1 Introduction

This document describes an LWE-based fully homomorphic encryption scheme. We begin by defining a somewhat homomorphic encryption scheme in Section 3, and then build upon it to achieve a fully homomorphic encryption scheme in Section 4. We analyze the choice of parameters in Section 5 and finish by discussing an attempt at implementing the scheme in Section 6.

2 Notation

We denote matrices by boldface uppercase letters, and vectors by boldface lowercase letters. If $\mathbf{A}$ is a matrix and $\mathbf{v}$ is a vector, then $\mathbf{A}[i]$ denotes the i-th row of $\mathbf{A}$ and $\mathbf{v}[i]$ denotes the i-th element of $\mathbf{v}$. The elements and rows are indexed from 1. For a scalar value x we let $x^{(j)}$ denote the j-th least significant bit of x; the same notation extends to matrices, applied to each of its elements individually. If $\mathbf{A}, \mathbf{B}$ are matrices then $(\mathbf{A}, \mathbf{B})$ denotes the vertical stacking and $[\mathbf{A}, \mathbf{B}]$ denotes the horizontal stacking of these matrices. If S is a finite set, we let $x \xleftarrow{\$} S$ denote sampling from S a value x uniformly at random. If $\mathcal{D}$ is a distribution, we write $x \leftarrow \mathcal{D}$ to denote sampling according to $\mathcal{D}$. For an integer z, we let $z \pmod q$ and $[z]_q$ denote the remainder of z modulo q.

3 Somewhat Homomorphic Encryption Scheme

All ciphertexts of this scheme belong to $\mathbb{Z}_q^{n\ell \times n}$.

SWHE.Setup(n, ℓ)
$\beta \leftarrow \lceil \sqrt{n} \rceil$; $q \leftarrow 2^\ell$
$pp \leftarrow (n, \ell, \beta, q)$
return pp

Kg
$\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$; return $\mathbf{s}$

SWHE.Enc$(\mathbf{s} \in \mathbb{Z}_q^{n-1}, m \in \mathbb{Z}_2)$
$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{(n-1)\times n\ell}$; $\mathbf{e} \leftarrow \chi_\beta^{n\ell} \in \{-\beta, \ldots, \beta\}^{n\ell}$
$\mathbf{b} \leftarrow \mathbf{A}^T\mathbf{s} + \mathbf{e} \pmod q \in \mathbb{Z}_q^{n\ell}$
$\mathbf{G} \leftarrow (\mathbf{I}, 2\mathbf{I}, 4\mathbf{I}, \ldots, 2^{\ell-1}\mathbf{I}) \in \mathbb{Z}_q^{n\ell \times n}$
$\mathbf{C} \leftarrow [\mathbf{A}^T, \mathbf{b}] + m\mathbf{G} \pmod q$
return $\mathbf{C}$

SWHE.Dec$(\mathbf{s} \in \mathbb{Z}_q^{n-1}, \mathbf{C})$
$\overline{\mathbf{s}} \leftarrow (-\mathbf{s}, 1) \pmod q \in \mathbb{Z}_q^n$
$\mathbf{c} \leftarrow \mathbf{C}[n\ell] \in \mathbb{Z}_q^{1\times n}$
$v \leftarrow \mathbf{c} \cdot \overline{\mathbf{s}} \pmod q \in \mathbb{Z}_q$
if $q/4 < v < 3q/4$ then return 1
else return 0

Note that the Setup procedure creates and returns public parameters pp. It is implicitly assumed that all other procedures take pp as their first argument and start by extracting $(n, \ell, \beta, q) \leftarrow pp$.

SWHE.Neg$(\mathbf{C})$
$\mathbf{C}_{\mathsf{res}} \leftarrow [\mathbf{G} - \mathbf{C}]_q$
return $\mathbf{C}_{\mathsf{res}}$

SWHE.Add$(\mathbf{C}_1, \mathbf{C}_2)$
$\mathbf{C}_{\mathsf{res}} \leftarrow [\mathbf{C}_1 + \mathbf{C}_2]_q$
return $\mathbf{C}_{\mathsf{res}}$

SWHE.Mult$(\mathbf{C}_1, \mathbf{C}_2)$
$\overline{\mathbf{C}}_1 \leftarrow \left[\mathbf{C}_1^{(0)}, \ldots, \mathbf{C}_1^{(\ell-1)}\right] \in \left[\mathbb{Z}_2^{n\ell\times n}, \ldots, \mathbb{Z}_2^{n\ell\times n}\right] = \mathbb{Z}_2^{n\ell\times n\ell}$
$\mathbf{C}_{\mathsf{res}} \leftarrow \overline{\mathbf{C}}_1 \cdot \mathbf{C}_2 \pmod q \in \left[\mathbb{Z}_2^{n\ell\times n\ell} \times \mathbb{Z}_q^{n\ell\times n}\right]_q = \mathbb{Z}_q^{n\ell\times n}$
return $\mathbf{C}_{\mathsf{res}}$

Parameters. The scheme can use an arbitrary modulus q. However, the change of q may require some further modifications to the scheme, such as making sure that the matrix $\mathbf{G}$ contains the row vector $(0, \ldots, 0, q/2)$ that makes the decryption possible. It is required that $\beta \geq \sqrt{n}$ in order to prove that LWE is as hard as worst-case lattice problems, so we set $\beta = \lceil \sqrt{n} \rceil$. For the sake of simplicity, in this work we assume that χ_β is a uniform distribution with support $\{-\beta, \ldots, \beta\} \subset \mathbb{Z}$.

