

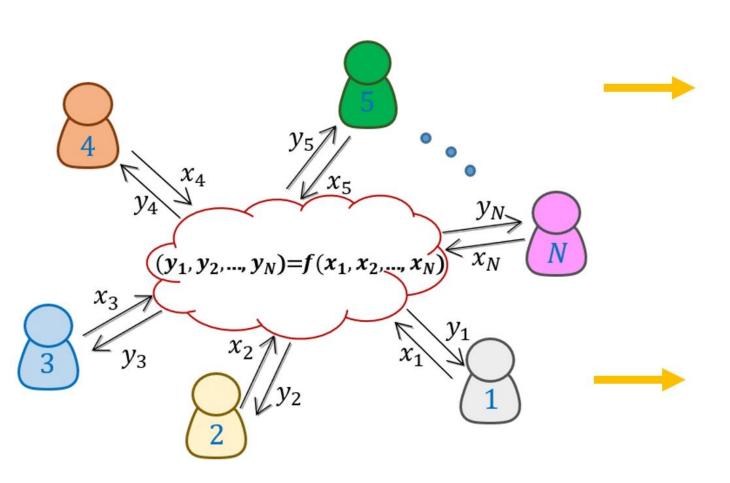
ECE 209AS (03/19/2020) Final Presentation

P21: Privacy Preserving Inferencing for Medical Cyberphysical Systems

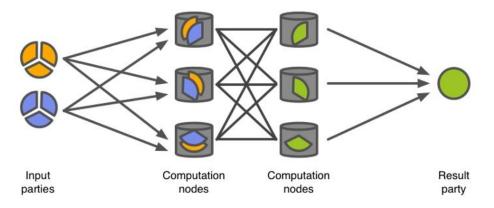
Team Members: Swapnil Sayan Saha, Vivek Jain and Brian Wang



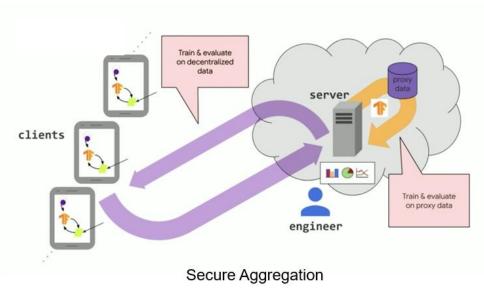
Problem Statement and Importance



Perform collaborative computation without revealing raw data



Secure Multi-Party Computation



Problem Statement and Importance (contd.)

Protocols not tailored for edge computing

Not scalable for large-scale sensor networks

Current Approach

Dependence on TTP services

Scalable with low computational overhead

Desired Approach No external cloudservices or TTP

Difficult to integrate in existing systems

SA and SMPC protocols running on the edge

Solves
integrability
and
scalability
issues

Overall Project Goals and Specific Aims

Specific Aims

Implement state-of-the-art SMPC and SA protocols in small-scale virtual MCPS, with computation occurring at the edge.

Benchmark standard performance, privacy and security metrics of implemented SMPC and SA.

Tune the parameters of state-of-the-art SMPC and SA protocols, focusing on reducing computational overhead.

Implement a scalable and robust starexchange MCPS topology embracing SMPC protocols void of centralized cloud inferencing.

Deliverables / Goals

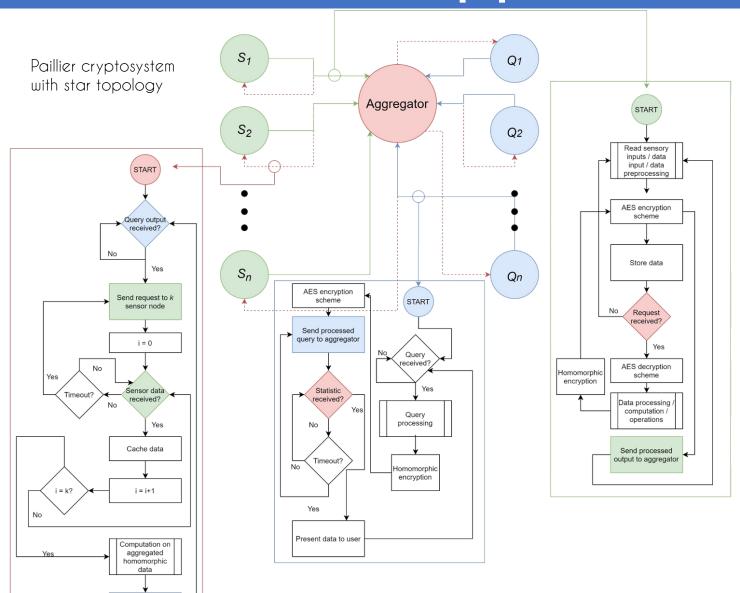
Benchmark and tune SMPC and SA protocols for resource-constrained settings

