

FHE Bootstrapping in less than a Second

Léo Ducas¹ Daniele Micciancio

University of California, San Diego

Eurocrypt, April 2015

¹Now at CWI, Amsterdam, The Netherlands

Outline

Introduction/Summary

The new NAND gate

Simpler Refreshing

Conclusion

The evolution of FHE

Fully Homomorphic Encryption has seen drastic changes since Gentry's first proposal:

- ▶ [Rivest,Adleman,Dertouzos'78]: Open problem
- ▶ [Gentry'09]: ideal lattices, sparse subset-sum, squashing, etc.
- ▶ [Gentry,Halevi'11],[Brakerski,Vaikuntanathan'11]: no squash
- ▶ [Brakerski,Vaikuntanathan'11]: Subexponential LWE
- ▶ [Brakerski'12],[Alperin-Sheriff,Peiert'14]: (Polynomial) LWE
- ▶ Many more works improving efficiency, etc.

The evolution of FHE

Fully Homomorphic Encryption has seen drastic changes since Gentry's first proposal:

- ▶ [Rivest,Adleman,Dertouzos'78]: Open problem
- ▶ [Gentry'09]: ideal lattices, sparse subset-sum, squashing, etc.
- ▶ [Gentry,Halevi'11],[Brakerski,Vaikuntanathan'11]: no squash
- ▶ [Brakerski,Vaikuntanathan'11]: Subexponential LWE
- ▶ [Brakerski'12],[Alperin-Sheriff,Peiert'14]: (Polynomial) LWE
- ▶ Many more works improving efficiency, etc.

Still, all schemes have a common ingredient:

Key technique

Gentry's FHE bootstrapping

The Problem

FHE Bootstrapping/Refreshing is an expensive process:

- ▶ [Halevi, Shoup'14,'15] HElib: 6-30 mins

The Problem

FHE Bootstrapping/Refreshing is an expensive process:

- ▶ [Halevi, Shoup'14,'15] HElib: 6-30 mins

Mitigating the cost of bootstrapping (previous approaches):

- ▶ SIMD-FHE: Perform many refresh operations in parallel
- ▶ Noise control: allow more computation before refreshing
- ▶ HElib: Cost can be amortized over ≈ 1000 binary ciphertext.

The Problem

FHE Bootstrapping/Refreshing is an expensive process:

- ▶ [Halevi, Shoup'14,'15] HElib: 6-30 mins

Mitigating the cost of bootstrapping (previous approaches):

- ▶ SIMD-FHE: Perform many refresh operations in parallel
- ▶ Noise control: allow more computation before refreshing
- ▶ HElib: Cost can be amortized over ≈ 1000 binary ciphertext.

Question

How fast can we refresh **a single** ciphertext?

Contributions

Question

How fast can we refresh for a single ciphertext ?

²i.e. not ridiculously slower

Contributions

Question

How fast can we refresh for a single ciphertext ?

We give a proof of concept solution in 0.6 seconds:
amortized cost comparable² to [HElib], but without the delay...

Two new techniques:

- ▶ a new, cheap NAND gate
- ▶ a simpler refreshing procedure using ring structure

²i.e. not ridiculously slower

The new NAND gate

Base: Start from LWE encryption with message space: \mathbb{Z}_t , $t \geq 2$.

Idea: Different message space for input ($t = 4$) and output ($t = 2$).

The new NAND gate

Base: Start from LWE encryption with message space: \mathbb{Z}_t , $t \geq 2$.

Idea: Different message space for input ($t = 4$) and output ($t = 2$).

Advantages:

- ▶ Cost of computing Homomorphic NAND is negligible (similar to a single private key cryptographic operation.)
- ▶ Excellent noise growth: ϵ grows only by a small constant factor.
- ▶ Substantially simplifies the task faced by the Refreshing procedure.

A simpler refreshing procedure

Base: General approach of [Alperin-Sheriff,Peikert'14] + Ring variant of [Gentry,Sahai,Waters'13] Homomorphic encryption.

