : return s_i ≠ c
```

- Recover all entries of $s$, one at a time

# (Decisional) LWE Assumption

- In the rest of the course we will just assume that DLWE is hard
- There are several variants of the assumption:
  - Uniform vs. small secret $s$
  - Different (always small) error distributions $e \leftarrow \chi$
  - Fixed vs unbounded number of samples $m$
  - Different values of $q$
  - Concrete hardness assumptions
- By and large all variants are equivalent up to polynomial reductions

# How to Encrypt with LWE

- Fix secret `s` **`in`** $\mathbb{Z}_q^n$
- LWE samples $(a_i, b_i)$ where $a_i \in \mathbb{Z}_q^n$ and $b_i \in \mathbb{Z}$
- Polynomially many samples $(a_i, b_i)$ for $i = 1, 2, ...$
- DLWE: the $b_i$ values are pseudorandom
- Idea: use $b_i$ as a one-time pad to encrypt a messate `m`

# LWE Symmetric Encryption

```
Gen():
  s ← Z_q^n
  return s

Enc(s,m):
  a ← Z_q^n
  e ← χ
  b = ⟨a,s⟩ + e + m

Dec(s,(a,b)):
  return (b - ⟨a,s⟩)
```

Is this a valid encryption scheme?

# Symmetric Encryption

```
SKE(Gen,Enc,Dec)
  Gen: () → sk
  Enc: (sk,m) → c
  Dec: (sk,c) → m
```

Correctness: for every `sk` ← `Gen()` and `m` ← `[M]`, `r` ← `[R]`:

```
    Dec(sk,Enc(sk,m;r)) = m
```

**Question**

*Is this a valid encryption scheme?*

# Correcting from errors

- Ciphertext modulus $q$
- Message modulus $p$ (assume $p$ divides $q$)
- Message space: $m \in \mathbb{Z}_p$

```
Enc(s,m) = (a,b) where
   a ← Z_q^n, e ←χ
   b = ⟨a,s⟩ + e + (q/p)m

Dec(s,(a,b)) = round(c*p/q)) where
   c = b - ⟨a,s⟩ mod q
```

**Lemma**

*If* $|e| < \beta$ *then* `Dec(s,Enc(s,m;a,e))= m`

**Question**

*For what value of* $\beta$ *is the lemma correct?*

# IND-CPA security for symmetric encryption

```
INDCPAgameSKE(b:{0,1})
   sk ← Gen()
   b' ← A[LR]
   return b':{0,1}

LR(m_0,m_1):
   ct ← Enc(sk,m_b)
   return ct
```

- Similar LR security definition can be given also for PKE: `A[LR](pk)` is given pk and oracle access to `LR`
- Previous PKE INDCPAgame allows only one query to LR

**Question**

*Why can restrict PKE INDCPAgame to one query?*

# Security of LWE symmetric encryption

- Assume $|e| < \beta = q/(2p)$ for all $e \leftarrow \chi$
- Is LWE INDCPAgameSKE secure?

**Theorem**

*Assume DLWE holds for a given $q(n)$ and any $m = poly(n)$. Then LWE symmetric encryption is INDCPA secure, i.e., any adversary* `Adv` *has negligible advantage in the INDCPAgameSKE distinguishing game.*

# RR-CPA security

- LWE encryption satisfies a stronger security property: ciphertext indistinguishability from random

```
INDCPAgameSKE(b:{0,1})
  sk ← Gen()
  b' ← A[RR]
  return b':{0,1}
```

RR($m$):
  $ct_0 \leftarrow$ Enc(sk, $m$)
  $ct_1 \leftarrow \mathbb{Z}_q^{n+1}$
  **return** $ct_b$

- "Real or Random" oracle RR
- RR-CPA security also provides a form of anonimity

# LeftRight vs RealRand security

**Theorem**

*If a (SKE or PKE) scheme is* `INDCPA-RR` *secure, then it is also* `INDCPA-LR` *secure.*

**Remark**

*A (SKE or PKE) scheme can be* `INDCPA-LR` *secure, but not* `INDCPA-RR` *secure.*

# Compact LWE Encryption

- Ciphertext expansion: `bitsize(ct)/ bitsize(m)`
- Compact LWE SKE (`Gen`,`Enc`,`Dec`)

```
Gen():
   S ← Z_q^{l×n}
   return S

Enc(S,m) = (a,b)
   a ← Z_q^n
   e ← χ^l
   b = Sa + e + round((p/q)m)

Dec(S,(a,b)):
   c ← b - S^t a) mod q
   return round(c*p/q)
```

**Theorem**

*Compact LWE SKE is correct and INDCPA-RR secure*

# Ciphertext Expansion

Compact LWE encryption:

- Key $S \in \mathbb{Z}_q^{l \times n}$
- Message $m \in \mathbb{Z}_p^l$
- Encryption `Enc(S,m)= (a,b)` where `b = Sa + e + mp/q`
- Ciphertext $(a, b) \in \mathbb{Z}_q^{n+l}$

**Question**

*What is the ciphertext/plaintext size ratio?*

- Example:
  - `Enc(f,x;r)= (f(r),H(r)`$\oplus$`m)` where $f : \{0,1\}^k \to \{0,1\}^k$
  - `Enc(f,.)`$:\{0,1\}^m \to \{0,1\}^{m+k}$
  - Ciphertext expansion: $(m+k)/m = 1 + (k/m)$

Section 5

# Linearity

# LWE Symmetric Encryption

```
Gen():
  s ← Z_q^n
  return s

Enc(s,m):
  a ← Z_q^n
  e ← χ
  b = ⟨a,s⟩ + e + (q/p)m
  return (a,b)

Dec(s,(a,b)):
  d = b - ⟨a,s⟩ mod q
  return (round(d*p/q))
```

# Compact (Matrix) LWE

```
Gen():
  S ← Z_q^{l×n}
  return S

Enc(S,M) = (A,B)
  A ← Z_q^{n×w}
  E ← χ^{l×w}
  B = SA + E + round((p/q)M) mod q

Dec(S,(A,B)):
  D ← B - SA mod q
  return round(D*p/q)
```

Notation:

- $[A, B]$: horizontal concatenation
- $(A, B)$: vertical concatenation

# Linearity of the LWE function

- Let `LWE(S,X;A,E)= SA + X + E` be the *raw* LWE function
- Encryption: `Enc(S,M)= LWE(S,(q/p)M; A,E)` for random `A,E`
- Linear properties:

```
LWE(S,X;A,E) + LWE(S,X';A',E')
       = LWE(S,X+X';A+A',E+E')

LWE(S,X;A,E) - LWE(S,X';A',E')
       = LWE(S,X-X';A-A',E-E')

c*LWE(S,X;A,E) = LWE(S,c*X; c*A,c*E)
```

# Linearity of the LWE function

- Let `LWE(S,X;A,E)= SA + X + E` be the *raw* LWE function
- Encryption: `Enc(S,M)= LWE(S,(q/p)M; A,E)` for random `A,E`
- Linear properties:

```
LWE(S,X;A,E) + LWE(S,X';A',E')
       = LWE(S,X+X';A+A',E+E')

LWE(S,X;A,E) - LWE(S,X';A',E')
       = LWE(S,X-X';A-A',E-E')

c*LWE(S,X;A,E) = LWE(S,c*X; c*A,c*E)
```

- Key Homomorphism:

```
LWE(S,X;A,E) + LWE(S',X';A,E')
     = LWE(S+S',X+X'; A, E+E')
```

- Ciphertexts must use the same `A`!

# Linearity of Ciphertexts

Ciphertexts that "encrypt" `X` under `S` with error `E`.

**Definition**

`LWE(S,X;E) = { (A,B): B = LWE(S,X;A,E)}`

`LWE(S,X;`$\beta$`) = { (A,B) : B = LWE(S,X;A,E),` $|E|_\infty < \beta$ `}`

# Linearity of Ciphertexts

Ciphertexts that “encrypt” X under S with error E.

**Definition**

LWE(S,X;E) = { (A,B): B = LWE(S,X;A,E)}

LWE(S,X;$\beta$) = { (A,B) : B = LWE(S,X;A,E), $|E|_\infty < \beta$ }

- LWE(S,X;E)+ LWE(S,X';E')$\subseteq$ LWE(S,X+X';E+E')
- LWE(S,X;E)- LWE(S,X';E')$\subseteq$ LWE(S,X-X';E-E')
- c$*$LWE(S,X;E)$\subseteq$ LWE(S,c$*$X; c$*$E)

# Linearity of Ciphertexts

Ciphertexts that "encrypt" X under S with error E.

**Definition**

LWE(S,X;E) = { (A,B): B = LWE(S,X;A,E)}

LWE(S,X;$\beta$) = { (A,B) : B = LWE(S,X;A,E), $|E|_\infty < \beta$ }

- LWE(S,X;E)+ LWE(S,X';E')$\subseteq$ LWE(S,X+X';E+E')
- LWE(S,X;E)- LWE(S,X';E')$\subseteq$ LWE(S,X-X';E-E')
- c*LWE(S,X;E)$\subseteq$ LWE(S,c*X; c*E)

**Question**

LWE(S,X;$\beta$) + LWE(S,X';$\beta'$) $\subseteq$LWE(S,X+X';$\beta + \beta'$) ?

**Question**

LWE(S,X;$\beta$) - LWE(S,X';$\beta'$) $\subseteq$LWE(S,X+X';$\beta - \beta'$) ?

# Message and Ciphertext Operations

- Addition:
  - $M_0 + M_1 \in \mathbb{Z}_q^{l \times w}$
  - $(A_0, B_0) + (A_1, B_1) = (A_0 + A_1, B_0 + B_1) \in \mathbb{Z}_q^{(n+l) \times w}$
- Subtraction
  - $M_0 - M_1 \in \mathbb{Z}_q^{l \times w}$
  - $(A_0, B_0) - (A_1, B_1) = (A_0 - A_1, B_0 - B_1) \in \mathbb{Z}_q^{(n+l) \times w}$
- Scalar multiplication
  - $c \cdot M \in \mathbb{Z}_q^{l \times w}$
  - $c \cdot (A, B) = (c \cdot A, c \cdot B) \in \mathbb{Z}_q^{(n+l) \times w}$
- Arbitrary linear transformations . . .

# Additive Homomorphism Encryption

- Homomorphic Encryption supporting the *addition* of ciphertexts

```
sk ← Gen()
c0 ←Enc(sk, m0)
c1 ←Enc(sk, m1)
c = c0 + c1
m = m0 + m1
Dec(sk,c) ?= m
```

**Question**

*Does LWE encryption satisfy the addititive homomorphic property? For what error bound $|\chi| < \beta$?*

**Question**

*Is ciphertext* c *distributed according to* Enc($m_0$+$m_1$)*?*

# Summation

- Homomorphic Encryption supporting the *addition* of ciphertexts

