Private Set Intersection from Homomorphic Encryption

Kim Laine
Cryptography Research Group
Microsoft AI & Research

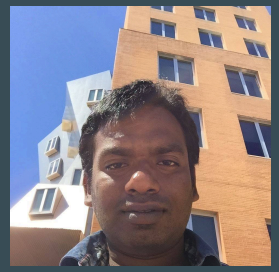
WAHC 2017: 5th Workshop on Encrypted Computing and Applied Homomorphic Cryptography
Sliema, Malta

Cryptography Research Group

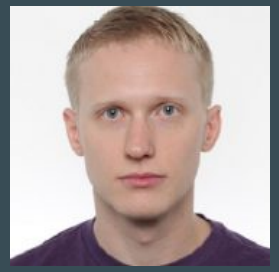
Kristin Lauter
Principal Researcher

Ran Gilad-Bachrach
Researcher

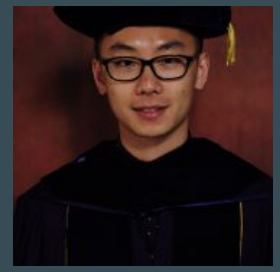
Melissa Chase
Researcher



Ranjit Kumaresan
Researcher



Kim Laine
Researcher



Hao Chen
Post-Doc Researcher

Intern Program

- Microsoft Research has an extensive internship program
- Again this year we will have 5 or 6 interns working on theory, code, and applications
- Internships mostly in the summer, but also throughout the year
- Let me know if you are interested in an internship!

Our Work

- Heavy focus on secure computation
- Tools
  - Homomorphic encryption
  - Garbled circuits
  - Differential privacy
  - Hardware methods
- Work on primitives, protocols, applications, implementations

Simple Encrypted Arithmetic Library

- SEAL project started in 2015
- Goals
  - Develop new library following good engineering practices
  - Written in C++11, includes .NET wrappers
  - Must be usable by non-crypto experts
  - No external dependencies
- Uses Fan-Vercauteren scheme
- Available at http://sealcrypto.codeplex.com
- Actively developed

Simple Encrypted Arithmetic Library

- Security based on RLWE
- Plaintext space $\mathbb{Z}\downarrow t\,[x]/(x\uparrow n+1)$, restrict to n a power of 2
- Ciphertext space $\mathbb{Z}\downarrow q\,[x]/(x\uparrow n+1)\times\mathbb{Z}\downarrow q\,[x]/(x\uparrow n+1)$ (q typically hundreds of bits)
- n fixed: smaller q more secure, larger less secure
- Noise growth in multiplication $\propto t$, noise ceiling $\approx q/t$
- Want small t to get deeper circuits with fixed q
- Note that increasing q requires increasing n for security
- Want large t to increase the size of plaintexts

Simple Encrypted Arithmetic Library

- Choose t a prime such that $2n|(t-1)$
- Then $x\uparrow n+1$ splits completely modulo t, and CRT says

$\mathbb{Z}\downarrow t\,[x]/(x\uparrow n+1) \cong \mathbb{Z}\downarrow t \times \ldots \times \mathbb{Z}\downarrow t$

- Ring isomorphism implies that both addition and multiplication are respected
- As long as t is large enough can do n-fold SIMD operations
- Often n is very large (at least 1024)
- SEAL implements batching in PolyCRTBuilder class
- Question: How to use this efficiently?
  1. Do same computation many times to improve amortized performance (easy)
  2. Use as a tool when performing one computation involving parallel components (non-trivial)

iDASH Competition

- iDASH Secure Genome Analysis competition in 2016
- Query encrypted dataset of genetic mutations
- 40 bit strings
- Up to $100{,}000$ mutations (can easily scale to much larger)
- Message expansion 10-12x
- Improved amortized performance for many queries
- Running time $\approx 2\ sec$ on 100,000 size dataset
- Know many ways of significantly improving performance from this

Punchline: *Surprisingly good performance for a dataset query task!*

Private Set Intersection

- iDASH task close to Private Set Intersection with asymmetric set sizes
- Could homomorphic encryption work well for PSI?
- More challenging security model, so big changes needed
- Is there any hope in competing with state-of-the-art OT extension –based PSI protocols?
- Example [Pinkas et al., 2016]:
  - On Intel Xeon CPU E5-2699 v3 @ 2.3GHz and 256GB of RAM
  - 16M and 4096 intersection in 40s computation (single-threaded)
  - 480MB of communication
  - Symmetric set size case has very similar performance