DECRYPTION. Let $\mathbf{a} = \mathbf{A}^T[n\ell]$, $\mathbf{g} = \mathbf{G}[n\ell] = (0, \ldots, 0, q/2)$, and $e = \mathbf{e}[n\ell]$. Then

$$\begin{aligned}\mathbf{c} \cdot \bar{\mathbf{s}} &= ([\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e] + m\mathbf{g}) \cdot (-\mathbf{s}, 1) \\ &= [\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e + m \cdot q/2] \cdot (-\mathbf{s}, 1) \\ &= (-\mathbf{a} \cdot \mathbf{s}) + (\mathbf{a} \cdot \mathbf{s} + e + m \cdot q/2) \\ &= m \cdot q/2 + e.\end{aligned}$$

Therefore, the decryption is correct as long as

$$\|\mathbf{e}\|_\infty \leq \beta < q/4. \tag{1}$$

More generally, this requirement applies to the current (rather than the initial) error bound of the ciphertext. Also note that as long as the error bound remains strictly less than $q/4$, it should never happen that $v = q/4$ or $v = 3q/4$ holds in $\mathsf{SWHE.Dec}$.

ERROR BOUNDS. The error bound of a fresh ciphertext is β. Consider ciphertexts $\mathbf{C}_1 = [\mathbf{A}_1, \mathbf{A}_1\mathbf{s} + \mathbf{e}_1] + m_1\mathbf{G}$ and $\mathbf{C}_2 = [\mathbf{A}_2, \mathbf{A}_2\mathbf{s} + \mathbf{e}_2] + m_2\mathbf{G}$ with error bounds β_1 and β_2, respectively. Then the error bound of $\mathsf{Neg}(\mathbf{C}_1)$ is β_1, and the error bound of $\mathsf{Add}(\mathbf{C}_1, \mathbf{C}_2)$ is $\beta_1 + \beta_2$. For multiplication, we have

$$\begin{aligned}\overline{\mathbf{C}}_1 \cdot \mathbf{C}_2 &= \overline{\mathbf{C}}_1 \cdot ([\mathbf{A}_2, \mathbf{A}_2\mathbf{s} + \mathbf{e}_2] + m_2\mathbf{G}) \\ &= [\overline{\mathbf{C}}_1 \cdot \mathbf{A}_2, \overline{\mathbf{C}}_1\mathbf{A}_2\mathbf{s} + \overline{\mathbf{C}}_1\mathbf{e}_2] + m_2\mathbf{C}_1 \\ &= [\overline{\mathbf{C}}_1 \cdot \mathbf{A}_2 + m_2\mathbf{A}_1, (\overline{\mathbf{C}}_1\mathbf{A}_2 + m_2\mathbf{A}_1)\mathbf{s} + (\overline{\mathbf{C}}_1\mathbf{e}_2 + m_2\mathbf{e}_1)] + m_1m_2\mathbf{G}.\end{aligned}$$

Hence, the error bound of $\mathsf{Mult}(\mathbf{C}_1, \mathbf{C}_2)$ is

$$n\ell\beta_2 + |m_2|\beta_1. \tag{2}$$

Note that as a result of performing homomorphic operations, m_2 may be greater than 1.

4 Fully Homomorphic Encryption Scheme

In order to build a FHE scheme, we add a bootstrapping functionality to the SWHE scheme described in Section 3. Bootstrapping is done by the $\mathsf{Recrypt}$ procedure and consists of two parts: key switching and homomorphic decryption. The key switching subroutine takes a ciphertext with a high error bound and produces a new ciphertext that encrypts the same plaintext message using another SWHE scheme with a smaller modulus. The resulting ciphertext has a slightly increased error bound. The smaller modulus size now allows to run a subroutine that homomorphically decrypts the produced ciphertext, such that the result is again encrypted using the original SWHE scheme and has a small error bound.

$\underline{\mathsf{FHE.Setup}(n, c)}$
$(n_0, \ell_0, n_1, \ell_1) \leftarrow \mathsf{Param}(n)$
$pp_0 \leftarrow \mathsf{SWHE.Setup}(n_0, \ell_0)$
$pp_1 \leftarrow \mathsf{SWHE.Setup}(n_1, \ell_1)$
$pp \leftarrow (pp_0, pp_1)$
return pp

$\underline{\mathsf{FHE.Kg}}$
$\mathbf{s}_0 \xleftarrow{\$} \mathsf{SWHE.Kg}(pp_0) \in \mathbb{Z}_{q_0}^{n_0-1}$
$\mathbf{s}_1 \xleftarrow{\$} \mathsf{SWHE.Kg}(pp_1) \in \mathbb{Z}_{q_1}^{n_1-1}$
$\mathsf{evk}^{\mathsf{hdec}} \xleftarrow{\$} \mathsf{FHE.Kg}^{\mathsf{hdec}}(\mathbf{s}_0, \mathbf{s}_1)$
$\mathsf{evk}^{\mathsf{kswitch}} \xleftarrow{\$} \mathsf{FHE.Kg}^{\mathsf{kswitch}}(\mathbf{s}_0, \mathbf{s}_1)$
$\mathsf{evk} \leftarrow (\mathsf{evk}^{\mathsf{hdec}}, \mathsf{evk}^{\mathsf{kswitch}})$
return $(\mathsf{evk}, \mathbf{s}_1)$

$\underline{\mathsf{FHE.Enc}(\mathbf{s}_1 \in \mathbb{Z}_{q_1}^{n_1-1}, m \in \mathbb{Z}_2)}$
return $\mathsf{SWHE.Enc}(pp_1, \mathbf{s}_1, m)$

$\underline{\mathsf{FHE.Dec}(\mathbf{s}_1 \in \mathbb{Z}_{q_1}^{n_1-1}, \mathbf{C} \in \mathbb{Z}_{q_1}^{n_1\ell_1 \times n_1})}$
return $\mathsf{SWHE.Dec}(pp_1, \mathbf{s}_1, \mathbf{C})$

