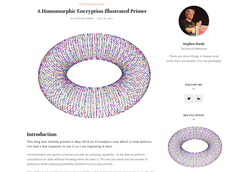
A Homomorphic Encryption Illustrated Primer

BY STEPHEN HANDY - JULY 06, 2021



Introduction

This blog was initially posted in May 2018 on a Facebook.com site which is now defunct. I've updated it and moved it to this site in July 2021 for posterity.

Homomorphic encryption (HE) is a powerful tool for computing quantity. In its ideal implementation, calculations on data without knowing what the data is. The fact you work out the answer to questions while keeping potentially sensitive secrets secure is pretty cool.

One of the most promising approaches (and the subject of some recent standardization efforts) is based on the ring-learning-with-errors problem (RLWE). This approach is fairly complicated to attempt and this is what will be discussed in depth here. There are a number of implementations you can use to try things out:

Library	URL	License	Language
SEAL	https://github.com/microsoft/SEAL	MS Research Limited Use License	C++
NFLlib	https://github.com/quarkslab/NFLlib	GPLv3	C++
FV-NFLlib	https://github.com/CryptoLab/FV-NFLlib	GPLv3	C++
Lattigo	https://github.com/ldsec/lattigo/v2	MIT	Golang C++
PALISADE	https://gitlab.si.pala.palisade.org/PALISADE/homomorphic-encryption-library	Liberal	C++
SEAL	https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/	Microsoft Research Limited Use License	C#
HELib	https://github.com/shai-bi/HElib	Apache 2.0	C++
JUPC	https://gitlab.dfki.de/mpc-framework/jupc/jupc-hom-enc	LGPLv3	Java

These encryption schemes appear complicated and white-hysterical, but hopefully this post will give you a clear idea of how and why they work.

The overall structure of the blog post will be:

- a little bit of a mathematical introduction
- how to encrypt and decrypt with
- homomorphic addition and multiplication

This is quite a lot of material so we jump to the fun bits.

Some introductory maths

These homomorphic encryption schemes are based on a hard computation problem known as the ring-learning-with-errors problem (RLWE). Before we can talk about RLWE, we need to define what is these schemes is represented by polynomials both when it is encrypted (ciphertext) and when it is unencrypted (the plaintext).

These are almost the normal polynomials that everyone studies at school. Things like:

$$x^2 + 2x + 1$$

There are some differences, the first being that the coefficients (and all other numbers) are in the cyclotomic relation to some other whole number f (that is, $\bmod f$). Usually $f = 2k$ then this is like a 2k-hour clock, where adding 2 hours is 2. In this, all the coefficients of the polynomial are created like this:

The second, and trickier, thing that is different about these polynomials is that this idea of using remainders applies not just to the coefficients of the polynomials, but also to the polynomials themselves.

We define a special polynomial called the polynomial modulus, and only consider the remainder of polynomial when they have been divided by the polynomial modulus. The specific form of this polynomial modulus is $x^n + 1$ where n is a power of 2. For example, if we have the polynomial modulus $x^4 + 1$, x to the polynomial x^4 is -1.

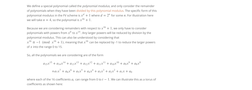
Because we are considering remainders with respect to $x^n + 1$, we only have to consider polynomials of degree $n - 1$ or less. This keeps polynomials with increasing degrees from growing in size. For example, if we are in the ring with polynomial modulus $x^4 + 1$, the polynomial $x^3 + 1$ is fine, but $x^4 + 1$ is not. Instead, x^4 can be replaced by -1 to reduce the larger powers of x into the range $n - 1$.

So, all the polynomials we are considering are of the form:

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + a_{n-3}x^{n-3} + \ldots + a_2x^2 + a_1x^1 + a_0x^0$$

$$= a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + a_{n-3}x^{n-3} + \ldots + a_2x^2 + a_1x^1 + a_0 \quad \text{as } x^n = -1$$

where each of the n coefficients a_i can range from 0 to $f - 1$. We can illustrate this as a bunch of coefficients as shown here:



As illustrated above, multiplication of some $2x^3 \bmod x^4 + 1$ takes this term (represented by the red dot above) and shifts it forward a place around the circle, and then multiplies by 2 to get the Cyan coloring $(2x^4 \rightarrow 2x^1)$ that consider coefficients $\bmod 5$ (since $f = 5$) rather than $\bmod 2$.

