Implementation of FULLY HOMOMORPHIC ENCRYPTION in Hardware

Amir Koupaei

December 3, 2013

Outline

- Homomorphic Encryption Concept
- Gentry-Halevi Fully Homomorphic Encryption Scheme
- Optimization Algorithms
- Efficient Implementations on ASIC and GPU

Homomorphic Encryption

- An encryption scheme is homomorphic on a given operation if the computation can be performed on encrypted inputs
  - Given:
    - $Enc(m_1) = c_1$
    - $Enc(m_2) = c_2$
  - If *Enc* is homomorphic for function *f*
    - $Enc(f(m_1,m_2)) = f(c_1,c_2)$
- Attractive for cloud computing
- Fully Homomorphic Encryption is homomorphic for all functions/manipulations
- First suggested by Rivest, Adleman, and Dertouzos in 1978
- Gentry proposed the first possible scheme in 2009

A Simple Example

- For x,y ϵ {0,1}
  - AND(x,y) = xy
  - OR(x,y) = 1 – (1-x)(1-y)
  - Not(x) = 1-x
- Fully Homomorphic Encryption is achieved if a scheme can add, subtract, and multiply the underlying message in encrypted format
- Encryption
  - c = m' + pq, m' = m mod 2, p is the key, q is a random number
- Decryption
  - m = $[c]_p$ mod 2, where $[c]_p$ = c mod p, for c in (-p/2, p/2)

Need for Recryption

- Ciphertext c is a near-multiple of p
- m' has the same parity as m, and is called the noise in the ciphertext
- Consider $c = c_1 + c_2$
  - $c = m'_1 + m'_2 + pq'$
  - If $| m'_1 + m'_2 | < p/2$ (i.e. noise is small enough), c can be correctly decrypted
- For the Enc/Dec scheme to work, we need to keep the noise term small.
- Requires an additional intermediate step call *Recryption*

Gentry-Halevi FHE Scheme

- First software implementation reported in 2010
- Has four basic primitives
  - Key Gen.
  - Encrypt
  - Decrypt
  - Recrypt

GH-FHE Key Generation

- Generates public-secret key pair
  - Public key: (d,r)
  - Private key: w
- Choose a random n-dimensional integer lattice
  - i.e. v = [v_0, v_1, ... , v_{n-1}]
  - v is a t-bit singed integer
- Compute v(x) and w(x) as following

$$v(x) = \sum_{i=0}^{n-1} v_i \cdot x^i \qquad w(x) = v^{-1}(x) \, mod \, x^n + 1$$

- Compute d, r, and w
  - d = w(x) . v(x) mod (x^n + 1)
  - r = w_1 / w_0 mod d
  - w is an odd w_i

GH-FHE Encryption

- Given Public Key (d,r)
- Generate a random noise vector u = [u_0, u_1, … , u_{n-1}]
  - u ϵ {-1, 0, 1}, P{u = 0} = *p*
- Compute the following

$$c = [b + u(r)]_d = [b + 2\sum_{i=0}^{n-1} u_i \times r^i]_d$$

  - $[N]_d$ = N mod d in range (-d/2, d/2)

GH-FHE Decryption

- Given Private Key w
- Compute
  - $m = [c.w]_d \bmod 2$

GH-FHE Recryption

- At a high level, the ciphertext is decrypted with an encrypted private key
- The purpose is to reduce noise in the ciphertext
- Compute:

$$m = [c \times w]_d \bmod 2 = [\sum_S c \times x_i \times R^{l_i}]_d \bmod 2$$

$$= [\sum_S c \times x_i \times R^{l_i}]_d - [\left\lfloor \sum_S c \times x_i \times R^{l_i} \right\rfloor]_d$$

- Where:

$$\sum_{i=0}^{s} w_i = w$$

$$w_i = x_i \times R^{l_i} \bmod d$$

  - R is a constant
  - $l_i \in \{1, 2, \ldots, s\}$
  - x_i is random

Optimized Algorithms

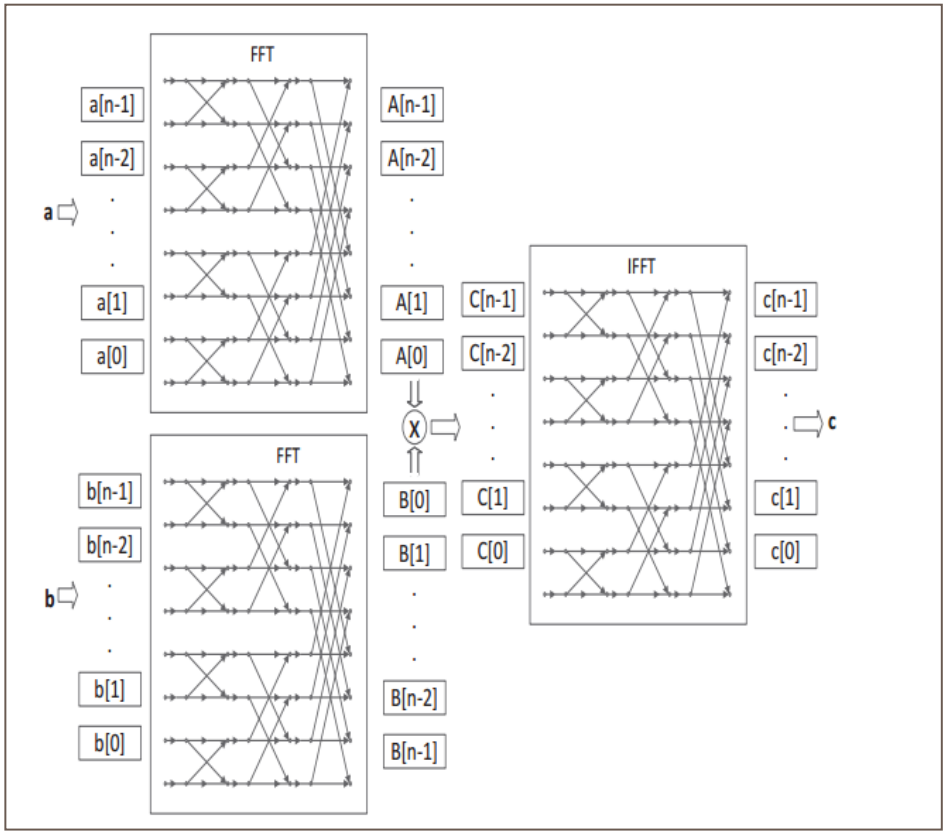
- Parameter sizes proposed by Gentry and Halevi

Setting	Dimension n	Determinant Size d
Small	2,048	785,006
Medium	8,192	3,148,249
Large	32,768	12,625,500

- Modular addition and subtraction of large numbers
  - The carry-lookahead algorithm avoids the long carry chain associated with the carry-ripple adder
- Modular multiplication of large numbers
  - Efficient multiplication is necessary for the decryption primitive
  - The Schonhage-Strassen Algorithm is the most suitable for large-number multiplication
- Modular Reduction
  - Can be efficiently performed using the Barrett Reduction Algorithm

Schonhage-Strassen Algorithm

- Multiplication algorithm based on Fast Fourier Transform
- Effective for parallel computation of large-number multiplication
  - Breaks large inputs A and B into series of words a(n) and b(n)
  - Compute the FFT of A and B, treating each word as a time-domain sample
  - Multiply the FFT results component by component C[i] = A[i] × B[i]
  - Compute the Inverse FFT
  - Resolve the carries
    - If c[i] ≥ b, set c[i+1] = c[i+1] + (c[i] div b), c[i] = c[i] mod b



Barrett Reduction Algorithm

- Goal: Compute r = t mod M
  - q = ceil [$\log_2$ (M)]
  - μ = floor [2^q / M]
  - r = t – M × floor [t.μ / 2^q]
  - While r ≥ M

    r = r – M

    End
  - Return r

Implementation in ASIC

- ASIC Implementation: 90 nm TSMC
- Comparison with CPU implementation for small setting
  - Considerable improvement in throughput
  - Significant improvement in chip area

	ASIC	Xeon
Encrypt	18.1 ms	1800 ms
Decrypt	16.1 ms	20 ms
Recrypt	3.1 sec	32 sec

Implementation in GPU

Small Setting: Dimension 2048				
Operation	CPU	C2050	GTX 690	Speedup
Encrypt	1.08 sec	6.3 msec	6.2 msec	174
Decrypt	14 msec	2.5 msec	1.84 msec	7.6
Recrypt	17.8 sec	1.8 sec	1.32 sec	13.5
Medium Setting: Dimension 8192				
Operation	CPU	C2050	GTX 690	Speedup
Encrypt	10.6 sec	37 msec	24 msec	442
Decrypt	70 msec	13 msec	7.2 msec	9.7
Recrypt	96.3 sec	12.4 msec	8.4 sec	11.5
Large Setting: Dimension 32768				
Operation	CPU	C2050	GTX 690	Speedup
Encrypt	82 sec	1.4 sec	850 msec	96
Decrypt	295 msec	90 msec	29 msec	10.1
Recrypt	800 sec	531 sec	337 sec	2.37

QUESTIONS?