WPI

Accelerating LTV Based Homomorphic Encryption in Reconfigurable Hardware

Yarkın Doröz, Erdinç Öztürk, Erkay Savaş, Berk Sunar

Worcester Polytechnic Institute
100 Institute Road, Worcester, MA, USA, 01609

VERNAM Group

Security & Privacy @ WPI

Homomorphic Encryption

- What is it: Permits computations on encrypted data
- Partially homomorphic encryption
  - Permits evaluation of restricted circuit on encrypted inputs
  - E.g. Goldwasser Micali (XOR), Paillier (Int. Adds), BGN (2-DNF)
- Fully homomorphic encryption (Gentry 2009)
  - Allows efficient evaluation of arbitrary circuits
  - Gentry's blueprint:
    - Builds on *somewhat homomorphic encryption scheme* (SWHE):
    - Supports homomorphic additions and only few levels of multiplication
    - Add Bootstrapping SWHE -> FHE
- Many new constructions: lattice and integer based
  - Orders of magnitude improvement each year since 2010!
- Efficiency still remains as biggest problem!

Overview of FHE Implementations

Table 1. Overview of specialized FHE Implementations. GH-FHE: Gentry & Halevi's FHE scheme; CMNT-FHE: Coron et al.'s FHE schemes [10, 11] [22]; NTRU based FHE, e.g. [27, 34]

Design	Scheme	Platform	Performance
CPU			
AES [25]	BGV-FHE	2.0 GHz Intel Xeon	5 min / AES block
AES [16]	NTRU-FHE	2.9 GHz Intel Xeon	55 sec / AES block
Full FHE [31]	NTRU-FHE	2.1 GHz Intel Xeon	275 sec / per bootst.
GPU			
NTT mul / reduction [35]	GH-FHE	NVIDIA C250 GPU	0.765 ms
NTT mul [35]	GH-FHE	NVIDIA GTX 690	0.583 ms
AES [14]	NTRU-FHE	NVIDIA GTX 690	7 sec / AES block
FPGA			
NTT transform [37]	GH-FHE	Stratix V FPGA	0.125 ms
NTT modmul / enc. [7]	CMNT-FHE	Xilinx Virtex7 FPGA	13 msec / enc.
ASIC			
NTT modmul [17]	GH-FHE	90nm TSMC	2.09 sec
Full FHE [18]	GH-FHE	90nm TSMC	3.1 sec / recrypt

LTV

- Proposed by Lopez-Alt, Tromer and Vaikuntanathan
- Variant of NTRU Encryption by Stehlé and Seinfeld
- Designed as a multi-user, leveled FHE scheme

The construction:

- Operations are performed in $R_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ where q is the prime coefficient modulus
- Sequence of primes $q_0 > q_1 > \ ... > q_d$, a different q_i for each level
- Error distribution χ, a truncated discrete Gaussian distribution, for sampling random polynomials

LTV - Key Generation

Private Key:

$$f^{(i)}(x) = 2u^{(i)} + 1$$

Public Key:

$$h^{(i)}(x) = 2g^{(i)}\left(f^{(i)}\right)^{-1}$$

Evaluation Keys:

$$\zeta_\tau^{(i)}(x) = h^{(i)}s^{(i)} + 2e_\tau^{(i)} + 2^\tau f^{(i-1)}$$

For each $i = 0, 1, \ldots, d$

- $\left\{h^{(i)}, f^{(i)}, \zeta_\tau^{(i)}\right\} \in R_q$
- $\left\{g^{(i)}, u^{(i)}, s_\tau^{(i)}, e_\tau^{(i)}\right\} \in \mathrm{X}$
- $\tau = [0,\ \lfloor \log q_i \rfloor]$