Polynomials add this form have a very rich structure and many nice properties. They are a subset of the ring-learning-with-errors problem, which is considered computationally hard, and, for example, lattice based ring-based cryptosystems are a number of cryptosystems and symbols.

Encryption using polynomial rings

Now that we have introduced some properties of the polynomial rings that are used in the homomorphic encryption, it is time to see how these polynomials are used to encrypt and decrypt data. In brief, show how the private and public keys are generated, and then how they are used for encryption and decryption.

The private and public keys

Encryption takes a plaintext and converts it to a ciphertext using a public key which is derived from a secret key. Decryption takes a ciphertext and converts it back into a plaintext using the matrix pair of keys derived from the private key.

More specifically, the plaintext is a polynomial from the ring with polynomial modulus $x^n + 1$, f is set to $f = 2$ above, and coefficient modulus t. The encryption ciphertext is a polynomial which is represented by two polynomials from the ring with the same polynomial modulus, but a coefficient modulus q which is typically much larger than t.

For example, the polynomial modulus may be $x^{4096} + 1$, meaning the polynomials in both the plaintext and ciphertext have 4096 coefficients. The plaintext coefficient modulus t and ciphertext polynomial ring coefficient modulus q are typically chosen to be powers of two, for example $t = 2^8$ and $q = 2^{128}$. This means the plaintext polynomials have coefficients in the range 0 to $t - 1$ and the ciphertext polynomials have coefficients in the range 0 to $q - 1$.

The public key is then defined as the pair of polynomials, $\mathbf{pk} = (p_0, p_1)$, where the coefficients of the polynomials p_0 and p_1 are randomly selected from the ciphertext coefficients modulus q.

For the example given above, the first polynomial of the public key is constructed as:

$$p_0 = -(a_{n-1}x^{n-1}(b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \ldots b_1x^1 + e_0)$$

$$\bmod x^n + 1 \bmod q \quad \text{where } b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \ldots b_1x^1 + e_0 \in \{0 \ldots 2\}$$

The first multiplication uses the polynomial a which has coefficients from the entire range of 0 to $q - 1$ and the secret key polynomial b which has coefficients of 0, 1 or -1. The result is a random sum of powers of polynomial multiplication with respect to the polynomial modulus, $\bmod q$ and an addition of the coefficients of e, and the peak error term is further reduced by multiplying by -1 and taking the remainder $\bmod q$ to ensure it is in the range.

Revealing the coefficients of both the public key polynomials leaks information about the secret key b (or $-b$ or $1 - b$). This indicates the inclusion of the error term in the scheme. If e is 0, b is easily found by solving the equation. The error term e is small, so b is still very close to the solution which was current at the time of writing.

For the prime n second key, which we call a, we generate a random polynomial with coefficients of either -1, 0 or 1. For example:

$$a = x^{n-1} + x^{n-2} + x^{n-3} + \ldots x^2 + x^1 + x^0$$

We define a second public key using coefficients extracted from the same ciphertext modulus with coefficients modulus q, which we call p_2:

$$p_2 = a(x^{n-1}(b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \ldots b_1x^1 + e_0)$$

$$\bmod x^n + 1 \bmod q \quad \text{where } b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \ldots b_1x^1 + e_0 \in \{0 \ldots 2\}$$

The public key is then defined as the pair of polynomials, $\mathbf{pk} = (p_0 \text{ or } p_1, p_2)$, where the coefficients of the polynomials p_0 and p_1 are randomly selected from the ciphertext coefficients modulus q.

Encryption

The encryption process looks a little like the public key generation process.

Encryption takes a plaintext, which is a polynomial with coefficients modulus t, and converts it to a pair of polynomials with coefficients modulus q. As an illustration, we will encrypt a very simple polynomial message: $m = 5 + 6x^2 + 3x^3$ - with only two non-zero coefficients.

