String Matching on Homomorphically Encrypted Data

Kim Laine
Microsoft Research, Redmond

at

Indian Institute of Science
Bangalore, India 2017

Problem

- iDASH 2016 Secure Genome Analysis competition
- Encrypt dataset of up to $100{,}000$ genetic mutations (.VCF file)
- Store in cloud
- Later query for presence of a mutation (encrypted query)
- Query for presence of more than one mutation (multiquery)
- Can this kind of use-case be practical?
- Need efficient string matching using HE

Homomorphic Encryption

- Evaluate circuits on encrypted data
- Leveled fully homomorphic encryption
- Bounded depth circuits
- Parameters set according to circuit

String Matching

- Strings S_1 and S_2 of known length ℓ
- Homomorphic encryptions $\mathrm{Enc}(S_1)$ and $\mathrm{Enc}(S_2)$
- How to homomorphically compute $\mathrm{Enc}(S_1 == S_2)$?
  - Encrypt each bit $S_{1,1}, S_{1,2}, \ldots, S_{1,\ell}, S_{2,1}, S_{2,2}, \ldots, S_{2,\ell}$
  - Compute homomorphically $\prod_i \left[1 - \left(S_{1,i} - S_{2,i}\right)^2\right]$
  - In many cases too expensive for applications
  - Can still be practical if done right

- For better performance need different strategy
- Instead compute $\sum_i (S_{1,i} - S_{2,i})^2$
- Match indicated by 0, no match by non-zero
- Cheap to compute
  - Squaring is faster than multiplication
  - Sum very fast
  - Result less useful than in first approach
- This is the approach we will take in this talk
- Many approaches, each with pros and cons

Homomorphic Encryption

- Plaintext space $\mathbb{Z}_t \times \mathbb{Z}_t \times \cdots \times \mathbb{Z}_t$ (n factors)
- Suppose t prime and $2n \mid (t-1)$
- Suppose $2n \mid (t-1)$ so batching is supported
- $\sum_i (S_{1,i} - S_{2,i})^2$ can be evaluated if $t > \ell$

Improvements I

- Instead of encrypting S_1 and S_2 bit-wise encrypt them in chunks of integers mod a base b to make ℓ smaller (ℓ_b)
- $S_{1,1}, S_{1,2}, \ldots, S_{1,\ell_b}, S_{2,1}, S_{2,2}, \ldots, S_{2,\ell_b}$ now all integers mod b
- $\sum_i (S_{1,i} - S_{2,i})^2$ still works if $t > \ell_b \cdot b^2$
- Bigger t means more noise (bump up parameters?)
- $\ell_b < \ell$ makes evaluation faster, better message expansion, smaller encrypted query and dataset

- Want to compare one string X to many others quickly
  - Plaintext space is $\mathbb{Z}_t^n$ so can encrypt vectors
  - $X = X_1 X_2 \dots X_{\ell_b}$ compared to $Q_i = Q_{i,1} Q_{i,2} \dots Q_{i,\ell_b}$ for $i = 1 \dots N = n$
  - Each part an integer mod b
  - Batches

    $\overline{X_1} = (X_1, X_1, \dots, X_1)$, $\overline{X_2} = (X_2, X_2, \dots, X_2)$, …, $\overline{X_{\ell_b}} = (X_{\ell_b}, X_{\ell_b}, \dots, X_{\ell_b})$

    $\overline{Q_1} = (Q_{1,1}, \dots, Q_{n,1})$, $\overline{Q_2} = (Q_{1,2}, \dots, Q_{n,2})$, …, $\overline{Q_{\ell_b}} = (Q_{1,\ell_b}, \dots, Q_{n,\ell_b})$
  - Compute $\sum_i (\overline{X_i} - \overline{Q_i})^2$
  - If X matched Q_i then the i-th slot is 0, otherwise non-zero
- Good performance since $n \geq 4096$

n=4

1	0	1	1	0	0	0	0	1	0
1	0	1	1	0	0	0	0	1	0
1	0	1	1	0	0	0	0	1	0
1	0	1	1	0	0	0	0	1	0

Query

n=4

1	1	1	0	1	1	1	0	1	1
0	0	0	1	1	0	0	0	0	1
1	0	1	1	0	0	0	0	1	0
0	1	0	1	1	0	1	0	0	0

Dataset

What about datasets bigger than n?