A real-time scalable and robust privacy preserving inferencing system at the edge for MCPS

Attack Model

Semi-honest Adversary: All aggregator participating nodes (semi honest/ malicious) External Sensor nodes malicious SA **SMPC** try to spy on adversary at other sensors' any point in the data system Zero-knowledge Similar assumptions as proofs for in secure authentication of Adversary: aggregation nodes and statistics Query nodes and aggregator

Technical Approach (Secure Agg.)



Key Generation:

$$pk = (n, g)
 - n = pq, GCD(pq, (p-1)(q-1)) = 1
 - g ∈ ℤn²*
 sk = (λ, μ)$$

$$sk = (\lambda, \mu)$$

$$- \lambda = LMC(p-1, q-1)$$

$$- \mu = (\frac{g^{\lambda} \mod n^{2}-1}{n})^{-1} \mod n$$

Encrypt message into ciphertext:

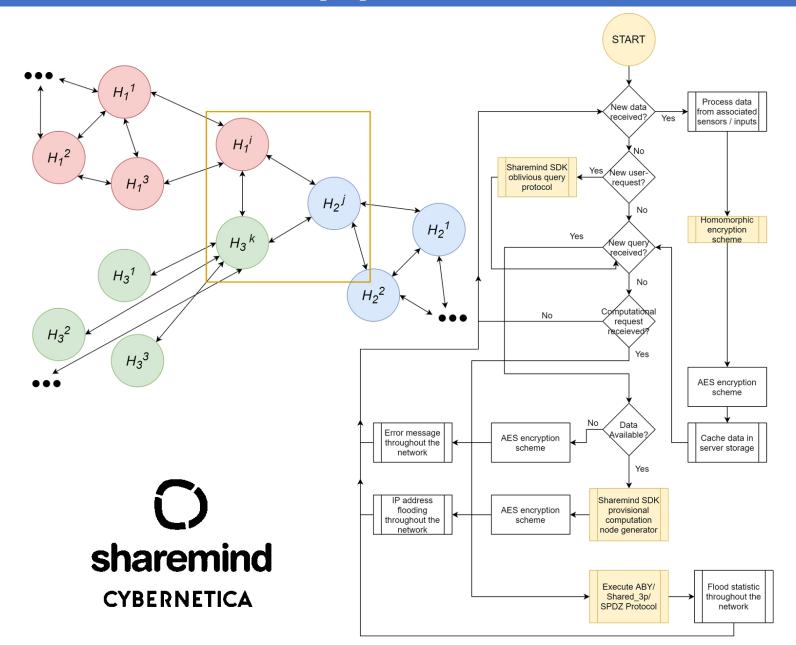
•
$$c = g^m \cdot r^n \mod n^2, r \in \mathbb{Z}_n$$

Decrypt ciphertext into message:

Homomorphic property:

$$D(E(m_1,r_1)\cdot E(m_2,r_2) \bmod n^2) = m_1+m_2 \bmod n. \ D(E(m_1,r_1)\cdot g^{m_2} \bmod n^2) = m_1+m_2 \bmod n. \ D(E(m_1,r_1)^k \bmod n^2) = km_1 \bmod n. \ D(E(m_1,r_1)^k \bmod n^2) = km_1 \bmod n.$$

Technical Approach (SMPC)



Technical Approach (SMPC)

SPDZ:

- An input $a \in \mathbb{F}_{p^k}$ is represented as $< a > = (\delta, (a_1, ..., a_n), (\gamma(a)_1, ..., \gamma(a)_n)),$ a_i is a share of a and $\gamma(a)_i$ is the MAC share authenticating a under a SPDZ global key a (not revealed until end). Player i holds a_i , $\gamma(a)_i$ and δ is public.
- Correct SPDZ execution: $a = \sum_i a_i$, $\alpha(a + \delta) = \sum_i \gamma(a)_i$
- Two phases offline: generates precomputed values (independent of the function); online: executes designated function using the values.