Idea: Implement arithmetic mod q **in the exponent**

A simpler refreshing procedure

Base: General approach of [Alperin-Sheriff,Peikert'14] + Ring variant of [Gentry,Sahai,Waters'13] Homomorphic encryption.

Idea: Implement arithmetic mod q **in the exponent**

Improvement over [AP14]:

- ▶ Theoretical speed-up of $\tilde{\Omega}(\log^3 q)$
- ▶ Smaller final error.

Combined with the problem simplification brought by our cheap NAND computation, this results in bootstrapping cost ≈ 0.6 second, at estimated ≈ 100 -bit security level.

Outline

Introduction/Summary

The new NAND gate

Simpler Refreshing

Conclusion

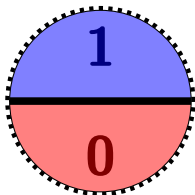
LWE encryption

Idea: use an LWE sample as a mask

$$\text{Enc}_s(m) = (\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e + m \cdot q/2)$$

$$\text{Dec}_s(\mathbf{a}, b) = \lfloor 2(b - \langle \mathbf{a}, \mathbf{s} \rangle)/q \rfloor$$

$$m \cdot \frac{q}{2} + e$$



binary messages

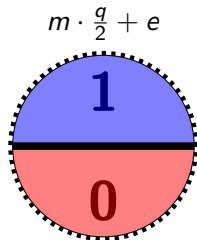
$$\text{LWE}_s^2(m, q/4)$$

LWE encryption

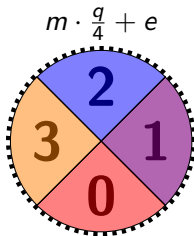
Idea: use an LWE sample as a mask

$$\text{Enc}_s(m) = (\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e + m \cdot q/2)$$

$$\text{Dec}_s(\mathbf{a}, b) = \lfloor 2(b - \langle \mathbf{a}, \mathbf{s} \rangle)/q \rfloor$$



binary messages
 $\text{LWE}_s^2(m, q/4)$



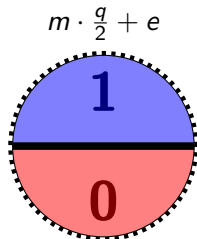
Messages in \mathbb{Z}_4
 $\text{LWE}_s^4(m, q/8)$

LWE encryption

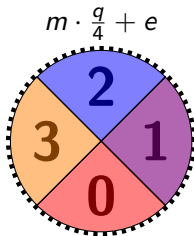
Idea: use an LWE sample as a mask

$$\text{Enc}_s(m) = (\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e + m \cdot q/2)$$

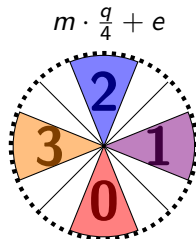
$$\text{Dec}_s(\mathbf{a}, b) = \lfloor 2(b - \langle \mathbf{a}, \mathbf{s} \rangle)/q \rfloor$$



binary messages
 $\text{LWE}_s^2(m, q/4)$



Messages in \mathbb{Z}_4
 $\text{LWE}_s^4(m, q/8)$

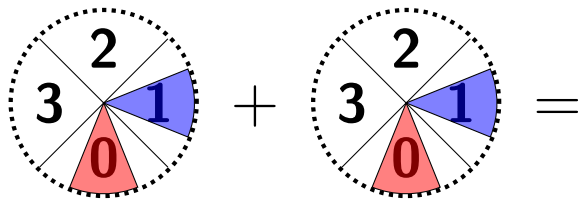


Smaller error
 $\text{LWE}_s^4(m, q/16)$

A Cheaper NAND gate

Idea, use: $m_1 \wedge m_2 \Leftrightarrow m_1 + m_2 = 2 \bmod 4$.