- Suppose dataset has size $N = nB$
- $\overline{Q_{j,i}}$ denotes i-th ciphertext in j-th batch
- Query still same ℓ_b ciphertexts
- Encrypted dataset now $B \cdot \ell_b$ ciphertexts
- Need to compute

$$\prod_{j=1}^{B} \sum_{i=1}^{\ell_b} \left(\overline{X_i} - \overline{Q_{j,i}}\right)^2$$

- In result the i-th slot is 0 if an only if there was a match in any of the batches in the i-th position
- In many cases B is small (n is large) so product is not bad

n=4

1	0	1	1	0	0	0	0	1	0
1	0	1	1	0	0	0	0	1	0
1	0	1	1	0	0	0	0	1	0
1	0	1	1	0	0	0	0	1	0

Query

n=4

n=4

1	1	1	0	1	1	1	0	1	1
0	0	0	1	1	0	0	0	0	1
1	0	1	1	0	0	0	0	1	0
0	1	0	1	1	0	1	0	0	0
0	0	1	0	0	0	1	1	1	1
0	1	1	0	1	0	1	0	0	1
0	0	0	1	0	0	0	0	0	1
1	1	0	0	1	1	0	0	0	0

Dataset

- In practice we want to make $k > 1$ queries
- Can we get better amortized performance in this case?

- In practice we want to make $k > 1$ queries
- Can we get better amortized performance in this case?

Yes.

We use hashing techniques familiar from PSI protocols.

Permutation-Based Cuckoo Hashing

- Hash the dataset
- Permutation-based hashing encodes a part of the item in the bin index
- Shortened strings faster to compare after matching bins
- We know bin index from query so get less wasteful queries
- $X = X_L \,||\, X_R$ a string of length $\ell = \ell_L + \ell_R$
- $H: \{0,1\}^{\ell_L} \rightarrow \{0,1\}^{\ell_R}$ a hash function

- Hash table of size n (plaintext), suppose $N' < n$ items
- Insert X_L in bin with index $\mathrm{Loc}(X) = H(X_L) \oplus X_R$
- Let $\ell_R = \log_2 n$
- If two string inserted in same bin match, the strings match
- Need to insert items without any collisions

- d-cuckoo hashing
  - Consider d hash functions $H_1, H_2, \dots, H_d$
  - d location functions $\mathrm{Loc}_i(X) = H_i(X_L) \oplus X_R$
  - When inserting X, if $\mathrm{Loc}_i(X)$ is occupied, kick out previous value and re-insert with different hash function
  - Succeeds w.h.p. when n large enough compared to N'
  - Reasonable values $d = 3,4,5$, and $\frac{N'}{n} = 1.05 \ \dots 1.2$

Improvements II

- Suppose dataset is only $N' < n$ items
- d-cuckoo hash dataset to one hash table of size n
  - Fill empty bins with dummy values
  - Split strings $Q_{i,L}$ in bins into $\ell_{L,b} < \ell_b$ base b chunks
  - Encrypt ($\ell_{L,b}$ ciphertexts)
- Hash query string using all location functions (no cuckoo)
  - Cannot know in which location item was inserted so have to check them all
  - Fill empty bins with dummy values
  - Query hash table only has d items in it, other bins empty
  - Encrypt, and match