Compiler Bytecode VM (online) Program Output

ABY

- Arithmetic, binary and Yao 3PC
- 3PC with secret sharing for privacy preserving machine learning and database joins (PSI, Union, etc.); secure against semi-honest adversaries;
- Randomly goes back and forith between A, B and Y.

 $[[x]]^A \rightarrow [\overline{[x]}]^B$

 $[[x]]^A \leftarrow [\overline{[x]}]^B$

Yao Garbled

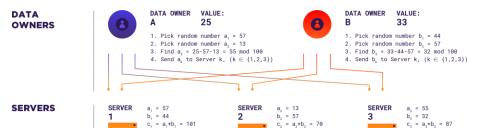
Circuit

Arithmetic

Share

Shared3p

- Sharemind's proprietary MPC.
- 3PC with secret sharing; secure against semi-honest adversaries
- Uses the additive secret sharing scheme in the ring $Z_2(32)$.





ANALYST X

Analyst calculates c = 1+79+87 = 158 = 58 mod 100

Analyst learns, that the sum of A's and B's score is 58 without learning the scores of either of them



Share
$$[[x]]^A = \sum_i x_i$$

$$[[x]]^B = x_1 \oplus x_2 \oplus x_3$$

$$[[x]]^Y = LSB(x_1 \oplus x_2)$$

$$x = x_1 + x_2 + x_3$$

Technical Approach (oblivious functions)

k = 0:

Implemented SA operations (Language: Python)

Mean	Convolution	Linear Regression	
Vector Sum			

Implemented SMPC operations (Language: SecreC)

Shuffle	Quicksort	Outer join		
Union	Intersection	MAD		
Mean	Median	Upper Quantile		
Lower Quantile	Minimum	Maximum		
StdDev	Variance	Vector Sum (VS)		
Outlier_MAD Outlier_Quantile Linear Re		Linear Regression		
Obv_Insert				

^{*} mean implemented for all 3 protocols

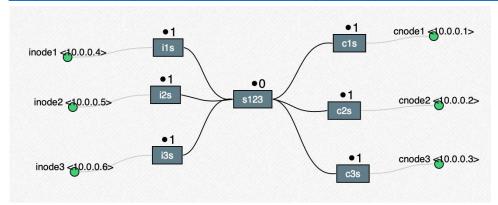
```
time
                 public_key, private_key = paillier.generate_paillier_keypair(n_length=1024)
                 elapsed_total_add = 0
                                                                             c in range(n_clients):
                 elapsed_total_mult = 0
                                                                             X.append(data[step * c: step * (c + 1)])
                      i in range(100):
                                                                         key_length = 1024
                      a = np.random.randint(0, high=256)
                      b = np.random.randint(0, high=256)
                      encrypted_a = public_key.encrypt(a)
                                                                         class Client:
                      encrypted_b = public_key.encrypt(b)
                      start time = time.process time()
                           j in range(10000):
                                                                                  self.pubkey = pubkey
                                                                                  self.privatekey = privatekey
                            adder = a+b
                                                                                  self.X = X
                      elapsed = (time.process_time() - st
                      elapsed_total_add += elapsed
                                                                              def mean(self):
                                                                                          np.mean(self.X)
                      start time = time.process time()
                           j in range(10000):
                           mult
                                   = a * b
                                                                                          self.pubkey.encrypt(self.mean())
                                   (time.process_time() - st
                      elapsed :
                                                                              def decrypt_mean(self, mean_server):
                                                                                          self.privatekey.decrypt(mean server)
 //secure union and intersection using oblivious functions
  pd_shared3p float64[[1]] intersectAB(size(joinAB));
                                                                                              f.pubkey.encrypt(np.sum(np.square(self.X)))
 uint k = 0;
▼ for(uint i = 0; i < size(sharedA); ++i){</pre>
    for(uint i = 0; i < size(partyB); ++i){
                                                                                              .privatekey.decrypt(mean_server)
         pd_shared3p bool[[0]] truecond = true;
         pd_shared3p bool[[0]] falsecond = false;
         pd_shared3p bool[[0]] cond = choose(sharedard --
         if(declassify(cond)){
             intersectAB[k] = sharedA[i];
                                                                                                test1.sc - Ot Creator
             k++:
                                                                                            /* Secure MPC Emulation Demo. (c) 2020 Swapnil Savan Saha */
  intersectAB = intersectAB[0:k];
                                                                                            import shared3p;
  pd_shared3p float64[[1]] unionAB(size(joinAB));
                                                                                            import shared3p random
                                                                                            import shared3p sort;
                                                                                             import shared3p_statistics_summary;
                                                                                             import shared3p_statistics_outliers;
for(uint i = 0; i <size(sharedA); ++i){</pre>
                                                                                             import shared3p_statistics_distribution;
      unionAB[k] = sharedA[i];
                                                                                             import shared3p_statistics_regression;
                                                                                             import oblivious;
                                                                                             import shared3p_oblivious;
▼ for(uint i = 0; i <size(partyB); ++i){</pre>
      f = 0;
                                                                                            domain pd_shared3p shared3p;
      for(uint j=0; j<size(sharedA); ++j){</pre>
                                                                                            domain pd spdz fresco spdz fresco:
         pd_shared3p bool[[0]] truecond = true;
pd_shared3p bool[[0]] falsecond = false;
                                                                                            domain pd aby aby:
          pd_shared3p bool[[0]] cond = choose(party
                                                                                            //secure user defined function to calculate vector sum
                                                                                            template<domain D : shared3p, type T>
          if(declassify(cond)) {
                                                                                        21 T D T[[1]] vecSum(D T[[1]] x, D T[[1]] y,D T[[1]] z){
             f = 1;
                                                                                                return sqrt((x*x)+(y*y)+(z*z));
                                                                                                 000001, 62.611, -7.92}
                                                                                       -777, 62.611, -7.92}
                                                                                     Process returned status: 0
                                                                                     Estimated running time: 46312685 microseconds (46 seconds)
                                                                                      (This is the estimated running time of the program on the Sharemind Application Server, running on a
```