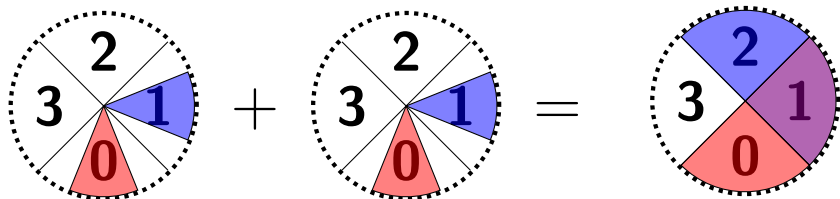
Consider binary messages $\{\mathbf{0}, \mathbf{1}\}$ encrypted with $t = 4$:



A Cheaper NAND gate

Idea, use: $m_1 \wedge m_2 \Leftrightarrow m_1 + m_2 = 2 \bmod 4$.

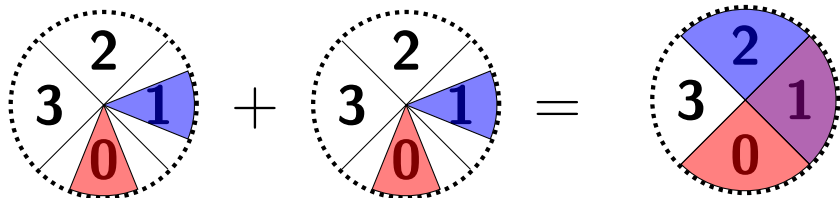
Consider binary messages $\{0, 1\}$ encrypted with $t = 4$:



A Cheaper NAND gate

Idea, use: $m_1 \wedge m_2 \Leftrightarrow m_1 + m_2 = 2 \bmod 4$.

Consider binary messages $\{\mathbf{0}, \mathbf{1}\}$ encrypted with $t = 4$:

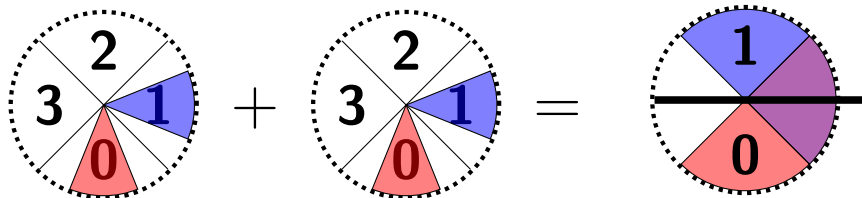


Consider it as a ciphertext for $t = 2$ and rotate.

A Cheaper NAND gate

Idea, use: $m_1 \wedge m_2 \Leftrightarrow m_1 + m_2 = 2 \bmod 4$.

Consider binary messages $\{\mathbf{0}, \mathbf{1}\}$ encrypted with $t = 4$:

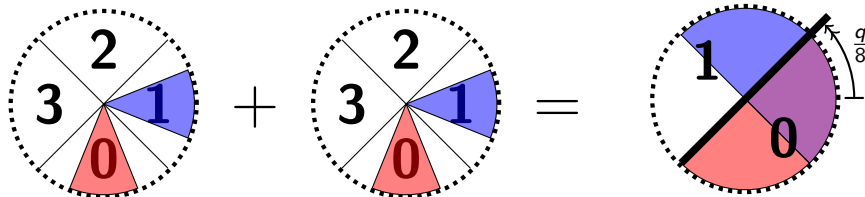


Consider it as a ciphertext for $t = 2$ and rotate.

A Cheaper NAND gate

Idea, use: $m_1 \wedge m_2 \Leftrightarrow m_1 + m_2 = 2 \bmod 4$.

Consider binary messages $\{\mathbf{0}, \mathbf{1}\}$ encrypted with $t = 4$:

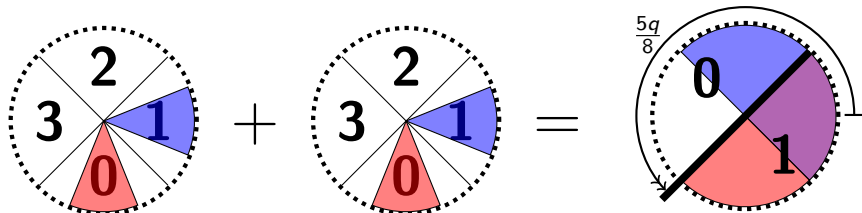


Consider it as a ciphertext for $t = 2$ and rotate.

