KU LEUVEN

Faster Private Decision Tree Evaluation for Batched Input from Homomorphic Encryption

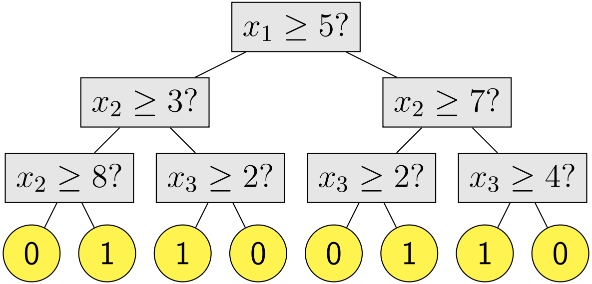
Kelong Cong[1], **Jiayi Kang**[2], Georgio Nicolas[2], and Jeongeun Park[3]
[1]Zama, [2]COSIC, KU Leuven, and [3]NTNU

SCN 2024, September 11

Outline

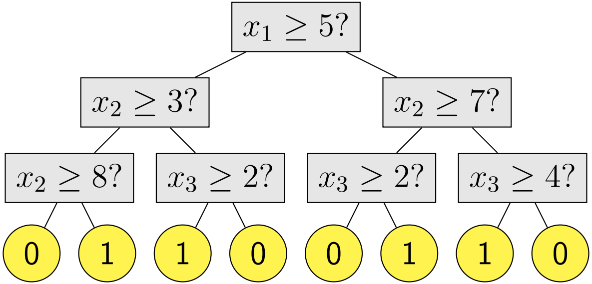
Private Decision Tree Evaluation (PDTE)

- Given n feature values, evaluating a decision tree outputs a classification label (e.g. 0/1)



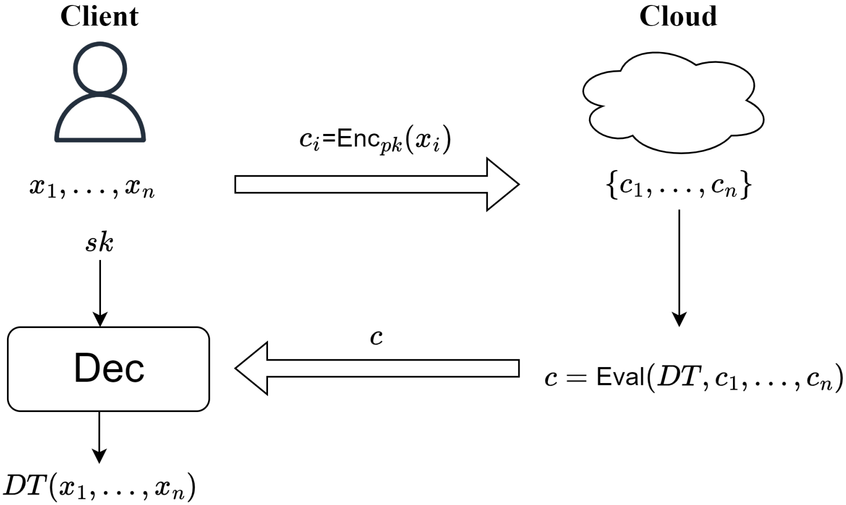
Private Decision Tree Evaluation (PDTE)

- Given n feature values, evaluating a decision tree outputs a classification label (e.g. 0/1)



- Simple machine learning algorithm with broad applications
  - credit scoring, biometric authentication,...
  - sensitive data requires enhanced-privacy

PDTE from Homomorphic Encryption



KU LEUVEN

PDTE from Homomorphic Encryption

- SortingHat [CDPP22] uses TFHE for single-query scenarios
- Level Up [MNLK23] uses the levelled BFV scheme, which supports SIMD (Single-Instruction Multiple-Data) operations

KU LEUVEN

PDTE from Homomorphic Encryption

- SortingHat [CDPP22] uses TFHE for single-query scenarios
- Level Up [MNLK23] uses the levelled BFV scheme, which supports SIMD (Single-Instruction Multiple-Data) operations
- Can we further exploit the SIMD capacity for batched queries?