$\underline{\mathsf{FHE.Recrypt}(\mathsf{evk}, \mathbf{C} \in \mathbb{Z}_{q_1}^{n_1\ell_1 \times n_1})}$
$(\mathsf{evk}^{\mathsf{hdec}}, \mathsf{evk}^{\mathsf{kswitch}}) \leftarrow \mathsf{evk}$
$\mathbf{c} \leftarrow \mathsf{KSwitch}(\mathsf{evk}^{\mathsf{kswitch}}, \mathbf{C}) \in \mathbb{Z}_{q_0}^{1 \times n_0}$
return $\mathsf{HDec}(\mathsf{evk}^{\mathsf{hdec}}, \mathbf{c}) \in \mathbb{Z}_{q_1}^{n_1\ell_1 \times n_1}$

The Setup procedure returns public parameters pp and we assume that they are available to all other procedures of the FHE scheme. The choice of parameters done by Param is discussed in Section 5. All ciphertexts in the Neg, Add and Mult procedures defined below belong to $\mathbb{Z}_{q_1}^{n_1\ell_1\times n_1}$.

$\underline{\mathsf{FHE.Neg}(\mathbf{C})}$	$\underline{\mathsf{FHE.Add}(\mathbf{C}_1,\mathbf{C}_2)}$	$\underline{\mathsf{FHE.Mult}(\mathbf{C}_1,\mathbf{C}_2)}$
return $\mathsf{SWHE.Neg}(pp_1,\mathbf{C})$	return $\mathsf{SWHE.Add}(pp_1,\mathbf{C}_1,\mathbf{C}_2)$	return $\mathsf{SWHE.Mult}(pp_1,\mathbf{C}_1,\mathbf{C}_2)$

4.1 Key Switching

Given a plaintext encrypted under key $\mathbf{s}_1$, the key switching produces an encryption of the same plaintext under a different key $\mathbf{s}_0$. First, the key generation subroutine $\mathsf{Kg}^{\mathsf{kswitch}}$ uses key $\mathbf{s}_0$ in order to "encrypt" each of the (n_1-1) individual ℓ_1-bit values of $\mathbf{s}_1$. Then the KSwitch procedure takes an arbitrary ciphertext encrypted under key $\mathbf{s}_1$ and uses the homomorphism of the scheme in order to "decrypt" it inside the protected domain of $\mathbf{s}_0$. Informally, it computes

$$\mathsf{enc}_{\mathbf{s}_0}\left(\frac{q_0}{q_1}\mathbf{s}_1\right)\cdot\mathbf{c}=\mathsf{enc}_{\mathbf{s}_0}\left(\frac{q_0}{q_1}\cdot\mathbf{s}_1\mathbf{c}\right)=\mathsf{enc}_{\mathbf{s}_0}\left(\frac{q_0}{2}m+\frac{q_0}{q_1}e\right),$$

where e is the error bound of the initial ciphertext. This also demonstrates the conceptual difference between key switching and bootstrapping: the former does not decrease the error bound.

$\underline{\mathsf{FHE.Kg}^{\mathsf{kswitch}}(\mathbf{s}_0\in\mathbb{Z}_{q_0}^{n_0-1},\mathbf{s}_1\in\mathbb{Z}_{q_1}^{n_1-1})}$

$\beta_0\leftarrow\lceil\sqrt{n_0}\rceil$
for $i=1,\ldots,n_1-1$ do
  $\mathbf{A}_i\xleftarrow{\$}\mathbb{Z}_{q_0}^{(n_0-1)\times\ell_1}$
  $\mathbf{e}\leftarrow\chi_{\beta_0}^{\ell_1}\in\{-\beta_0,\ldots,\beta_0\}^{\ell_1}$
  $\mathbf{x}\leftarrow\dfrac{\mathbf{s}_1[i]\cdot q_0}{q_1}\cdot(1,2,\ldots,2^{\ell_1-1})\in\mathbb{R}^{\ell_1}$
  $\mathbf{b}\leftarrow\mathbf{A}_i^T\mathbf{s}_0+\mathbf{e}+\lfloor\mathbf{x}\rceil\pmod{q_0}\in\mathbb{Z}_{q_0}^{\ell_1}$
  $\mathbf{C}_i\leftarrow\left[\mathbf{A}_i^T,\mathbf{b}\right]\in\mathbb{Z}_{q_0}^{\ell_1\times n_0}$
$\mathsf{evk}^{\mathsf{kswitch}}\leftarrow\{\mathbf{C}_i\}_i$; return $\mathsf{evk}^{\mathsf{kswitch}}$

$\underline{\mathsf{FHE.KSwitch}(\mathsf{evk}^{\mathsf{kswitch}},\mathbf{C}\in\mathbb{Z}_{q_1}^{n_1\ell_1\times n_1})}$

$\{\mathbf{C}_i\}_i\leftarrow\mathsf{evk}^{\mathsf{kswitch}}$; $\mathbf{c}\leftarrow\mathbf{C}[n_1\ell_1]\in\mathbb{Z}_{q_1}^{n_1}$
$c\leftarrow\mathbf{c}[n_1]\in\mathbb{Z}_{q_1}$; $r\leftarrow\lfloor(q_0/q_1)\cdot c\rceil\pmod{q_0}\in\mathbb{Z}_{q_0}$
$\mathbf{c}^*\leftarrow[0,\ldots,0,r]\in\mathbb{Z}_{q_0}^{1\times n_0}$
for $i=1,\ldots,(n_1-1)$ do
  $c\leftarrow\mathbf{c}[i]\in\mathbb{Z}_{q_1}$; $\mathbf{a}\leftarrow\left[c^{(0)},\ldots,c^{(\ell_1-1)}\right]\in\mathbb{Z}_2^{1\times\ell_1}$
  $\mathbf{x}\leftarrow\mathbf{a}\cdot\mathbf{C}_i\pmod{q_0}\in\left[\mathbb{Z}_2^{1\times\ell_1}\times\mathbb{Z}_{q_0}^{\ell_1\times n_0}\right]_{q_0}=\mathbb{Z}_{q_0}^{1\times n_0}$
  $\mathbf{c}^*\leftarrow\mathbf{c}^*-\mathbf{x}\pmod{q_0}\in\mathbb{Z}_{q_0}^{1\times n_0}$
return $\mathbf{c}^*\in\mathbb{Z}_{q_0}^{1\times n_0}$