A Cheaper NAND gate

Idea, use: $m_1 \wedge m_2 \Leftrightarrow m_1 + m_2 = 2 \bmod 4$.

Consider binary messages $\{\mathbf{0}, \mathbf{1}\}$ encrypted with $t = 4$:



Consider it as a ciphertext for $t = 2$ and rotate.

HomNAND:

$$\text{LWE}_s^4(m_1, \frac{q}{16}) \times \text{LWE}_s^4(m_2, \frac{q}{16}) \rightarrow \text{LWE}_s^2(m_1 \bar{\wedge} m_2, \frac{q}{4})$$
$$(\mathbf{a}_1, b_1) \quad , \quad (\mathbf{a}_2, b_2) \quad \mapsto \quad (\mathbf{a}_1 + \mathbf{a}_2, b_1 + b_2 + \frac{5q}{8})$$

Lightweight Refreshing

We have **HomNAND**:

$$\text{LWE}_s^4 \left(m_1, \frac{q}{16} \right) \times \text{LWE}_s^4 \left(m_2, \frac{q}{16} \right) \rightarrow \text{LWE}_s^2 \left(m_1 \bar{\wedge} m_2, \frac{q}{4} \right)$$

To build an FHE we require a relaxed function **LightRefresh**:

$$\mathbf{LightRefresh} : \text{LWE}_s^2 (m, q/4) \rightarrow \text{LWE}_s^4 (m, q/16)$$

whereas previous works required:

$$\mathbf{Refresh} : \text{LWE}_s^2 (m, q/4) \rightarrow \text{LWE}_s^2 (m, E), E \ll q.$$

As usual, we will use Key Switching, Mod Switching, and Homomorphic Decryption.

Outline

Introduction/Summary

The new NAND gate

Simpler Refreshing

Conclusion

Decryption using an Accumulator

$$\text{Dec}_s(\mathbf{a}, b) = \text{msb}(b - \langle \mathbf{a}, \mathbf{s} \rangle \bmod q)$$

$\text{Dec}_s(\mathbf{a}, b)$:

$acc \leftarrow b$

for $i = 1$ to n :

$acc \leftarrow acc - a_i \cdot s_i \bmod q$

Return $\text{msb}(acc)$

Decryption using an Accumulator

$$\text{Dec}_s(\mathbf{a}, b) = \text{msb}(b - \langle \mathbf{a}, \mathbf{s} \rangle \bmod q)$$

Homomorphic decryption given $E(\mathbf{s})$:

$E(\text{acc}) \leftarrow b$

for $i = 1$ to n :

$E(\text{acc}) \leftarrow E(\text{acc}) - E(a_i \cdot s_i) \bmod q$

Return $\text{LWE}(\text{msb}(\text{acc}))$

We need to perform this algorithm given Encryption of the key \mathbf{s} .

Decryption using an Accumulator

$$\text{Dec}_s(\mathbf{a}, b) = \text{msb}(b - \langle \mathbf{a}, \mathbf{s} \rangle \bmod q)$$

Homomorphic decryption given $E(\mathbf{s})$:

```
 $E(acc) \leftarrow b$   
for  $i = 1$  to  $n$ :  
     $E(acc) \leftarrow E(acc) - E(a_i \cdot s_i) \bmod q$   
Return  $\text{LWE}(\text{msb}(acc))$ 
```

We need to perform this algorithm given Encryption of the key \mathbf{s} .

- ▶ Initialization
- ▶ Addition mod q with a **fresh ciphertext** $E(a \cdot s_i)$
- ▶ A final test for the msb, in LWE form

Implementing the Accumulator

The framework follows from [AP14]:

- ▶ Use (\times) -homomorphism to implement $(\mathbb{Z}_q, +)$
- ▶ Use $(+)$ -homomorphism to implement msb testing.