```
sk ← Gen()
c_1 ←Enc(sk, m_1)
c_2 ←Enc(sk, m_2)
...
c_k ←Enc(sk, m_k)
c = c_1 + c_2 + ... + c_k
m = m_1 + m_2 + ... + m_k
Dec(sk,c) =? m
```

**Question**

*For any given bound $|\chi| < \beta$, what is the largest value of k for which one can add k ciphertexts?*

# Subtraction and Scalar multiplication

- Subtraction $m_0 - m_1$ : similar to addition $m_0 + m_1$
- $\pm 1$-linear combinations: similar to summation
- Scalar multiplication $c \cdot m$: error grows by a factor $c$
- Ciphertexts can be multiplied only by small scalars!

# Concatenation

- `LWE(S,X;A,E)= SA + X + E`
  - $S \in \mathbb{Z}_q^{k \times n}$
  - $A \in \mathbb{Z}_q^{n \times w}$
  - $X, E \in \mathbb{Z}_q^{k \times w}$
- The same `S` can be used with messages `X` with any number of columns $w$
- Message Concatenation `X | X' = [X,X']`

**Definition**

`(A,B) | (A',B')= ([A,A'],[B,B'])`

**Theorem**

`LWE(S,X;A,E)| LWE(S,X';A',E)`$\subseteq$`LWE(S,[X,X'];[A,A'],[E,E'])`

# Linear Transforms

- Left multiplication by a constant matrix: `M` $\to$ `M T`
- Ciphertext `C = LWE(S,M;E)`
- Notice: `M` and `C` have the same number of columns
- We can apply `T` to `C`: `C` $\to$ `CT`

**Theorem**

`LWE(S,X;A,E)*T` $\subseteq$`LWE(S,XT;AT,ET)`

`LWE(S,X;E)*T` $\subseteq$`LWE(S,XT;ET)`

Special case:

- Addition: `C + C' = [C|C']T` for `T=(I,I)`
- Subtraction: `C - C' = [C|C']T` for `T=(I,-I)`

# Constant Messages

**Question**

*Can you compute an LWE encryption of a message* `M` *without knowing the secret key* `S`*?*

- I pick `S` $\leftarrow$ `Gen()` and keep it secret
- Goal: find ciphertext `C` such that `Dec(S,C)= M`

# Constant Messages

**Question**

*Can you compute an LWE encryption of a message* `M` *without knowing the secret key* `S`*?*

- I pick `S` $\leftarrow$ `Gen()` and keep it secret
- Goal: find ciphertext `C` such that `Dec(S,C)= M`
- Let `(A,B) = (0,(q/p)M)`
- `Dec(S,(A,B))= (p/q)(B - SA)= M`
- We write `Const(M)` for the constant ciphertext `(0,(q/p)M)`
- Remarks:
  - The ciphertext `C` is independent of `S`
  - `C = LWE((q/p)M;0)` is a "noiseless" encryption of `M`

# Constant Messages as Homomorphic properties

- `LWE(S,M;E)+ LWE((q/p)M';0)= LWE(S,M+M';E)`
- Homomorphism for "nullary functions" $f_M() = M$
  - Given an empty sequence of ciphertexts [], produce an encryption of $f_M([]) = M$
- Homomorphism for unary functions $f_M(M') = M + M'$
  - Given an encryption of $M'$, produce an encryption of the shifted message $M + M'$

# Circular security

- A PKE scheme is "circular secure" if one can securly publish the encryption `Enc(pk,sk)`.
- A SKE scheme is "circular secure" if one can securly publish the encryption `Enc(sk,sk)`.

**Definition**

A PKE scheme `(Gen,Enc,Dec)` is circular secure if `(Gen',Enc',Dec)` is IND-CPA secure where

```
Gen'():
  (sk,pk) ← Gen()
  ct ← Enc(pk,sk)
  pk' = (pk,ct)

Enc'((pk,ct),msg) = Enc(pk,msg)
```

# Application: Public key encryption

- Can we transform Secret Key Encryption to Public Key Encryption?
  - Not in general: black box separations
  - Impagliazzo's worlds: Minicrypt vs Cryptomania

# Application: Public key encryption

- Can we transform Secret Key Encryption to Public Key Encryption?
  - Not in general: black box separations
  - Impagliazzo's worlds: Minicrypt vs Cryptomania
- What if we start from an Additively Homomorphic SKE scheme?
  - Black box separation results break down
- What about a weakly (bounded) additive scheme?
- What about our LWE SKE scheme?

# PKE: Construction

- Start from SKE (`Gen`,`Enc`,`Dec`)
- Construct a PKE (`Gen'`,`Enc'`,`Dec`)

```
Gen'():
   sk ← Gen()
   for i=1..n
      pk[i] ← Enc(sk,0)
   pk = pk[1..n]
   return (sk,pk)

Enc'(pk,msg):
   for i=1..n
     r[i] ← {0,1}
   ct = Const(msg) + sum { pk[i] : r[i] = 1 }
   return ct
```

# Correctness of PKE

```
Dec(sk, msg+Enc(sk,0)+...+Enc(sk,0))
    = msg + 0 + ... + 0 = msg
```

**Theorem**

*If SKE is (1-hop) homomorphic under constant increment and n-summation, then PKE is correct.*

**Theorem**

*If SKE is (1-hop) homomorphic under constant increment and hn-summation, then PKE is correct and homomorphic under constant increment and n-summation.*

# Correctness of PKE

```
Dec(sk, msg+Enc(sk,0)+...+Enc(sk,0))
    = msg + 0 + ... + 0 = msg
```

**Theorem**

*If SKE is (1-hop) homomorphic under constant increment and n-summation, then PKE is correct.*

**Theorem**

*If SKE is (1-hop) homomorphic under constant increment and hn-summation, then PKE is correct and homomorphic under constant increment and n-summation.*

**Question**

*Assume SKE is an IND-CPA secure and homomorphic. Is PKE secure?*

- For what value of $n$?
- Certainly not secure for $n = 1$ (or even $n = 0$!)
- What about large $n$?
- How large?
- Answer: Secure, for large enough $n$ and any additively homomorphic SKE [Rothblum, TCC 2011]

# The case of LWE SKE

- Consider the PKE scheme obtained from our LWE-based SKE

```
Gen'():
  S ← Gen()
  P = Enc(S,0)|..|Enc(S,0) = Enc(S,[0..0])
  return (S,P)

Enc'(P,M):
  R ← {0,1}*
  PR + Const(M)
```

**Theorem**

*LWE PKE is RR-IND secure.*

# Universal Hashing

## Definition

A function family $H = \{h : X \to Y | h\}$ is 2-universal if for any $a, b \in X$,

$$\{(h(a), h(b)) | h \in H\} \equiv \{(f(a), f(b)) | f : X \to Y\}$$

- Let $(X, +)$ be an additive group
- For any vector $a \in X^n$, define the subset-sum function $h(a, S) = \sum\{a_i : i \in S\}$

## Question

*Which of the following function families is 2-universal?*

1. $\{h_a : S \to h(a, S) | a \in X^n\}$
2. $\{h_S : a \to h(a, S) | S \subseteq \{1, .., n\}\}$
3. *Both*
4. *Neither*

# Universal Hashing (continued)

- $h_a(S) = \sum_{i \in S} a_i$ is not 2-universal
- What about $g_{a,b}(S) = b + h_a(S)$?
  - Yes, this is 2-universal
  - Prove it as an exercise
- $\{h_a : \{0,1\}^n \to X\}_a$ still satisfies a weaker property which is enough for our purposes

**Definition**

For any $a \neq b$, $\Pr_h\{h(a) = h(b)\} = 1/|X|$

- We will refer to this weaker property as 2-universal'

# Universal Hashing: proof

**Lemma**

*For any group $(X, +)$, the function family $\{h_a(S) = \sum_{i \in S} a_i\}_a$ is 2-universal', i.e., for all $S \neq T$ we have*
$\Pr_h\{h(S) = h(T)\} = 1/|X|$

Proof.

- Let $j \in S \setminus T$
- Fix $a_i$ for all $i \neq j$
- Let $T' = T \setminus S$ and $S' = S \setminus (T \cup \{i\})$
- $c = \sum_{i \in T'} a_i - \sum_{i \in S'} a_i$ does not depend on $a_j$
- $h_a(S) = h_a(T)$ iff $a_j = c$
- $\Pr\{a_j = c\} = 1/|X|$

# Leftover Hash Lemma

**Lemma**

*For any 2-universal' family* $\{h : X \to Y | h \in H\}$, *the distributions*

- $\{(h, h(x)) | h \leftarrow H, x \leftarrow X\}$
- $\{(h, y) | h \leftarrow H, y \leftarrow Y\}$

*are within statistical distance* $\Delta \leq \sqrt{|Y|/|X|}$.

Proof Steps:

1. If $H$ is 2-universal', then $(H, H(X))$ has small collision probability
2. If $(H, H(X))$ has small collision probability, then it is statistically close to uniform

# Collision Probability and Uniformity

- $Z, Z'$ i.i.d., with $\Pr\{Z = z\} = p(z)$

**Definition**

Collision Probability:
$C(Z) = \Pr\{Z = Z'\} = \sum_z p(z)^2$

- $\sum_z (p(z) - 1/|Z|)^2 = C(Z) - 1/|Z|$
- Norm inequality: $\forall v \in R^n. \|v\|_1 \leq \sqrt{n}\|v\|_2$
- $\Delta(Z, U) = \frac{1}{2}\sum_z |p(z) - 1/|Z||$

# Collision Probability and Uniformity

- $Z, Z'$ i.i.d., with $\Pr\{Z = z\} = p(z)$

**Definition**

Collision Probability:
$C(Z) = \Pr\{Z = Z'\} = \sum_z p(z)^2$

- $\sum_z (p(z) - 1/|Z|)^2 = C(Z) - 1/|Z|$
- Norm inequality: $\forall v \in R^n. \|v\|_1 \leq \sqrt{n}\|v\|_2$
- $\Delta(Z, U) = \frac{1}{2}\sum_z |p(z) - 1/|Z||$

- $\Delta \leq \frac{1}{2}\sqrt{|Z|}\sqrt{\sum_z (p(z) - 1/|Z|)^2}$
- $\Delta \leq \frac{1}{2}\sqrt{|Z|C(Z) - 1}$

# Collision Probability of Universal Hashing

- $Z = (H, H(X))$, 2-universal function family $H : X \to Y$
- Collision Probability of $Z$:

  $C(Z) = \Pr(h = h', h(x) = h'(x') | h, h' \leftarrow H, x, x' \leftarrow X)$
- $C = \frac{1}{|H|} \Pr_{x,x'}[\Pr_h(h(x) = h(x'))]$

# Collision Probability of Universal Hashing

- $Z = (H, H(X))$, 2-universal function family $H : X \to Y$
- Collision Probability of $Z$:

  $C(Z) = \Pr(h = h', h(x) = h'(x') | h, h' \leftarrow H, x, x' \leftarrow X)$
- $C = \frac{1}{|H|} \Pr_{x,x'}[\Pr_h(h(x) = h(x'))]$
- Union bound:
  - $\Pr(x = x') = 1/|X|$
  - If $x \neq x'$, then $\Pr_h(h(x) = h(x')) \leq 1/|Y|$

# Collision Probability of Universal Hashing

- $Z = (H, H(X))$, 2-universal function family $H : X \to Y$
- Collision Probability of $Z$:

  $C(Z) = \Pr(h = h', h(x) = h'(x') | h, h' \leftarrow H, x, x' \leftarrow X)$
- $C = \frac{1}{|H|} \Pr_{x,x'}[\Pr_h(h(x) = h(x'))]$
- Union bound:
  - $\Pr(x = x') = 1/|X|$
  - If $x \neq x'$, then $\Pr_h(h(x) = h(x')) \leq 1/|Y|$
- $C \leq \frac{1}{H}(\frac{1}{|X|} + \frac{1}{|Y|})$

# Collision Probability of Universal Hashing

- $Z = (H, H(X))$, 2-universal function family $H : X \to Y$
- Collision Probability of $Z$:

  $C(Z) = \Pr(h = h', h(x) = h'(x') | h, h' \leftarrow H, x, x' \leftarrow X)$

- $C = \frac{1}{|H|} \Pr_{x,x'}[\Pr_h(h(x) = h(x'))]$
- Union bound:
  - $\Pr(x = x') = 1/|X|$
  - If $x \neq x'$, then $\Pr_h(h(x) = h(x')) \leq 1/|Y|$
- $C \leq \frac{1}{H}(\frac{1}{|X|} + \frac{1}{|Y|})$
- Using $|Z| = |H| \cdot |Y|$ we get

$\Delta \leq \frac{1}{2}\sqrt{|Z|C - 1} = \frac{1}{2}\sqrt{|Y|/|X|}$

# Security of LWE PKE