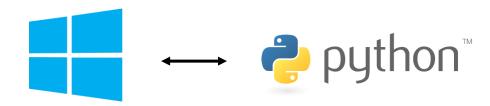
Experimental Setup

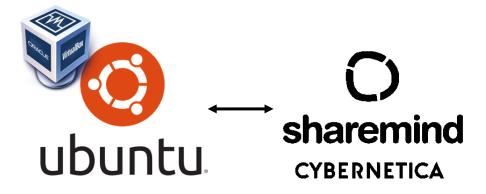
Preliminary Benchmarking and Prototyping:

SDN Narmox Spear - Mininet

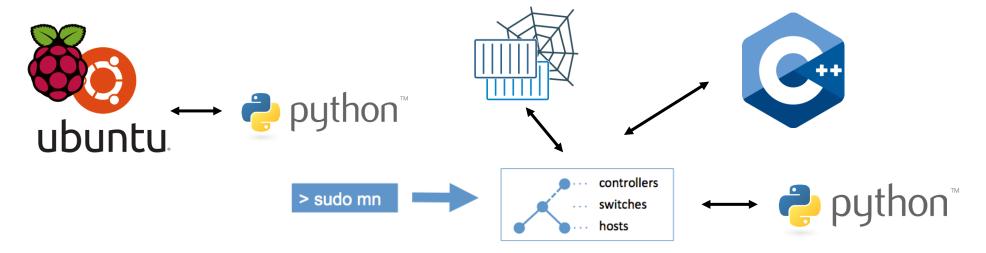
http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet







Real-time Benchmarking/Implementation



Success Metrics

Low latency and execution time of PPI system in resource-constrained setting

Integrability of code with existing system

No compromise in security for performance

O need for trusted third party

Implementation / Demo

Real-time secure aggregation demo on Mininet:

https://www.youtube.com/watch?v=DHPKwDjj1ag

Real-time SMPC demo on Mininet:

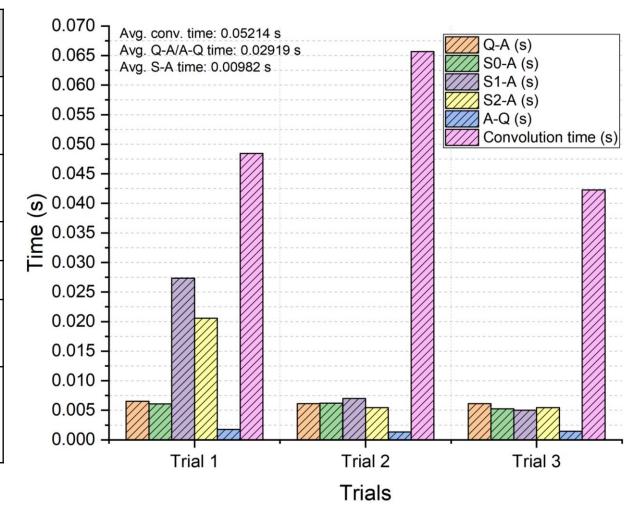
https://www.youtube.com/watch?v=t_OHudujrkc