[AP14] builds $(\mathbb{Z}_q, +)$ as an **outer structure**:
convolution products and CRT.

We optimize this construction in the case of cyclotomic ring

$$\mathcal{R} = \mathbb{Z}[X]/(X^N + 1).$$

We embed \mathbb{Z}_q in the group $(\langle X \rangle, \times)$, the group of roots of unity.

Ring version of [GSW13]

$Q = 2^k$, Gadget matrix: $\mathbf{G} = [\mathbf{I}, 2\mathbf{I}, 4\mathbf{I} \dots 2^{k-1}\mathbf{I}]^t \in \mathbb{Z}_Q^{n+1 \times (n+1)k}$.

$$E_s(m) = [\mathbf{A}, \mathbf{A}s + \mathbf{e}] + m \cdot \mathbf{G}$$

Dec_s: extract an LWE_s ciphertext (last row) and decrypt.

Supports Add. and Mult. for **small messages** $m \in \{-1, 0, 1\}$.

Ring version of [GSW13]

$Q = 2^k$, Gadget matrix: $\mathbf{G} = [\mathbf{I}, 2\mathbf{I}, 4\mathbf{I} \dots 2^{k-1}\mathbf{I}]^t \in \mathbb{Z}_Q^{n+1 \times (n+1)k}$.

$$\mathbf{E}_s(m) = [\mathbf{A}, \mathbf{A}s + \mathbf{e}] + m \cdot \mathbf{G}$$

Dec_s: extract an LWE_s ciphertext (last row) and decrypt.

Supports Add. and Mult. for **small messages** $m \in \{-1, 0, 1\}$.

Cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, $2N = q$ is a power of 2.

Generalized Gadget matrix: $\mathbf{G} = u \cdot [\mathbf{I}, b\mathbf{I}, b^2\mathbf{I} \dots b^{k-1}\mathbf{I}]^t \in \mathcal{R}_Q^{2 \times 2k}$.

$$\mathbf{E}_s(m \in \mathbb{Z}_q) = [\mathbf{a}, \mathbf{a} \cdot s + \mathbf{e}] + X^m \cdot \mathbf{G}$$

Supports addition for all message $m \in \mathbb{Z}_q$.

The group of roots of unity and msb

$$\begin{array}{c|cccc|cccc} m & 0 & 1 & \dots & \frac{q}{2}-1 & \frac{q}{2} & \frac{q}{2}+1 & \dots & q-1 \\ X^m & 1 & X & \dots & X^{N-1} & -1 & -X & \dots & -X^{N-1} \end{array}$$

Take the vector representation of X^m

$$\mathbf{x}_m \left| \begin{array}{c|c|c|c|c} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} & \ddots & \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} & \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ -1 \\ \vdots \\ 0 \end{bmatrix} & \ddots & \begin{bmatrix} 0 \\ 0 \\ \vdots \\ -1 \end{bmatrix} \end{array} \right.$$

Sum all the coordinates

$$\langle \mathbf{1}, \mathbf{x}_m \rangle \left| \begin{array}{c|c|c|c|c} 1 & 1 & \dots & 1 & -1 & -1 & \dots & -1 \end{array} \right.$$

$$\frac{\langle \mathbf{1}, \mathbf{x}_m \rangle + 1}{2} = \frac{(-1)^{\text{msb}(m)} + 1}{2} = \text{msb}(m).$$

Improvements

Improvement over the bootstrapping of [AP14]:

- ▶ Generic $\tilde{\Omega}(n)$ speed-up from Ring structure
- ▶ An extra $\tilde{\Omega}(\log^3 q)$ speed-up by **embedding**
- ▶ Error after bootstrapping reduced by $O(\sqrt{n} \log n)$.

In addition to our new NAND gate, implementation becomes reasonable.