```
Gen(): S,E ← ...
       P = Enc(S,[0..0]) = (A,SA+E)
       return (S,P)

Enc(P,M): R ← {0,1}*
          return PR + Const(M)
```

**Theorem**

*LWE PKE is RR-IND secure.*

# Security of LWE PKE

```
Gen(): S,E ← ...
       P = Enc(S,[0..0]) = (A,SA+E)
       return (S,P)

Enc(P,M): R ← {0,1}*
          return PR + Const(M)
```

**Theorem**

*LWE PKE is RR-IND secure.*

Proof:

1. Assume `Adv` breaks PKE
2. LWE Assumption: `P = (A,SA+E)`≈`(A,B)`
3. `Adv` breaks RR-CPA when `P` is uniform
4. If `P` is uniform, then `(P,PR)` is close to uniform

# Details

Claim: `(P,PR)` is close to uniform

- Enough to look at a single column `(P,Pr)`
  - Statement for matrix `(P,PR)` follows by hybrid argument
- `P: r` $\rightarrow$`Pr` is 2-universal
  - Columns of P belong to a group $(\mathbb{Z}_q^{n+l}, +)$
  - r selects a subset of the columns of P
  - Apply Leftover Hash Lemma

# Homomorphic PKE

- `Enc(P,M)= PR + Const(M)`
- `Enc(P,M)+ Enc(P,M')= PR + Const(M)+ PR' + Const(M')= P(R+R')+ Const(M+M')`
- `Enc(P,M)+ Enc(P,M')`$\approx$`Enc(P,M+M')`
    - Noise: `E+E'`

# Homomorphic PKE

- `Enc(P,M)= PR + Const(M)`
- `Enc(P,M)+ Enc(P,M')= PR + Const(M)+ PR' + Const(M')= P(R+R')+ Const(M+M')`
- `Enc(P,M)+ Enc(P,M')`≈`Enc(P,M+M')`
  - Noise: `E+E'`
- `[ Enc(P,M)| Enc(P,M')] = Enc(P,[M|M'])`
  - Noise: `[E|E']`
- `Enc(P,M)T` ≈`Enc(P,MT)`
  - Noise: `ET`
  - T must be small

# Encoding modulo q

- Ciphertext modulus $q$. Assume $q = 2^k$
- Plaintest modulus $p \ll q$, e.g., `p=2`. Use scaling `Const(msg)= (0,(q/p)msg)` to allow error correction and correct decryption

# Encoding modulo q

- Ciphertext modulus $q$. Assume $q = 2^k$
- Plaintest modulus $p \ll q$, e.g., `p=2`. Use scaling `Const(msg)= (0,(q/p)msg)` to allow error correction and correct decryption
- What if we want to encrypt `msg` $\in \mathbb{Z}_q$?

# Encoding modulo q

- Ciphertext modulus $q$. Assume $q = 2^k$
- Plaintest modulus $p \ll q$, e.g., p=2. Use scaling `Const(msg)= (0,(q/p)msg)` to allow error correction and correct decryption

- What if we want to encrypt `msg` $\in \mathbb{Z}_q$?

- Idea:
  - write $msg = \sum_i m_i 2^i$, where $m_i \in \{0, 1\}$
  - Encrypt each bit individually: `Enc`$(m_0)$, . . . ,`Enc`$(m_k)$

# Encoding modulo q

- Ciphertext modulus $q$. Assume $q = 2^k$
- Plaintest modulus $p \ll q$, e.g., p=2. Use scaling `Const(msg)= (0,(q/p)msg)` to allow error correction and correct decryption

```
Enc(m: {0,1}^k) = (a,Sa+e+(q/2)m)

bitDecomp(msg: Z_q) =
  for i=0..k-1
    m[i] = (msg >> i) mod 2
  return m[]

Enc'(msg: Z_q) =
  return (Enc(bitDecomp(msg))
```

# Linear Encoding

- Bit encoding: $(\texttt{msg}:\mathbb{Z}_q)\rightarrow(\texttt{m[*]}:\{0,1\}^k)$
  - good: works for any message space
  - bad: breaks linear homomorphic properties
- We need to use a linear encoding function:
  - $(\texttt{msg}:\mathbb{Z}_q)\rightarrow(\texttt{m[*]}:\mathbb{Z}_q^k)$
  - $\texttt{msg} \rightarrow \texttt{msg}*\texttt{(1,2,4,8,...)}$

# Linear Encoding

- Bit encoding: (msg:$\mathbb{Z}_q$)$\rightarrow$(m[*]:$\{0,1\}^k$)
  - good: works for any message space
  - bad: breaks linear homomorphic properties
- We need to use a linear encoding function:
  - (msg:$\mathbb{Z}_q$)$\rightarrow$(m[*]:$\mathbb{Z}_q^k$)
  - msg $\rightarrow$msg*(1,2,4,8,...)
- Column encoding:
  - **pow2col** = (1,2,4,8,...)
  - Enc'(S,msg)= LWE(S,msg***pow2col**)= (a,b)
- Row encoding:
  - **pow2row** = [1,2,4,8,...]
  - Enc'(s,msg)= LWE(s,msg***pow2row**)= (A,b)

# Decoding modulo $q$

**Question**

- *Can you decrypt* `Enc'(S,msg)= LWE(S,msg*pow2col)= (a,b)`*?*
- *Can you decrypt* `Enc'(s,msg)= LWE(s,msg*pow2row)= (A,b)`*?*
- *For what error bound* $|e|_\infty < \beta$*?*

# Decryption algorithm

- Enc'(s,msg)= LWE(s,msg*pow2row)= (A,b) where
  b = sA+e+msg*pow2row

```
Dec'(s,(A,b)):
  msg ← 0
  for i=0...(k-1)
    ct ← (A[k-i-1],b[k-i-1] - msg*2^(k-i))
    m[i] ← Dec(s,ct)
    msg ← msg+m[i]<<(i)
  return msg
```

# Decryption algorithm

- `Enc'(s,msg)= LWE(s,msg*pow2row)= (A,b)` where `b = sA+e+msg*pow2row`

```
Dec'(s,(A,b)):
  msg ← 0
  for i=0...(k-1)
    ct ← (A[k-i-1],b[k-i-1] - msg*2^(k-i))
    m[i] ← Dec(s,ct)
    msg ← msg+m[i]<<(i)
  return msg
```

**Theorem**

`(Gen,Enc',Dec')` *is a valid encryption algorithm for* $\beta = q/4$

**Question**

*Does a similar algorithm work for* `pow2col`*?*

# Arbitrary linear transformations

- Starting point: `Enc()` linearly homomorphic for small `t`
  - `Enc(P,m)*t` $\approx$`Enc(P,mt)`
  - problem: error grows by a factor `t`

# Arbitrary linear transformations

- Starting point: `Enc()` linearly homomorphic for small `t`
  - `Enc(P,m)*t` $\approx$`Enc(P,mt)`
  - problem: error grows by a factor `t`
- What about computations modulo q?
  - `pow2row = [1,2,4,8,...]`
  - `Enc'(s,msg)= LWE(s,msg*pow2row)= (A,b)`
- Multiplying by any $t \in \mathbb{Z}_q$
  - Compute `tBin[] = bitDecomp(t)`
  - Compute scalar product with vector `tBin[]`

# Correctness of scalar multiplication

```
Enc'(s,msg) * tBin[]
   = LWE(s,msg*pow2row;e) * tBin[]
   = LWE(s,msg*pow2row*tBin[];e*tBin[])
   = LWE(s,msg*t;e')
```

- **pow2row** * tBin[] = $\sum_i 2^i \cdot$tBin[i] = t

# Correctness of scalar multiplication

```
Enc'(s,msg) * tBin[]
    = LWE(s,msg*pow2row;e) * tBin[]
    = LWE(s,msg*pow2row*tBin[];e*tBin[])
    = LWE(s,msg*t;e')
```

- `pow2row * tBin[]` = $\sum_i 2^i$·`tBin[i] = t`
- if $|e| < \beta$, then $|e'| = |\sum_i e_i \cdot tBin[i]| \leq k \cdot \beta$
- Error grows only by $k = \log q$

# Correctness of scalar multiplication

```
Enc'(s,msg) * tBin[]
   = LWE(s,msg*pow2row;e) * tBin[]
   = LWE(s,msg*pow2row*tBin[];e*tBin[])
   = LWE(s,msg*t;e')
```

- **pow2row** `* tBin[] =` $\sum_i 2^i$`·tBin[i] = t`
- if $|e| < \beta$, then $|e'| = |\sum_i e_i \cdot tBin[i]| \leq k \cdot \beta$
- Error grows only by $k = \log q$
- Problem:
    - result `msg*t` is a value modulo q
    - `Enc(s,msg*t;e')` is not properly encoded
    - we need an encryption of `msg*t*`**pow2row**

# Constant Multiplication algorithm

- `Enc'(s,msg)= LWE(s,msg*pow2row)`
- `Enc'(s,msg)* bitDecomp(t)= LWE(s,msg*t;e')`

```
CMul(C,t):
  T = bitDecomp(t * pow2row)
  return C * T
```

Proof:

# Extensions and Generalizations

- Matrix messages

  `M` $\otimes$ **`pow2row`** `= [M,M*2,M*4,M*8,...]`

- Arbitrary message modulus:

  `round(m*(q/p),m*(q/p)/2,m*(q/p)*4,...)`

- Other gadgets, e.g., based on Chinese Remainder Theorem

  - $q = \prod_i p_i$ product of small primes
  - encoding vector **`crtRow`** `= [q/`$p_1$`,q/`$p_2$`,...,q/`$p_k$`]`
  - **`crtRow`** $*$ `crtDecomp(t)= t`

# Summary

At this point we have an encryption algorithm

```
Enc'(S,M) = LWE(S,M ⊗ pow2row)
```

with message space $\mathbb{Z}_q^{w \times l}$, and supporting the homomorphic evaluation of the following operations:

- `Const(M)`: noiseless encryption of `M`
- `(+)`: addition of ciphertexts
- `(-)`: subtraction of ciphertexts
- `CMul(.,T)`: multiplication by any linear transformation modulo `q`

Section 6

# Key Switching

# Remember Proxy Re-encryption?

- Primary key: `(pk,sk)`
- Secondary key: `(pk1,sk1)`
- Re-encryption key: `rk = Enc(pk1,sk[1..k])`
- Input ciphertext `c = Enc(pk,m)`
- Decryption function $f_c$`(sk) = Dec(sk,c)`

**Question**

*What is the result of the following computation?*

`Eval(pk1,`$f_c$`,rk)`

# Remember Proxy Re-encryption?

- Primary key: `(pk,sk)`
- Secondary key: `(pk1,sk1)`
- Re-encryption key: `rk = Enc(pk1,sk[1..k])`
- Input ciphertext `c = Enc(pk,m)`
- Decryption function $f_c$`(sk) = Dec(sk,c)`

**Question**

*What is the result of the following computation?*

`Eval(pk1,` $f_c$`,rk)`

**Question**

*Can you implement proxy re-encryption using LWE?*

# LWE-based Proxy Re-encryption?

