Contents

1 Introduction **1**

2 The scheme **3**
2.1 Bootstrapping 6

3 Full version **7**
3.1 Regev's encoding scheme 7
3.2 Key-switching 9
3.3 Full scheme 10

4 Security **11**

5 Applications **12**

1 Introduction

References:

- The exposition is mostly based on Boaz Barak and Zvika Brakerski's blog posts `http://windowsontheory.org/2012/05/01/the-swiss-army-knife-of-cryptography/`, `http://windowsontheory.org/2012/05/02/building-the-swiss-army-knife/`. (See also references there.) See also Barak's lecture notes at `http://www.cs.princeton.edu/courses/archive/spring10/cos433/`.
- [Bra12], Brakerski's paper with the full scheme. (We follow this because it is the simplest.)
- [Reg09], basic encoding scheme that Brakerski's scheme is based on.
- [Gen09], original FHE scheme based on ideal lattices.
- [Reg09], [Pei09]: reduction from GapSVP to LWE to breaking the scheme. Regev depends on hardness for quantum algorithms for security; Peikert reduces it to classical.
- `https://en.wikipedia.org/wiki/Homomorphic_encryption`

Basic public-key encryption is simple: Using a system like RSA, Alice can choose a public key and secret key and give everyone the public key; now anyone can encrypt a message using the secret key and only Alice can read the messages.

However, after encrypting a message, there's *a priori* not anything they can do with the encrypted message (ciphertext). They can't perform *computation* with the ciphertext.

Consider the following situation, where Alice wants other people to be able to compute with encrypted data.

- Alice has a sensitive data (medical records, financial records, passwords, etc.) that she needs to do various computations on.
- Bob runs a cloud computing service.
- Alice does not have much computing power so she can't do the computations herself. She has to outsource the computation to Bob. She doesn't want Bob to be able to read her data, so she gives the encrypted data to Bob.
- Alice tells Bob the program she'd like to run on the data. But how can Bob carry out Alice's computation on the encrypted data?

Fully homomorphic encryption solves this problem. First we give the most basic version. Let $B = \{0, 1\}$.

Definition (Homomorphic encryption)**:** A **homomorphic encryption scheme** consists of the following.

1. A way to generate keys (a secret key and an evaluation key): a randomized algorithm
$$(sk, evk) \leftarrow KeyGen(1^n).$$
2. An *probabilistic* encryption algorithm $E_{sk} : B \to B^n$ using the secret key:
$$c \leftarrow E_{sk}(b).$$
3. A decryption algorithm $D_{sk} : B^n \to B$
$$b \leftarrow D_{sk}(c).$$
4. A way to compute on encrypted data without knowing the secret key. Since any computation can be expressed as a circuit of XOR and AND, which correspond to $+$ and $\times$ modulo 2, this means that there are two algorithms $+_{pk}$ and $\times_{pk}$ such that produce a result decoding to the right answer with probability $1 - \varepsilon$:
$$\begin{aligned} D_{sk}(+_{evk}(E_{sk}(m_1), E_{sk}(m_2))) &= m_1 + m_2 \\ D_{sk}(\times_{evk}(E_{sk}(m_1), E_{sk}(m_2))) &= m_1 m_2. \end{aligned}$$
[1]

Moreover, given $E_{sk}(b)$, it is difficult for a polynomial-time adversary to recover b.

[1]It's sufficient to have
$$\begin{aligned} +_{evk}(E_{sk}(m_1), E_{sk}(m_2)) &\approx_\varepsilon E_{sk}(m_1 + m_2) \\ \times_{evk}(E_{sk}(m_1), E_{sk}(m_2)) &\approx_\varepsilon E_{sk}(m_1 m_2). \end{aligned}$$
where $\approx_\varepsilon$ means that the distributions are ε apart in statistical distance. (Both the LHS and RHS are distributions, as E_{sk} is a randomized algorithm.)

Here's a more official version. The above version isn't public-key, as the encoding function depends on the secret key. We remedy this in the full definition.

Definition 1.1: A **homomorphic encryption scheme** consists of the following. (See §2.2 of [Bra12].)