We now demonstrate the correctness of the transformation and find the error bound of the produced ciphertext. The KSwitch procedure returns the result of the following expression:

$$\left[0,\ldots,0,\left\lfloor\frac{q_0}{q_1}\cdot\mathbf{c}[n_1]\right\rceil\right]-\sum_{i=1}^{n_1-1}\left[\mathbf{c}[i]^{(0)},\ldots,\mathbf{c}[i]^{(\ell_1-1)}\right]\cdot\left[\mathbf{A}_i^T,\mathbf{A}_i^T\mathbf{s}_0+\mathbf{e}_i+\left\lfloor\frac{\mathbf{s}_1[i]\cdot q_0}{q_1}\cdot(1,2,\ldots,2^{\ell_1-1})\right\rceil\right]$$

modulo q_0. First, we remove the rounding operations, by rewriting them as follows:

$$\left\lfloor\frac{q_0}{q_1}\cdot\mathbf{c}[n_1]\right\rceil=\frac{q_0}{q_1}\cdot\mathbf{c}[n_1]+h,$$

$$\left\lfloor\frac{\mathbf{s}_1[i]\cdot q_0}{q_1}\cdot(1,2,\ldots,2^{\ell_1-1})\right\rceil=\frac{\mathbf{s}_1[i]\cdot q_0}{q_1}\cdot(1,2,\ldots,2^{\ell_1-1})+\mathbf{h}_i,$$

where $h \in \mathbb{R}$ such that $|h| \leq 1/2$, and $\mathbf{h}_i \in \mathbb{R}^{\ell_1}$ such that $\|\mathbf{h}_i\|_\infty \leq 1/2$. Next, we denote the bit-vector $\left[\mathbf{c}[i]^{(0)}, \ldots, \mathbf{c}[i]^{(\ell_1-1)}\right]$ by $\vec{\mathbf{c}[i]}$, and we denote $\left(-\sum_{i=1}^{n_1-1} \vec{\mathbf{c}[i]}\mathbf{A}_i^T\right) \in \mathbb{Z}_{q_0}^{1\times(n_0-1)}$ by $\mathbf{v}$. Finally, we assume that the error bound of the ciphertext $\mathbf{C} \in \mathbb{Z}_{q_1}^{n_1\ell_1\times n_1}$ is αq_1. Then we can simplify the above expression, as follows:

$$
\begin{aligned}
&= \left[\mathbf{v}, \mathbf{v}\cdot\mathbf{s}_0 + h - \sum_{i=1}^{n_1-1} \vec{\mathbf{c}[i]}(\mathbf{e}_i + \mathbf{h}_i) + \frac{q_0}{q_1}\left(\mathbf{c}[n_1] - \sum_{i=1}^{n_1-1} \mathbf{s}_1[i]\vec{\mathbf{c}[i]}(1, 2, \ldots, 2^{\ell_1-1})\right)\right] \bmod q_0 \\
&= \left[\mathbf{v}, \mathbf{v}\cdot\mathbf{s}_0 + h - \sum_{i=1}^{n_1-1} \vec{\mathbf{c}[i]}(\mathbf{e}_i + \mathbf{h}_i) + \frac{q_0}{q_1}\left(\mathbf{c}[n_1] - \sum_{i=1}^{n_1-1} \mathbf{s}_1[i]\mathbf{c}[i]\right)\right] \bmod q_0 \\
&= \left[\mathbf{v}, \mathbf{v}\cdot\mathbf{s}_0 + h - \sum_{i=1}^{n_1-1} \vec{\mathbf{c}[i]}(\mathbf{e}_i + \mathbf{h}_i) + \frac{q_0}{q_1}\left(\frac{q_1}{2}m + \alpha q_1\right)\right] \bmod q_0 \\
&= \left[\mathbf{v}, \mathbf{v}\cdot\mathbf{s}_0 + \left(h + \alpha q_0 - \sum_{i=1}^{n_1-1} \vec{\mathbf{c}[i]}(\mathbf{e}_i + \mathbf{h}_i)\right) + \frac{q_0}{2}m\right] \bmod q_0
\end{aligned}
$$

Therefore, the error bound of the resulting ciphertext is

$$1/2 + \alpha q_0 + n_1\ell_1(\beta_0 + 1/2), \tag{3}$$

which must be strictly less than $q_0/4$.

4.2 Homomorphic Decryption

Let $\mathbf{c} \in \mathbb{Z}_{q_0}^{n_0}$ be the row of a SWHE ciphertext matrix (encrypted under key $\mathbf{s}_0$) that is required for decryption. The core decryption step requires to compute $v = \mathbf{c}\cdot(-\mathbf{s}_0, 1) \pmod{q_0} \in \mathbb{Z}_{q_0}$, which can be done as follows:

$$v = \mathbf{c}[n_0] - \sum_{i=1}^{n_0-1} \mathbf{s}_0[i]\cdot\mathbf{c}[i] = \mathbf{c}[n_0] - \sum_{i=1}^{n_0-1}\sum_{j=0}^{\log_2 q_0-1} (\mathbf{s}_0[i]\cdot 2^j)\cdot\mathbf{c}[i]^{(j)} \pmod{q_0}.$$

The ciphertext decrypts to 1 whenever $q/4 < v < 3q/4$.