```
sk[1..n] ∈ Z_q^n
sk'[1..n] ∈ Z_q^n
Enc(sk,msg) = LWE(sk,msg*pow2row) = (A[],b[])
rk[i] = Enc(sk',sk[i])

Dec'(sk,(A,b))[j] = b[j] - ∑_i sk[i] * A[i,j]
                  ≈ msg*pow2row
Dec(sk,(A,b)) = decode(Dec'(sk,(A,b)))
```

**Question**

*Can you compute* `Dec'` *homomorphically?*
*Does it give you a proxy re-encryption scheme?*

# LWE-based Proxy Re-encryption

```
Enc(sk,msg) = LWE(sk,msg*pow2row)
rk[i] = Enc(sk',sk[i])
Dec'(sk,(A,b)) = b[j] - ∑_i sk[i] * A[i,j]
```

Goal: homomorphically evaluate the function

```
f_{A,b}(sk) = Dec'(sk,(A,b))
Eval(f_{A,b},rk) = ?
```

# LWE-based Proxy Re-encryption

```
Enc(sk,msg) = LWE(sk,msg*pow2row)
rk[i] = Enc(sk',sk[i])
Dec'(sk,(A,b)) = b[j] - ∑_i sk[i] * A[i,j]
```

Goal: homomorphically evaluate the function

$f_{A,b}$`(sk) = Dec'(sk,(A,b))`
`Eval(`$f_{A,b}$`,rk) = ?`

Solution: `Eval(`$f_{A,b}$`,rk) = ct`

`ct[j] = Const(b[j]) - ` $\sum_i$ `CMul(rk[i],A[i,j])`

# Key Switching

- Generalize proxy re-encryption:
    - `sk,sk'` may have different dimensions and moduli
    - `Enc(sk,.)`, `Enc'(sk',.)` may use different plaintext moduli and message encodings
- Example
    - Message space `msg`: $\mathbb{Z}_p$
    - Ciphertext modulus q
    - `sk[1..n]`, `sk'[1..n]` $\in \mathbb{Z}_q^n$
    - `Enc(sk,m)= LWE(sk,(q/p)*msg) mod q`
    - Evaluation key: `rk[i] = Enc(sk',sk[i])`
- Do you see any problem?

# Key Switching

Source scheme:

msg: $\mathbb{Z}_p$
sk[1..n] $\in \mathbb{Z}_q^n$
Enc(sk,msg) = LWE(sk,$\frac{q}{p}$msg) = (a[],b) mod q

Target scheme:

msg': $\mathbb{Z}_q$
sk'[1..n'] $\in \mathbb{Z}_q^{n'}$
Enc'(sk',msg') = LWE(sk',msg'***pow2row**)

Evaluation:

ek[i] = Enc'(sk',sk[i])
KeySwitch(ek,(a[],b)) =
  Const(b) - $\sum_i$ CMul(a[i],ek[i])

# Correctness

msg: $\mathbb{Z}_p$; sk[1..n] $\in \mathbb{Z}_q^n$
msg': $\mathbb{Z}_q$; sk'[1..n'] $\in \mathbb{Z}_q^{n'}$

Enc(sk,msg) = LWE(sk,$\frac{q}{p}$msg) = (a[],b) mod q
Enc'(sk',msg') = LWE(sk',msg'*pow2row)

ek[i] = Enc'(sk',sk[i])

KeySwitch(ek,(a[],b))
 = Const(b) - $\sum_i$ CMul(a[i],ek[i])

# Correctness

msg: $\mathbb{Z}_p$; sk[1..n] $\in \mathbb{Z}_q^n$
msg': $\mathbb{Z}_q$; sk'[1..n'] $\in \mathbb{Z}_q^{n'}$

Enc(sk,msg) = LWE(sk,$\frac{q}{p}$msg) = (a[],b) mod q
Enc'(sk',msg') = LWE(sk',msg'*pow2row)

ek[i] = Enc'(sk',sk[i])

KeySwitch(ek,(a[],b))
 = Const(b) - $\sum_i$ CMul(a[i],ek[i])

 = Const(b) - $\sum_i$ CMul(a[i],Enc'(sk',sk[i]))

# Correctness

```
msg:  ℤ_p;  sk[1..n] ∈ ℤ_q^n
msg': ℤ_q;  sk'[1..n'] ∈ ℤ_q^{n'}

Enc(sk,msg) = LWE(sk,(q/p)msg) = (a[],b) mod q
Enc'(sk',msg') = LWE(sk',msg'*pow2row)

ek[i] = Enc'(sk',sk[i])

KeySwitch(ek,(a[],b))
 = Const(b) - ∑_i CMul(a[i],ek[i])

 = Const(b) - ∑_i CMul(a[i],Enc'(sk',sk[i]))

 = LWE(sk',b - ∑_i a[i]*sk[i])
 = LWE(sk',(q/p)msg + e)
 = Enc(sk',msg)
```

# Remarks

- Source and Target schemes may use different moduli
  - `Enc'(sk',msg')= LWE(sk',`$\frac{q}{q}$`msg'*pow2row)`

# Remarks

- Source and Target schemes may use different moduli
  - `Enc'(sk',msg')= LWE(sk',`$\frac{q}{q}$`msg'*pow2row)`
- Input ciphertext may use compact (matrix) LWE
  - `Enc(SK,msg[])= LWE(SK,`$\frac{q}{p}$`msg[])`
  - `RK' = Enc'(SK',SK)`

# Remarks

- Source and Target schemes may use different moduli
  - `Enc'(sk',msg')= LWE(sk',`$\frac{q}{q}$`msg'*pow2row)`
- Input ciphertext may use compact (matrix) LWE
  - `Enc(SK,msg[])= LWE(SK,`$\frac{q}{p}$`msg[])`
  - `RK' = Enc'(SK',SK)`
- Key Switching:
  - Input: `Enc(sk,msg: mod p): mod q`
  - Switching Key: `Enc'(sk',sk: mod q): mod q'`
  - Output: `Enc(sk',msg: mod p): mod q'`

# Remarks

- Source and Target schemes may use different moduli
    - `Enc'(sk',msg')= LWE(sk',`$\frac{q}{q}$`msg'*pow2row)`
- Input ciphertext may use compact (matrix) LWE
    - `Enc(SK,msg[])= LWE(SK,`$\frac{q}{p}$`msg[])`
    - `RK' = Enc'(SK',SK)`
- Key Switching:
    - Input: `Enc(sk,msg: mod p): mod q`
    - Switching Key: `Enc'(sk',sk: mod q): mod q'`
    - Output: `Enc(sk',msg: mod p): mod q'`
- Input/Output can use arbitrary encoding, e.g.,
    - Input: `Enc(sk,msg)= LWE(sk,msg*pow2row)`
    - Output: `Enc(sk',msg)= LWE(sk',msg*pow2row)`

# Sub-key Switching

- Application: reduce key size `SK` $\rightarrow$ `SK'`
- Always: `SK`,`SK'` must have the same number of rows
- Often `SK` is a "sub-matrix" of `SK = [SK',SK'']`
- Switching Key

```
[RK',RK”] = Enc'(SK',SK)
           = Enc'(SK',[SK' | SK''])
           = [Enc'(SK',SK') | Enc'(SK',SK”)]
```

- But `RK'` is publicly known! (remeber circular security?)
- Can use a smaller switching key `RK” = Enc'(SK',SK”)`

**Question**

*Does it work? What if* `SK”=[]`*? Then,* `RK”=[]` *and* `SK = SK'`*!*
*Is it trivial? Is it useful?*

# Modulus switching

- Subkey switching from SK to SK'=SK can still be useful to change the ciphertext modulus from q to q'
- So far we used the simplifying assumption that $p|q$
- Switching from q to q' requires a switching key with
  - plainext modulus q
  - ciphertexst modulus q'
  - but if q|q', this only allows to increase the modulus
- (Sub-)Key Switching works also for $p \not| q$
  - but introduces a "small" rounding error
  - for subkey switching the rounding error if proportional to SK
  - switching to a smaller modulus requires "small" key SK

# Subkey and Modulus Switching

- Subkey switching
  - Input: `ct = Enc([SK',SK"],m)` and `RK = Enc'(SK',SK")`
  - `SubkeySwitch(RK,ct)= ct'` such that `Dec(SK',ct')= m`

# Subkey and Modulus Switching

- Subkey switching
  - Input: `ct = Enc([SK',SK"],m)` and `RK = Enc'(SK',SK")`
  - `SubkeySwitch(RK,ct)= ct'` such that `Dec(SK',ct')= m`

**Question**

*Give explicit description of* `SubkeySwitch` *algorithm*

# Subkey and Modulus Switching

- Subkey switching
  - Input: `ct = Enc([SK',SK"],m)` and `RK = Enc'(SK',SK")`
  - `SubkeySwitch(RK,ct)= ct'` such that `Dec(SK',ct')= m`

**Question**

*Give explicit description of* `SubkeySwitch` *algorithm*

- Modulus switching
  - Assume SK has small entries
  - Input: `ct = Enc(SK,m)mod q` and nothing else
  - `ModSwitch(ct)= ct' mod q'` such that `Dec(SK,ct')= m`

# Subkey and Modulus Switching

- Subkey switching
  - Input: `ct = Enc([SK',SK"],m)` and `RK = Enc'(SK',SK")`
  - `SubkeySwitch(RK,ct)= ct'` such that `Dec(SK',ct')= m`

**Question**

*Give explicit description of* `SubkeySwitch` *algorithm*

- Modulus switching
  - Assume `SK` has small entries
  - Input: `ct = Enc(SK,m)mod q` and nothing else
  - `ModSwitch(ct)= ct' mod q'` such that `Dec(SK,ct')= m`

**Question**

*Give explicit description of* `ModSwitch` *algorithm*

Section 7

# Multiplication

# What we have done so far

Simple LWE Encryption: private key encryption supporting

- small message modulus ($p \ll q$)
- homomorphic addition
- homomorphic multiplication by small constants
- enough to obtain public key encryption
- circular security (for small keys)

Extended LWE Encryption to support

- large message modulus ($p = q$)
- homomorphic multiplication by arbitray constants
- circular security (for arbitary keys)
- key switching

# Next: Homomorphic Multiplication

**Problem**

*Given* `Enc(sk,msg[0])` *and* `Enc(sk,msg[1])`*, compute a ciphertext* `ct` *such that* `Dec(sk,ct)= msg[0]*msg[1]`

- Can this be done for our LWE encryption scheme?
- Can it be done with the help of some additional key material?
- Yes, in fact, there are multiple ways to do it
  - Nested encryption
  - Homomorphic decryption
  - Tensor product

# Method 1: Nested Encryption

- `msg[0],msg[1]` $\in \mathbb{Z}_q$
- `ct[0] = Enc(sk[0],msg[0])`
- `ct[1] = Enc(sk[1],msg[1])`
- Multiply encryption of `msg[0]` by `ct[1]`

```
ct[0] * ct[1]
= Enc(sk[0],msg[0]) * ct[1]
= Enc(sk[0],msg[0]*ct[1])
```

- Inner multiplication:

```
msg[0]*ct[1]
= msg[0]*Enc(sk[1],msg[1])
= Enc(sk[1],msg[0]*msg[1])
```

- Final result: `Enc(sk[0],Enc(sk[1],msg[0]*msg[1]))`

# Details

- `ct[1] = Enc(sk[1],msg[1])` is a vector!
    - `ct[0] = Enc(sk[0],msg[0]*I)`
    - `(msg[0]*I)*ct[1] = msg[0]*ct[1]`
- `msg[0]*Enc(sk[1],msg[1];e[1])= Enc(sk[1],msg[0]*msg[1]; msg[0]*e[1])`
    - Assume `msg[0]` is small (e.g., `,10,1`)
    - May set `Enc(sk[1],msg[1])= LWE(sk[1],(q/2)*msg[1])`
- Using `Enc(sk[0],Enc(sk[1],msg))`
    - Keep nesting?
    - Ciphertexts get larger and larger!

# Key Nesting

- Recall: `Enc(S,M)= LWE(S,M)= (A,S*A + E + M)`
- Claim: Nested encryption `Enc(Z,Enc(S,M))= Enc(Z◇S,M)`

**Question**

*For what key* `Z◇S`*?*

# Key Nesting

- Recall: `Enc(S,M)= LWE(S,M)= (A,S*A + E + M)`
- Claim: Nested encryption `Enc(Z,Enc(S,M))= Enc(Z◇S,M)`

**Question**

*For what key* `Z◇S`*?*

- `S:`$\mathbb{Z}$`[k,n]`, `Z:`$\mathbb{Z}$`[n+k,n]`
- `Z = (Zn,Zk)` where `Zn[n,n]` and `Zk[k,n]`
- `Z◇S = [S*Zn + Zk, S] = [S,I]Z`
  - `Enc(Z,Enc(S,m;e);e')= Enc(Z◇S,m; e")`
  - `e" = e + [S,I]e'`
  - Key `S` needs to be small!

# Nested Encryption + (Sub)Key Switching

Combine nested multiplication with key switching:

- Input keys: `Z,S`
- Evaluation key: `W = Enc(S,[S,I]Z;F)`
- Input ciphertexts:
  - `CT[0] = Enc(Z,msg[0]*I;E[0])`
  - `CT[1] = Enc(S,msg[1]*I;E[1])`
- Output: `SubkeySwitch(W,CT[0]*CT[1])= Enc(S,msg*I;E)`
  - `msg = msg[0]*msg[1]`
  - `E = msg[0]*E[1] + [S,I]*E[0]*X + F*Y` for binary matrices `X,Y`
- Key `S` needs to be small!
- Security Assumption: Standard LWE

# Method 1.5: Homomorphic Decryption

- Assume both ciphertexts use the same key `S`
- Nested Encryption:
  1. Homomorphic multiplication: `Enc(S,msg[0])*CT[1]`
  2. Key Switching: Homomorphic multiplication by `[S,I]`
- Method 1: `Eval([S,I],Enc(S,msg[0])*CT[1])`
- Combine the two homomorphic multiplications:
  - Bring `[S,I]` inside the first ciphertext
  - `Enc(S,msg[0]*[S,I])* CT[1]`
- Define a new LWE encryption variant:

```
Enc#(S,msg) = Enc(S,msg*[S,I])

Enc#(S,msg[0]) * Enc(S,msg[1])
= Enc(S,msg[0]*[S,I]*Enc(S,msg[1]))
= Enc(S,msg[0]*msg[1])
```

# Security

```
Enc#(S,msg) = Enc(S,msg*[S,I])
            = [Enc(S,msg*S),Enc(S,msg*I)]
```

- Circular security:
  - Can compute `Enc(S,msg*S)= msg*Enc(S,S)` without knowing `S`
  - Problem: `msg*(-I,0)` reveals `msg`!
  - Solution: `Enc(S,0)+ msg*Enc(S,S)`

**Theorem**

`Enc` *is secure under the LWE assumption*

# Remarks

- Second encryption scheme can be chosen arbitrarily

$$\mathtt{Enc}^{\#}(S, m_0) * \mathtt{Enc}(S, m_1) = \mathtt{Enc}(S, m_0 m_1)$$
$$\mathtt{Enc}^{\#}(S, m_0) * \mathtt{Enc}^{\#}(S, m_1) = \mathtt{Enc}^{\#}(S, m_0 m_1)$$

- No need for key switching
  - Product $\mathtt{Enc}(S, m_0 m_1)$ uses the same key as the input
  - Key $S$ does not have to be small
  - No evaluation key!
- Enc is a homomorphic encryption scheme supporting
  - Ciphertext addition
  - Ciphertext multiplication
  - without any evaluation key!
- Too good to be true?

# Error growth

- $\text{Enc}^{\#}(m_0; E_0) * \text{Enc}^{\#}(m_1; E_1) = \text{Enc}^{\#}(m_0 m_1; E)$
    - Error: $E \approx m_0 * E_1 + E_0 * X$
- Multiplying many ciphertexts
    - `CT[i] =` $\text{Enc}^{\#}(m_i; E_i)$
    - Assume $m_i \in 0, 1$
    - Given `CT[1],...,CT[k]`
    - Goal: compute `CT[1]*...*CT[k] =` $\text{Enc}^{\#}(\prod_i m_i)$
- How? Several options (multiplication is associative):
    - Left to right multiplication chain
    - Right to left multiplication chain
    - Binary tree (minize circuit depth)

**Question**

*What order is best?*

# Arithmetic and Boolean operations

- Addition
  - Can add polynomially many ciphertexts
  - Error grows by polymomial factor (e.g., $O(log(n))$ bits)
- Multiplication
  - Assume **binary** message space
  - Can multiply polynomially many ciphertexts in a **chain**
  - Error grows by polymomial factor (e.g., $O(log(n))$ bits)

# Arithmetic and Boolean operations

- Addition
  - Can add polynomially many ciphertexts
  - Error grows by polymomial factor (e.g., $O(log(n))$ bits)
- Multiplication
  - Assume **binary** message space
  - Can multiply polynomially many ciphertexts in a **chain**
  - Error grows by polymomial factor (e.g., $O(log(n))$ bits)

- Bit operations:
  - $m_0, m_1 \in \{0, 1\}$
  - $m_0 \wedge m_1 = m_0 \cdot m_1$
  - $\neg m_0 = 1 - m_0$
  - $m_0 \vee m_1 = \neg(\neg m_0 \wedge \neg m_1)$
- Conditional: $(b, m_0, m_1) \mapsto m_b$
  - $m_b = (1 - b) \cdot m_0 + b \cdot m_1$
- Arbitrary log-depth boolean circuits

# Method 2: Tensor and Key Switch

- Why? Efficiency! Allows SIMD operations using polynomial rings
- Ciphertext as a function
  - $f_C$`(S)= Dec'(S,C)= [S,I]C`
  - $f_C$ is linear in `[S,I]`
- Product ciphertext `C = C0*C1`
  - Goal: `Dec'(S,C)= Dec'(S,C0)*Dec'(S,C1)`
  - $f_{C0,C1}$`(S)= Dec'(S,C0)*Dec'(S,C1)` is bilinear in `[S,I]`
- Tensor product: `Z = [S,I]`$\otimes$`[S,I] = [S`$\otimes$`S,S,S,I]`
  - Any bilinear function of `[S,I]` is linear in `Z`
  - `C = C0`$\otimes$`C1` decrypts to $m_0 \cdot m_1$ under `Z`

# Mixed product property

**Theorem**

*For any* $A, B, X, Y$,

$$(A \otimes B) \cdot (X \otimes Y) = (A \cdot X) \otimes (B \cdot Y)$$

# Mixed product property

**Theorem**

*For any $A, B, X, Y$,*

$$(A \otimes B) \cdot (X \otimes Y) = (A \cdot X) \otimes (B \cdot Y)$$

$$\begin{aligned}([S, I] \otimes [S, I]) \cdot (C_0 \otimes C_1) &= ([S, I]C_0) \otimes ([S, I]C_1) \\ &= (X_0 + E_0) \otimes (X_1 + E_1)\end{aligned}$$

Result: $X_0 \otimes X_1 + X_0 \otimes E_1 + E_0 \otimes X_1 + E_0 \otimes E_1$

- Assume scalar messages: $x_0 \otimes x_1 = x_0 \cdot x_1$
- Messages must be encoded: $x_i = \frac{q}{p} m_i$

# Encoding issues

- Encode scalar messages: $x_i = \frac{q}{p} m_i$
- Product: $(x_0 + e_0)(x_1 + e_1) = x_0 x_1 + x_0 e_1 + e_0 x_1 + e_0 e_1$
- Issues:
  - Error terms $x_0 e_1 + e_0 x_1 = \frac{q}{p}(m_0 e_1 + e_0 m_1)$ are too large
  - Main term $x_0 x_1 = (q/p)^2 m_0 m_1$ is not properly encoded
- Solutions:
  - **Modular arithmetics**: assume $\gcd(q, p) = 1$, and multiply result by $p \pmod q$
  - **Modulus lifting**: Compute the product modulo $q^2$, and then switch to smaller modulus $q$

# Modular arithmetics

- Compute $c = p \cdot c_0 \otimes c_1 \pmod{q}$
- Output $c$ decrypts (under $\mathsf{sk} \otimes \mathsf{sk}$) to

$$p(x_0 + e_0)(x_1 + e_1) = \frac{q}{p}(-qm_0 m_1) + pe_0 e_1$$

# Modular arithmetics

- Compute $c = p \cdot c_0 \otimes c_1 \pmod q$
- Output $c$ decrypts (under $\mathsf{sk} \otimes \mathsf{sk}$) to

$$p(x_0 + e_0)(x_1 + e_1) = \frac{q}{p}(-qm_0 m_1) + pe_0 e_1$$

- Assume $q = -1 \pmod p$
  - Error growth: $\beta \mapsto p\beta^2$
- Arbitrary $q, p$
  - Multiply result by $(-q)^{-1} \pmod p$
  - Error growth: $\beta \mapsto p^2\beta^2$
- Modulus switching can be used to reduce $\beta$ to a fixed polynomial $\sigma = \|s\|_1 = O(n)$, and substantially slow down the error growth

# Modulus Lifting

- Compute $c = p \cdot c_0 \otimes c_1 \pmod{q^2}$
- Assume key $\|s\|_1 < \sigma$ has small entries
- Analyze the relative error: $c_i$= Enc($m_i$;(q/p)$e_i$)

**Theorem**

*The product* $p(c_0 \bmod q) \otimes (c_1 \bmod q)$ *is an encryption of* $m_0 m_1 \pmod{p}$ *under key* $s \otimes s \pmod{q^2}$ *with error* $(q^2/p)e$

$$e \leq 3e_0 e_1 + \frac{p}{2}(\sigma + 1)(e_0 + e_1)$$

# Relative error growth

- Fixed polynonials $\beta \approx \sqrt{n}$, $\sigma = \|s\|_1 \approx O(n^{1.5})$
- Modulus lifting error growth
  - relative error: input $(q/p)e_i$, output $(q^2/p)e$
  - assume $|e_i| < \epsilon$
  - output (multiplication) error $\approx p\sigma\epsilon$
- After $L$ levels of multiplications, error $\approx (p\sigma)^L\epsilon < 1$
- Input ciphertext modulus must be $q \approx (p\sigma)^L$
  - Better than modular arithmetics approach $q > (p\beta)^{2^L}$
  - Similar growth to modular arithmetics + modulus switching

# Tensoring + Key Switching

- Both methods produce a ciphertext under key `[S⊗S,I⊗S,S⊗I]`
- For scalar messages $I = [1]$ and `I⊗S = S⊗I = S`
- Can use subkey switching from `[S⊗S,S]` to `S`
- Evaluation key: Enc`(S,S⊗S)`
- Security:
  - Does not follow from circular security of LWE
- Using standard LWE:
  - Evaluation key Enc`(Z,S⊗S)`
  - Use a sequence of keys $S_0, ..., S_L$, one for each multiplicative level of circuit/computation
  - Can you still use subkey switching?

# Arithmetic computations using Tensor products

- Message encoding: $(q/p)m$
- Plaintext arithmetic modulo $p$ (both addition and multiplication)
- Error grows with multiplicative depth of the circuit
- Use small key $\|s\|_1 < \sigma$ to use modulus switching and slow down error growth
- Error at depth $L$: $\approx (p\sigma)^L < q$
  - $L = O(\log n)$: $q = n^{O(\log n)}$
  - $L = poly(n)$: $q = 2^{poly(n)}$
- Impact of modulus:
  - Efficiency: running time $poly(\log q)$
  - Security: requires hardness of of approximating lattice

Section 8

## FHE!!

# Bootstrapping

- Given (1-hop) (Gen,Enc,Dec,Eval) supporting functions

  $f_{c,c'}$(sk) = Dec(sk,c) **nand** Dec(sk,c')

- Define (multi-hop) FHE scheme with Func = { **nand** }

```
Gen'() = (sk,pk) ← Gen()
  ek ← Enc(pk,sk)
  return (sk,(pk,ek))

Enc'((pk,ek),m) = Enc(pk,m)