Encryption requires three more small polynomials. Two more polynomials drawn from the same set of discrete Gaussian distributions used to the error polynomial in the public key, and another polynomial set with only 0, 1 and -1 as its coefficients. The first of these is called u:

$$u = u_{n-1}x^{n-1} + u_{n-2}x^{n-2} + u_{n-3}x^{n-3} + \ldots u_1x^1 + u_0x^0$$

$$\bmod x^n + 1 \bmod q \quad \text{where } u_{n-1}x^{n-1} + u_{n-2}x^{n-2} + u_{n-3}x^{n-3} + \ldots u_1x^1 + u_0 \in \{0 \ldots 2\}$$

The second and third are called e_1 and e_2:

$$e_1 = e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + e_{n-3}x^{n-3} + \ldots e_1x^1 + e_0x^0$$

$$\bmod x^n + 1 \bmod q \quad \text{where } e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + e_{n-3}x^{n-3} + \ldots e_1x^1 + e_0 \in \{0 \ldots 2\}$$

and

$$e_2 = e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + e_{n-3}x^{n-3} + \ldots e_1x^1 + e_0x^0$$

These polynomials are used in the encryption process, then discarded.

The ciphertext is represented by two polynomials calculated as:

$\mathbf{ct} = (\mathbf{p_0}, \mathbf{p_1})$, $c_0 = p_0u + e_1 + \Delta m$ and $c_1 = p_1u + e_2$. Note where the message appears and how it appears. The value Δ is the ciphertext modulus q divided by the plaintext modulus t, and Δ is rounded down to 1.5 in our example - creating both cases from the range of $\bmod q$. That is the only change to the message. The ciphertext polynomials are the public key polynomials multiplied by the random term which has values from the range of $\bmod q$ and a small error polynomial multiplied by the random value. The relative nature of u changes the scale and u each encryption, thereby ensuring that the same plain text produces very different ciphertexts each time it is encrypted.

Homomorphic addition and multiplication work because the message is present in the ciphertext only with a scaling. The other terms are there to hide the message, and are scaled by the scaling so in doing so in a way that can be removed only if you know the secret key.

Evaluating the first element of the ciphertext explicitly using the polynomials above gives:

$$c_0 = p_0u + \Delta m + e_1 = (a(x^{n-1}(b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \ldots b_1x^1 + e_0)$$

$$\bmod x^n + 1 \bmod q \quad \text{where } b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \ldots b_1x^1 + e_0 \in \{0 \ldots 2\}$$

Expanding the public key in the first term of the ciphertext, we see that $\mathbf{p_0} = (-a(x^{n-1}(b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \ldots b_1x^1 + e_0)$ and $\mathbf{p_1} = a$. In this expression, the first two terms are "small", being the product of the random polynomial a and the small secret key polynomial b. The final two terms effectively hiding the secret term which is the message.

The second element of the ciphertext is calculated like this:

$$c_1 = p_1u + e_2 = (a(x^{n-1} + x^{n-2} + x^{n-3} + \ldots x^2 + x^1 + x^0)$$

$$\bmod x^n + 1 \bmod q \quad \text{where } a = x^{n-1} + x^{n-2} + x^{n-3} + \ldots x^2 + x^1 + x^0$$

Rounding these coefficients recovers our message:

$$m = \lfloor c_0 / \Delta \rceil$$

We get our message back by rounding the approximate coefficients after scaling to their nearest integer counterparts:

Green - noisy transformation , Pink - rounded result

Putting it in angstroms: we decrypt a ciphertext by calculating:

$$m = \lfloor \frac{t}{q}(c_0 + c_1 s) \rceil \bmod t$$

where $\lfloor \rceil$ denotes rounding to the nearest integer.

If the coefficients are too noisy, then they will end up closer to a different integer than the correct integer, and then the decryption will (mostly) fail and produce an incorrect result. In the example above, the largest error was 1.97/2, so there is still some headroom for more noise and still correct decryption. The amount of headroom for noise can be increased by increasing the value of Δ or larger q or smaller t.

Homomorphic Operations

One of the main features that is so much interest in these types of cryptography schemes is the ability to do math on the encrypted data. This means that someone who does not have the secret key can still change the ciphertext and get results that when they are decrypted, are meaningful. This means that you can set up an online business where you can provide encrypted data analysis services back to those who find this is an amazing capability that promises to let a new great range of services be created.