Ideal Functionality

- "Sender" holds set X of size $N{\downarrow}X$ (item length σ)
- "Receiver" holds set Y of size $N{\downarrow}Y$ (item length σ)
- PSI protocol outputs $X \cap Y$ to receiver
- Receiver learns nothing else; sender learns nothing

Basic Protocol

- Suppose plaintext space is $\mathbb{Z}_t$, where t is prime
  - E.g. use constant coefficients of plaintext polynomials

1. Sender and receiver agree on an FHE scheme
2. Receiver encrypts each element in its set, and sends them to sender
3. Sender receives $c_1, \ldots, c_{N_Y}$
4. Sender samples random non-zero $r_i \in \mathbb{Z}_t$ for each c_i
5. Sender homomorphically evaluates $r_i \prod_{x \in N_X} (c_i - x)$ and sends them to receiver
6. Receiver decrypts the ciphertexts: a value of 0 indicates a match

Computational cost is $O(N_X N_Y)$ (very bad); communication $2N_Y$ ciphertexts (also very bad due to FHE message expansion)

First Improvement: Batching

- In many FHE schemes plaintext space is $(\mathbb{Z}_t)^n$, where t is prime and chosen appropriately

1. Sender and receiver agree on an FHE scheme
2. Receiver encrypts its elements into N_Y/n ciphertexts, and sends them to sender
3. Sender receives $c_1,\ldots,c_{N_Y}$
4. Sender samples a random vector $r_i \in (\mathbb{Z}_t)^n$ with non-zero entries for each c_i
5. Sender homomorphically evaluates $r_i \prod_{x \in N_X} (c_i - x)$ and sends them to receiver
6. Receiver decrypts the ciphertexts: a value of 0 indicates a match

This has reduced both communication and computation by a factor of n

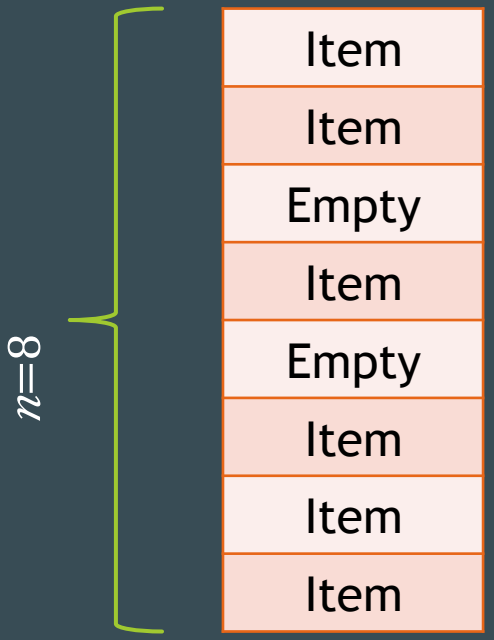
Second Improvement: Hashing

1. Sender and receiver agree on an FHE scheme and hash function
2. Receiver hashes its items into a hash table of size n with no collisions, encrypts the bins into one single ciphertext, and sends it to sender
3. Sender receives c
4. Sender hashes its set into a hash table of size n (collisions allowed), bin size B
5. Sender batches "columns" of its table into B plaintexts $x↓1$,..., $x↓B$
6. Sender samples a random vector $r∈(ℤ↓t)↑n$ with non-zero entries
7. Sender homomorphically evaluates $r∏i=1↑B▒(c−x↓i)$ and sends it to receiver
8. Receiver decrypts the ciphertext: a value of 0 indicates a match

Communication essentially reduced by factor of n (only one ciphertext);
computation B (expectation value $N↓X/n$)

Second Improvement: Hashing

Receiver's hash table



Sender's hash table

Item	Item	Empty	Empty	Empty	Empty	Empty
Item	Item	Item	Empty	Empty	Empty	Empty
Item	Item	Item	Item	Item	Empty	Empty
Item	Item	Item	Empty	Empty	Empty	Empty
Item	Item	Item	Empty	Empty	Empty	Empty
Item	Item	Item	Item	Item	Empty	Empty
Empty	Empty	Empty	Empty	Empty	Empty	Empty
Item	Item	Item	Item	Item	Item	Empty

$B=7$

Second Improvement: Hashing

- In practice we use permutation-based cuckoo hashing with 3 hash functions
  - Very good fill rate
  - No collisions on receiver's side
  - Encodes part of strings in bin index → shortens items
  - Very well known technique in PSI world