1. Key generation (public key, evaluation key, secret key):
$$(pk, evk, sk) \leftarrow Keygen(1^n).$$
2. Encryption: A probabilistic algorithm that encrypts using the public key.
$$c \leftarrow E_{pk}(m).$$
3. Decryption: Decrypt the ciphertext using the secret key.
$$m \leftarrow D_{pk}(c).$$
4. Homomorphic evaluation: Given a function (represented as an arithmetic circuit mod 2) taking l arguments, evaluate it on encrypted input. Note that it suffices to define Eval for $+$ and $\times$, as every circuit is built out of these basic operations.
$$c_f \leftarrow Eval_{evk}(f, c_1, \ldots, c_l)$$

We say

- The scheme is $L(n)$**-homomorphic** if for every depth L arithmetic circuit, the problem of decoding a evaluated function incorrectly is negligible:
$$\mathbb{P}[D_{sk}(Eval_{evk}(f, E(m_1), \ldots, E(m_l))) \neq f(m_1, \ldots, m_l)] = o\left(\frac{1}{\text{poly}(n)}\right).$$
- The scheme is **compact** if the decryption circuit D_{sk} is independent of f.
- The scheme is **fully homomorphic** if it is compact and poly(n)-homomorphic (i.e., p-homomorphic for any polynomial p).
- The scheme is **leveled** if it requires input 1^L for key generation.

2 The scheme

We first give a simple and "morally correct" version of the scheme. It's more intuitive to think about the scheme as being done over the reals, although for technical reasons the rigorous version is done with integers modulo some q instead. We'll see that in Section 3.

Here's the basic idea. We try to build a symmetric-key system first. Consider the reals modulo 2; then $(-1, 1]$ is a complete set of residue classes. Equality below is modulo 2.

- Key generation: The secret key is a random $s \in B^n \backslash \{0^n\}$.
- Encryption: $E_s(m) \in (-1,1]^n$ is a random solution $c \in (-1,1]^n$ to
$$\langle s, c\rangle \approx_\varepsilon m$$
where
  - $\langle s, c\rangle = \sum s_i c_i$, and
  - $a \approx_\varepsilon b$ means the distance in $\mathbb{R}/2\mathbb{Z}$ is less than ε: $|a-b|_{\mathbb{R}/2\mathbb{Z}} < \varepsilon$.

  (Technically, because we have finite precision, we'd have to take a large q and choose the entries of s to be a random multiples of $\frac{1}{q}$. Let's ignore this.)
- Decryption: $D_s(c) = \langle s, c\rangle$ rounded to 0 or 1.

We can obtain a random solution by choosing all coordinates of c arbitrarily, except for one i where $s_i = 1$; choose $e \in (-\varepsilon, \varepsilon)$ uniformly, and then solve $\langle s, c\rangle = m + e$ for the remaining coordinate.

Why is this secure? Noise is essential here. If we had a lot of equations $\langle s, c_i\rangle = m_i$, then we could recover s by solving a system of linear equations. However, Gaussian elimination is quite sensitive to noise, so if only know $\langle s, c_i\rangle \approx_\varepsilon m_i$, this problem is conjecturally hard. More on this in Section **??**.

As long as we keep the error under $\frac{1}{2}$ we can decrypt successfully.

(Also note that we can encrypt non-boolean values as well, and decrypt them to within ε. This will be important.)

Problem 2.1:
1. How would we define $+_{evk}(c_1, c_2)$. (Do we need even need an evaluation key?) What is the error?
2. What about for multiplication?

Let $c_i = E_{sk}(m_i)$.

- Addition is easy. Let $c_+ = c_1 + c_2$. Then it's clear that
$$\langle c_+, s\rangle = \langle c_1, s\rangle + \langle c_2, s\rangle \approx_{2\varepsilon} m_1 + m_2.$$
The error is 2ε.
- Multiplication is trickier. We have $\langle s, c_1\rangle = m_1$, $\langle s, c_2\rangle = m_2$, so (ignoring the ε)
$$m_1 m_2 = \sum_{i,j} c_{1i} c_{2j} s_i s_j.$$
We want this to equal $\langle s, c\rangle$. A first instinct might be to set $c = \sum_j s_j c_{1j} c_{2j}$ but that's not allowed, because we can't use the secret key.

  We want
$$\langle s, c_\times\rangle = \sum_{i,j} c_{1i} c_{2j} s_i s_j$$

so try setting

$$c_\times = \sum_{i,j} c_{1i}c_{2j}x_{ij}$$

for some x_{ij}. We see that we should choose x_{ij} such that $\langle s, x_{ij}\rangle = s_is_j$. This suggests that we let the evaluation key be $evk = (x_{ij} = E_s(s_is_j))_{1\le i,j\le n}$, so $c_\times$ is computable from evk and c_1, c_2. (Note s_is_j is nonboolean, but the encryption still works.)

Problem 2.2: This doesn't work. What's the problem?

Firstly, is our scheme still secure? We have to make sure that giving away the encodings of s_i, s_j doesn't hurt (called "circular security"). This isn't a major problem as it seems quite plausible. There is a way to not rely on this assumption by "switching keys"—more on this later.