In order to run homomorphic decryption, it is necessary to precompute the encryptions of $\mathbf{s}_0[i]\cdot 2^j$ (under key $\mathbf{s}_1$) for all possible combinations of i and j. We use an encoding that encrypts a value $\mathbb{Z}_{q_0}$ as q_0 separate ciphertexts. Specifically, for any i and j, the value $x = \mathbf{s}_0[i]\cdot 2^j$ is represented by $\{\mathbf{e}_{i,j,k}\}_{k\in\mathbb{Z}_{q_0}}$, where $\mathbf{e}_{i,j,k}$ encrypts 1 if and only if $k = x$.

Assume we want to compute $\{\mathbf{C}_k^{\mathsf{sum}}\}_{k\in\mathbb{Z}_{q_0}} = \sum_{i=1}^{n_0-1}\sum_{j=0}^{\log_2 q_0-1} (\mathbf{s}_0[i]\cdot 2^j)\cdot\mathbf{c}[i]^{(j)}$. Note that the bit-value $\mathbf{c}[i]^{(j)}$ serves as a flag indicating whether the encryption of $\mathbf{s}_0[i]\cdot 2^j$ is included in the sum. So whenever $\mathbf{c}[i]^{(j)} = 1$, we add $\{\mathbf{e}_{i,j,k}\}_{k\in\mathbb{Z}_{q_0}}$ (an encryption of $\mathbf{s}_0[i]\cdot 2^j$) to an intermediate (encrypted) sum $\{\mathbf{C}_k\}_{k\in\mathbb{Z}_{q_0}}$. Denote the result of this homomorphic addition by $\{\mathbf{C}_k^*\}_{k\in\mathbb{Z}_{q_0}}$. Then for any $k \in \mathbb{Z}_{q_0}$ the value of $\mathbf{C}_k^*$ can be computed as a convolution $\sum_{a=0}^{q_0-1} \mathbf{C}_{k-a}\cdot\mathbf{e}_{i,j,a}$.

Once $\{\mathbf{C}_k^{\mathsf{sum}}\}_{k\in\mathbb{Z}_{q_0}}$ is found, it only remains to process (informally) $v = \mathbf{c}[n_0] - \{\mathbf{C}_k^{\mathsf{sum}}\}_{k\in\mathbb{Z}_{q_0}}$. Here $\mathbf{c}[n_0] = z + \frac{q_0}{2}m + e$ for some $z \in \mathbb{Z}_{q_0}$, $m \in \mathbb{Z}_2$ and $|e| < q_0/4$, and $\{\mathbf{C}_k^{\mathsf{sum}}\}_{k\in\mathbb{Z}_{q_0}}$ is an encryption of z. The decryption of $\mathbf{c}$ is 1 iff $\frac{q_0}{4} < \left[\frac{q_0}{2}m + e\right]_{q_0} < \frac{3q_0}{4}$, which is equivalent to $\frac{q_0}{4} < [\mathbf{c}[n_0] - z]_{q_0} < \frac{3q_0}{4}$. So in order to get the encryption of m under key $\mathbf{s}_1$, we need to add up ciphertexts $\mathbf{C}_z^{\mathsf{sum}}$ for all values $z \in \mathbb{Z}_{q_0}$ that satisfy this inequality.

$\underline{\mathsf{FHE.Kg}^{\mathsf{hdec}}(\mathbf{s}_0, \mathbf{s}_1)}$
for $i = 1, \ldots, (n_0 - 1)$ do
  for $j = 0, \ldots, (\log_2 q_0 - 1)$ do
  $x \leftarrow \mathbf{s}_0[i] \cdot 2^j \pmod{q_0} \in \mathbb{Z}_{q_0}$
    for $k = 0, \ldots, (q_0 - 1)$ do
      if $(k = x)$ then $\mathbf{e}_{i,j,k} \xleftarrow{\$} \mathsf{SWHE.Enc}(pp_1, \mathbf{s}_1, 1)$
      else $\mathbf{e}_{i,j,k} \xleftarrow{\$} \mathsf{SWHE.Enc}(pp_1, \mathbf{s}_1, 0)$
$\mathsf{evk}^{\mathsf{hdec}} \leftarrow \{\mathbf{e}_{i,j,k}\}_{i,j,k}$; return $\mathsf{evk}^{\mathsf{hdec}}$

$\underline{\mathsf{FHE.Add}^{\mathsf{hdec}}(i, j)}$
if $\mathsf{empty}_{\mathbf{C}}$ then
  for $k = 0, \ldots, (q_0 - 1)$ do $\mathbf{C}_k \leftarrow \mathbf{e}_{i,j,k}$
  $\mathsf{empty}_{\mathbf{C}} \leftarrow \mathsf{false}$; return
for $k = 0, \ldots, (q_0 - 1)$ do $\mathbf{C}_k^* \leftarrow \mathsf{FHE.SetZeros}^{\mathsf{hdec}}$
for $u = 0, \ldots, (q_0 - 1)$ do
  for $v = 0, \ldots, (q_0 - 1)$ do
    $k \leftarrow u + v \pmod{q_0} \in \mathbb{Z}_{q_0}$
    $\mathbf{C}_{\mathsf{prod}} \leftarrow \mathsf{SWHE.Mult}(pp_1, \mathbf{C}_u, \mathbf{e}_{i,j,v})$
    $\mathbf{C}_k^* \leftarrow \mathsf{SWHE.Add}(pp_1, \mathbf{C}_k^*, \mathbf{C}_{\mathsf{prod}})$
for $u = k, \ldots, (q_0 - 1)$ do $\mathbf{C}_k \leftarrow \mathbf{C}_k^*$