Homomorphic addition

The simplest case is the addition of two encrypted numbers. Let us assume we have encrypted two polynomials, m_1 and m_2, with the same public key.

$$\mathbf{c_1} = (\mathbf{p_0}u_1 + e_1 + \Delta m_1, \mathbf{p_1}u_1 + e_2)$$

$$\mathbf{c_2} = (\mathbf{p_0}u_2 + e_3 + \Delta m_2, \mathbf{p_1}u_2 + e_4)$$

Note that we needed to use two different small polynomials u_1 and u_2, as well as 4 small noise polynomials $e_1 \ldots e_4$ to do this.

If we just add the corresponding ciphertext elements, we get a new ciphertext:

$$\mathbf{c} = \mathbf{c_1} + \mathbf{c_2} = (\mathbf{p_0}(u_1 + u_2) + e_1 + e_3 + \Delta(m_1 + m_2), \mathbf{p_1}(u_1 + u_2) + e_2 + e_4)$$

Because the message is present in the ciphertext scaled by Δ, the result of the addition is the same form as an encryption of $m_1 + m_2$. For example, if $m_1 = 1 + 2x^2$ and $m_2 = 1 + x^2$:

$$\mathbf{c} = \lfloor \frac{t}{q}(\mathbf{p_0}(u_1 + u_2) + e_1 + e_3 + \Delta(m_1 + m_2), \mathbf{p_1}(u_1 + u_2) + e_2 + e_4) \rceil$$

The approximate decryption (before rounding) will be:

$$\lfloor \mathbf{p_0}u_1 + \mathbf{p_1}u_1 + e_1 + e_2 \rceil$$

This means that the messages, m_1 and m_2 will decrypt correctly as long as the noise terms are not too large. There are three types of noise terms:

$$e_1 + e_2 + e_3$$

$$\mathbf{p_0}u_1 + \mathbf{p_1}u_1$$

$$\mathbf{p_0}(u_1 + u_2) + \mathbf{p_1}(u_1 + u_2)$$

When we are worried about it when these terms become large enough that one of the coefficients wraps around the coefficient modulus q and gets a different decryption will fail as the rounding operation at the end of the decryption process will round to the wrong number.

If we consider just the first term then we are adding up polynomials with coefficients drawn from the small set of discrete Gaussian distributions. Each coefficient in the sum is a random coefficient by a positive coefficient, and the result will be lower, same, or higher, with coefficients still based the same sign, and so the result will be larger. We can do many additions before the noise term grows large enough to cause a problem. The number of additions that are okay. The distribution of coefficients in a sum between two values centered 1, 5 and 30 (since polynomials combined over several multiplied totals).

When we're asked 30 of these error polynomials, there is a reasonable chance that some of the coefficients will wrapped. Note that adding half of all of the sums, so decryption will not produce the correct result.

The other two terms represent different cases - the second term is an error polynomial multiplied by the sum of some number of small polynomials with coefficients of -1, 0 or 1. The multiplication leads to much larger sums. A coefficient of the result of the product of an error polynomial and a random polynomial is the sum of n random numbers. The distribution of coefficients in the random polynomial is irrelevant, it simply scales the results. The result is the error term scaled by a factor of the size of the polynomials, $\sqrt{n}$.

Drawing the same distributions as above this time shows that it is much larger than the first term, and that even for very small polynomials, there is some danger of getting an incorrect decryption, even after you've scaled the result.

The third term is similar - a sum of a set of error polynomials, multiplied by a "small polynomial". This has an error distribution that looks like this:

Combined, we can plot the growth of the maximum coefficient of the three terms combined, as a function of the number of additions that have occurred. This is shown below as a whole, to give an idea of the variability of these maximum values. Note the mean of the error is close to zero. This is the distribution of the largest coefficient in magnitude.

This shows that for the parameters that we have chosen, there is a high probability of errors for more than 300 additions, and that even before that there is a significant chance of incorrect decryption. This is because summing the maximum errors will lead to large sums that will wrap around to an incorrect decryption when $q = 128$ is in form. To have more headroom for operations such as multiplication, we need to increase q or Δ or decrease t to allow for more amount of noise typically consumed by the number of operations that are allowed.

Unfortunately, the amount of noise introduced by the homomorphic multiplication of ciphertexts is much larger again.