Key Findings (SA)

SA standalone benchmark metrics:

Parameter	17-6700 HQ, 16 GB RAM	Raspberry Pi 4
Memory Usage	~ 8 Mb	~ 36 Mb
CPU Usage	17.6%	99.7%
Key Generation (mS)	259.37	105.66
Encryption (mS)	13.75	40.01
Decryption (mS)	15.63	12.32
Scalar Addition (nS)	46.88	336.33
Scalar Multiplication (nS)	109.38	374.01

SA Real-time benchmark metrics:



Key Findings (SMPC)

SMPC Standalone benchmark metrics:

Preliminary benchmark results* (-microseconds (CPU usage)):

	ABY	Shared 3p	SPDZ Fresco
Scalar Addition (+ encryption)	11373 (9%)	104 (11%)	18149 (10%)
Scalar Multiplication (+ encryption)	8055 (11%)	397 (14%)	25685 (10%)

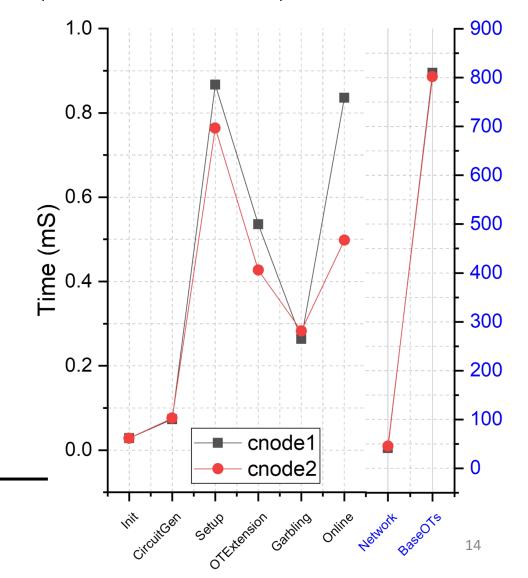
^{*} on single core AMD64 architecture, 1 GB RAM (RAM usage: 173 MB)

Total: 1.73 seconds

cnode 1: 0.854 seconds

cnode2: 0.849 seconds

ABY SMPC realtime benchmark metrics (Millionaire's Problem):



Key Findings (Overall)

SA suitable for real-time low-resource computations (e.g. queries)

Negligible network latency

Overall Findings

MPC suitable for offline high-resource computations (e.g. ML)

Security vs performance: Shared3p vs SPDZ

Prior Work and Relative Novelty

Work	Functionality	n-party?	Malicious security?	Practical
Nikolaenko et al. [60]	ridge regression	no	no	_
Hall et al. [45]	linear regression	yes	no	_
Gascon et al. [38]	linear regression	no	no	_
Cock et al. [21]	linear regression	no	no	_
Giacomelli et al. [39]	ridge regression	no	no	_
Alexandru et al. [5]	quadratic opt.	no	no	_
SecureML [58]	linear, logistic,	no	no	_
Shokri&Shmatikov [70]	deep learning deep learning	not MPC (heuristic)	no	-
Semi-honest MPC [7]	any function	yes	no	
Malicious MPC [28, 41, 11,	any function	yes	yes	no
Our proposal, Helen: regular	ized linear models	yes	yes	yes

Our work

- Zheng et al. [1] hypothesized that it is not possible to achieve robust and practical MPC using existing state-of-the-art protocols on the edge.
- Helen requires powerful server-class machines to operate.

Our benchmark shows that it is possible to solve classical MPC (and SA) problems and queries on resource-constrained edge devices using existing state-of-the-art MPC (and SA) protocols.

[1]. Zheng, Wenting, et al. "Helen: Maliciously secure coopetitive learning for linear models." 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019.