PDTE from Homomorphic Encryption

- SortingHat [CDPP22] uses TFHE for single-query scenarios
- Level Up [MNLK23] uses the levelled BFV scheme, which supports SIMD (Single-Instruction Multiple-Data) operations
- Can we further exploit the SIMD capacity for batched queries?

Batched PDTE

- Evaluate one decision tree for multiple samples in parallel
- Example application: a bank outsources a credit-scoring decision tree and needs evaluations for various applicants without revealing their profiles

Outline

KU LEUVEN

Folklore bit-wise comparator

- Two s-bit numbers $\mathbf{a}, \mathbf{b}$ can be compared using recursion

$$\mathsf{GT}(\mathbf{a}, \mathbf{b}) = \theta_{GT}(\mathbf{a}[1], \mathbf{b}[1]) + \theta_{EQ}(\mathbf{a}[1], \mathbf{b}[1]) \cdot \mathsf{GT}(\mathbf{a}[2, s], \mathbf{b}[2, s])$$

Folklore bit-wise comparator

- Two s-bit numbers $\mathbf{a}, \mathbf{b}$ can be compared using recursion

$$\mathsf{GT}(\mathbf{a}, \mathbf{b}) = \theta_{GT}(\mathbf{a}[1], \mathbf{b}[1]) + \theta_{EQ}(\mathbf{a}[1], \mathbf{b}[1]) \cdot \mathsf{GT}(\mathbf{a}[2, s], \mathbf{b}[2, s])$$

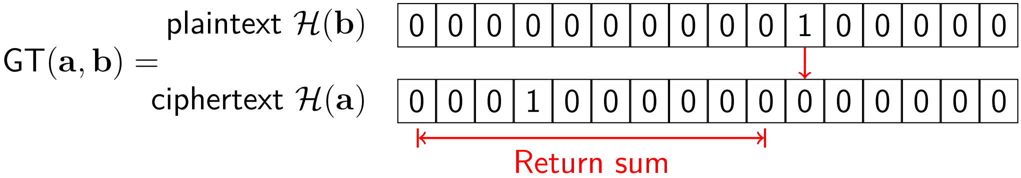
- For ciphertext $\mathbf{a}$ and plaintext $\mathbf{b}$,
  - Bit comparisons θ_{EQ} and θ_{GT} are at most degree 1
  - The total number of multiplications is $s - 1$
  - The minimum multiplicative depth is $\log s$

Comparing one-hot encoded numbers

- Two s-bit numbers $\mathbf{a}, \mathbf{b}$ are encoded into $\mathcal{H}(\mathbf{a}), \mathcal{H}(\mathbf{b}) \in \{0,1\}^{2^s}$

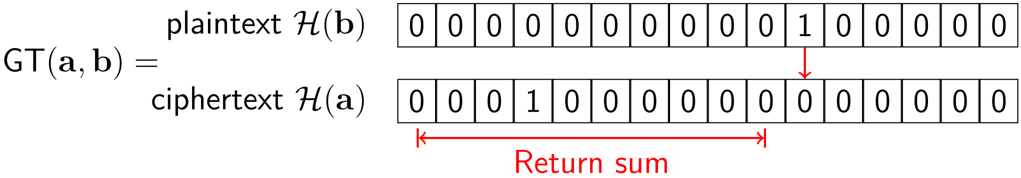
Comparing one-hot encoded numbers

- Two s-bit numbers $\mathbf{a}, \mathbf{b}$ are encoded into $\mathcal{H}(\mathbf{a}), \mathcal{H}(\mathbf{b}) \in \{0,1\}^{2^s}$
- The ciphertext $\mathcal{H}(\mathbf{a})$ and plaintext $\mathcal{H}(\mathbf{b})$ can be compared as