Second Improvement: Hashing
Cuckoo hashing

- Cuckoo hashing uses several hash functions (e.g. 3 or 4)
- Item x is hashed as $H{\downarrow}i\,(x)$ for some i
- If this bin is empty:
  - Insert item to bin
- If this bin is not empty:
  - Kick out previous value y from bin
  - Insert x to bin
  - Re-insert y using $H{\downarrow}j\,(y)$ for some random $j{\neq}i$
- Sender needs to hash with all hash functions; only receiver uses cuckoo hashing

This results in very high fill-rate, and success with probability that can be estimated.

Second Improvement: Hashing
Permutation-based hashing

- For simplicity, consider hashing scheme with only one hash function H
- To hash item x, split it as $x\downarrow L \,||\, x\downarrow R$
- Insert $x\downarrow L$ at location determined by $H(x\downarrow L)\oplus x\downarrow R$
- If two insertion coincide in a particular location, the items are equal
- This shortens string lengths from σ bits to $\sigma - \log\downarrow 2\, n$ bits

- With cuckoo hashing, need to append hash function index to inserted string
- For security, need to ensure hashing succeeds except with negligible probability

Essentially, a part of the strings is encoded in the hash table bin index, effectively reducing the length of the items (better performance for FHE).

Second Improvement: Hashing

Hashing to smaller representation

- If items are very long, can start by hashing to short enough representation
- Need to ensure no collisions occur in hashing (birthday paradox)
- If can do $2 \log\downarrow 2 (N\downarrow X + N\downarrow Y) + \lambda - 1$ length items (λ statistical security parameter) then can do arbitrary length items

Second Improvement: Hashing

- Circuit has depth $\log\downarrow 2\ B$, which can still be unfortunately bad (perhaps 4-32)
- What is the best way to evaluate polynomial $r\prod i=1\uparrow B\blacksquare(c-x\downarrow i\ )$?

Evaluating $r\prod i=1\uparrow B\,(c-x\downarrow i\ )$

- Expand as $rc\uparrow B+ra\downarrow B-1\ c\uparrow B-1+\ldots+ra\downarrow 1\ c+ra\downarrow 0$
- Here $a\downarrow i=\mathrm{Sym}\downarrow i\,(x\downarrow 1\ ,x\downarrow 2\ ,\ldots,x\downarrow B\,)$ are elementary symmetric polynomials in the sender's plaintext data
- Sender pre-computes $ra\downarrow i$ and stores them (receiver-independent operation)
  - Not *true* offline pre-processing: for security proof hash functions need to be chosen together with receiver, and used only once
- Computing elementary symmetric polynomials is easy
- In an online phase sender still needs to compute $c, c\uparrow 2\ ,c\uparrow 3\ ,\ldots,\ c\uparrow B$
  - Depth $\log\downarrow 2\ B$

Third Improvement: Windowing

- Easy to improve evaluation of $r\prod i=1\uparrow B\blacksquare(c-x\downarrow i\ )$ by increasing communication
- Receiver sends $c, c\uparrow 2\ , c\uparrow 4\ , \ldots, c\uparrow\lfloor \log\downarrow 2\ B\rfloor$ to sender instead of just c
- Increases communication R→S to $\log\downarrow 2\ B$
- Circuit depth goes down to $\log\downarrow 2\ \log\downarrow 2\ B$ (perhaps 2-5)

- More generally use "windowing" (window size ℓ)
- Receiver sends encrypted powers $c\uparrow i\cdot 2\uparrow j\ ,\ i=1,\ldots,2\uparrow j-1$, and $j=0,\ldots,\log\downarrow 2\uparrow\ell\ B$
- Reduces depth to $\log\downarrow 2\ \ log\downarrow 2\uparrow\ell\ B=\log\downarrow 2\ \log\downarrow 2\ B\ -\log\downarrow 2\ \ell$
- Communication from receiver to sender increases to $2\uparrow\ell-1/\ell\ \log\downarrow 2\ B$ (substantial!)
- Still need to compute B powers of c
  - Multi-threading non-trivial