Secondly, we ignored the ε to our peril. We have the (sad) fact that if $k \equiv 0 \pmod 2$ and r is a real number, it may not be that $kr \equiv 0 \pmod 2$. (Multiplication in $\mathbb{R}/2\mathbb{Z}$ is not well-defined.) Let's work out the error. Denote the errors by $(n_1, n_2, n_{ij} \in \mathbb{Z},\ e_1, e_2, e_{ij} \in (-\varepsilon, \varepsilon))$

$$\begin{aligned}\langle s, x_{ij}\rangle &= s_is_j + 2n_{ij} + e_{ij}\\ \langle s, c_1\rangle &= m_1 + 2n_1 + e_1\\ \langle s, c_2\rangle &= m_2 + 2n_2 + e_2.\end{aligned}$$

Then

$$\begin{aligned}\langle s, c_\times\rangle &= \sum c_{1i}c_{2j}\langle s, x_{ij}\rangle\\ &= \sum c_{1i}c_{2j}(s_is_j + e_{ij} + 2n_{ij})\\ &= \langle s, c_1\rangle\langle s, c_2\rangle + \sum c_{1i}c_{2j}e_{ij} + \sum c_{1i}c_{2j}n2n_{ij}\\ &= (m_1 + 2n_1 + e_1)(m_2 + 2n_2 + e_2) + O(n^2\varepsilon) + \sum c_{1i}c_{2j}2n_{ij}\\ &= m_1m_2 + 2n_1m_2 + 2n_2m_1 + O(n\varepsilon + \varepsilon^2) + O(n^2\varepsilon) + \sum c_{1i}c_{2j}2n_{ij}.\end{aligned}$$

The red terms are a problem! The n's are integers, but the c_{bi}'s are not.

Problem 2.3: How can we fix this?

This wouldn't pose a problem if the c's were 0/1's. So let's break up c into bits using its binary expansion. Assuming the precision is 2^{-N} (with the roundoff error negligible), let c^{ijk} be integer vectors such that

$$c_{1i}c_{2j} = \sum_{k=0}^{N} 2^{-k}c^{ijk}.$$

(This is in $[0, 2)$ rather than $(-1, 1]$, but we can shift by a multiple of 2.) Then we can rewrite

$$\begin{aligned}\langle s, c_\times\rangle &= \sum_{i,j} c_{1i}c_{2j}s_is_j \\ &= \sum_{i,j,k} 2^{-k}c^{ijk}s_is_j \\ &= \sum_{i,j,k} c^{ijk}(2^{-k}s_is_j).\end{aligned}$$

We instead let $evk = (x_{ijk})$ where $\langle s, x_{ijk}\rangle \approx_\varepsilon 2^{-k}s_is_j$, and let

$$c_\times = \sum_{i,j,k} c^{ijk}x_{ijk}.$$

Then defining n_{ijk}, e_{ijk} as the errors,

$$\begin{aligned}\langle s, c_\times\rangle &= \sum c^{ijk}\langle s, x_{ijk}\rangle \\ &= \sum c^{ijk}(2^{-k}s_is_j + e_{ijk} + 2n_{ijk}) \\ &\equiv \sum c_{1i}c_{2j}s_is_j + O(n^3\varepsilon) \pmod 2.\end{aligned}$$

as now $c^{ijk} \in \mathbb{Z}$ and hence $c^{ijk}2n_{ijk} \equiv 0 \pmod 2$. Thus we have defined $c_\times$ with error $O(n^3\varepsilon)$. (It can be reduced to $O(n\varepsilon)$, but we won't do that here.)

Note that L layers of circuits will make the error $O(n^{3L}\varepsilon)$:

$$\langle s, Eval_{evk}(f, E_s(m_1), \ldots)\rangle \approx_{\varepsilon n^{3L}} f(m_1, \ldots).$$

This is bad because for a polynomial number of layers, it would force us to set the inital error too small, which makes the scheme not secure. Later we'll see how to reduce the error by bootstrapping.

Problem 2.4: We gave a symmetric-key protocol (use the same key to encode and decode). How can we make this a public-key protocol?

Here's a generic way to turn a symmetric-key FHE scheme into a public-key FHE scheme.

If the symmetric-key FHE scheme comes with a re-randomization algorithm—an algorithm using only pk that takes a ciphertext encoding b and returns a ciphertext with the same distribution as $E_s(b)$ (over its randomness), then we can give $E_s(0), E_s(1)$ as the public key, and the public encoding algorithm would be to take one of these and re-randomize it. We don't have this feature in our scheme. The full version will involve some randomness that allows re-randomization.[2]

2.1 Bootstrapping

We show that as long as we can do homomorphic encryption with noise level $\varepsilon = n^{-O(\lg n)}$, and decryption can be done using a $\lg n$-depth circuit, we can do fully homomorphic encryption.

