

# Gentry's Fully Homomorphic Encryption Scheme

(over the integers)

# Agenda

- Definitions
  - (Fully) Homomorphic Encryption
  - Approximate GCD and SSSP
  - Semantic Security
- History and Applications
- Construction of a Partially Homomorphic Scheme
- Construction of a Fully Homomorphic Encryption Scheme
  - Bootstrappability
  - Tweaking the Initial Scheme
  - Achieving Bootstappability
- Open Questions

# Definitions

- Homomorphic Encryption
  - $D(f_i(c_1, c_2, \dots)) = f_i(p_1, p_2, \dots)$ 
    - $f_i$  is a set of functions
- Fully Homomorphic Encryption
  - $f_i$  *includes every computable function*
- Approximate GCD and SSSP
  - Approximate GCD: Given set of  $pq_i + n_i$ , determine  $p$
  - SSSP: Given a set of size  $n$  and target  $z$ , select a subset  $A$  of size  $k$  where  $k \ll n$ ,  $\text{sum}(A) = z$
- Semantic Security
  - Given a plaintext and a ciphertext, it is computationally infeasible to determine if the plaintext encrypts to the ciphertext

# History and Applications

- Proposed in 1978, shortly after RSA, by Rivest, Adelman, and Dertouzos.
  - "On Data Banks and Privacy Homomorphisms"
- Partially Homomorphic Schemes: Paillier's, ElGamal, Boneh-Goh-Nissim

**RSA**



# Properties of Homomorphic Encryption

- *Correctness* □
  - $D(f_i(c_1, c_2, \dots)) = f_i(p_1, p_2, \dots)$ 
    - Holds true for all  $f$  in  $f_i$
- *Compactness*
  - The size of the produced ciphertext by *Evaluate* does not depend on the size of the circuit
- *Circuit Privacy*
  - The output of *Evaluate* does not reveal anything about the circuit used (except the output itself)

# Construction of a Partially Homomorphic Encryption Scheme

Security parameter  $\lambda$ .

Private Key Generation()

Random  $\lambda^2$ -bit odd integer "k"

Encrypt(k,plaintext)

plaintext is single bit in  $\{0,1\}$

m is a random  $\lambda$ -bit integer whose LSB is the plaintext

q is a random  $\lambda^5$ -bit integer

ciphertext  $c = m + kq$

Decryption(k,c)

$(c \bmod k) \bmod 2$

# Construction of a Partially Homomorphic Encryption Scheme

- Security of the scheme
  - Approximate GCD
- Efficiency of the scheme
- What makes it partially homomorphic?
  - Turing Machines and circuits
  - Can we do better than a Turing Machine?
- Can we make the scheme a public key encryption scheme?
- What sorts of functions can we compute?
- Can we make the scheme fully homomorphic?
  - If we could find a way to remove noise...

# Bootstrappability

We *can* remove noise! Decrypting the ciphertext and re-encrypting the resulting plaintext will do just that!

But what if we don't want private key holder to do *any* work outside the initial encryption and final decryption?

The solution is to have the client decrypt and re-encrypt every time the noise becomes large - homomorphically!

So the question becomes "can the partially homomorphic scheme handle the circuit version of its decryption function?"

For this scheme, not yet.



# Tweaking the Initial Scheme

Recall that the decryption function is  $(c \bmod k) \bmod 2$ .

$$(c \bmod k) \bmod 2 = \text{LSB}(c) \text{ XOR } \text{LSB}(\lceil c/k \rceil)$$

$\lceil c/k \rceil$ , then, must be difficult to compute. Is there a way that we can "cheat" so that computing  $\lceil c/k \rceil$  will introduce less noise?

What if we introduced a "shortcut" that would allow the server to compute  $c/k$  homomorphically?

# Tweaking the Initial Scheme

- Key Generation\*()
  - $y = \{y_1, y_2, \dots, y_\beta\}$  in  $[0, 2)$  such that there exists subset of size  $\alpha$  which sums to  $1/k$
  - Public key contains  $y$ , encryptions of 0, and an encryption of the private key
  - Private key contains  $k$  (as before) and the solution  $s$  to  $y$
- Encrypt\*( $k, \text{plaintext}$ )
  - Recall that ciphertext  $c = m + kq$
  - Compute  $z = \{z_1, z_2, \dots, z_i\}$  where  $z_i = cy_i$
- Decryption\*( $k, c$ )
  - $(c \bmod k) \bmod 2$ 
    - $= \text{LSB}(k) \text{ XOR } \text{LSB}(\lceil c/k \rceil)$
    - $= \text{LSB}(k) \text{ XOR } \text{LSB}(\lceil \sum s_i z_i \rceil)$

# Achieving Bootstrappability

By replacing the two long multiplications by many small additions the encryption scheme becomes bootstrappable for correctly sized parameters:

$$\sum s_i z_i = \sum s_i c y_i = c \sum s_i y_i = c * 1/k = c/k$$

$\sum s_i z_i$  can be obtained by computing a polynomial whose coefficients are the hamming weights of the terms  $s_i z_i$ .

This computation (decryption) adds noise of  $N \alpha \log(\alpha)$  bits. The scheme becomes bootstrappable when the noise is less than  $P-4$  bits.

When  $\alpha = \lambda / \text{polylog}(\lambda)$ , the tweaked scheme becomes feasible.

# Open Questions

- Computationally feasible FHE.
  - This scheme requires  $\lambda^{10}$  work.
  - The lattice based scheme requires  $\lambda^6$  work.
  - Is there a way to lower these requirements?
- FHE without bootstrapping.
  - Bootstrapping, while clever, slows computation.
- FHE over other domains or trapdoor functions.
- FHE with UTM may have very interesting applications.
  - Can we think of other killer cryptographic applications?