Can we compute arbitrary functions on encrypted data without decrypting it first? Yes! We recursively apply a scheme that can compute some key functions on encrypted data, thus “bootstrapping” it into a fully homomorphic scheme.

What is fully homomorphic encryption?

Let $pk$ be a public key, $sk$ a private key, $F$ a circuit (think function) on $t$ inputs, $m_1, m_2, \ldots, m_t$ a collection of plaintexts, and $\langle c_1, c_2, \ldots, c_t \rangle$ a collection of ciphertexts, which are encryptions of the $m_i$.

Public key homomorphic scheme:

- $\text{KeyGen}$, $\text{Encrypt}$, $\text{Decrypt}$, $\text{Evaluate}(pk, F, \langle c_1, c_2, \ldots, c_t \rangle)$
- $\text{Evaluate}$ outputs a single new ciphertext $c$. A scheme is correct if $c$ decrypts (via $\text{Decrypt}$) to $m = F(m_1, m_2, \ldots, m_t)$.

If a scheme is correct for a circuit or class of circuits, it is said to be homomorphic on those circuits. If it is correct for all boolean circuits, it is said to be fully homomorphic.

Desired properties of a FHE scheme

Let $c$ denote the ciphertext output by $\text{Evaluate}$.

- **Compactness.** The size of $c$ is polynomial in the security parameter, and is independent of the size of the circuit $F$.
- **Circuit privacy.** The ciphertext $c$ does not reveal anything about the circuit $F$, beyond the result of evaluating it.
- **Chosen plaintext security.** In particular, this requires that encryption be non-deterministic. 
  
  *Note:* Chosen ciphertext security is not possible because homomorphic schemes are by definition malleable.

Bootstrapping

A somewhat homomorphic encryption scheme (SHE) is one that is only homomorphic for certain circuits. SHE’s usually introduce noise every time $\text{Evaluate}$ performs an operation. Thus, after too many operations, the resulting ciphertext can no longer be correctly decrypted by $\text{Decrypt}$.

Gentry [1] proposed the idea of bootstrapping, which is a way to convert a somewhat homomorphic scheme into a fully homomorphic one, if the SHE scheme can handle a circuit a little more complex than its own decryption function.

- A SHE scheme is said to be bootstrappable if it can correctly evaluate the circuits $\text{Decrypt-then-add}$ and $\text{Decrypt-then-multiply}$, because any boolean circuit can be constructed via sequences of additions and multiplications.
- A bootstrappable SHE scheme can be converted into a FHE scheme.

If the scheme is bootstrappable, then we can utilize a new function, $\text{Recrypt}$ to decrypt an arbitrary ciphertext, $c$ and perform a single add or multiply, then encrypt the result. Since any
arbitrary circuit can be constructed through repetition of adds and multiplies, repetition of Recrypt lets us achieve fully homomorphic encryption.

**Example: Fully Homomorphic Encryption Over the Integers** [3]

**Parameters** are \( n \) (bit-length of secret key), \( y \) (bit-length of integers in public key), \( h \) (bit-length of the noise), \( k \) (secondary noise parameter), and \( t \) (number of integers in the public key).

**KeyGen**
The secret key is a random odd \( n \)-bit integer \( p \).
The public key is a tuple \( \langle x_0, x_1, ..., x_t \rangle \) such that \( x_i = pq_i + r_i \), where \( q_i \) are random integers between 0 and \( 2^y/p \), and the \( r_i \) are random integers in \((-2^h, 2^h)\). The public key values are relabeled so that \( x_0 \) is the largest. Repeat the process if \( x_0 \) is even or \( x_0 \% p \) is odd.

**Encrypt**
Choose a random integer \( r \) from \((-2^k, 2^k)\). Calculate the sum of a random subset of the values in the public key to get a value \( S \). The ciphertext is \( c = (m + 2r + 2S) \% x_0 \).

**Evaluate**
Apply the circuit to the input ciphertexts, as integers.

**Decrypt**
Compute \( m = (c \% p) \% 2 \).

The semantic security of the above scheme is reducible to the approximate GCD problem, which is in essence a generalization of the problem of finding the secret key from the public key.

The intuition as to why this scheme works is that all ciphertexts are of the form \( m + pq + 2r \), even after multiplications or additions with other ciphertexts. If \( 2r < p \), then reducing modulo \( p \), and then modulo 2, will leave just \( m \), the plaintext.

The issue arises when the \( 2r \) term becomes too large in relation to \( p \). We choose \( 2r \) to be small at first, but each addition and multiplication will increase its value. In particular, if we do an operation on two ciphertexts with noise greater than \( p/2 \), the resulting \( 2r \) term could have a multiple of \( p \) in it, so that \( 2r \% p \% 2 \) would have no guarantee of being even.

Thus the scheme is somewhat, but not fully homomorphic. However, after a process known as “squashing the decryption circuit,” (embedding a hint about the secret key into the public key to make decryption easier) it can be bootstrapped into a fully homomorphic scheme.

**References**


2