[2]Actual construction omitted because I'm not quite sure how it works...

The obstacle is that the error rate could increase rapidly (exponentially in the above scheme). Bootstrapping is a way to "reset" the error back to a small quantity. We can use it after every fixed number of layers as needed.

We show how to bootstrap for our scheme. The idea is to homomorphically evaluate the *decryption circuit itself.* Alice needs to provide the encryption of her own secret key $E_{pk}(s)$ (as part of evk). Let $D_s(c) =: f_c(s)$ be the decryption function. (D_s is secret, but f_c is public.) Note f_c is computable with a $O(\lg n)$-depth circuit (as it basically just involves adding some subset of n numbers). Given

$$\langle s, c\rangle \approx_{\frac{1}{2}} m,$$

because f_c is depth $\lg n$, we have

$$\langle s, Eval_{evk}(f_c, E_s(s))\rangle \approx_{\varepsilon n^{O(\lg n)}} f_c(s) = D_s(c) = m;$$

we've reduced the error back down to $\varepsilon n^{O(\lg n)}$. We just need the scheme to be secure at noise $\boxed{\varepsilon = n^{-O(\lg n)}}$.

Note that this error is independent of the original error ($\frac{1}{2}$) because we are not treating c as an encrypted message to be operated on homomorphically—we are treating it as an input to an algorithm (f) that is always correct (indeed, we curried c into f above).

3 Full version

3.1 Regev's encoding scheme

We outline the scheme as presented in Brakerski. We note several changes from the nonrigorous version that we presented using reals.

1. Rather than working with real numbers in $(-1, 1]$, we work with integers in $(-\frac{q}{2}, \frac{q}{2}]$. 0 and 1 are encoded with $0, \frac{q}{2}$ (assume for simplicity $2 \mid q$).

2. We split up the evaluation of multiplication into 2 phases. Recall that multiplication seemed difficult because $m_1m_2 \approx \sum_{i,j} c_{1i}c_{2j}s_is_j$ is not in the right form. However, it is in the right form if we *switch keys* to the key $s \otimes s$. (We'll actually further decompose s into bits before tensoring.)

   The first part is multiply $c \otimes c = (c_{1i}c_{2j})$ but to give a result encoded with $s \otimes s$. (Actually, we'll encode with $\mathsf{BitDecomp}(s) \otimes \mathsf{BitDecomp}(s)$.)

   Note that what we did for multiplication is actual very general. We essentially gave a way to switch from the key $s \otimes s = (s_is_j)$ to the key s.

   The second part is this key-switch protocol.

3. If we switch back to s, then we have to ask the question: if we encode information about s using s, is our protocol still secure? In our original protocol we were sending bits $E_s(2^{-k}s_is_j)$.

This assumption of "circular security" is reasonable, but it's not a standard assumption, and we would much rather rely purely on standard assumptions. The key-switch protocol allows us to do away with this assumption by simply switching to a completely new key for each level of the circuit. We'd have to know the number of levels first, though.

4. Before, c is such that $\langle s, c\rangle = m$: there is just 1 equation. However, this doesn't allow a person without the sk to randomly generate a random encoding of 0 or 1 without us giving away the sk (because we'd give away the kernel, and hence the vector s).

   The solution is to choose c that approximately satisfies a *system* of equations $\langle s_i, c\rangle \approx_\varepsilon m_i + \varepsilon$. This allows us to hide the sk.

The following basic (nonhomomorphic) encoding scheme was first proposed by Regev, and then turned into a fully homomorphic scheme by Brakerski. Let $\mathbb{Z}_q := (-\frac{q}{2}, \frac{q}{2}]$, and $[n]_q$ denote the element in $\mathbb{Z}_q$ that is $\equiv n \pmod q$.

1. Key generation: Given the parameter n, generate $s \sim \mathbb{Z}_q^n$. Let $N = (n+1)(\lg q + O(1))$.

   Now generate random vectors $a_i \in \mathbb{Z}_q^n, 1 \le i \le N$ and errors $e_i \sim \chi$ and output $pk = (a_i, \langle a_i, s\rangle + e_i)_{i=1}^n$ as the public key.

   Equivalently, generate $A \in \mathbb{Z}_q^{N\times n}$ and $e \sim \chi^N$.

   $$P = \begin{pmatrix} s^T A^T + e^T \\ -A^T \end{pmatrix}.$$

   (Motivation: The idea is that we want to give the public a randomized algorithm for encoding a message. A first attempt is to give P such that for a random $r \in \{0,1\}^n$, $\langle Pr, s\rangle \approx_\varepsilon 0$. But this gives away the kernel of $\langle \bullet, s\rangle$ and hence x. The hardness of learning with noise suggests that we can hide s by adding noise.