Eval'((pk,ek),nand,c,c')
  = EvalC(pk,f_{c,c'},ek)
```

# LWE Homomorphic Encryption

- Goal: homomorphic evaluation of

  $f_{c,c'}$`(sk) = Dec(sk,c)` **nand** `Dec(sk,c')`

- LWE-based cryptosystem
  - Supports bounded depth addition and multiplication
  - Bit operations: `x` **nand** `y = 1 - (1-x)\cdot (1-y)`

- Key Switching

```
ek[i] = Enc'(sk',sk[i])
KeySwitch(ek,(a[],b)) =
  Const(b) - ∑_i CMul(a[i],ek[i])
```

- Homomorphic evaluation of `Dec'(a,b)= b - Sa`

# Not enough

- Key switching only computes the linear part of `Dec`
- We also need to round the result to `decode(b - Sa)`
- Is this really needed?
  - Yes, `b - Sa = (q/p)m + e`
  - Key switching gives a noisy encryption of `(q/p)+e`
  - Without rounding, noise keeps getting bigger

**Questions**

- Can we express rounding as a polynomial function (mod q)?
- What is the degree of the polynomial?

# Error growth and bounded computation

We have seen two methods to multiply ciphertexts:

- Tensor products
  - error growth $\sim \beta \to \beta\sigma$
  - can evaluate arbirary circuits with **multiplicative depth** $L$
  - even for $L = \log n$, requires superpolynomial modulus $q > \sigma^L \approx n^{O(\log n)}$
- Nested Encryption / Homomorphic Decryption
  - asymmetric error growth: $(m_0, e_0) \times (m_1, e_1) \to m_0 e_1 + e_0 \beta$
  - can evaluate arbitrary multiplication **chains** of $L$ **fresh** encryptions of **binary** messages
  - even for large $L$, polynomial modulus $q \approx L\beta^2$ is enough

# Roadmap

For each multiplication method

1. Describe/analyze a bootstrapping algorithm
2. Homomorphically evaluate the algorithm using an appropriate cryptographic data structure (encrypted accumulator)
3. Implement the cryptographic data structure using LWE

# Cryptographic accumulators

- Cryptographic Data Structure `ACC[v]`
  - Holds a value $v \in V$ in encrypted form
  - Input Encryption scheme: `Enc'`
  - Output Encryption scheme: `Enc''`
- Operations on `ACC[v]`
  - Given `Enc'(x)`, update `ACC[v]` $\rightarrow$`ACC[f(v,x)]`
  - Given `ACC[v]`, output `Enc''(f(v))`
- Bootstrapping:
  - Bootstrapping key: `Enc'(s)`
  - Final output: `Enc''(m)`

# Boostrapping problem

- Assume $p = 2, m \in \{0, 1\}$
- Decryption Algorithm:
  - Input: $a[1..n] \in \mathbb{Z}_q^n$, $b \in \mathbb{Z}_q$
  - Secret key: $s[1..n] \in \mathbb{Z}^n$
  - Compute $d = b - \sum_i a[i]s[i] + (q/4) \pmod q$
  - Round $d$ to $MSB(d) = \lfloor 2d/q \rfloor$
- Homomorphic Computation:
  - Given `Enc(s[i])`
  - Compute `Enc(MSB(d))`
- Simplifying assumption:
  - $s[i] \in \{0, 1\}$
  - without loss of generality using $(a, 2a, 4a, ...)$

# Ripple-carry addition

- Standard schoolbook method
    - using binary digits
    - add $n$ numbers at a time
    - *carry* in $\{0, \ldots, n\}$
- Input digits are encrypted

# Ripple-carry accumulator

- Parameters:
- Message space `V = {v',...,v''}`
- Input: `Enc'(x)=` $\texttt{Enc}^{\#}$`(x)`
- Output: `Enc''(x)= LWE(x)`
- `ACC[x] = (Enc''("x=v"): v` $\in$ `V)`
    - `Init(v)= ACC[v]`
    - Function application: `f(ACC[v])= ACC[f(v)]`
    - Selection:
      `Enc'(b)? ACC[v0] : ACC[v1] = ACC[b?v0:v1]`
    - Output: `p(ACC[v])= Enc''(p(b))`

# Bootstrapping algorithm

```
b+q/4 = ∑_j 2^j b[j]
a[i] = ∑_j 2^j a[i,j]
ACC  ← ACC[0]
for h = 0..k-1
  ACC[x] ← f(ACC[x]) where f(x) = (x/2) + b[h]
  forall i,j
    if (a[i,j] = 1)
      ACC[x + s[i]] ← Enc'(s[i]) ? ACC[x] :
         ACC[x+1]
return (even(ACC[x])) = Enc''(even(x))
```

# Carry-save accumulator

- Parameters: bit length `k`
- `ACC[x] = (x0,x1)`
  - `x = x0+x1` (modulo $2^k$)
  - `x0[0,..,k-1]` and `x1[0,..,k-1]`
  - redundant representation
- Operations:
  - add `y` to `ACC`
  - compute `MSB(ACC)`

# Carry-save addition

```
Add(ACC(x0,x1),y):
  x0'[i]   = (x0[i] + x1[i] + y[i]) mod 2
  x1'[i+1] = (x0[i] + x1[i] + y[i] > 1)
  return ACC(x0',x1')
```

# MSB computation

- Standard MSB computation
  - addition `x0+x1` with carry propagation
  - `O(log(k))` depth circuit where `k=log(q)`
- Can also add in `log(k)` depth
  - Compute both `MSB(ACC)` and `MSB(ACC+1)`
  - `ACC[k]`: *k*-bit accumulator
  - Recursive algorithm: split `ACC[k] = (HiACC[k/2],LoACC[k/2])`

```
MSBs(ACC=(HiACC,LoACC)):
  parallel:
    hi[0,1] = MSBs(HiACC)
    lo[0,1] = MSBs(LoACC)
  out[0] = hi[lo[0]]
  out[1] = hi[lo[1]]
  return out
```

# Bootstrapping algorithm

```
ACC[0] = b+q/4
for i=1..n
  ACC[i] = s[i]*a[i]
ACC = Sum(ACC[0],...,ACC[n])
return MSB(ACC)

Sum(ACC[0..n])
  if n=0
    then return ACC[0]
    else h=n/2
         ACC0 = Sum(ACC[0..h-1])
         ACC1 = Sum(ACC[h..n])
         (x0,x1) = ACC1
         return (ACC0 + x0) + x1
```

# Summary

Bootstrapping functions can be computed by

1. $O(n \log q)$-long sequence of multiplications, or
2. $\log(n) + \log\log(q)$-depth arithmetic circuits

Error growth:

1. Using `LWE` $\odot$: final error $\approx O(n) \cdot \beta$
2. Using `LWE` $\otimes$: final error $\approx \sigma^{\log n + \log\log q} = \sigma^{O(\log n)}$

Parameters $\beta(n), \sigma(n)$: fixed polynomials in $n$

Modulus:

1. polynomial modulus $q(n) \approx O(n)\beta = n^{O(1)}$
2. quasipolynomial $q(n) = n^{O(\log n)}$

# Summary (security)

- Hardness of lattice problems within factor $\gamma \approx q/\beta$
  1. LWE $\odot$: polynomial $\gamma = n^{O(1)}$
  2. LWE $\otimes$: quasipolynomial $\gamma = n^{O(\log n)}$
- Circular security assumption
  - Needed by tensor product multiplication / keyswitching
  - Needed to apply bootstrapping
  - Not needed for leveled homomorphic encryption

# Summary (security)

- Hardness of lattice problems within factor $\gamma \approx q/\beta$
  1. LWE $\odot$: polynomial $\gamma = n^{O(1)}$
  2. LWE $\otimes$: quasipolynomial $\gamma = n^{O(\log n)}$
- Circular security assumption
  - Needed by tensor product multiplication / keyswitching
  - Needed to apply bootstrapping
  - Not needed for leveled homomorphic encryption

**Question**

*Remove circular security assumption:*

- *Can you build (unbounded) FHE from standard LWE?*
- *Can you build (unbounded) linearly homomorphic HE?*

# Efficiency

- Main security parameter $n > 100$ (typically, $n \approx 1000$)
- Modulus $q(n) < 2^n$ has bitsize $\log q < n$
- Assume 1GHz, arithmetic operations modulo $q$
- Bootstrapping: homomorphically evaluate decryption algorithm (once or twice per gate)

**Question**

*Can you estimate the cost of a single FHE operation?*

Section 9

# Project Info

# Implementation and Libraries

Libraries:

- SEAL
- HElib
- PALISADE
- Lattigo
- ...

Interface:

- try to hide math as much as possible
- offer encoding, decoding and SIMD operations

# Project:

- Use one of the libraries
- Open ended, do anything you want
- Goal: demonstrate you managed to use the library
- Extra points: do something interesting
- Submission: pdf report describing your work + supporting code

Teams:

- You can work in pairs if you like
- Larger teams only if doing something more substantial
- Indivudual project required to use for master competency

# Project Deadlines:

Deadlines:

- Next lecture (Tue, Dec 1): need to know what you are doing (team, library)
- End of finals week (Fri, Dec 18): project submission (canvas, pdf+code)

In the meantime:

- in class, mathematics underlying Ring LWE
- used by the libraries
- useful to understand/improve the libraries
- not required to use the libraries

Section 10

# Ring LWE

# (In)efficiency of LWE

Standard LWE

- Ciphertexts: $(a, b) \in \mathbb{Z}_q^{(n+1)\times\log q}$ store one value (mod $p$)
- Ciphertext size: $O(n \log q)$
- Addition, Scalar multiplication: $T \approx n \log q$
- Ciphertext multiplication: $T \approx n^2 \log^2 q$

Compact LWE

- Ciphertexts: $(a, b) \in \mathbb{Z}_q^{(2n)\times\log q}$ store $n$ values (mod $p$)
- Amortized ciphertext size: $O(\log q)$
- Amortized addition, scalar multiplication: $T \approx \log q$
- Ciphertext multiplication?

# Ring LWE

- Generalize LWE using a ring $R$ instead of $\mathbb{Z}$
- Ring of polynomials $\mathbb{Z}[X]$
- Monic irreducible $p(X)$ of degree $n$
  - e.g., $p(X) = X^n - 1$
- Quotient ring $R = \mathbb{Z}[X]/p(X)$
  - isomorphic to $(\mathbb{Z}^n, +)$
  - convolution product
  - $R_q = R/qR$
- Ring LWE
  - Key: $s(X) \in R$
  - Ciphertexts $(a, b) \in R_q^2$
  - Messages: $m \in R_p$

# Ring LWE vs Compact LWE

Both methods:

- Encrypt $n$ values (mod $p$) using $O(n)$ values (mod $q$)
- Efficient (linear time) vector addition and scalar multiplication

Multiplication:

- Compact LWE: tensor multiplication, cost $O(n^2)$
- Ring LWE: polynomial multiplication, cost $O(n \log n)$ using FFT

Applications / Programming model:

- Addition, scalar multiplication: SIMD
- Multiplication: convolution is usually not what you want
- Encode data to perform SIMD multiplication

# Data encoding

- Polynomial representation
  - $p(x_1), \ldots, p(x_n) \in \mathbb{Z}_q^n$
  - $p(x) = a_0 + a_1 x_1 + \ldots a_{n-1} x^{n-1} \equiv \mathbb{Z}_q^n$
  - Polynomial multiplication: SIMD multiplication of evaluation representations
- Quasilinear time transformations:
  - $(y_1, \ldots, y_n) \to (a_0, \ldots, a_{n-1})$: polynomial interpolation
  - $(a_0, \ldots, a_{n-1}) \to (y_1, \ldots, y_n)$: polynomial evaluation
- Other operations:
  - SIMD: great to run same program on $n$ data sets
  - Need also to *pack*,*unpack*,*shuffle*, etc. for general computations

# Security

- Is Ring LWE secure?
- For what rings?

Short answer:

- Working modulo $p(X) = X^n - 1$ is not a good idea
- Better to work with *cyclotomic* polynomials
- SWIFFT ring: $p(X) = X^n + 1$ where $n = 2^k$

Useful both for

- security, pseudorandomness, search/decision reductions
- efficient implementation using Number Theoretic Transform (NTT)

# Implementation and Libraries

Libraries:

- SEAL
- HElib
- PALISADE
- Lattigo
- ...

Interface:

- try to hide math as much as possible
- offer encoding, decoding and SIMD operations

# Cyclic lattices

- A lattice is cyclic if it is closed under $rot(v_1, ..., v_n) = (v_n, v_1, v_2, ..., v_{n-1})$
- Equivalently
  - view vectors as coefficients of a polynomial
  - lattice is closed under $rot(v(X)) = X * v(X) \mod (X^n - 1)$
- Commonly used in coding theory (over finite fields)
  - cyclic codes: linear code, closed under rotation
  - equivalently, set of polynomials in $\mathbb{F}[X]/(X^n - 1)$, closed under multiplication by $X$

# Generators

**Theorem**

*Any cyclic code over finite a field $\mathbb{F}$ can be written as*

$$C = \{g(X) \cdot f(X) \mod (X^n - 1) | f(X)\}$$

*for some $g(X)$*

Proof.

## Generators

**Theorem**

*Any cyclic code over finite a field $\mathbb{F}$ can be written as*

$$C = \{g(X) \cdot f(X) \mod (X^n - 1) | f(X)\}$$

*for some $g(X)$*

Proof.

**Question**

*Is the same true for cyclic lattices?*

# Cyclic lattices and one-way functions

- NTRU (1998): public key encryption, efficient, no proof
- First provable construction, (M., FOCS 2002): one-way function
  - $R_q = \mathbb{Z}[X]/(q, X^n - 1)$
  - key: $a_1(X), \ldots, a_m(X) \in R_q$
  - input: $v_1(X), \ldots, v_m(X) \in \{0,1\}^n \subset R_q$
  - output: $w(X) = \sum_i a_i(X) \cdot v_i(X) \in R_q$
  - compression function: $m = 2n \log_2(q)$
- One-way: given $a_1, \ldots, a_m$ and $w$,
  - easy to find $v_1, \ldots, v_m \in R_q$ such that $\sum_i a_i v_i = w \in R_q$
  - hard to find $v_1, \ldots, v_m \in \{0,1\}^n$
- Intuition: Compact knapsack, circulant matrices

# Compact knapsack, circulant matrices

- Polynomials: $a(X) \in \mathbb{Z}[X]/(X^n - 1)$
- Equivalently: $A \in \mathbb{Z}^{n \times n}$ circulant matrix
  - $a_1 + a_2 \equiv A_1 + A_2$
  - $a_1 \cdot a_2 \equiv A_1 \cdot A_2$
- Compact knapsack

# Collision resistance?

- Regular knapsack:
  - given random $a_1, \ldots, a_m \in \mathbb{Z}_q$
  - $m = 2\log_2(q)$
  - collisions exist
  - collisions are hard to find
- Compact knapsack:
  - given random $a_1, \ldots, a_m \in \mathbb{Z}_q[X]/(X^n - 1)$
  - $m = 2n\log_2(q)$
  - collisions exist

**Question**

*Are collisions hard to find?*

# Collisions in compact knapsacks

- Multiply each "circulant" matrix $a_i$ by the all-one vector
- Find collision in $\mathbb{Z}_q$
- Algebraic decription:
  - multiply each $a_i(X)$ by $u(X) = (1 + X + X^2 + ....)$
  - Notice $(X^n - 1) = u(X) \cdot (X - 1)$
  - CRT: $R \equiv (\mathbb{Z}[X]/(X-1)) \times (\mathbb{Z}[X]/u(X))$
  - Multiplication by $u(X)$ maps $R$ to $\mathbb{Z}[X]/(X-1) \equiv \mathbb{Z}$

# Anti-Cyclic lattices

- A lattice is anticyclic if it is closed under $rot(x_1, ..., x_n) = (-x_n, x_1, x_2, ..., x_{n-1})$
- Equivalently: work in $R = \mathbb{Z}[X]/(X^n + 1)$
- Questions:
  1. Are compact knapsacks over $R$ collision resistant?
  2. Does $(X^n + 1)$ have small degree factors?

# Anti-Cyclic lattices

- A lattice is anticyclic if it is closed under $rot(x_1, ..., x_n) = (-x_n, x_1, x_2, ..., x_{n-1})$
- Equivalently: work in $R = \mathbb{Z}[X]/(X^n + 1)$
- Questions:
  1. Are compact knapsacks over $R$ collision resistant?
  2. Does $(X^n + 1)$ have small degree factors?

**Theorem**

*$X^n + 1$ is irreducible if and only if $n$ is a power of 2*

# Roots of Unity

- $\omega_m = \exp(2\pi\imath/m) \in \mathbb{C}$, primitive $m$th root of unity
- Observation: $X^m - 1 = \prod_{k=0}^{m-1}(X - \omega_m^k)$

$$\begin{aligned} X^m - 1 &= \prod_{d|m} \prod_{\gcd(k,m)=d} (X - \omega_m^k) \\ &= \prod_{d|m} \prod_{k \in \mathbb{Z}_{m/d}^*} (X - \omega_{m/d}^k) \end{aligned}$$

**Definition**

Cyclotomic Polynomial: $\Phi_m(X) = \prod_{k \in \mathbb{Z}_m^*}(X - \omega_m^k) \in \mathbb{C}[X]$

- Question: does $\Phi_m$ have integer coefficients?

# Division Theorem

- $(R, +, *, 0, 1)$: any ring
- $R[X]$: polynomials with coefficients in $X$

**Theorem**

*For any $a(X) \in R[X]$ and monic $b(X) \in R[X]$, there exists unique $q(X), r(X) \in R[X]$ such that*

- $a(X) = q(X) * b(X) + r(X)$
- $\deg(r(X)) < \deg(b(X))$

# Division Algorithm

