Homomorphic Encryption

Cody Tinker

What is Homomorphic Encryption (HE)?

- Homomorphism: A structure-preserving map between two algebraic structures
  - $f : (A, \circ) \mapsto (B, *)$
  - $f(a \circ b) = f(a) * f(b)$
- An encryption scheme that allows for encrypted data to be evaluated with some operations in the scope of the encrypted group of which the result of the operation will decrypt to the result of a corresponding operation performed on the plaintext data in the scope of the plaintext group.

Motivation

- Privacy of volatile sensitive data
  - Medical records in a database
- Privacy of outsourced processing
  - Cloud computing is a continuously growing field
  - As it grows larger, data will need to be kept secret

Types of Homomorphic Encryption

- Partially Homomorphic Encryption (PHE)
  - Allows one type of operation on encrypted data without limit.
  - Used in a few practical applications: e-voting, Private Information Retrieval (PIR)
  - Example Schemes: RSA, Goldwasser-Micali, El-Gamal, Benaloh, Paillier, ect.
- Somewhat Homomorphic Encryption (SWHE)
  - Allows a set of operations on encrypted data usually a limited number of times.
  - Could be used in applications where the depth of the evaluation operation is constant or known.
  - Example Schemes: Polly Cracker, Boneh-Goh-Nissim (BGN)
- Fully Homomorphic Encryption (FHE)
  - Allows an unlimited number of operations on encrypted data without limit.
  - This is the “holy grail”.
  - Theoretically, can be used for any application.
  - The main focus

Types of FHE Schemes

- Lattice-Based
  - Gentry's original scheme (2009)
  - Security based on the hardness of lattice problems (post quantum): CVP, SVP
  - Large public key sizes and ciphertext
- Integer-Based
  - Van Dijk's scheme (2010)
  - Motivation was to create conceptually simple scheme
  - Security based on the hardness of the Approximate-Greatest Common Divisor (AGCD)
  - Large public key sizes
- (R)LWE-Based (Main focus of FHE)
  - Brakerski and Vaikunthanathan's original scheme (2011)
  - Proved scheme is circularly secure
  - Security based on the Ring-Learning With Error problem (reduced to hard lattice problems) (post quantum).
  - Offers optimizations such as reduced computational complexity and smaller ciphertext sizes
- NTRU-like
  - Lopez-Alt et al. scheme based on NTRU-Encrypt scheme (2012)
  - Technically a (Levelled) Fully Homomorphic scheme (bounded number of public keys)

Rings

- A fundamental algebraic structure, much like groups and fields.
- Groups $\subseteq$ Rings $\subseteq$ Fields
- Consists of a set with two operations, often denoted by addition and multiplication.
- $(R, +, \cdot)$
- Polynomial Ring: K[x], in X over a field K
  - $a_0 + a_1X + a_2X^2 + a_3X^3 + \ldots$ such that $a_i \in K$
- Quotient Ring: $K[x]/(b_0 + b_1X + \ldots + b_2X^m)$
  - $a_0 + a_1X + a_2X^2 + \ldots \pmod{b_0 + b_1X + \ldots + b_2X^m}$
  - Mod term often $X^n + 1$; n is a power of 2

Must satisfy the following axioms with $a,b,c \in R$:

1. R is an abelian group under addition:
   - $(a + b) + c = a + (b + c)$
   - $a + b = b + a$
   - There exists an additive identity
   - There exists additive inverses
2. R is a monoid under multiplication:
   - $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
   - There usually exists a multiplicative identity (though not required)
3. Multiplication is distributive with respect to addition:
   - $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
   - $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$

Learning With Errors (LWE)

"Given $q \geq 2$ and $n \in Z^+$, the Learning With Errors problem can be simply stated as recover $s \in Z_q^n$ from a sequence of "approximate" random linear equations." []

Example:

$12s_1 + 3s_2 + 7s_3 + 3s_4 \approx 9 \bmod 15$
$3s_1 + 1s_2 + 3s_3 + 7s_4 \approx 3 \bmod 15$
$4s_1 + 8s_2 + 9s_3 + 3s_4 \approx 1 \bmod 15$
$14s_1 + 13s_2 + 2s_3 + 1s_4 \approx 12 \bmod 15$
⋮
$11s_1 + 5s_2 + 10s_3 + 7s_4 \approx 3 \bmod 15$

Solve for s. (s is actually [2, 13, 5, 0])

Alternatively, the problem can be states as "given a set of samples, determine whether they originated from the $A_{s,X}$ oracle for some s or whether they follow the uniform distribution on $Z_q^n \times Z_q$." []

In other words, it is hard to distinguish a set of samples generated from $(a, \langle as \rangle + e)$ where a is a sampled vector from Z_q^n uniformly at random and e is an element sampled from Z_q based on some distribution X from samples (a, u) where u is sampled from Z_q uniformly at random.

The problem is reduced to hard lattice problems.

Ring-Learning With Errors (R-LWE)

Extension of LWE but with ring elements.