   First, an easier way to do the encoding is to make the dot product by $\begin{pmatrix} 1 \\ s \end{pmatrix}$ rather than s, so that $\mathbf{m} = \begin{pmatrix} m\frac{q}{2} \\ 0 \\ \vdots \end{pmatrix}$ decodes to m. We want the encoding to then be $m + Pr$.

   Without noise, P would be $\begin{pmatrix} s^T A^T \\ -A^T \end{pmatrix}$ so $P\begin{pmatrix} 1 \\ s \end{pmatrix} = 0$. But as we said, $s^T A^T$ and A^T gives away s. Now we add noise, so it's the P given above.)

2. Encryption: Sample $r \in \{0,1\}^N$ and output

   $$c := [Pr + \left\lfloor \frac{q}{2} \right\rfloor \begin{pmatrix} m \\ 0 \\ \vdots \end{pmatrix}]_q \in \mathbb{Z}_q^{n+1}.$$

3. Decryption: Dot product with $\begin{pmatrix}1\\s\end{pmatrix}$ and round to 0 or $\frac{q}{2}$ modulo q.

$$m := \left[2\frac{[\left\langle c, \begin{pmatrix}1\\s\end{pmatrix}\right\rangle]_q}{q} \right]_2 .$$

Lemma 3.1 (Encryption noise)**:** We have

$$\left\langle c, \begin{pmatrix}1\\s\end{pmatrix}\right\rangle = \frac{q}{2}m + e \pmod q$$

for some $|e| \le NB$. Decryption succeeds whenever the noise $|e| < \frac{q}{4}$.

Proof.

$$\left\langle c, \begin{pmatrix}1\\s\end{pmatrix}\right\rangle = \left\langle \begin{pmatrix}s^TA^T + e^T\\ -A^T\end{pmatrix} r + \frac{q}{2}\begin{pmatrix}m\\0\\ \vdots\end{pmatrix}, \begin{pmatrix}1\\s\end{pmatrix}\right\rangle$$
$$= \cancel{s^TA^Tr} + e^Tr + \cancel{\langle -A^Tr, s\rangle} + \frac{q}{2}m$$

□

Theorem 3.2: If $\mathsf{DLWE}_{n,q,\chi}$ (see next section) is hard (for polynomial time), then for any $m \in \{0,1\}$, the distribution $(P, c = E_{pk}(m))$ is indistiguishable (in polynomial time) from uniform over $\mathbb{Z}_q^{N\times(n+1)} \times \mathbb{Z}_q^{n+1}$.

The proof is standard: given a distinguisher for distinguishing the encoded messages, one constructs a distinguisher for DLWE. See §5 of Regev [Reg09].

3.2 Key-switching

Next we give the key-switching algorithm. We really already gave it but not in its full generality. Our "source" key s corresponds to (s_is_j) before, and our "target" key t corresponds to s before. Recall what we did was that we encoded 2^ks_i and the binary digits of c.

For $x \in \mathbb{Z}^n$, write $x = \sum_{k=0}^{\lceil \lg q\rceil -1} x_k 2^k$ and define

$$BitDecomp_q(x) = (x_0, \ldots, x_{\lceil \lg q\rceil -1}) \in \{0,1\}^{n\lceil \ln q\rceil}$$
$$PowersOfTwo_q(x) = [(x, 2x, \ldots, 2^{\lceil \lg q\rceil -1}x)]_q \in \mathbb{Z}_q^{n\lceil \ln q\rceil}.$$

A first attempt at key-switching is just encode the message again with the new key: the new ciphertext is (for a random $A_{s\to t}$)

$$c_t = \begin{pmatrix}t^TA_{s\to t} + e^T + s\\ -A_{s\to t}^T\end{pmatrix} c_s.$$

Again the problem is the error: The error is $e^T c_s$ which could be large because e having small entries does not mean $e^T c_s$ is small when c_s could be anything mod q. Breaking c_s up into bits solves this problem.