Prior Work and Relative Novelty

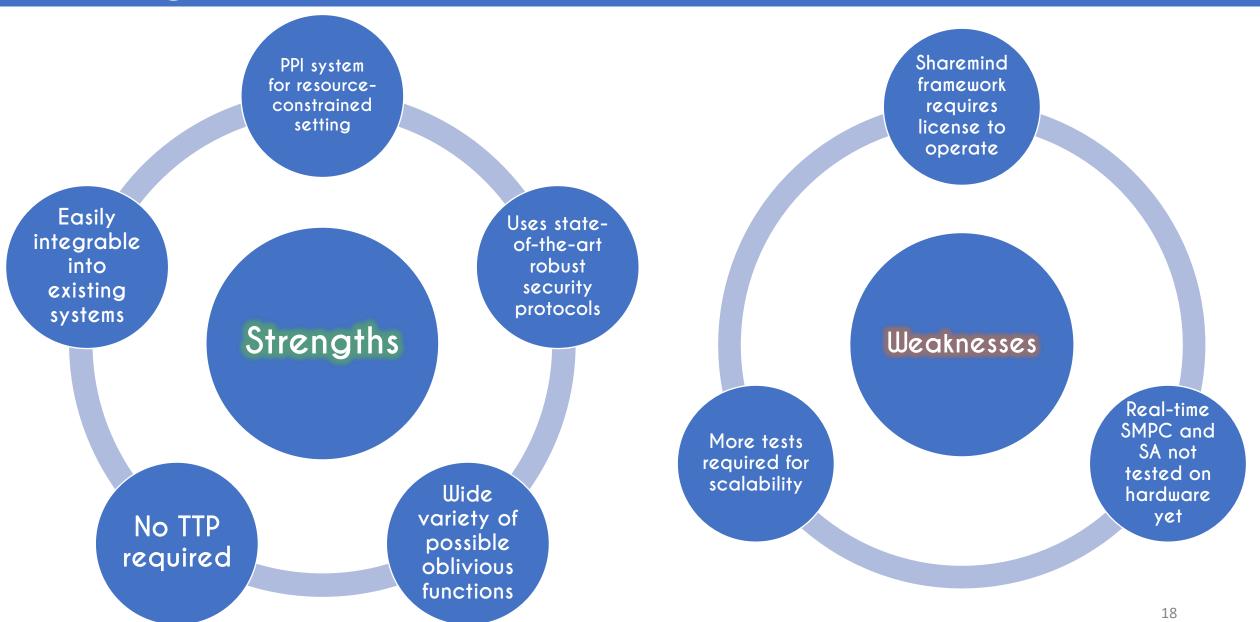
- Several MPC and SA architectures have been proposed in literature for MCPS (populated in website)
- All of the proposals, while secure from an information-theoretic view, require some notion of cloud services or external trusted third party, breaking Lindell's [1] recommendations.

"...the key challenge in secure MPC is computational resource, and common errors in secure MPC include assuming semi honest behavior precluding collusions, input dependent flow, deterministic encryption and having a false notion of the absolute actions an adversary may take (rather than mathematical proof)..."

Our proposed architectures do not require any external party in the pipeline, yet achieving collaborative computing goals.

[1]. Lindell, Yehuda, and Benny Pinkas. "Secure Multiparty Computation for Privacy-Preserving Data Mining." *Journal of Privacy and Confidentiality* 1.1 (2009).

Strengths, Weaknesses and Future Directions



Secure MPC

Member Contributions



Swapnil Sayan Saha

- Overseeing website/GitHub repo.
- Survey of literature pertinent to the fundamentals and latest advances in SMPC, with applications in MCPS.
- Implementation and preliminary benchmarking of basic SMPC protocols and custom oblivious functions in computer simulation.
- Implementation, dependency installation and benchmarking of SMPC protocols in Raspberry Pi hardware simulation environment.
- Tuning SMPC protocols for resource-constrained environments.



CUR

Vivek Jain

- Survey of literature pertinent to application of secure aggregation and SMPC for loT sensor networks and MCPS
- Implementation and preliminary benchmarking of basic secure aggregation protocols and custom oblivious functions in computer simulation.
- Benchmarking secure aggregation protocols (in real-time) in Raspberry Pi environment and Mininet.
- Handling networking mechanisms in Mininet.
- Tuning secure aggregation protocols for resourceconstrained environments.



Brian Wang

- Survey of literature pertinent to application of secure multiparty computation in medical cyberphysical systems and clinical decision support systems.
- Formulating SMPC and aggregation architecture for MCPS.
- Implementation of Mininet software simulation.
- Implementation and benchmarking (real-time) of SMPC and secure aggregation protocols in Mininet and Raspberry Pi.
- Handling networking mechanisms in Mininet and Raspberry Pi 4.



THANK YOU

https://github.com/swapnilsayansaha/BVSece209as/https://swapnilsayansaha.github.io/BVSece209as/