```
divRem :: Poly → Poly → Poly
divRem a b =
    if (deg a < deg b)
    then (0,a)
    else let aL = leadingTerm a
             bL = leadingTerm b
             qL = aL / bL
             a' = a - b*qL
             (q',r) = divRem a' b
             q = qL + q'
         in divRem (q, r)
```

- Dividing by $b(X)$ requires divisions by the leading coefficient of $b$
- If $R$ is a *field*, we can divide by any **non-zero** $b(X)$:
- If $b(X)$ is **monic**, division is possible in *any ring* $R$

# Polynomial Division: Example

**Question**

*Divide* $a(X) = 5X^8 + 4X^6 - 5X^3 + 4$ *by* $b(X) = X^3 - X + 7$

# Polynomial Division: Example

**Question**

*Divide* $a(X) = 5X^8 + 4X^6 - 5X^3 + 4$ *by* $b(X) = X^3 - X + 7$

Solution:

- quotient: $q(X) = 5X^5 + 9X^3 - 35X^2 + 9X - 103$
- remainder: $r(X) = 254X^2 - 166X + 725$

# Remarks about Division Algorithm

- Division Algorithm:
  $(a(X), b(X) \in R[X]) \mapsto (q(X), r(X) \in R[X])$

- For any subring $S \subseteq R$, and $a(X), b(X) \in S[X]$
  - Result of dividing $a(X)$ by $b(X)$ is in $S[X]$
  - Division as polynomials in $R[X]$ or as polynomials in $S[X]$ produces the same result

# Polynomial GCD

- $\mathbb{F}[X]$: polynomials with coefficients in a **field** $\mathbb{F}$
- The Greatest Common Divisor (gcd) of $a(X), b(X) \in \mathbb{F}[X]$ is a polynomial $g(X) \in \mathbb{F}[X]$ such that
  - $g(X)$ divides $a(X)$ and $b(X)$
  - any $d(X) \in \mathbb{F}[X]$ that divides both $a(X)$ and $b(X)$ also divides $g(X)$

**Theorem**

*For any $a(X), b(X) \in \mathbb{F}[X]$*

$$\gcd(a(X), b(X)) = u(X)a(X) + v(X)b(X)$$

*for some $u(X), v(X) \in \mathbb{F}[X]$.*

# Euclid's Algorithm

- Input: $a(X), b(X) \in \mathbb{F}[X]$
- Output: $u(X), v(X) \in \mathbb{F}[X]$ such that $u(X)a(X) + v(X)b(X) = \gcd(a(X), b(X))$
- Invariant: $\gcd(a(X), b(X)) = \gcd(b(X), a(X) \mod b(X))$

```
euclid :: (Poly,Poly) → (Poly,Poly)
euclid (a,b) =
    if (deg b ≡ 0)
    then (1,0)
    else let (q,r) = divRem b a
             (u,v) = euclid (b,r)
         in (-q*v , u+v)
```

- Base case: `1*a+0*b = a = gcd(a,b)`
- Indunction: `(-qv)a+(u+v)b = ub + v(b-qa)= ub+vr`

# Remarks about Euclid Algorithm