- Typically dataset has size $N \gg n$
- Let $B = N/N'$ and split dataset into B parts
- Let $\epsilon = n/N'$, where ϵ is typically between 1.05 and 1.2
- Use B completely separate hash tables
- Query still uses only one hash table, $\ell_{L,b}$ ciphertexts
- Use again

$$\prod_{j=1}^{B} \sum_{i=1}^{\ell_{L,b}} \left(\overline{X_i} - \overline{Q_{j,i}}\right)^2$$

- Encrypted dataset is B times bigger than encrypted query

Improvements III – Multiqueries

- Want to perform k queries
- Naively: Repeat above k times
- Instead of hashing only one string to query hash table, hash all k
- k-multiquery table has $k \cdot d$ non-empty bins
- Works if no collisions occur (can estimate probability)
- If collisions occur, might need to split to more tables
- For small k equally efficient as single queries
  - Computation and query size

Example

- iDASH Secure Genome Analysis Competition
- Problem 3: Testing for Genetic Diseases on Encrypted Genomes
- Private queries/multiqueries on encrypted dataset of mutations in genome (SNPs) stored in cloud
- Support for up to 100,000 mutations, 5-fold multiqueries
- Can realistic performance be achieved?

- After parsing files, each row 40 bits
- File sizes 10,000 rows and 100,000 rows
- Up to 5-multiqueries (but we can easily do much more)
- $d = 4$, $n = 8192$, results in items of length $\ell_L = 29$ bits
- $b = 2^{10}$ results in $\ell_{L,b} = 3$
- $N' = 7700$ gives $\epsilon = 1.064$ (hashing likely to succeed)
- $B_{small} = 2$ and $B_{large} = 13$

- We use SEAL v2.1 (Simple Encrypted Arithmetic Library)
- Easy-to-use homomorphic encryption library
- Developed at MSR since 2015
- Uses Fan-Vercauteren RLWE-based HE scheme
- Freely available at http://sealcrypto.codeplex.com
- Intel Xeon E5-1620 v3 @ 3.50 GHz with 16 GB RAM

Small example: ($N = 10{,}000$)

SEAL parameter	Value
Polynomial modulus	$x^{8192} + 1$
Ciphertext coeff modulus	$2^{155} - 2^{25} + 1$
Plaintext coeff modulus	3686401 (22 bits)
Decomposition bit count	78
Computational security level	Very high

Large example: ($N = 100{,}000$)

SEAL parameter	Value
Polynomial modulus	$x^{8192} + 1$
Ciphertext coeff modulus	$2^{253} - 2^{21} + 5 \cdot 2^{14} + 1$
Plaintext coeff modulus	3686401 (22 bits)
Decomposition bit count	52
Computational security level	≈ 100 bits [APS15]

Small example: $(N = 10{,}000)$

Operation	Time (ms)
Encoding dataset	115
Encrypting dataset	120
Encoding 5-multiquery	9
Encrypting 5-multiquery	60
Evaluating matching function	225
Decrypting response	15

Large example: $(N = 100{,}000)$

Operation	Time (ms)
Encoding dataset	1098
Encrypting dataset	956
Encoding 5-multiquery	9
Encrypting 5-multiquery	75
Evaluating matching function	2003
Decrypting response	19

Small example: $(N = 10{,}000)$

Data	Size (KB)
Original file (.VCF)	557
Parsed file	196
Encrypted dataset	2305
Encrypted 5-multiquery	1158
Encrypted response	386

Large example: $(N = 100{,}000)$

Data	Size (KB)
Original file (.VCF)	5490
Parsed file	1920
Encrypted dataset	19971
Encrypted 5-multiquery	1545
Encrypted response	515

Conclusions

- The scenario might actually be realistic and practical!
- Performance is surprisingly good
  - O(1 sec), 10x message expansion (can be improved)
  - But enabling more complicated queries reduces performance
- Performance-optimized libraries (e.g. NFLlib) will yield better results in larger examples (larger parameters)
- Similar ideas can be used for Private Set Intersection

Thank you for your attention

Contact: kim.laine@microsoft.com