$\underline{\mathsf{FHE.HDec}(\mathsf{evk}^{\mathsf{hdec}}, \mathbf{c} \in \mathbb{Z}_{q_0}^{1\times n_0})}$
$\{\mathbf{e}_{i,j,k}\}_{i,j,k} \leftarrow \mathsf{evk}^{\mathsf{hdec}}$
$\mathbf{C}_{\mathsf{res}} \leftarrow \mathsf{FHE.SetZeros}^{\mathsf{hdec}}$
for $i = 0, \ldots, (q_0 - 1)$ do
  $\mathbf{C}_i \leftarrow \mathsf{FHE.SetZeros}^{\mathsf{hdec}}$
$\mathsf{empty}_{\mathbf{C}} \leftarrow \mathsf{true}$
for $i = 1, \ldots, (n_0 - 1)$ do
  for $j = 0, \ldots, (\log_2 q_0 - 1)$ do
    $u \leftarrow \mathbf{c}[i] \in \mathbb{Z}_{q_0}$
    if $(u^{(j)} = 1)$ then $\mathsf{FHE.Add}^{\mathsf{hdec}}(i, j)$
for $i = 0, \ldots, (q_0 - 1)$ do
  $x \leftarrow \mathbf{c}[n_0] - i \pmod{q_0}$
  if $q_0/4 < x < 3q_0/4$ then
    $\mathbf{C}_{\mathsf{res}} \leftarrow \mathsf{SWHE.Add}(pp_1, \mathbf{C}_{\mathsf{res}}, \mathbf{C}_i)$
return $\mathbf{C}_{\mathsf{res}} \in \mathbb{Z}_{q_1}^{n_1\ell_1\times n_1}$

$\underline{\mathsf{FHE.SetZeros}^{\mathsf{hdec}}}$
return $\{0\}^{n_1\ell_1\times n_1} \in \mathbb{Z}_{q_1}^{n_1\ell_1\times n_1}$

Note that $\mathsf{FHE.HDec}$ will produce an incorrect result whenever $\mathbf{c}[1] = \ldots = \mathbf{c}[n_0 - 1] = 0$, since no ciphertexts $\mathbf{e}_{i,j,k}$ will participate in the sum. However, for a reasonable value of n_0 this will happen only with a negligible probability.

Let β_1 be the error bound of ciphertexts $\{\mathbf{e}_{i,j,k}\}_{i,j,k}$. If the current error bound of $\{\mathbf{C}_u\}_u$ in HDec is β_{res} and the procedure $\mathsf{Add}^{\mathsf{hdec}}$ is called for some parameters i and j, then after its termination the error bound of $\{\mathbf{C}_u\}_u$ becomes:

$$\sum_{v=0}^{q_0-1}(n_1\ell_1\beta_1 + |m_v|\beta_{\mathsf{res}}) = q_0 n_1\ell_1\beta_1 + \beta_{\mathsf{res}},$$

where m_v is the plaintext bit encrypted by $\mathbf{e}_{i,j,v}$. It follows that the output ciphertext of the procedure HDec has the error bound of size

$$(q_0/2 - 1) \cdot n_0 \log_2 q_0 \cdot (q_0 n_1\ell_1\beta_1), \tag{4}$$

where the factor $(q_0/2 - 1)$ arises from computing the result $\mathbf{C}_{\mathsf{res}}$ as a sum of individual bit encryptions. For a correct decryption, we require (4) to be strictly less than $q_1/4$.

5 Choice of Parameters

Given ar arbitrary security parameter n for the entire FHE scheme, we want to find the values of n_0, ℓ_0, n_1, ℓ_1 that provide the best performance of the scheme while also guaranteeing its correctness. Note that there is no reason for n_0 or n_1 to be greater than n, so we set

$$n_0 = n_1 = n. \tag{5}$$

It remains to choose the optimal values for ℓ_0 and ℓ_1.

Recall that in this work we assume $\beta_i = \lceil \sqrt{n_i} \rceil$ and $q_i = 2^{\ell_i}$ for $i \in \{0, 1\}$. According to (5), we have $\beta_0 = \beta_1$, so we will refer to this value simply as β.

In general, we want to set ℓ_0 as small as possible to ensure the efficiency of the homomorphic decryption step, wheareas ℓ_1 should be chosen large enough to ensure its correctness. Since (1) requires $\beta < q_0/4$, it follows that

$$\ell_0 > \log_2 \beta + 2 = \log_2 \lceil \sqrt{n} \rceil + 2, \tag{6}$$

and hence $q_0 = 2^{\ell_0} > 4 \cdot \lceil \sqrt{n} \rceil$. However, in the next section we will see that ℓ_0 must be considerably larger in order to ensure the correctness of decryption.

5.1 Relative Error Bounds

According to (4), the relative error bound of a ciphertext produced by FHE.Recrypt is

$$\alpha_{\mathsf{Recrypt}} = \frac{(q_0/2 - 1) \cdot n_0 \log_2 q_0 \cdot (q_0 n_1 \ell_1 \beta_1)}{q_1}.$$

We want to be able to perform at least one multiplication between two ciphertexts with this error bound. By (2), the relative error bound after performing the multiplication is

$$\alpha_{\mathsf{Recrypt+Mult}} = \alpha_{\mathsf{Recrypt}} \cdot (n_1 \ell_1 + 1) = \frac{(q_0/2 - 1) \cdot n_0 \log_2 q_0 \cdot (q_0 n_1 \ell_1 \beta_1) \cdot (n_1 \ell_1 + 1)}{q_1}.$$

Finally, by (3), the relative error bound after performing the key switching is

$$\begin{aligned} \alpha_{\mathsf{Recrypt+Mult+KSwitch}} &= \frac{(q_0/2 - 1) \cdot n_0 \log_2 q_0 \cdot (q_0 n_1 \ell_1 \beta_1) \cdot (n_1 \ell_1 + 1)}{q_1} + \frac{1/2 + n_1 \ell_1 (\beta_0 + 1/2)}{q_0} \\ &= \frac{(q_0/2 - 1) n^2 \ell_0 q_0 \ell_1 \beta (n \ell_1 + 1)}{q_1} + \frac{1/2 + n \ell_1 (\beta + 1/2)}{q_0}. \end{aligned} \tag{7}$$

In order to ensure the correctness of the FHE scheme, we require that $\alpha_{\mathsf{Recrypt+Mult+KSwitch}} < 1/4$.