```
euclid :: (Poly,Poly) → (Poly,Poly)
euclid (a,b) =
    if (deg b ≡ 0)
    then (1,0)
    else let (q,r) = divRem b a
             (u,v) = euclid (b,r)
         in (-q*v , u+v)
```

- Euclid Algorithm works over a field:
  - Even if $b(X)$ is monic, $r(X) = b(X) \mod a(X)$ may not be
- If $a(X), b(X) \in R[X]$ have coefficients in a domain $R \subseteq F$, then we can compute $\gcd(a(X), b(X)) \in \mathbb{F}[X]$

# Cyclotomic Polynomials

- $X^m - 1 = \prod_{d|m} \Phi_m(X)$

**Theorem**

$$\Phi_m(X) \in \mathbb{Z}[X]$$

# Cyclotomic Polynomials

- $X^m - 1 = \prod_{d|m} \Phi_m(X)$

**Theorem**

$$\Phi_m(X) \in \mathbb{Z}[X]$$

Proof:

- For $m = 1$, $\Phi_1(X) = (X - 1)$
- For $m > 1$, $b(X) = \prod_{m>d|m} \Phi_d(X)$ is in $\mathbb{Z}[X]$ by induction
- Compute $(q(X), r(X)) = divRem(X^m - 1, b(X))$ in $\mathbb{Z}[X]$
- $r(X) = 0$ because $b(X)$ divides $X^m - 1$
- $\Phi_m(X) = q(X)$ is in $\mathbb{Z}[X]$

# Irreducibility of Cyclotomics

**Theorem**

$\Phi_m(X) \in \mathbb{Z}[X]$ *is irreducible*

**Theorem**

$C_m \equiv \mathbb{Z}[X]/\Phi_m(X) = \mathbb{Z}[\omega_m]$

- simple proof, helps intuition
- Algebraic Number Fields
  - finite dimentional extensions of $\mathbb{Q}$
  - key concepts: field extensions, vector spaces
- Algebraic Number Rings
  - finite dimensional extensions of $\mathbb{Z}$, i.e., lattices
  - key concepts: ring extensions, modules over a ring

# Factoring primes in Cyclotomic rings

- $\Phi_m(X) \in \mathbb{Z}[X]$: $m$th cyclotomic polynomial
- $\Phi_m(X)$ is irreducible in $\mathbb{Z}[X]$
- Let $p$ be a prime, and assume $\gcd(m, p) = 1$
- Question: if $\Phi_m(X)$ irreducible also in $\mathbb{Z}_p[X]$?
- Answer: no, and this is very useful

**Question**

*Question: What's the factorization of $\Phi_m(X)$ modulo $p$?*

Technically, this is the problem of factoring (the ideal generated by) the prime $p$ in the ring of polynomials modulo $\Phi_m(X)$

# Factoring primes in Cyclotomic rings

- $\Phi_m(X) \in \mathbb{Z}[X]$: $m$th cyclotomic polynomial
- $\Phi_m(X)$ is irreducible in $\mathbb{Z}[X]$
- Let $p$ be a prime, and assume $\gcd(m, p) = 1$
- Question: if $\Phi_m(X)$ irreducible also in $\mathbb{Z}_p[X]$?
- Answer: no, and this is very useful

**Question**

*Question: What's the factorization of $\Phi_m(X)$ modulo $p$?*

Technically, this is the problem of factoring (the ideal generated by) the prime $p$ in the ring of polynomials modulo $\Phi_m(X)$

> *"The obvious mathematical breakthrough would be development of an easy way to factor large prime numbers"*
> *(Bill Gates, The Road Ahead, p. 265)*

# Motivation

- $R = \mathbb{Z}[X]/\Phi_m(X)$
- $R_p = R/(pR) \equiv \mathbb{Z}[X]/\langle \Phi_m(X), p \rangle_{\mathbb{Z}[X]}$
- Equivalently, $R_p \equiv \mathbb{Z}_p[X]/\Phi_m(X)$
- The structure of $R_p$ is equivalently described by
  - the factorization of $(pR)$ in $R$, or
  - the factorization of $\Phi_m$ in $\mathbb{Z}_p[X]$

Section 11

# ANT

# Basic Algebra

Review of basic algebraic structures:

- (Commutative) monoids and groups
- Rings and Fields
- Modules and Vector spaces

Some common examples:

- $\mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$: the fields of rational, real and complex numbers
- $\mathbb{Z}, \mathbb{Z}_n$: the rings of integers, and integers modulo $n$
- $R[X]$: The ring of polynomials with coefficients in $R$

# Monoids and Groups

- A monoid $(A, *, 1)$ is a set $A$ with a binary operation $(*) : A \times A \to A$ and unit element $1 \in A$ such that
  - $(x * y) * z = x * (y * z)$ (associativity)
  - $1 * x = x * 1 = x$ (identity)
- A monoid is commutative if
  - $x * y = y * x$ (commutativity)
- An element $x$ is invertible if there is a $y$ such that $x * y = y * x = 1$
- A group is a monoid such that all elements are invertible
- Abelian group: commutative groups, additive notation $(A, +, 0)$, additive inverse $-x$

# Rings and Fields

- A (commutative) Ring $(R, +, *, 1, 0)$ is a set with two binary operations such that
- $(R, +, 0)$ is an abelian group
- $(R, *, 1)$ is a (commutative) monoid
- $x * (y + z) = x * y + x * z$ and $(x + y) * z = x * z + y * z$ (distributivity)
- Subring $S \subseteq R$, subset of a ring closed under $+, *, 0, 1$
- A commutative ring $(F, +, *, 1, 0)$ such that all nonzero elements are invertible is called a *Field*
- A subring of a field $R \subseteq F$ is called an *Integral Domain*

# Modules and Vector Spaces

- Let $(R, +, *, 0, 1)$ be a commutative ring
- An $R$-module is an additive group $(A, +, 0)$ with a scalar multiplication operation $(*) : R \times A \to A$ such that
  - $r * (s * a) = (r * s) * a$
  - $(r + s) * a = r * a + s * a$
  - $r * (a + b) = r * a + r * b$
- If $R$ is a field, then $A$ is called a *Vector Space*
  - Linear independence
  - Dimension
  - Basis

# Submodules and Quotients

- Let $(A, +, 0)$ be an $R$-module
- An $R$-submodule of is
  - a subgroup $B \subseteq A$
  - closed under scalar multiplication: $R * B \subseteq B$
- Quotient group: $A/B = \{[a]_B : a \in A\}$, $[a]_B = a + B$
  - also an $R$-module with $r * [a]_B = [r * a]_B$
- Special case:
  - $R$ is an $R$-module
  - $R$-submodules $I \subseteq R$ are called *ideals*
  - $R/I$ is also a ring with $[a] * [b] = [a * b]$

# Integral and Algebraic Numbers

- Domain $R \subseteq F$: subring of a field $F$
- $\alpha \in F$ is **algebraic** over $R$ if $m(\alpha) = 0$ for some $m(X) \in R[X]$
- $\alpha \in F$ is **integral** over $R$ if $m(\alpha) = 0$ for some **monic** $m(X) \in R[X]$
- Examples:
  - $\alpha = \sqrt{2}$ is integral over $\mathbb{Z}$ because $m(\alpha) = 0$ for $m(X) = X^2 - 2$
  - $\alpha = 1/\sqrt{2}$ is algebraic over $\mathbb{Z}$ because $m(\alpha) = 0$ for $m(X) = 2X^2 - 1$, but is it not integral

# Minimal Polynomial

- Field Extension $F \subseteq E$
- Let $\alpha \in E$ be algebraic over $F$
- Ring homomorphism: $h_\alpha : F[X] \to E$, where $h_\alpha(p(X)) = p(\alpha)$
- $I = \ker(h_\alpha)$: set of polynomials $p$ such that $p(\alpha) = 0$
- $I \subseteq F[X]$ is a non-zero ideal
- **Minimal polynomial**: smallest degree monic polynomial $m(X) \in I$
- $I = F[X] \cdot m(X)$, i.e., $p(\alpha) = 0$ iff $m(X) | p(X)$

# Irreducibility

- Let $m(X)$ be the minimal polynomial of $\alpha$
- $m(X)$ is irreducible:
  - If $m(X) = a(X) \cdot b(X)$, then $a(\alpha) \cdot b(\alpha) = m(\alpha) = 0$,
  - either $a(\alpha) = 0$ or $b(\alpha) = 0$.
  - either $a(X) = c \cdot m(X)$ or $b(X) = c \cdot m(X)$
- $F[\alpha] \equiv F[X]/m(X)$ are isomorphic
- isomorphism: $h_\alpha : F[X]/m(X) \to F[\alpha]$

# Algebraic Extensions

- Algebraic $\alpha \in E \subseteq F$
- Minimal polynomial $m(\alpha) = 0$ of degree $n = \deg(m(X))$
- $F[\alpha] \equiv F^n$ as an $F$-vector space with basis $\alpha^0, \alpha^1, \ldots, \alpha^{n-1}$:
  - $\alpha^0, \alpha^1, \ldots, \alpha^{n-1}$ are linearly independent
  - $\alpha^0, \alpha^1, \ldots, \alpha^{n-1}$ generate $F[\alpha]$

# Extension fields

**Theorem**

*$F[\alpha] = F(\alpha)$ is a field*

Proof:

- Let $p(\alpha) \in F[\alpha]$ **for** some $p(X) \in F[\alpha]$, $\deg(p) < n$
- $\gcd(p(X), m(X)) \in \{1, m(X)\}$ because $m(X)$ is irreducible
- If $\gcd = m(X)$, **then** $p(X) = m(X)$ **and** $p(\alpha) = 0$
- If $\gcd = 1$, **then** $u(X)p(X) + v(X)m(X) = 1$
- $u(\alpha) \cdot p(\alpha) = 1$

# Factoring primes in Cyclotomic rings

- $\Phi_m(X) \in \mathbb{Z}[X]$: $m$th cyclotomic polynomial
- $\Phi_m(X)$ is irreducible in $\mathbb{Z}[X]$
- Let $p$ be a prime, and assume $\gcd(m, p) = 1$
- Question: if $\Phi_m(X)$ irreducible also in $\mathbb{Z}_p[X]$?
- Answer: no, and this is very useful

**Question**

*Question: What's the factorization of $\Phi_m(X)$ modulo $p$?*

- Since $\gcd(m, p) = 1$, we have $p \in \mathbb{Z}_m^*$
- Let $d = o(p)$ be the order of $p$ in $\mathbb{Z}_m^*$
- $p^d = 1 \mod m$, equivalently, $m | (p^d - 1)$
- Let $GF(p^d)$ be the finite field with $p^d$ elements
- The multiplicative group $GF(p^d)^*$ is cyclic of order $p^d - 1$
- There is an element $\zeta \in GF(p^d)$ of order $m$
- $\zeta^d = 1$ in $GF(p^d)$
- $o(\zeta^k) = m$ for all $k \in \mathbb{Z}_m^*$
- $\Phi_m(X) = \prod_{k \in \mathbb{Z}_m^*} (X - \zeta^k)$ splits in $GF(p^d)$

**Theorem**

*The minimal polynomials of all $\zeta^k$ over $\mathbb{Z}_p$ have degree $d$*

- Let $l(X) \in \mathbb{Z}_p[X]$ be the minimal polynomial of $\zeta$
- $\mathbb{Z}_p[\zeta] \equiv \mathbb{Z}_p[X]/l(X)$ is a field
  - of size $p^{\deg(l)}$
  - containing an element $\zeta$ of order $m$
- $m = o(\zeta)$ divides $p^{\deg(l)} - 1 = |\mathbb{Z}_p[\zeta]^*|$
- $p^{\deg(l)} = 1 \mod m$
- by definition of $d = o(p)$ and $\deg(l) = d$

- When $\gcd(m, p) = 1$ $\Phi_m(X) \in \mathbb{Z}_p[X]$ factors into a product of $\varphi(m)/d$ distinct degree $d = o(p \mod m)$ polynomials
- For arbitrary $m$, factorization of $\Phi_m(X)$ modulo $p$ is obtained using the following theorem.

**Theorem**

*For any $m' = mp^k$ with $\gcd(m, p) = 1$,*

$$\Phi_{m'}(X) = (\Phi_m(X))^{\varphi(p^k)} \mod p$$

# Proof

- Frobenius map $(x \mapsto x^p) : GF(p^k) \to GF(p^k)$ satisfies:
  - $(x + y)^p = x^p + y^p$ (from binomial expansion)
  - $a^p = a$ for $a \in \mathbb{Z}_p \subseteq GF(p^k)$ (Lagrange)
- $\mathbb{Z}_p[X]$ is a domain:
  - $a(X)b(X) = a(X)c(X)$ cancels to $b(X) = c(X)$

- Using these two properties:
    - $(X^{mp^k} - 1) = (X^m - 1)^{p^k} = \prod_{d|m} \Phi_d(X)^{p^k}$
    - $(X^{mp^k} - 1) = \prod_{d|m} \prod_{i \le k} \Phi_{dp^i}(X)$
    - So, by induction on $m$:

$$\prod_{i \le k} \Phi_{mp^i}(X) = \Phi_m(X)^{p^k}$$

    - Canceling equality for $k-1$ from equality for $k$:

$$\Phi_{mp^k}(X) = \Phi_m(X)^{p^k - p^{k-1}} = \Phi_m(X)^{\varphi(p^k)}$$

# Factoring modulo a prime power

- $\Phi_m(X) = \prod_i F_i(X) \mod p$ with irredudible $F_i(X) \in \mathbb{Z}_p[X]$
- Lift each $F_i(X) \mod p$ to a factor $G_i(X) \mod p^k$
- $\Phi_m(X) = \prod_i G_i(X) \mod p^k$ with $F_i(X) = G_i(X) \mod p$
- $G_i(X)$ is irreducible, because any factorization $\mod p^k$ gives also a factorization $\mod p$

**Theorem**

*(Lifting) Let* $a(X)b(X) = c(X) \mod p$ *with* $\gcd(a(X), b(X)) = 1$*. For every* $k$*, there are* $a'(X) = a(X) \mod p$ *and* $b'(X) = b(X) \mod p$ *such that* $a'(X)b'(X) = c(X) \mod p^k$

- Let $u(X), v(X)$ such that $a(X)u(X) + b(X)v(X) = 1$ mod $p$
- Assume $a(X)b(X) = c(X) + p^k d(X)$ by induction
- Let $a'(X) = a(X) - p^k v(X)d(X)$ and $b'(X) = b(X) - p^k u(X)d(X)$
- $a'(X)b'(X)$ mod $p^{k+1} = a(X)b(X) - p^k(a(X)u(X) + b(X)v(X))d(X) = a(X)b(X) - p^k d(X) = c(X)$