CSE208: Advanced Cryptography (FHE)

Daniele Micciancio

UCSD

Winter 2023

Section 1

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?

$sk[1..n] \in \mathbb{Z}_q^n$

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

$Enc(sk, msg) = LWE(sk, msg * pow2col) = (A[], b[])$

$rk[i] = Enc(sk', sk[i])$

$$Dec'(sk,(A,b))[j] = b[j] - \sum_i sk[i] * A[i,j] \approx msg * pow2col$$

$Dec(sk,(A,b)) = decode(Dec'(sk,(A,b)))$

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*pow2col)
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*pow2col)
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]) - ∑_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'*pow2col)

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'*pow2col)

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'*pow2col)

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: $\mathbb{Z}_p$; 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'*pow2col)

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]))

= LWE(sk',b - $\sum_i$ a[i]*sk[i])

= LWE(sk', $\frac{q}{p}$msg + e)

= Enc(sk',msg)

Remarks

- Source and Target schemes may use different moduli
  - Enc'(sk',msg')= LWE(sk', $\frac{q}{q}$msg'*pow2col)

Remarks

- Source and Target schemes may use different moduli
  - Enc'(sk',msg')= LWE(sk', $\frac{q}{q}$msg'*pow2col)
- 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'*pow2col)
- 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', q/q msg'*pow2col)`
- Input ciphertext may use compact (matrix) LWE
  - `Enc(SK,msg[])= LWE(SK, 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*pow2col)`
  - Output: `Enc(sk',msg)= LWE(sk',msg*pow2col)`

Sub-key Switching

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

$$
\begin{aligned}
[\texttt{RK'}, \texttt{RK''}] &= \texttt{Enc'}(\texttt{SK'}, \texttt{SK}) \\
&= \texttt{Enc'}(\texttt{SK'}, [\texttt{SK'} \mid \texttt{SK''}]) \\
&= [\texttt{Enc'}(\texttt{SK'}, \texttt{SK'}) \mid \texttt{Enc'}(\texttt{SK'}, \texttt{SK''})]
\end{aligned}
$$

- But $\texttt{RK'}$ is publicly known! (remeber circular security?)
- Can use a smaller switching key $\texttt{RK''} = \texttt{Enc'}(\texttt{SK'}, \texttt{SK''})$

Question

Does it work? What if $\texttt{SK''}=[]$*? Then,* $\texttt{RK''}=[]$ *and* $\texttt{SK} = \texttt{SK'}$*!*
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*