Comparing one-hot encoded numbers

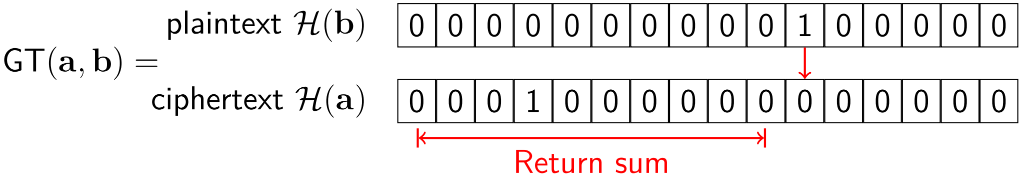
- Two s-bit numbers $\mathbf{a}, \mathbf{b}$ are encoded into $\mathcal{H}(\mathbf{a}), \mathcal{H}(\mathbf{b}) \in \{0,1\}^{2^s}$
- The ciphertext $\mathcal{H}(\mathbf{a})$ and plaintext $\mathcal{H}(\mathbf{b})$ can be compared as



- The number of multiplications is zero, but the bitlength is 2^s instead of s

Comparing one-hot encoded numbers

- Two s-bit numbers $\mathbf{a}, \mathbf{b}$ are encoded into $\mathcal{H}(\mathbf{a}), \mathcal{H}(\mathbf{b}) \in \{0,1\}^{2^s}$
- The ciphertext $\mathcal{H}(\mathbf{a})$ and plaintext $\mathcal{H}(\mathbf{b})$ can be compared as



- The number of multiplications is zero, but the bitlength is 2^s instead of s
- Can we balance computation and communication?

Our constant-weight piece-wise comparator

- With a constant hamming weight h, an s-bit number $\mathbf{a}$ can be encoded into $\mathcal{CW}_{h,\ell}(\mathbf{a})$ of ℓ bits, where

$$\binom{\ell}{h} \geq 2^s \Rightarrow \ell \in O(\sqrt[h]{h!2^s} + h)$$

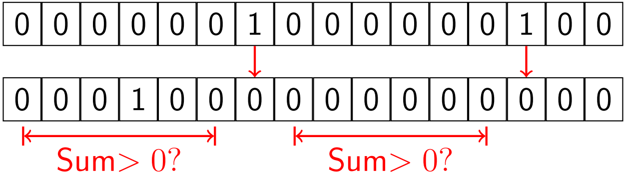
Our constant-weight piece-wise comparator

- With a constant hamming weight h, an s-bit number $\mathbf{a}$ can be encoded into $\mathcal{CW}_{h,\ell}(\mathbf{a})$ of ℓ bits, where

$$\binom{\ell}{h} \geq 2^s \Rightarrow \ell \in O(\sqrt[h]{h!2^s} + h)$$

- The ciphertext $\mathcal{CW}_{h,\ell}(\mathbf{a})$ and plaintext $\mathcal{CW}_{h,\ell}(\mathbf{b})$ can be compared piece-wise recursively

$\mathsf{GT}_h(\mathbf{a}, \mathbf{b}) =$ $\mathcal{CW}_{h,\ell}(\mathbf{b})$ / $\mathcal{CW}_{h,\ell}(\mathbf{a})$

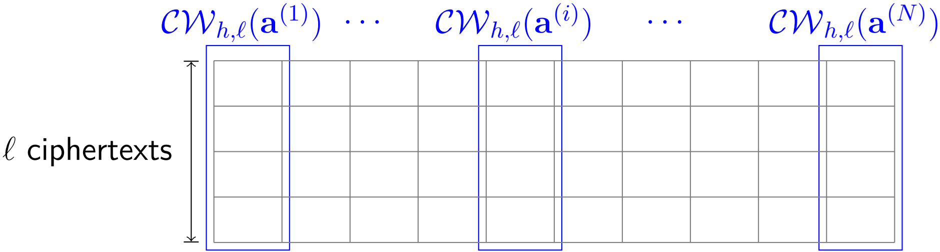


Our batched constant-weight piece-wise comparator

- The depth is $\mathcal{O}(\log h)$, which is independent of the input bitlength s