LTV – Encrypt / Decrypt

Encryption:

$$b \in \{0,1\}, \qquad \{s, e\} \in \mathrm{X}$$

$$c^{(i)} = h^{(i)}s + 2e + b$$

Decryption:

$$\mu = \left\lfloor c^{(i)} f^{(i)} \right\rceil_{q_i}$$

$$m = \mu \ (mod\ 2)$$

LTV - Evaluation

$$c_1^{(i)} = Encrypt(b_1) \qquad c_2^{(i)} = Encrypt(b_2)$$

$$Decrypt\left(c_1^{(i)} + c_2^{(i)}\right) = b_1 + b_2$$
$$Decrypt\left(c_1^{(i)} \cdot c_2^{(i)}\right) = b_1 \cdot b_2$$

$$\left(h^{(i)}s_1 + 2e_1 + b_1\right) + \left(h^{(i)}s_2 + 2e_2 + b_2\right)$$
$$\left(h^{(i)}s_1 + 2e_1 + b_1\right) \cdot \left(h^{(i)}s_2 + 2e_2 + b_2\right)$$

$$h^{(i)}\acute{s} + 2\check{e} + (b_1 + b_2)$$
$$h^{(i)^2}\acute{s} + 2\check{E} + (b_1 \cdot b_2)$$

LTV - Evaluation

Relinearization:

$$\check{c}^{(i-1)}(x) = \sum_{\tau} 2^{\tau} \check{c}_{\tau}^{(i-1)}(x)$$

$$\check{c}^{(i)}(x) = \sum_{\tau} \zeta_{\tau}^{(i)}(x) \check{c}_{\tau}^{(i-1)}(x)$$

Modulus Switch:

- Noise Reduction

$$\check{c}^{(i)}(x) = \left\lfloor \frac{q_i}{q_{i-1}} \check{c}^{(i)}(x) \right\rceil_2$$

Specialization of LTV

- Operations are performed in $R_{q_i} = \mathbb{Z}_{\mathrm{q_i}}[\mathrm{x}]/\langle\varphi_m(x)\rangle$
  - $\varphi_m(x) = m^{th}$ cyclotomic polynomial
  - Degree of $\varphi_m(x)$, n is $\phi(m)$
  - $\varphi_m(x)$ is factorized over F_2 into equal degree polynomials $F_i(x)$
  - Using CRT, we batch $S = n/t$ messages, t is the smallest integer that satisfies $m|(2^t - 1)$
- Select number of primes p_i with size of $\log p$ –noise cutting size
  - $q_i = \prod_{i \le j} p_j$
- Evaluation keys are promoted to the next level via

$$\zeta_\tau^{(i)}(x) = \zeta_\tau^{(0)}(x)\ (mod\ q_i)$$

Arithmetic Operations

- Large polynomial multiplications and relinearization operations are costly
  - convert input polynomials using CRT
    - Same degree, smaller word-size coefficients
- Pairwise polynomial product
  - computed using Number Theoretical Transform (NTT) multiplication
- Resulting polynomial is recovered from the partial products using inverse CRT
  - postpone ICRT until switching a level

Polynomial Multiplication

- The classical multiplication techniques are too costly:
  - Schoolbook algorithm have quadratic complexity $O(N^2)$
  - Karatsuba algorithm reduce the complexity to $O\left(N^{\log_2 3}\right)$
- Very large parameters/operands:
  - Need asymptotically better algorithms!
- We use Schönhage–Strassen algorithm to compute multiplications:
  - Number theoretic transform (NTT) has $O(N \log N \log\log N)$ complexity
  - Fast Fourier Transform is needed
    - We use Cooley-Tukey decomposition

Polynomial Multiplication

$$A(x) = \sum_{i=0}^{2N-1} A_i x^i \qquad A_i \in \mathbb{Z}_p$$

$$\check{\mathrm{A}}_i = \sum_{j=0}^{2N-1} A_j \cdot w^{ij} \ (mod\ p) \qquad w^{2N} = 1\ (mod\ p)$$

$$\check{\mathrm{C}}_i = \check{\mathrm{A}}_i \cdot \check{\mathrm{E}}_i \ \ (mod\ p) \text{ for all } i \in 2N$$

$$C_i = (2N)^{-1} \cdot \sum_{j=0}^{2N-1} \check{\mathrm{C}}_j \cdot w^{-ij}\ (mod\ p)$$