Homomorphic multiplication

Homomorphic multiplication is quite complicated, simply, but comes with more complications than performing addition. As noted above, the message appears in the first element of the ciphertext scaled by Δ. Multiplying two ciphertexts together, the message will be present in the first element of the ciphertext scaled with Δ^2. Which means that we need to scale the result down by Δ before decrypting to avoid wrapping with the modulus q. Which means that we need to be able to perform multiplication with the ciphertexts.

Homomorphic Encryption Illustrated Primer © 2021 Stephen Handy

masking terms.

The trick, then, to understanding the mechanism for homomorphic multiplication is to see how the masking terms are removed from the product of ciphertexts. To do this, the idea is to look at the ciphertexts as a simple polynomial in the powers of s, the secret key. This is about the 3rd different way of using polynomials in this post, so it is a bit confusing, but it is key to understanding how it works.

We can write the first part of the decryption process such that each element of the ciphertext is a coefficient of a polynomial in s:

$$\left[\mathbf{ct}_0 + \mathbf{ct}_1 s^1\right]_q$$

Remember that the components of $\mathbf{ct}$ and s are polynomials themselves, so this equation is a polynomial times one (s^0) added to a polynomial times another polynomial, then all that taken modulo $x^d + 1$ and modulo q.

Now, we saw above that decryption produces a quantity that is independent of the mask terms au.

$$\left[\mathbf{ct}_0 + \mathbf{ct}_1 s^1\right]_q \rightarrow \frac{q}{t} m + noise$$

Well, consider now two ciphertexts $\mathbf{a}$ and $\mathbf{b}$, defined as the encryption of two messages, m_1 and m_2, that can be decrypted so:

$$\left[\mathbf{a}_0 + \mathbf{a}_1 s^1\right]_q \rightarrow \frac{q}{t} m_1 + n_1$$

$$\left[\mathbf{b}_0 + \mathbf{b}_1 s^1\right]_q \rightarrow \frac{q}{t} m_2 + n_2$$

where n_1 and n_2 represent the noise in the ciphertexts.

If we take the product of them we have:

$$\left[\mathbf{a}_0 + \mathbf{a}_1 s^1\right]_q \left[\mathbf{b}_0 + \mathbf{b}_1 s^1\right]_q \rightarrow \left(\frac{q}{t} m_1 + n_1\right)\left(\frac{q}{t} m_2 + n_2\right)$$

The expression on the right hand side is independent of the masks used in calculating $\mathbf{a}$ and $\mathbf{b}$, so the left hand side must be independent of them as well.

If we expand the lefthand side in terms of s (and for convenience multiply by $\frac{t}{q}$) then we have

$$mult(\mathbf{a}, \mathbf{b}) = \mathbf{c}_0 + \mathbf{c}_1 s + \mathbf{c}_2 s^2$$

with

$$\mathbf{c}_0 = \left[\frac{t}{q} \mathbf{a}_0 \mathbf{b}_0\right]_q$$

$$\mathbf{c}_1 = \left[\frac{t}{q} (\mathbf{a}_1 \mathbf{b}_0 + \mathbf{a}_0 \mathbf{b}_1)\right]_q$$

$$\mathbf{c}_2 = \left[\frac{t}{q} \mathbf{a}_1 \mathbf{b}_1\right]_q$$

Doing this means that we can work out the components of a new ciphertext that has one more element than the original ciphertexts, and that can be decoded properly using just powers of the secret key s.

The definition of decryption is extended to include this extra term:

$$\left[\left[\frac{t}{q}\left[\mathbf{ct}_0 s^0 + \mathbf{ct}_1 s^1 + \mathbf{ct}_2 s^2\right]_q\right]\right]_t$$

That just adds another term which is a polynomial times the square of a polynomial. There is lots of bookkeeping to do, but it is just school level algebra (up to the modulus part!). This is a generalisation of the decryption step, and it allows us to decrypt the results of a homomorphic multiplication.