Our batched constant-weight piece-wise comparator

- The depth is $\mathcal{O}(\log h)$, which is independent of the input bitlength s
- For BFV with N SIMD slots, the following packing method allows the comparison between N encrypted $\mathbf{a}$ and 1 plaintext $\mathbf{b}$



Range Cover Comparator (RCC) [SBC+07]

- Given two s-bit numbers $\mathbf{a}, \mathbf{b}$,

$$\mathsf{GT}(\mathbf{a}, \mathbf{b}) \iff \mathbf{a} \in [\mathbf{b}+1, 2^s - 1]$$

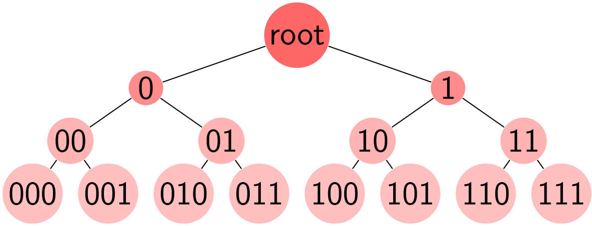
- If $\mathbf{a} \in [\mathbf{b}+1, 2^s-1]$, then $PE(\mathbf{a}) \cap RC(\mathbf{b}+1, 2^s-1) = \emptyset$; otherwise, they will intersect at one node.

Range Cover Comparator (RCC) [SBC+07]

- Given two s-bit numbers $\mathbf{a}, \mathbf{b}$,

$$\mathsf{GT}(\mathbf{a}, \mathbf{b}) \iff \mathbf{a} \in [\mathbf{b}+1, 2^s-1]$$

- If $\mathbf{a} \in [\mathbf{b}+1, 2^s-1]$, then $PE(\mathbf{a}) \cap RC(\mathbf{b}+1, 2^s-1) = \emptyset$; otherwise, they will intersect at one node.



A binary interval tree containing $[0, 7]$. For example, the point encoding of the number 5 is $PE(5) = \{1, 10, 101\}$ and the range cover of $[1, 7]$ is $RC(1, 7) = \{1, 01, 001\}$.

Batched RCC

- 1 GT comparator $\iff$ s equality checks (one for each level)

Batched RCC

- 1 GT comparator $\iff$ s equality checks (one for each level)
- In [MNLK23], these equality checks are realized in constant-weight encodings for s bit numbers
  - But the operands in equality checks are i bits for $i = 1, 2, \ldots, s$
  - Restriction of their packing method

Batched RCC

- 1 GT comparator $\iff$ s equality checks (one for each level)
- In [MNLK23], these equality checks are realized in constant-weight encodings for s bit numbers
  - But the operands in equality checks are i bits for $i = 1, 2, \ldots, s$
  - Restriction of their packing method
- When comparing N encrypted $\mathbf{a}$ and 1 plaintext $\mathbf{b}$
  - We encode the point encoding $PE(\mathbf{a}^{(i)})$ in SIMD slots using $\mathcal{CW}_{h^i,\ell^i}(\cdot)$
  - This improves the amortized storage and computation costs

Performance

		Amortized Computational Time	Amortized Client-to-server Communication Cost	Multiplicative Depth
Folklore bit-wise [MNLK23]	$\perp$	1982 μs	3 kb	4
RCC [MNLK23]	$h = 2$	8340 μs	1342 kb	1
	$h = 4$	1526 μs	136 kb	2
	$h = 8$	1308 μs	70 kb	3
Our CW piece-wise	$h = 2$	18 μs $(\mathbf{72}\times)$	52 kb	2
	$h = 4$	39 μs $(\mathbf{33}\times)$	5 kb	4
Our batched RCC	$h_s = 2$	41 μs $(\mathbf{32}\times)$	180 kb	1
	$h_s = 4$	82 μs $(\mathbf{16}\times)$	38 kb	2

Table: Performance of different batched ciphertext-plaintext comparators for 16-bit numbers in BFV with $N = 2^{14}$ and $t = 65537$.