1. $SwitchKeyGen_{q,\chi}(s,t)$: Let $\widehat{n_s} := n_s \lceil \lg q \rceil$. Choose $A_{s,t} \in \mathbb{Z}_q^{\widehat{n_s} \times n_t}, e \sim \chi^{\widehat{n_s}}$ randomly. Output
$$P_{s\to t} = \begin{pmatrix} t^T A_{s\to t} + e_{s\to t} + PowersOfTwo_q(s) \\ -A_{s\to t} \end{pmatrix} \in \mathbb{Z}_q^{\widehat{n_s} \times (n_t+1)}.$$
2. $SwitchKey_q(P_{s\to t}, c_s)$: Output
$$c_t := [P_{s\to t} BitDecomp_q(c_s)]_q.$$

Security follows in the same way as before. The additional error is
$$\langle BitDecomp_q(c_s), e_{s\to t}\rangle \le n_s \lceil \lg q \rceil B.$$

3.3 Full scheme

As we mentioned, to get a fully homomorphic scheme under standard assumptions, we generate as many keys as there are levels; each time we add or multiply we automatically switch to the tensored key (we don't have to do this for addition, but we want to be consistent); then we use key-switching to get to the next key.

We'll just give the functions here. We have all the ingredients for the proof, but we'll refer the reader to §4 of [Bra12]. Let "Regev" refer to the scheme in Section 3.1.

- KeyGen$(1^L, 1^n)$. Let
$$\begin{aligned} s_0, \ldots, s_L &\sim Regev.SecretKeyGen(1^n) \\ P_0 &\sim Regev.PublicKeygen(s_0) \\ \widetilde{s}_{i-1} &:= BitDecomp((1, s_{i-1}))^{\otimes 2} \in \{0,1\}^{((n+1)^2 \lceil \lg q \rceil)^2}, \\ P_{i-1\to i} &\sim SwitchKeyGen(\widetilde{s_{i-1}}, s_i) \\ (sk, pk, evk) &= (s_L, P_0, (P_{i-1\to i})_{i\in[L]}). \end{aligned}$$

- $E_{pk}(m) = Regev.E_{pk}(m)$.

- To give evaluation, it suffices to give addition and multiplication at level i. (For addition, we tensor it up for consistency with multiplication.)
$$\begin{aligned} \widetilde{c}_+ &= PowersOfTwo(c_1 + c_2) \otimes PowersOfTwo((1, 0, \ldots, 0)) \\ c_+ &= SwitchKey(P_{i-1\to i}, \widetilde{c_+}) \\ \widetilde{c}_\times &= \left\lfloor \frac{2}{q} PowersOfTwo(c_1) \otimes PowersOfTwo(c_2) \right\rceil \\ c_\times &= SwitchKey(P_{i-1\to i}, \widetilde{c_\times}) \end{aligned}$$

- Decryption $D_s = \mathit{Regev}.D_s(c)$.

A hybrid argument shows that security follows from hardness of $\mathrm{DLWE}_{n,q,\chi}$. (Considering replacing the outputs of the levels one by one, starting from the last level. Each level has the encoding of the previous level's key; if you can't distinguish it from uniform you can't find the previous level's key...)

Theorem 3.3: The scheme above with $n, q, |\chi| \le B, L$ is L-homomorphic when

$$\frac{q}{B} \ge O(n(\lg q))^{L+O(1)}.$$

Proof. Calculation shows that the starting encoding has error $O(n^2(\lg q)^3 B)$, and each next layer increases error by a factor of $O(n \lg q)$. □

Now we apply bootstrapping. As an inner product modulo q, the decoding function has circuit depth $O(\lg n + \lg\lg q)$. Thus homomorphically evaluating it gives error $B(n \lg q)^{O(\lg n + \lg\lg q)}$. We can recover when error is $\frac{q}{B} > \frac{1}{4}$; therefore, hardness of $\mathrm{DLWE}_{n,q,\chi}$ for some value $\frac{q}{B} \ge (n \lg q)^{O(\lg n + \lg\lg q)}$ gives secure leveled FHE.

By taking $q = \widetilde{O}(2^{\frac{n}{2}})$ and using facts from the next section, we can reduce from GapSVP.

Theorem 3.4: If $\mathsf{GapSVP}_{n^{O(\lg n)}}$ is hard, then this scheme is a leveled fully homomorphic encryption scheme.

If the scheme is "weak circular secure" (basically it's safe to give out encodings of the key itself), then we get non-leveled homomorphism (because we don't have to)

4 Security

The security of this scheme can be traced back to the security of the shortest vector problem, which is conjecturally hard.

$$\text{(Breaking the FHE scheme)} \le \text{(Learning with errors)} \le \text{(Shortest vector problem)}$$
$$\text{(FHE secure)} \Leftarrow \text{(Hardness of learning with errors)} \Leftarrow \text{(Hardness of shortest vector problem)}.$$

Definition 4.1: The **decisional learning with errors** problem $\mathsf{DLWE}_{n,m,q,\chi}$ with parameters $(n, m, q(n), \chi(n))$ is to distinguish (with $\Omega\left(\frac{1}{\mathrm{poly}(n)}\right)$ advantage) between m samples chosen according to the following two distributions.