Polynomial Multiplication

- Cooley-Tukey
  - SW: Recursive
    - $\check{A}_i = \sum_{j=0}^{2N-1} A_j \cdot w^{ij} \ (mod\ p)$

$$= \sum_{j=0}^{N-1} A_{2j} \cdot w^{i(2j)} + w^i \sum_{j=0}^{N-1} A_{2j+1} \cdot w^{i(2j)} \ (mod\ p)$$

  - HW: Non-Recursive

ALGORITHM 1: Iterative Version of Number Theoretic Transformation

input : $A(x) = A_0 + A_1 x + \ldots + A_{N-1}x^{N-1}$, $N = 2^n$, and w
output: $\mathcal{A}(x) = \mathcal{A}_0 + \mathcal{A}_1 x + \ldots + \mathcal{A}_{N-1}x^{N-1}$

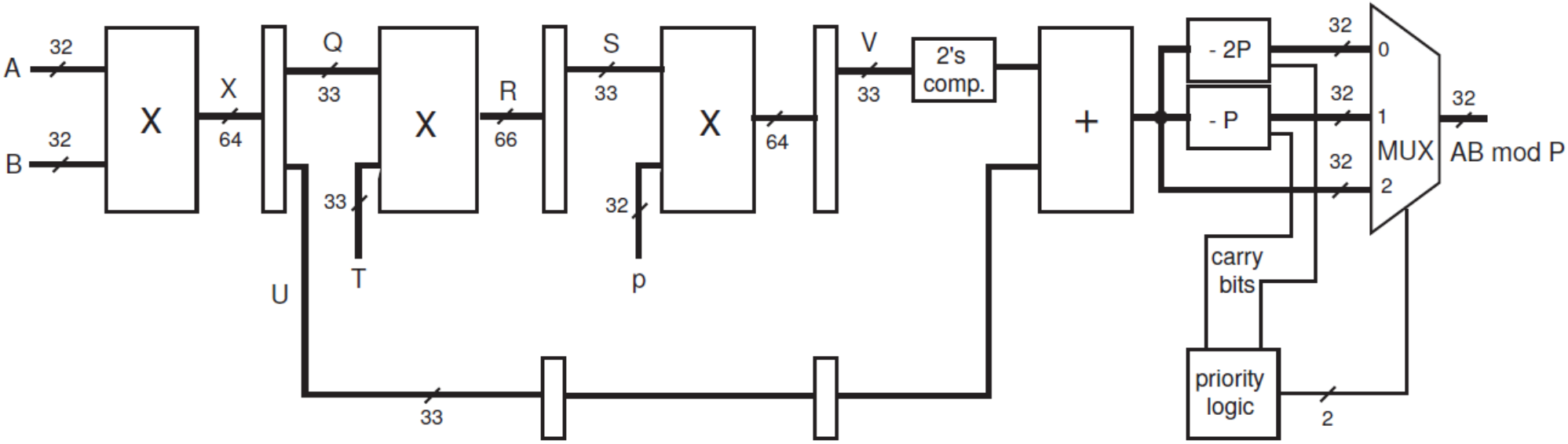
```
for i = N to 2N − 1 do
    A_i = 0 ;
end
(A_0, A_1, ..., A_{2N−1}) ← Permutation(A_0, A_1, ..., A_{2N−1});
for M = 2 to 2N do
    for j = 0 to 2N − 1 do
        for i = 0 to M/2 − 1 do
            x ← i × 2N/M ;
            I ← j + i ;
            J ← j + i + M/2 ;
            A[I] ← A[I] + w^{x mod 2N} × A[J] mod p ;
            A[J] ← A[I] − w^{x mod 2N} × A[J] mod p ;
            i ← i + 1;
        end
        j ← j + M;
    end
    M ← M × 2;
end
```