The right summand of (7) is roughly $\dfrac{n^{1.5} \ell_1}{q_0}$, so we can lower bound the values of q_0 and ℓ_0. Specifically, one can expect $\ell_0 = c_0 \log_2 n$ for some constant c_0. Likewise, for a fixed value of q_0, the right summand upper bounds ℓ_1, while the left summand provides a lower bound for it. In general, it seems there is no good way to get good non-asymptotic parameter estimates.

In order to get some asymptotic estimates, we let $\ell_0 = c_0 \log_2 n$, $\ell_1 = c_1 \log_2 n$ and $\beta = n^{0.5}$, hence $q_0 = n^{c_0}$ and $q_1 = n^{c_1}$. We rewrite (7) as follows:

$$\alpha_{\mathsf{Recrypt+Mult+KSwitch}} \approx c_0 c_1^2 n^{3.5 + 2c_0 - c_1} (\log_2 n)^3 + c_1 n^{1.5 - c_0} \log_2 n. \tag{8}$$

Each of the two summands has to be small, such that their sum is less than $1/4$. Hence, a possible parameter setting is $q_0 = \tilde{O}(n^{1.5})$ and $q_1 = \tilde{O}(n^{6.5})$.

5.2 Memory Usage and Time Complexity

In this section, we assume that ℓ_1 is reasonably small, so that q_1 fits in a single machine word and hence does not introduce potentially costly computations involving long arithmetic. (Ideally, $(q_1)^2$ should also fit in a single word.)

First, we determine the time complexity of all FHE procedures:

$\mathsf{FHE.Setup}$	$\mathsf{FHE.Enc}$	$\mathsf{FHE.Dec}$	$\mathsf{FHE.Neg}$	$\mathsf{FHE.Add}$	$\mathsf{FHE.Mult}$
$O(1)$	$O(n^2\ell_1)$	$O(n)$	$O(n^2\ell_1)$	$O(n^2\ell_1)$	$O(n^3\ell_1^2)$

The procedure $\mathsf{FHE.Mult}$ multiplies two matrices of sizes $n\ell_1 \times n\ell_1$ and $n\ell_1 \times n$. The indicated complexity of $\mathsf{FHE.Mult}$ assumes that we perform the multiplication in a straightforward way. However, it is possible to reduce this asymptotic estimate (e.g. by using fast matrix multiplication). For simplicity, we denote the running time of $\mathsf{FHE.Mult}$ by T_{Mult}. The complexity of $\mathsf{FHE.Dec}$ assumes that the procedure does not need to read the entire ciphertext matrix, but only the row that is required for decryption.

$\mathsf{FHE.Kg}^{\mathsf{kswitch}}$	$\mathsf{FHE.KSwitch}$	$\mathsf{FHE.Kg}^{\mathsf{hdec}}$	$\mathsf{FHE.Add}^{\mathsf{hdec}}$	$\mathsf{FHE.HDec}$
$O(n^2\ell_1)$	$O(n^2\ell_1)$	$O(n^3\ell_0\ell_1 2^{\ell_0})$	$O(2^{2\ell_0} \cdot T_{\mathsf{Mult}})$	$O(n\ell_0 2^{2\ell_0} \cdot T_{\mathsf{Mult}})$

The procedure $\mathsf{FHE.Kg}^{\mathsf{hdec}}$ encrypts $n\ell_0 2^{\ell_0}$ bits, where the time complexity of each encryption is $O(n^2\ell_1)$. The complexity of $\mathsf{FHE.HDec}$ consists of $n\ell_0$ calls to $\mathsf{FHE.Add}^{\mathsf{hdec}}$. The total time complexity of $\mathsf{FHE.Recrypt}$ is $O(n\ell_0 2^{2\ell_0} \cdot T_{\mathsf{Mult}})$, with an asymptotic estimate roughly $\tilde{O}(n^{6.5})$, assuming that T_{Mult} can be done in $O(n^{2.5}\ell_1^2)$.

Finally, we determine the memory usage of our scheme, listing the sizes of a ciphertext and the evaluation keys:

Ciphertext	$\mathsf{evk}^{\mathsf{kswitch}}$	$\mathsf{evk}^{\mathsf{hdec}}$
$n^2\ell_1^2$ bits	$n^2\ell_0\ell_1$ bits	$n^3\ell_0\ell_1^2 2^{\ell_0}$ bits

The evaluation key $\mathsf{evk}^{\mathsf{hdec}}$ contains $n\ell_0 2^{\ell_0}$ ciphertexts, each of size $n^2\ell_1^2$ bits.

5.3 Concrete Parameters

An easy way to find concrete parameters for different values of n is to write a script that does an exhaustive search over all possible values of ℓ_0 and ℓ_1 and picks the smallest combination that satisfies (7). However, it is important to note that (7) is based on the worst-case scenario, and in practice it might be possible to use smaller parameters such that the correctness of the scheme holds with an overwhelming probability.