1. m samples chosen according to the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$
2. m samples chosen according to the distribution $A_{s,\chi}$, defined by choosing a random $a \sim \mathbb{Z}_q^n$, a noise $e \sim \chi$, and outputting

$$(a, [\langle a, s\rangle + e]_q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q.$$

(When m is omitted, we mean $m = \mathrm{poly}(n)$.)

Definition 4.2 (2.5 in [Pei09])**:** For a lattice Λ, let $\lambda_1(\Lambda)$ denote the shortest nonzero vector.

The gap-shortest vector problem GapSVP_γ takes input (β, d) where β is the basis of a n-dimensional lattice $\Lambda = \mathcal{L}(\beta)$ and $d > 0$ and outputs

- YES if $\lambda_1(\Lambda) \le d$,
- NO if $\lambda_1(\Lambda) > \gamma(n)d$.

Learning with errors can be reduced to GapSVD. GapSVD is thought to be hard, even for quantum algorithms. As a result it forms the basis for many lattice cryptographic protocols. This is to be contrasted with more common protocols depending on factoring and hardness of the discrete log problem, which are known to be breakable by quantum algorithms.

Theorem 4.3 (DLWE$\le$GapSVP, Cor. 2.1 in [Bra12])**:** Let $q = 2^r$ (for simplicity). There is a distribution χ such that if $\mathsf{DLWE}_{n,q,\chi}$ can be solved in polynomial time, then

1. there is a efficient quantum (BQP) algorithm that solves $\mathsf{GapSVP}_{\widetilde{O}(nq/B)}$ in n dimensions.
2. if $q \ge \widetilde{O}(2^{\frac{n}{2}})$ there is also an efficient classical algorithm:

$$\mathsf{DLWE}_{n,q,\chi} \le \mathsf{GapSVP}_{\widetilde{O}(nq/B)}.$$

Polynomial-time algorithms for GapSVP_γ are known only for $\gamma = 2^{n\frac{\ln\ln n}{\ln n}}$, so $q = 2^{n/2}$ is a "safe" setting of parameters.

5 Applications

In cryptography, the usefulness of a concept lies not just in its inherent use but also in the *other protocols* it facilitates. Fully homomorphic encryption is a useful cryptographic "primitive" because other quite standard cyptographic protocols can be built on top of it.

We give two.

Problem 5.1: How can we use FHE to construct

1. zero-knowledge proofs?
2. two party secure computation?

Hint: you'll need a bit commitment scheme.

Definition 5.2 (Zero-knowledge proof)**:** Let $f : B^n \to B$ be a polynomial-time computable function. A zero-knowledge proof for f is a protocol where a Prover can convince a Verifier that she knows x such that $f(x) = 1$ while revealing no knowledge about x.

Formally, there is a probabilistic interactive protocol with completeness $\ge \frac{2}{3}$, soundness $\le \frac{1}{3}$, and *perfect zero knowledge* for every strategy of the Verifier, there is a probabilistic polynomial time algorithm the Verifier can run that gives the same distribution on the output as the interaction. I.e., the verifier can simulate the algorithm.

Definition 5.3 (k-party secure computation)**:** Let f be a function taking k inputs. A k-party secure evaluation protocol for $F : (B^n)^k \to (B^n)^k$ is a protocol such that the following holds. For every S and every cheating strategy for parties in S, there is a set of inputs $\{x_i\}_{i\in S}$ such that

1. Correctness: if the parties in $\overline{S}$ follow the protocol, then the ith output is $F(x_1, \ldots, x_k)_i$ or $\perp$.
2. Simulation: there is a simulator that takes as input $\{F(x_1, \ldots, x_k)_i\}_{i\in S}$, and outputs an interaction that is computationally indistinguishable from the point of view of all parties in S from the actual interaction. In other words, players don't learn anything about each other's inputs.

For example, two millionaires want to compute $f(x, y) = (x > y)$—which one of them has more money—without revealing their wealth.

ZKP: Here's a first attempt.

1. Alice takes the message x, encrypts it $x' = E_{pk}(x)$, and sends the result to Bob.
2. Bob computes $Eval_{evk}(f, x) = b'$, and sends the result to Alice.
3. Alice decrypts $D_s(b') = b$, and sends b to Bob.
4. Bob checks that $b = 1$.

Problem 5.4: What are two problems with this protocol? Fix them.
Does the FHE scheme need to be public-key?

1. How does Bob know Alice decrypted the result correctly, and isn't sending Alice back 1 all the time?

   Cf. How can Alice prove to color-blind Bob that his otherwise indistinguishable pairs of socks are 2 different colors? Bob randomly takes 2 socks from the same pair or different pairs and asks whether they are the same color.