To see how this all works explicitly, consider the expansion of $\mathbf{a}$ and $\mathbf{b}$, in terms of the encryption process

\{ {\bf a} = \left[{\bf pk}*0 u_1 + e*{11} + q m_1 / t, {\bf pk}*1 u_1 + e*{12} \right] \}

\{ {\bf b} = \left[{\bf pk}*0 u_2 + e*{21} + q m_2 / t, {\bf pk}*1 u_2 + e*{22} \right] \}

If we expand the definition of multiplication, along with partial decryption of the result (i.e. decrypt it to the point before we divide by q/t and round), then the resulting expression is complicated. However, because the components of each ciphertext are structured to be able to remove the masking term (au_i) during decryption, the result of this expansion does not depend on the masking term a from the public key at all!!! The expression obtained follows, which loosely shows the noise terms in the result of multiplication:

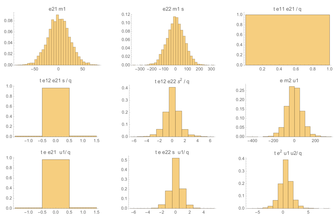
\{ {\bf c}*0 s^0 + {\bf c}*1 s^1 + {\bf c}*2 s^2 \} \{ = \frac{q}{t} m_1 m_2 + e*{22} m_1 s + e*{12} m_2 s + e m_2 u_1 + e m_1 u_2 + e*{21} m_1+e_{11} m_2 \}

$$+\frac{t}{q} e^2 u_1 u_2 + \frac{t}{q} e_{12} e_{22} s^2 + \frac{t}{q} e_{22} e s u_1 + \frac{t}{q} e_{12} e s u_2$$

$$+\frac{t}{q} e_{12} e_{21} s + \frac{t}{q} e_{11} e_{22} s + \frac{t}{q} e_{21} e u_1 + \frac{t}{q} e_{11} e u_2 + \frac{t}{q} e_{11} e_{21}$$

There are lots of terms here, but now we have removed the masks the question is, how large are the noise terms (everything apart from the first) compared to our "noise budget" of q/(2t)?

To get a feeling for this, we simulated a large number of multiplications of encrypted random messages, for $d = 16, t = 7$, and $q = 7168 = 1024 \times t$. The distribution of the coefficient size of each type of error term is shown below. Note that the total error needs to be larger than $q/(2t) = 512$ to cause a decryption error. Of these terms, the terms involving an error polynomial, the message, and the secret key , $e_{22} m_1 s + e_{12} m_2 s$, are the largest contributors.



This plot shows that for these parameters, the largest contribution is from the terms which involve an error polynomial multiplied by a message polynomial and the secret key. The largest coefficient of this noise is about 300. There are two of these terms, and the others are smaller. Combining all the error terms into one shows the total noise in the result of the multiplication. The distribution of these coefficients is shown here:

This shows that there is not quite enough headroom to do a single multiplication safely and then decrypt a correct result for these parameters (which are insecure anyway!) - around 1 coefficient in a 4000 will have an error larger than 512, leading to about a 1% error rate in decryption.

So, to summarise, one can do multiplication of ciphertexts if we treat them as polynomials in s - which thereby cancel out their own masking terms when decrypted - and multiply them as such, keeping track of the coefficients of the powers of s separately, and keeping track of the amount of noise so we can be confident that they decode correctly.

Relinearisation and other topics

The multiplication strategy outlined above allows us to do many multiplications, at the expense of increasing the size of our ciphertext by one polynomial for each multiplication. The fact that the ciphertext grows in size may be an issue. It turns out there are methods for reducing the ciphertext size back to two polynomials, at the cost of increasing the noise. This is known as *relinearisation*, as you are removing the quadratic and higher terms in the polynomial in s.

Another important technique that makes using this cryptographic scheme practical is the process of packing multiple messages into one plaintext to increase the throughput through parallelisation.

Relinearisation and these packed encoding mechanisms will be the subject of a subsequent blog post.

Conclusion

Roughly speaking, encryption is a masked embedding of a message in a polynomial ring with some added noise. Each ciphertext contains enough information to remove its own mask given the secret key. Because the embedding just involves a scaling of the message, addition and multiplication can still be performed on them, with some clever constructions to allow the masks to be removed afterwards. The security of the scheme comes from the difficulty in removing the masking in the presence of the noise when you don't know the private key. The difficulty of this problem leads to some excellent security properties, such as there being no known quantum algorithm for attacking these systems.

If you've got this far, we hope you now have a better appreciation of how Ring Learning with Errors homomorphic encryption schemes (or at least the FV version of these) work. Going through the process of writing this has certainly helped us.

Acknowledgements

Thanks to Wilko Henecka and Hamish Ivey-Law for substantially improving this post with their feedback!

#Homomorphic

@2021 Stephen Hardy adapted from Revolve