A visual explanation of homomorphic encryption (BFV)

Xuan Bach Le

You might not notice this but

you are using encryption everyday. Every time you send an email or FaceTime someone, encryption algorithms work behind the scenes to protect your private information from third parties along the way such as your email service or messaging platform.

Still, there are cases where you want third parties to do computations on your data while still keeping it private, such as diagnosing health conditions using medical data or verifying a voting ballot.

This is not possible with normal encryption schemes since they destroy any relationship between plain and encrypted data. That is until the invention of homomorphic encryption, which allows for addition and multiplication over encrypted data.

Let's learn how a specific homomorphic encryption scheme, **BFV**, works together!

What we need:

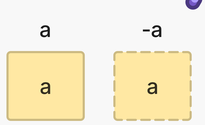
- **Small key** — A public polynomial sampled randomly over set of integers Z
- **Secret key** — A secret polynomial whose coefficients is either 0 or 1
- **Er1, Er2** — Small error polynomials sampled over a discrete normal distribution
- **Message** — A polynomial which contains our data in its coefficients, also called plaintext.

  For example, a message block containing (28, 56, -9, 15, -30) would be represented as:

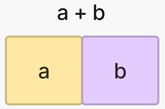
  $$28x^0 + 56x^1 - 9x^2 + 15x^3 - 30x^4$$

How does these colorful diagrams works?

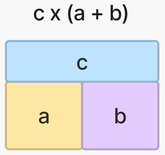
Each block represent a polynomial named for their role. A solid border represent a (+) sign, while a dashed line represent a (-) sign.



When two blocks are attached horizontally, it signifies addition.

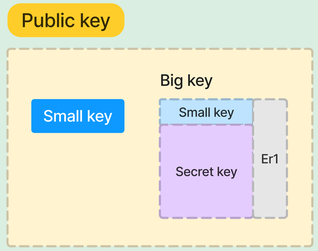


When a block is attached vertically, it means multiplication.

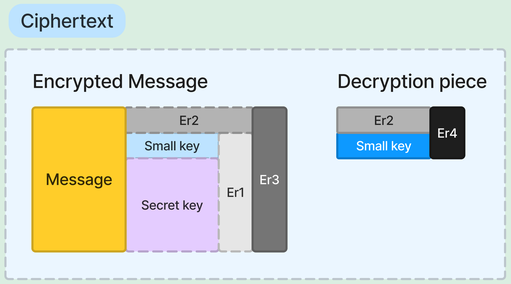


Encryption

First we will create a pair of public key by crafting the big key from the small key and secret key. The strength of encryption rely on the difficulty of finding the Secret key knowing the Public key pair.

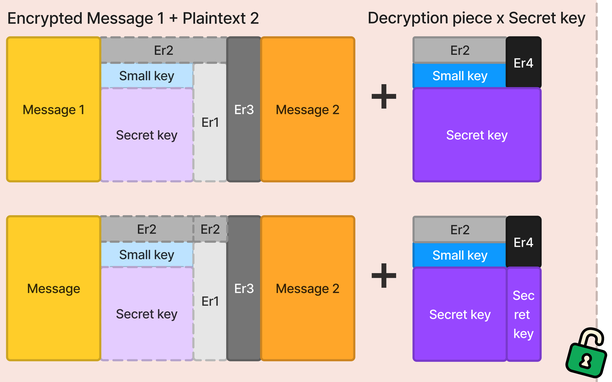


We then wrap the public key pair with error terms and add them to the plaintext message to create a pair of ciphertext, which contains the encrypted message and a decryption piece.



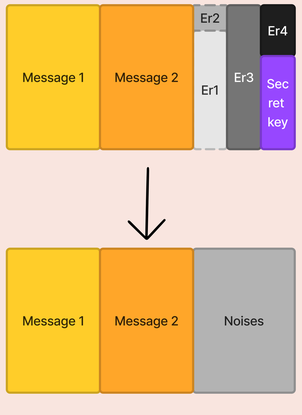
Add a plaintext to a ciphertext

Addition of a ciphertext and a plaintext is done by adding the plaintext message to the encrypted message. We then decrypt by multiplying the decryption piece with the secret key, then combine it with the encrypted message.



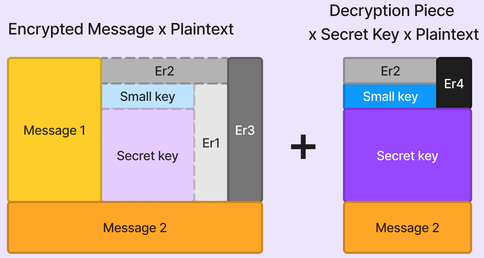
The combination will cancel out the redundancy in the encrypted message, giving us the plaintext messages and some small errors terms as Noises. As long as the Noises are below 0.5, we can round it to the correct result.

We can see that adding a plaintext to a ciphertext do not result in additional noise, which mean that we can do unlimited addition without corrupting the message.

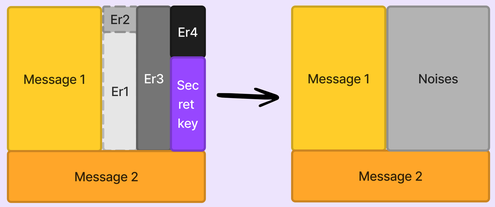


Multiply a ciphertext by a plaintext

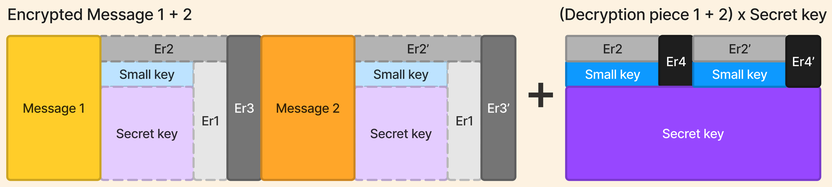
Multiplication of a ciphertext and a plaintext is done by multiplying both the encrypted message and decryption piece with the plaintext. We then decrypt in the same way as before.



We can see that multiplying a ciphertext by a plaintext also multiply the Noises, as such we can use this operator a certain amount of time before the Noises cross the threshold.

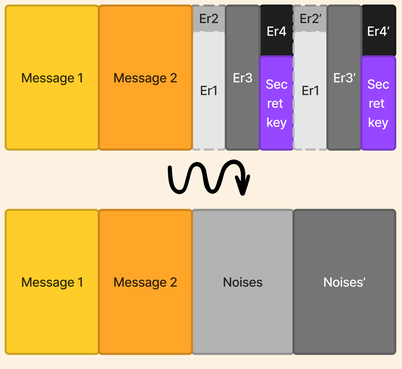


Add two ciphertexts



We can add two ciphertexts by adding the two encrypted messages and decryption pieces, then decrypt in the same way as before

We should notice that this operation also result in an additional layer of noise and as such we need to be careful to not let the noises grow above the decryptioon threshold.



Acknowledgement

The original paper outlining the BFV (Brakerski-Fan-Vercauteren) homomorphic encryption scheme was published in 2012. I simplified some aspects of the scheme, such as the integer and polynomial modulus, to make the explanation more accessible.

Fan, Junfeng, and Frederik Vercauteren. "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive (2012).

If you're curious, you can scan the QR code to the right to access a Python demo of the BFV homomorphic encryption on a Google Colab notebook. The original demo is written by OpenMined.

Thanks for reading!