   Bob sends Alice $E_{pk}(0)$ with $\frac{1}{2}$ probability and $Eval_{evk}(f, x')$ with $\frac{1}{2}$ probability. We want Alice *not to be able to distinguish* between the two when $f(x) = 0$. Bob rejects if he sent $E_{pk}(0)$ but Alice sent 1, or if Alice sends 0.

   Thus we need $Eval_{evk}(f, x')$ to be a randomized algorithm whose output is indistinguishable from $E_{pk}(f(x'))$. This can be accomplished by composing $Eval_{evk}(f, x')$ with a "rerandomization" algorithm.

   If Alice (Prover) is dishonest and sends Bob 1 with probability p (she can't tell what Bob did), then Bob rejects with probability $\frac{1}{2}$. If Alice is honest, she will always pass the test. This is a ZKP protocol with perfect completeness and $\frac{1}{2}$ soundness, which can be made arbitrarily small by repetition.

2. How does Alice know Bob carried out the computation correctly? (This is important because if Alice is reusing her secret key, we don't Bob to trick Alice into decoding other valuable information!)

   We use a commitment scheme.

   Definition 5.5: A **commitment scheme** is a protocol where Alice chooses a bit x, sends Bob b, and later reveals (x', y'), such that

   (a) (concealing) it is hard for Bob to recover x from b.

   (b) (binding) if $x \neq x'$, then Bob rejects, i.e., it is hard for Alice to find $(x' \neq x, y')$ that makes Bob accept.

   Here "hard" can be meant computationally, or perfectly (with infinite computational resources).

   For example, a injective one-way function f can be used to make a commitment scheme by having Alice pick random x and send $(f(x), b \oplus h(x))$ where h is a hard-core predicate. (Any OWF can be made to have a hard-core predicate.)

   Rather than give the result right away, Alice commits to an answer. Then Bob reveals all the randomness he used so that Alice can check his result. If it agrees, then Alice sends Bob the actual value.

2-party secure computation: Alice has x, Bob has y, and they want to compute $f(x, y)$. We'll just show how to achieve **honest but curious** security: when players don't deviate from the protocol they don't learn about each others inputs (other than what the output tells them.)[3] Here's the first attempt.

1. Alice generates a key (sk, pk, evk) and shares pk, evk with Bob. She sends $x' = E_{pk}(x)$ to Bob.

2. Bob encodes $y' = E_{pk}(y)$, computes $r' = Eval_{evk}(f, x', y')$, and sends r' to Alice.

3. Alice decodes r' to r and announces the result.

Problem 5.6: What are the problems and how do you fix them?

1. How does Bob know that Alice sent the right answer?

   Solution: Alice sends Bob a zero-knowledge proof that she knows s such that x (which she can commit to beforehand) encodes to x' and r encodes to r'.

[3]The Goldreich-Micali-Wigderson theorem ("compiler") can turn any multi-party computation protocol that is honest-but-curious secure into a secure protocol, provided there is a 1-way function. Also it is easy to go from 2 parties to k parties by splitting Alice in half—see Barak's notes.

2. How does Alice know Bob computed the function correctly?

   Solution: Bob sends Alice a zero-knowledge proof that he knows y such that $r' = Eval_{evk}(f, x', y')$ (he can commit to y beforehand).

Note that to compute a randomized function together, each party can generate randomness, and then XOR them together. Secure multi-party computation gives the ability to construct a virtual party "out of thin air" and do things like coin tossing, authenticated encryption (relay a message from one party to another), electronic voting and auctions, distributed signing and encryption...

References

[Bra12] Zvika Brakerski. "Fully homomorphic encryption without modulus switching from classical GapSVP". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7417 LNCS (2012), pp. 868–886. ISSN: 03029743. DOI: `10.1007/978-3-642-32009-5\_50`.

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing STOC 09* 19.September (2009), p. 169. ISSN: 07378017. DOI: `10.1145/1536414.1536440`. URL: `http://portal.acm.org/citation.cfm?doid=1536414.1536440`.

[Pei09] Chris Peikert. "Public-key cryptosystems from the worst-case shortest vector problem (extended abstract)". In: *STOC 2009: Proceedings of the 41st annual ACM symposium on Theory of computing* (2009), pp. 333–342. ISSN: 07378017. DOI: `10.1145/1536414.1536461`. URL: `http://portal.acm.org/citation.cfm?doid=1536414.1536461$%5Cbackslash$nhttp://dx.doi.org/10.1145/1536414.1536461`.

[Reg09] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *Journal of the ACM* 56.6 (2009), pp. 1–40. ISSN: 00045411. DOI: `10.1145/1568318.1568324`.