In the following table, we use the memory and time complexity estimates from the previous section. We let $T_{\mathsf{Mult}} = O(n^{2.5}\ell_1^2)$, and we assume that one can perform 10^9 operations per second. It is a very generous assumption, since our time complexity estimates did not take in account that we work with ℓ_1-bit numbers (and perform a lot of multiplications on them).

n	β	ℓ_0	ℓ_1	$\mathsf{FHE.Recrypt}$	$\mathsf{evk}^{\mathsf{hdec}}$
2	2	10	42	$20 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 3.3$ m	18 MB
3	2	11	45	$138 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 73$ m	147 MB
4	2	12	48	$805 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 17$ h	864 MB
5	3	12	52	$1 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 43$ h	2 GB

n	β	ℓ_0	ℓ_1	FHE.Recrypt	$\mathsf{evk}^{\mathsf{hdec}}$
6	3	13	53	$5.2 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 16$ d	8 GB
7	3	13	54	$6.1 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 27$ d	13 GB
8	3	13	55	$6.9 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 45$ d	20 GB
9	3	13	57	$7.8 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 72$ d	30 GB
10	4	14	58	$37 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 463$ d	90 GB
11	4	14	59	$41 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 669$ d	124 GB

The following are the estimates for the case when $\beta = 1$ is fixed irregardless of n:

n	β	ℓ_0	ℓ_1	FHE.Recrypt	$\mathsf{evk}^{\mathsf{hdec}}$
2	1	9	40	$4.7 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 42$ s	8 MB
3	1	10	42	$31 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 15$ m	59 MB
4	1	11	45	$184 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 4$ h	349 MB
5	1	11	47	$230 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 8$ h	742 MB
6	1	11	49	$276 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 17$ h	2 GB
7	1	12	50	$1.4 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 6$ h	5 GB
8	1	12	51	$1.6 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 9$ h	8 GB
9	1	12	52	$1.8 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 14$ d	12 GB
10	1	12	53	$2.0 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 21$ d	17 GB
11	1	12	54	$2.2 \cdot 10^9 \cdot T_{\mathsf{Mult}} = 30$ d	23 GB

And finally, a couple of estimates for a fixed $\beta = 0$:

n	β	ℓ_0	ℓ_1	FHE.Recrypt	$\mathsf{evk}^{\mathsf{hdec}}$
10	0	6	2	$0.25 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 0.3$ s	187 KB
50	0	8	2	$26 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 31$ m	123 MB
100	0	9	2	$235 \cdot 10^6 \cdot T_{\mathsf{Mult}} = 27$ h	3 GB

This is the only case when the bound $\ell_i > \log_2 \beta + 2$ from (6) is useful, applied to ℓ_1.

6 Implementation and Conclusions

The source code of a prototype implementation of the FHE scheme in C++ can be found here. The Python script used in Section 5.3 is also available here.

The prototype works correctly for all valid parameter settings that were tested, with an exception that was mentioned in 4.2. Specifically, for small values of n, there is a non-negligible chance of producing a ciphertext that looks like $(0, \ldots, 0, *)$, where $*$ is an arbitrary number. One example is $\mathbf{c} = (0, 8)$ for parameter settings $n = 2$, $\ell_0 = 4$, $\ell_1 = 2$ and $\beta = 0$. Then the homomorphic decryption algorithm fails because the sum

$$\sum_{i=1}^{n_0-1} \sum_{j=0}^{\log_2 q_0 - 1} (\mathbf{s}_0[i] \cdot 2^j) \cdot \mathbf{c}[i]^{(j)}$$

does not contain any ciphertexts, as $\mathbf{c}[i]^{(j)} = 0$ for all i and j. However, this also means that the ciphertext does not provide any privacy, as anyone can recover the message from $\mathbf{c}[n_0]$ and q_0. So we disregard this problem.

The following table shows some of the tested parameter settings, and the average time required to compute a single NAND gate followed by the recryption procedure:

n	β	ℓ_0	ℓ_1	Time
10	0	6	2	357 s
2	1	6	20	9 – 37 s (average 18 s) in 20 tests
4	1	6	20	520 – 1250 s (average 800 s) in 3 tests

Note that the test for $n = 10$ and $\beta = 0$ took roughly 1000 times longer than estimated in the previous section. One reason for that is an inefficient implementation; for example, we use a straightforward matrix multiplication algorithm. More generally, it shows that we can not assume that we are able to perform 10^9 operations per second.

All parameter settings for $\beta = \lceil\sqrt{n}\rceil$ from previous section require to use $\ell_1 > 32$, so it is no longer sufficient to use native 64-bit data types for multiplication. We perform multiplication by splitting one of the numbers in bits and then using the shift-and-add algorithm modulo q_1, which is similar to the square-and-multiply exponentiation algorithm.

Two of the tested parameter sets do not satisfy (7), so for we also indicate the number of tests that were run (all of them passed). The reason for trying such parameters is because any provably correct parameters for $\beta > 0$ are likely to be infeasible. I know that Michael's group managed to got some tests for $n = 4$, $\beta = 1$, $\ell_0 = 6$, and I picked $\ell_1 = 20$ at random. It would be valuable to find better ways to estimate the parameter settings that satisfy correctness. For example, we could allow the correctness of the scheme to fail with a small probability over the choice of random noise.

We performed a lot of tests for $n < 10$ and $\beta = 0$. We tested XOR, AND, NOT gates, as well as KSwitch and HDec procedures together and separately. However, we chose the $n = 10$, $\beta = 0$ case listed above as the most representative example.

It is interesting that the entire bootstrapping process is deterministic (apart from the generation of the evaluation keys). Is there a way to speed up the recryption procedure by precomputing anything?

Likewise, the evaluation key contains a lot of encryptions. It seems that we can not use the fact that most of them encrypt zeros, as we would likely be revealing some information about the bits of the encrypted secret key. However, is there a way to at least compress these encryptions and store only some "seeds"? Something similar was done for the keys of the integer FHE scheme.