Outline

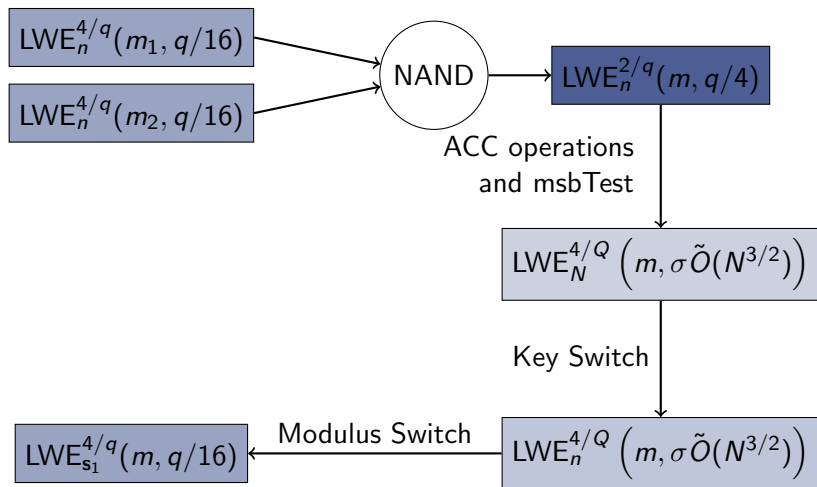
Introduction/Summary

The new NAND gate

Simpler Refreshing

Conclusion

The ciphertext cycle



Parameter Proposal

Parameters.

LWE parameters: $n = 500$ $q = 512.$
Ring-GSW parameters: $N = 1024$ $Q = 2^{32}.$
Gadget Matrix: $Q/8 \cdot [\mathbf{I}, 2^{11} \cdot \mathbf{I}, 2^{22} \cdot \mathbf{I}]$

Key Size.

Bootstrapping Key Size 1032 MB
Key Switching Key Size: +314 MB } $\leq 1.3\text{GB}$

Running time.

Per NAND gate: 48,000 FFTs ≈ 0.48 sec

Security.

Root-Hermite factor the LWE scheme $\delta_1 = 1.0064$
Root-Hermite factor Ring-GSW scheme $\delta_2 = 1.0064$

Proof of Concept Implementation

- ▶ Coded in 4 days·man
room for implementation level optimization.
- ▶ Reasonably concise: ≤ 600 lines of C++ code
[HElib]: $\approx 20,000$ lines
- ▶ Using FFT³ over \mathbb{C} at double precision
in dimension 2048 to obtain negacyclic-FFT.
potentially slower than 32-bits NTT in dimension 1024.

Result: Homomorphic NAND & refreshing in 0.61 seconds
on a single standard 64-bit core at 3Ghz.

Not much slower than the amortized cost of bootstrapping in
[HElib], no delays, hackable.

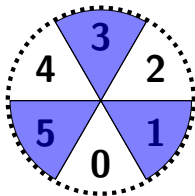
³FFTW library: The Fastest Fourier Transform in the West

Beyond binary gates

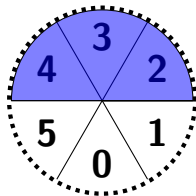
Replace msb by other membership testing function. Obtain :

- ▶ moderate fan-in symmetric gates
majority, threshold gates, **neurons**
- ▶ multiple outputs gates

Figure : Full-adder in **one refresh** (3-inputs, 2-outputs)



$$m_1 + m_2 + m_3 \in \{1, 3, 5\}$$
$$m_1 \oplus m_2 \oplus m_3$$



$$m_1 + m_2 + m_3 \in \{2, 3, 4\}$$
$$\text{carry}(m_1, m_2, m_3)$$

Open Source: <https://github.com/lucas/FHEW>

```
> click it, clone it, pull it, branch it,  
> hack it, use it, break it, fix it,  
> view it, code it, jam - debug it,  
> write it, cut it, paste it, save it,  
> load it, check it, quick - rewrite it,  
> commit, push it and pull-request it  
> ...  
> Homomorphic ... Homomorphic ...
```

Updated full paper: <https://eprint.iacr.org/2014/816>

Acknowledgements

The authors are thankful to Igors Stepanovs and Max Fillinger for helpful conversations and comments on this work.