Fourth Improvement: Partitioning

- When the sender's dataset is very large, it could make sense to partition it into several smaller datasets, and run PSI per each partition
- Sender hashes as usual, but splits has table (vertically) into α partitions
- Each partition contains B/α columns of the hash table
- Sender runs PSI for each column separately, and return α results to receiver
- Instead of evaluating $r\prod_{i=1}^{B}(c-x_i)$, now evaluate $r_j\prod_{i=1}^{B/\alpha}(c-x_i^{(j)})$
  - $x_i^{(j)}$ denotes the i-th column vector (plaintext polynomial) in the j-th partition
- Huge benefit from multi-threading
  - Each partition can be evaluated independently as soon as B/α powers of c are computed
- Circuit depth decreases further to $\log_2\log_2(B/\alpha)-\log_2\ell$
- Communication S→R increases to α ciphertexts (big! α could be e.g. 4-512)

Fifth Improvement: Modulus Switching

- Naively, big communication prevents big α. But …
- Since only decryption is done after communication S→R, can modulus switch to very small modulus before sending
- It now makes sense to take α quite large!
- Large α is very good for performance, multi-threading
- Large α is OK for communication
- Large α decreases depth (combine with smaller window size; better R→S)
- Hope to get depth 2-3

Further Difficulties: Function Privacy

- Unfortunately, IND-CPA security is not enough to protect sender's privacy and prove simulation-based security
- Noise growth in many schemes (FV in our implementation) depends on input plaintexts
- Need to have *function privacy* to protect sender's data
- To do this:
  1. Sender adds encryption of 0 to re-randomize output ciphertexts
  2. Sender floods the noise; need to upper-bound noise
  3. Sender's privacy then based on standard "smudging lemmas"
- Semi-honest security model is unfortunately critical

Implementation

- We implemented our protocol using SEAL v2.1 (experimental branch)
- ≥ 120 bit computational security level, 40 bit statistical security level
- Achieves significantly lower communication than [Pinkas et al. 2016]
- Much better advantage from multi-threading than [Pinkas et al. 2016], even when receiver single-threaded
- Only sender multi-threaded; receiver single-threaded
- On Intel Xeon CPU E5-2699 v3 @ 2.3GHz and 256GB of RAM

Results: Our Protocol vs. [Pinkas et al. 2016]

Our protocol

Parameters		Comm.	Total time (seconds)							
			10 Gbps		100 Mbps		10 Mbps		1 Mbps	
N_X	N_Y	Size (MB)	$T=1$	4	1	4	1	4	1	4
2^{24}	11041	23.2, †21.1	115.4	40.3	117.8	42.7	134.4	59.3	†290.8	†215.1
	5535	20.1, †12.5, ‡11.0	105.2	34.8	107.2	36.7	†120.3	†45.8	†211.1	‡132.7
2^{20}	11041	11.5	12.8	5.7	14.0	6.9	22.2	15.1	105.4	98.3
	5535	5.6	8.6	3.3	9.2	3.9	13.3	8.0	53.6	48.3
2^{16}	11041	4.1, †4.4	3.0	†1.7	3.4	†2.1	6.4	†5.3	36.0	35.0
	5535	2.6	1.8	0.9	2.0	1.2	3.9	3.1	22.5	21.7

[Pinkas et al. 2016]

Parameters		Comm.	Total time (seconds)							
			10 Gbps		100 Mbps		10 Mbps		1 Mbps	
N_X	N_Y	Size (MB)	$T=1$	4	1	4	1	4	1	4
2^{24}	11041	480.9	40.5	23.3	88.0	66.4	449.5	427.5	4084.8	4067.2
	5535	480.4	40.1	23.1	87.9	65.5	449.2	427.3	4080.6	4064.3
2^{20}	11041	30.9	3.3	2.1	7.0	5.6	29.8	28.3	263.7	262.1
	5535	30.4	3.1	2.0	6.8	5.0	29.0	27.9	260.0	259.6
2^{16}	11041	2.6	0.7	0.6	1.5	1.4	3.3	3.1	21.6	22.1
	5535	2.1	0.7	0.6	1.4	1.3	2.9	2.8	19.8	21.3

Next Steps

- Extend to longer strings by increasing FHE parameters; is there a better way?
- Get rid of function privacy requirement using hybrid scheme
  - But current approach is not too bad!
- What other advanced crypto primitives could be implemented efficiently with homomorphic encryption?

Thank You!

kim.laine@microsoft.com

Cryptography Research Group: https://www.microsoft.com/en-us/research/group/cryptography-research/
Homomorphic Encryption library SEAL: http://sealcrypto.codeplex.com