Outline

1. Batched PDTE: Background and Motivation
2. Batched Ciphertext-Plaintext Comparisons
3. Tree Traversal Methods
4. Batched PDTE: Performance and Conclusion

KU LEUVEN

Tree Traversal in PDTE

- In the plaintext evaluation of a depth-d decision tree, at most d decision nodes are evaluated

KU LEUVEN

Tree Traversal in PDTE

- In the plaintext evaluation of a depth-d decision tree, at most d decision nodes are evaluated
- In the homomorphic evaluation of a decision tree,
  - *all* the $m = \mathcal{O}(2^d)$ nodes need to be evaluated using ciphertext-plaintext comparisons
  - the encrypted comparison results are aggregated by tree traversal

Tree Traversal in PDTE

- In the plaintext evaluation of a depth-d decision tree, at most d decision nodes are evaluated
- In the homomorphic evaluation of a decision tree,
  - *all* the $m = \mathcal{O}(2^d)$ nodes need to be evaluated using ciphertext-plaintext comparisons
  - the encrypted comparison results are aggregated by tree traversal
- Tree traversal outputs an encrypted indicator array $\mathsf{Enc}(\mathbf{r})$
  - In SumPath [MNLK23], the array $\mathbf{r}$ contains only one zero value corresponding to the output leaf

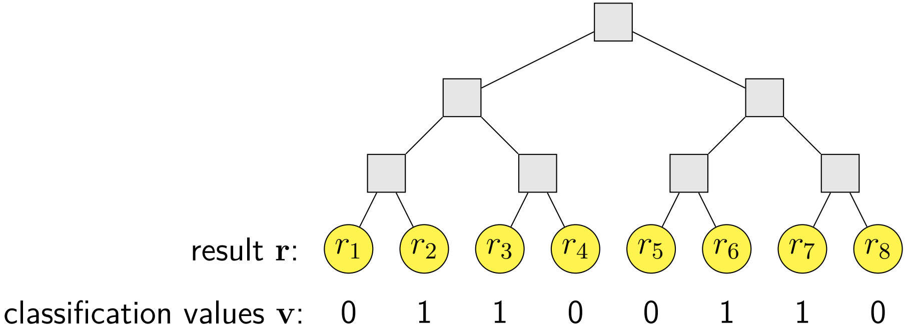
The SumPath method [MNLK23]

- PDTE with SumPath
  - The server sends $\mathsf{Enc}(\mathbf{r})$ of length $\mathcal{O}(2^d)$ to the client
  - The client decrypts, finds the only leaf with zero value and looks up the corresponding classification of this leaf

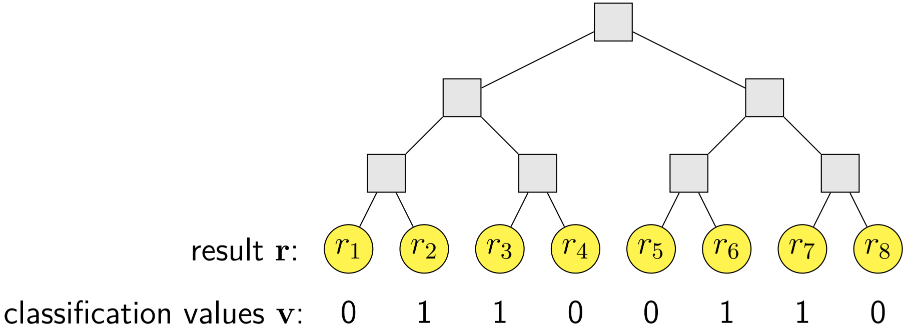
The SumPath method [MNLK23]