Architecture – 32-bit Modular Multiplier

- The homomorphic AES algorithm requires 31-bit cutting size
  - small CRT primes need to be 31-bits.
- We designed our hardware for 32-bit multiplications.
  - We require four 16x16-bit multiplications
    - 1 DSP unit can compute in 4 clock cycles
- We used Barrett's algorithm for modular reduction
  - 33x33-bit multiplications
  - 6 clock cycle latency

Architecture – 32-bit Modular Multiplier

- Using 3 multiplier and pipelining we compute modular multiplication in 19 clock cycles
  - Throughput: 4 clock cycles



Architecture - $2^{15}x2^{15}$ Polynomial Multiplier

- *K* modular multipliers
  - 3x*K* DSP units
  - power of two
- *K* Block RAMs
  - To feed multipliers with digits
  - Store the results
  - Block RAM indices generation
    - degree size (*N*), NTT level (*m*) and modular multiplier number (*K*)
  - We make sure to access in 4 clock cycles:
    - powers of primitive root *w*
    - polynomial coefficients

Architecture - 2^{15}x2^{15} Polynomial Multiplier

ALGORITHM 4: Parallel access to polynomial coefficients

input : $A(x) = A_0 + A_1x + \ldots + A_{2N-1}x^{2N-1}$, n, m, and $\kappa < n$
output: $B_i[j]$

```
1  mCnt ← 2^(m−1) − 1 ;          /* number of multiplications in a block */
2  bSize ← 2^m ;                                  /* size of a block */
3  BRAMCnt ← 2^(κ−2) ;                            /* number of BRAMs */
4  if bSize ≤ 2^(κ−2) then
       for t = 0 to 1024 do
           for i = 0 to BRAMCnt do in parallel
               for j = i + bSize − mCnt to i + bSize do
                   for k = 0 to 3 do
5                      Access BRAM_j[t + 2k] ;
6                      Access BRAM_j[t + 2k + 1] ;
                       k ← k + 1;
                   end
                   j ← j + 1;
               end
               i ← i + bSize;
           end
           t ← t + 8;
       end
   end
7  else
       for i = 0 to BRAMCnt do in parallel
           for j = 0 to 1024 do
               for k = 2^(m−κ+1) to 2^(m−κ+2) do
8                  Access BRAM_i[k + j] ;
                   k ← k + 1;
               end
               j ← j + 2^(m−κ+2);
           end
           i ← i + 1;
       end
   end
```