Example:

Let $R_q = Z_q[x]/\Phi(x)$

Let $a_i, b_i, s \in R_q$

$a_1 \cdot s \approx b_1 \bmod \Phi(x)$

$a_2 \cdot s \approx b_2 \bmod \Phi(x)$

$a_3 \cdot s \approx b_3 \bmod \Phi(x)$

$a_4 \cdot s \approx b_4 \bmod \Phi(x)$

$\vdots$

$a_m \cdot s \approx b_m \bmod \Phi(x)$

Solve for s.

Alternatively, the problem can be states as given a set of pairs (a_i, b_i), determine whether b_i originated from the $A_{s,X}$ oracle for some s or whether they follow the uniform distribution on $R_q \times R_q$.

Generally more efficient to compute and store elements related to this problem over LWE. For example, the samples $(a,b) \in R_q \times R_q$ of the R-LWE problem can replace n samples of $(x, y) \in Z_q^n \times Z_q$ of the LWE problem. This often results in a reduction of public and private keys by a factor of n over a system using LWE.

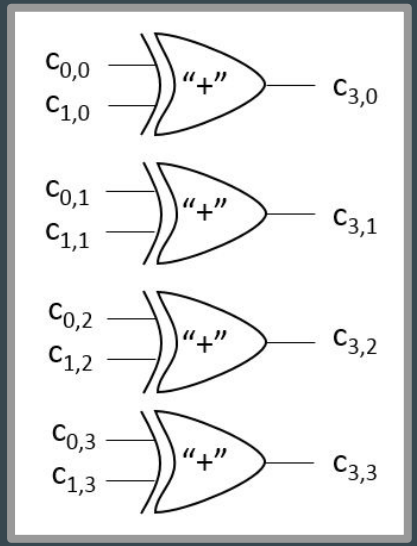
Security is still somewhat an open problem but there is a decent amount of confidence that there is not much of a security trade-off.

FHE Blueprint

- At minimum, can evaluate the following functions:
  - KeyGen: Generates secret and public keys
  - Enc: Encrypts plaintext data into ciphertext space
  - Dec: Decrypts ciphertext data into plaintext space
  - Eval: Performs some function *f()* with set of ciphertexts as inputs and outputs ciphertext corresponding to a functioned plaintext.
- Any Eval function that can compute for a functionally complete set of gates (ex. Binary Addition and Multiplication) can evaluate any circuit homomorphically.
- Squashing: “Squash” the decryption circuit so that its circuit depth can be evaluated by the scheme
  - Adds assumption of Sparse Subset Sum Problem to recovering the secret key
- Bootstrapping: “Re-encrypts” ciphertext to refresh the noise
  - Refresh ciphertext noise by encrypting ciphertext with a second layer and decrypting the first layer homomorphically
  - Requires the user to encrypt the secret key in order to decrypt the ciphertext homomorphically.
  - Assumes a semantically secure and circular secure scheme.
  - This is the real genius behind the design.

How It Works (Generally)

1. Structure binary string messages
   - $m_0 = 0110$; $m_1 = 1010$
2. Encrypt each bit of the messages
   - $c_{0,0} = E(0)$; $c_{0,1} = E(1)$; $c_{0,2} = E(1)$; $c_{0,3} = E(0)$
   - $c_{1,0} = E(1)$; $c_{1,1} = E(0)$; $c_{1,2} = E(1)$; $c_{1,3} = E(0)$
3. Apply ciphertext values to a binary circuit
   - Example: XOR component wise bits
   - $c_{2,0} = c_{0,1} \oplus c_{1,1} \equiv c_{0,1} + c_{1,1} = E(1)$
   - $c_{2,1} = c_{0,0} \oplus c_{1,1} \equiv c_{0,1} + c_{1,1} = E(1)$
   - $c_{2,2} = c_{0,0} \oplus c_{1,1} \equiv c_{0,1} + c_{1,1} = E(0)$
   - $c_{2,3} = c_{0,0} \oplus c_{1,1} \equiv c_{0,1} + c_{1,1} = E(0)$
4. Decrypt resulting ciphertext
   - $m_2 = D(c_{2,0}) = 1$; $m_3 = D(c_{2,1}) = 1$;
   - $m_4 = D(c_{2,2}) = 0$; $m_5 = D(c_{2,3}) = 0$
5. (Optional) Reconstruct resulting message
   - $m_6 = m_2||m_3||m_4||m_5 = 1100 = m_0 \oplus m_1$



Example XOR circuit

Schemes

Examples:

- Gen09 (Gentry original)
- DGHV10 (Van Dijk integer based scheme)
- BGV12 (Brakerski-Gentry-Vaikuntanathan)
- YASHE
- B/FV (Brakerski/Fan-Vercauteren)
- NTRU/LTV (Lopez-Alt-Tromer-Vaikuntanathan)
- GSW (Gentry-Sahai-Waters)

Open Source Libraries:

- SEAL [http://sealcrypto.org]
- HElib [https://github.com/shaih/HElib]
- NFLlib [https://github.com/CryptoExperts/FV-NFLlib]
- Palisade [https://git.njit.edu/groups/palisade]
- cuHE [https://github.com/vernamlab/cuHE]
- HEAAN [https://github.com/kimandrik/HEAAN]
- TFHE [https://tfhe.github.io/tfhe/]