- PDTE with SumPath
  - The server sends $\mathsf{Enc}(\mathbf{r})$ of length $\mathcal{O}(2^d)$ to the client
  - The client decrypts, finds the only leaf with zero value and looks up the corresponding classification of this leaf
- Drawbacks
  - $\mathcal{O}(2^d)$ server-to-client communication
  - Limited extension to tree ensembles such as random forests

The adapted SumPath method



The adapted SumPath method



- Obtain an encrypted unit vector $\mathbf{r}$, whose inner product with the plaintext classifications gives the encrypted classification value
- This requires an additional multiplicative depth $\log d$

Outline

1. Batched PDTE: Background and Motivation
2. Batched Ciphertext-Plaintext Comparisons
3. Tree Traversal Methods
4. Batched PDTE: Performance and Conclusion

Performance for 11-bit features

	SortingHat			Level Up ($h = 4$)			BPDTE_CW ($h = 2$)		
	Comparison	Traversal	Query Size	Comparison	Traversal	Query Size	Comparison	Traversal	Query Size
Breast	7 ms	178 ms	960 kb	139 μs	117 μs	310 kb	9 μs	139 μs	90 kb
	Total: 185 ms			Total: 256 μs			Total: 148 μs ($\mathbf{1.7}\times$)		
Heart	3 ms	47 ms	416 kb	156 μs	25 μs	135 kb	3 μs	18 μs	117 kb
	Total: 50 ms			Total: 181 μs			Total: 21 μs ($\mathbf{8.6}\times$)		
Steel	3 ms	59 ms	1056 kb	125 μs	34 μs	341 kb	4 μs	12 μs	297 kb
	Total: 62 ms			Total: 159 μs			Total: 16 μs ($\mathbf{9.9}\times$)		

Table: Amortized performance with batch size 16384 and input feature bitlength $s = 11$. The lattice dimension is 2^{11}, 2^{13} and 2^{14}, respectively.

Performance for 16-bit features

	Level Up ($h = 4$)			BPDTE_RCC ($h_s = 4$)			BPDTE_CW ($h = 2$)		
	Comparison	Traversal	Query Size	Comparison	Traversal	Query Size	Comparison	Traversal	Query Size
Breast	583 μs	159 μs	968 kb	75 μs	139 μs	1140 kb	17 μs	138 μs	1560 kb
	Total: 742 μs			Total: 214 μs ($\mathbf{3.4}\times$)			Total: 155 μs ($\mathbf{4.7}\times$)		
Heart	309 μs	34 μs	420 kb	20 μs	18 μs	494 kb	4 μs	18 μs	676 kb
	Total: 343 μs			Total: 38 μs ($\mathbf{9.0}\times$)			Total: 22 μs ($\mathbf{15.5}\times$)		
Steel	262 μs	46 μs	1065 kb	25 μs	12 μs	1254 kb	6 μs	12 μs	1716 kb
	Total: 308 μs			Total: 37 μs ($\mathbf{8.3}\times$)			Total: 18 μs ($\mathbf{17.1}\times$)		

Table: Amortized performance with batch size 16384 and input feature bit-length $s = 16$. The lattice dimension is 2^{13}, 2^{14} and 2^{14}, respectively.

Conclusion

Conclusion

- Two batched ciphertext-plaintext comparators
  - the constant-weight piece-wise comparator and the batched RCC comparator
  - up to $72\times$ speedup for 16-bit numbers

Conclusion

- Two batched ciphertext-plaintext comparators
  - the constant-weight piece-wise comparator and the batched RCC comparator
  - up to $72\times$ speedup for 16-bit numbers
- The adapted SumPath tree traversal method
  - $\mathcal{O}(1)$ server-to-client communication
- Batched PDTE protocols from combining these building blocks
  - up to $17\times$ faster than [MNLK23] in batch size 16384

Thank you for your attention!

ia.cr/2024/662