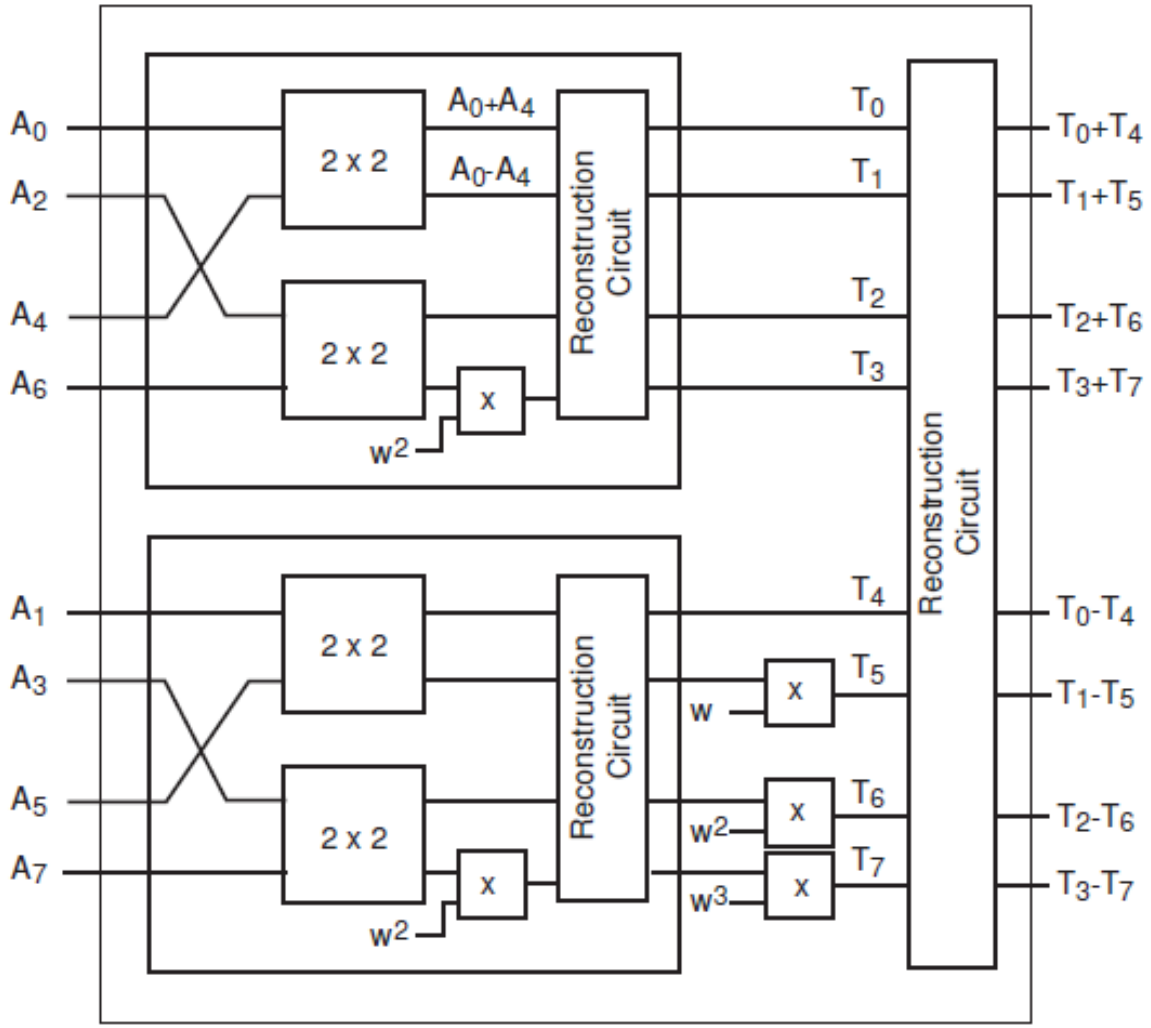
- Recall Our Recursive Algorithm:

$$\check{A}_i = \sum_{j=0}^{2N-1} A_j \cdot w^{ij} \ (mod\ p)$$

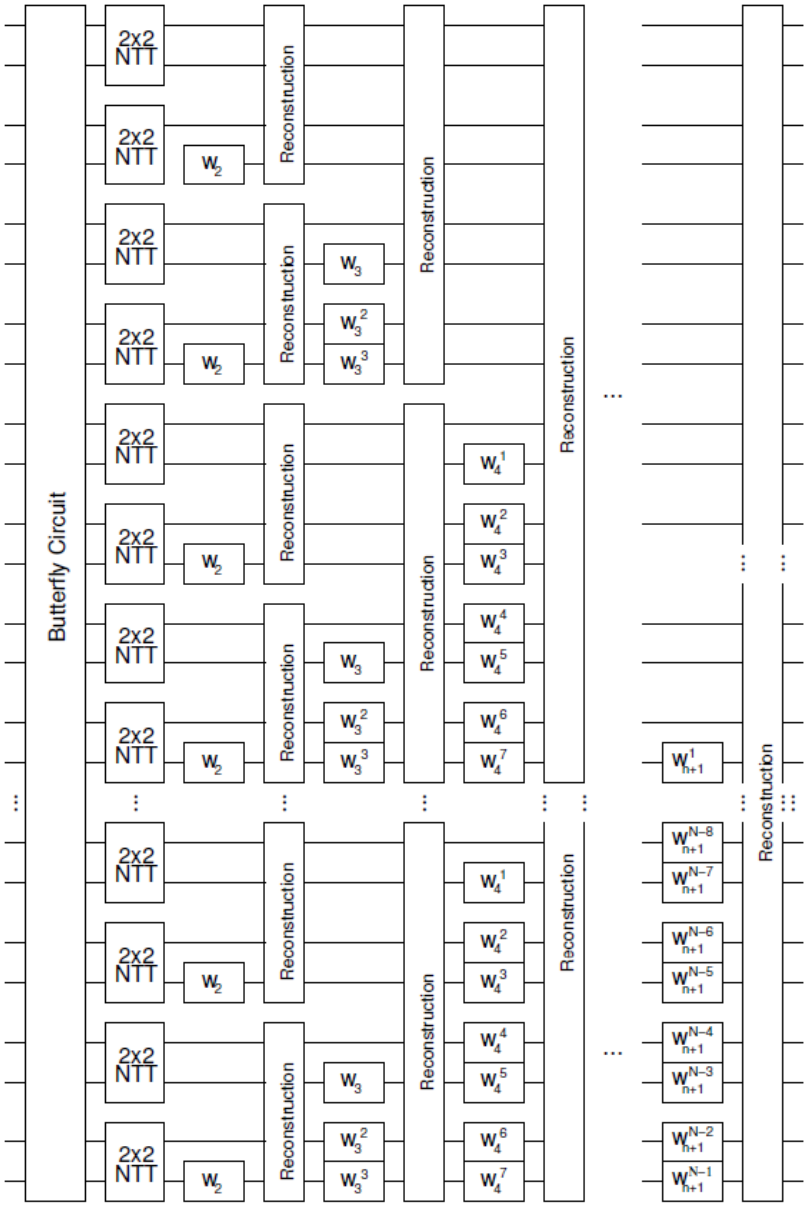
$$= \sum_{j=0}^{N-1} A_{2j} \cdot w^{i(2j)}$$

$$+ w^i \sum_{j=0}^{N-1} A_{2j+1} \cdot w^{i(2j)} \ (mod\ p)$$

Architecture – 2-point to 8-point NTT



Architecture – 64K-point NTT



Implementation Results

- We used Virtex-7 XC7VX690T FPGA
- The design reach 250 MHz clock
- The polynomial degree $n = 32{,}768$ and $\log p = 32$
- We choose the number of modular multiplier as $K = 64$

	Total	Used	Used (%)
Slice LUTs	433,200	219,192	50.59
Slice Registers	866,400	90,789	10.47
RAMB36E1	1470	193	13.12
DSP48E1	3600	768	21.33

Timing for Primitive Operations

- We include PCIe timing (8 Gbit/sec per lane)
- CRT is done on the CPU

	Timings (msec)		Timings (msec)
CRT	**89**	**Modulus Switch**	**89**
Multiplication	**9.51**	**Relinearization**	**526**
NTT conversions	6.25	CRT conversions	89
PCIe cost	3.26	NTT conversions	331
Modular Reduction	**19**	PCIe cost	52

Comparison

- AES is a 40 level circuit
  - 2,880 Relinearizations
  - 5,760 Modular Multiplications
  - 6,080 Modular Switching
- Total AES takes 15 minutes (2048 message slots)
  - Amortized time is 439 msec

	Mul (msec)	Speedup	Relin (sec)	Speedup	AES (sec)	Speedup
CPU [16]	970	1×	103	1×	55	1×
GPU [14]	340	2.8×	8.97	11.5×	7.3	7.5×
CPU (16-bit)	970	1×	6.5	16×	12.6	4.4×
FPGA (ours)	**9.5**	**102×**	**0.53**	**195×**	**0.44**	**125×**

Thank you!

Questions?