## Privacy-Preserving Systems (a.k.a., Private Systems)

### **CU Graduate Seminar**

Instructor: Roxana Geambasu

1

# Private Collaborative Learning

- Secure multi-party computation
  - Federated learning

## **Private Collaborative Learning**

### What If No Central Aggregation of Data?



### What If No Central Aggregation of Data? (cont.)



### What If No Central Aggregation of Data? (cont.)



## **Case 1: Money Laundering Detection**

- Banks want to detect money laundering using machine learning.
- Criminals conceal illegal activities across many banks.
- Banks want to jointly compute a model on customer transaction data, but cannot share data.





## Secure Multiparty Computation

- Parties emulate a trusted third party via cryptography.
- No party learns any party's input beyond the final result (trained model).
- Performance is a challenge, but for simple computations (such as computing linear models) and few parties (up to 10), this is practical.



## Case 2: Text Autocomplete

- Want to train a text
  - autocomplete model on many users' data but don't want to collect users' data in a central location.
- Each user trains a local, partial model, and then the cloud combines these models into a global model, which it ships back to the clients.

E	0
Existing	
Values	
Autocomplete	
Widget	

## **Federated Learning**

- Your phone personalizes the model locally, based on your usage (A)
- Many users' updates are aggregated (B) to form a consensus change (C) to the shared model
- The procedure is repeated as new data becomes available



## **Existing Systems**

MPC and FL are both practical. Here are a couple (of multiple!) example offerings:

- Inpher's XOR Secret Computing
- Google's <u>Tensorflow Federated</u>

### **Cited References**

(Yao82) Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In 23rd Annual Symposium on Foundations of Computer Science, 1982.

Private Collaborative Learning

### The End

### MPC Details and Demo

by Pierre Tholoniat

#### Introduction

#### • General MPC setting

- Multiple parties with private inputs
- Emulate a trusted party to compute a function on their inputs
- Without revealing anything else than the output
- How do MPC protocols work? How practical are they?
  - Pretty informal presentation
  - See the Pragmatic MPC textbook [1] and other references for details and proofs

#### Introduction

Two main threat models:

- Honest-but-curious adversary
  - Corrupt parties follow the protocol, but try to learn as much as they can
  - A.k.a passive or semi-honest adversary
- Malicious adversary
  - Corrupt parties can deviate from the protocol arbitrarily
  - A.k.a active adversary

- Today, we consider an honest-but-curious adversary
  - Simple setting to show essential techniques
  - Protocols can be converted from passive to active security

#### Outline

- 1. Shamir Secret Sharing
- 2. Evaluating Arithmetic Circuits with the BGW Protocol
- 3. MPC with Preprocessing: Beaver Triples
- 4. Implementation: Meta's Private Computation Framework

### 1. Shamir Secret Sharing

Shamir, 1979 [8]

Setting:

- *n* parties, threshold  $t \le n$
- A global secret  $y \in K := F_p$  is shared among parties
- Each party i has a share  $y_i$
- Notation for a sharing of y:  $[y] := (y_1, ..., y_n)$

Desired properties:

- Knowing  $k \ge t$  shares is sufficient to reconstruct y
- Knowing *k* < *t* shares doesn't reveal anything about *y*

#### How can secret-sharing be useful?

Example: secret key recovery

- Split your wallet key into n=5 backups servers
- Reconstruct the key from t servers when needed
- If t=1, a single corrupted server can steal your key
- If t=5, a single faulty backup prevents you from recovering your key
- If t=3, resilient against 2 corrupted colluding servers and 2 failures

We can also use secret-sharing for arbitrary MPC

#### Construction with polynomials

Lagrange interpolation:

- Fact: the only polynomial of degree  $\leq t-1$  with *t* roots or more is zero
- Consequence: any polynomial  $P \in K_{t-1}[X]$  is uniquely characterized by the list of coordinate pairs (P(x<sub>1</sub>), ..., P(x<sub>t</sub>)) for (x<sub>1</sub>, ..., x<sub>t</sub>) distinct field elements
- Lagrange coefficients:

$$P(X) = \sum_{i=1}^{t} P(x_i) \prod_{j \neq i} \frac{X - x_i}{x_j - x_i}$$

#### Construction with polynomials

Protocol:

- We (the secret owner/dealer) sample a random polynomial in  $K_{t-1}[X]$  such that P(0) = y
- Fix public non-zero interpolation points  $x_1, \dots, x_n$
- ${}^{\bullet}$
- Distribute  $y_i := P(x_i)$  to party  $i \in \{1, ..., n\}$ Any group of *t* parties can reconstruct y:  $y = P(0) = \sum_{i=1}^t P(x_i) \prod_{j \neq i} \frac{0 x_i}{x_j x_i} = \sum_{i=1}^t y_i \lambda_i$
- The Lagrange coefficients  $\lambda_i$  can be computed in advance, we just need a linear combination of the shares to reconstruct the secret

### 2. Evaluating Arithmetic Circuits with the BGW Protocol

Ben-Or, Goldwasser and Widgerson, 1988 [9]

Can we perform operations on a secret-shared input?

- Example application: split a private key into *n* shares, and sign a document without ever reconstructing the private key locally
- Any computation in  $F_p$  can be represented as an arithmetic circuit (why?) We just need to have secret-shared version of the + and x gates

Using multiple inputs:

- In the Shamir setting we had a trusted dealer that splits a secret into shares
- The dealer can be a (semi-honest) party that shares its own input with other parties
- We run multiple Shamir sharings in parallel and combine them with gates

#### **Addition Gate**

- Two inputs shared with Shamir's scheme:
  - Secret p, polynomial P such that p = P(0), shares  $P(x_1), ..., P(x_n)$
  - Secret q, polynomial Q such that q = Q(0), shares  $Q(x_1), ..., Q(x_n)$
- Output:
  - Desired output: r := p + q = P(0) + Q(0)
  - R := P + Q is a valid Shamir polynomial (degree  $\leq$  t-1 and R(0) = r)
  - Party i's share is  $R(x_i) = P(x_i) + Q(x_i)$
- Parties can construct their share of the output locally, without any interaction!

#### **Multiplication Gate**

- Two inputs shared with Shamir's scheme:
  - Secret p, polynomial P such that p = P(0), shares  $P(x_1), ..., P(x_n)$
  - Secret q, polynomial Q such that q = Q(0), shares  $Q(x_1), ..., Q(x_n)$
- Output:
  - Desired output: r := p \* q = P(0) \* Q(0)
  - R := P \* Q satisfies R(0) = r but has degree  $\leq 2(t-1)$ , not a valid sharing
  - Goal: find another polynomial R' with R'(0) = r and degree  $\leq$  t

#### Multiplication Gate – Degree Reduction

Goal: find another polynomial R' with R'(0) = r and degree  $\leq$  t-1

Reducing degree by resharing coefficients:

- Observation: with Lagrange's formula, we have  $R(0) = \sum \lambda_i R(x_i)$
- Each party i can create a Shamir sharing of  $R(x_i)$ :
  - Choose a degree t-1 polynomial R such that  $\vec{R}(0) = \vec{R}(x)$ Ο
  - Distribute  $R_i(x_i)$  to party j Ο

$$R(0) = \sum_{i=1}^{2t-1} \lambda_i \left( \sum_{j=1}^t \mu_j R_i(x_j) \right) = \sum_{j=1}^t \mu_j \left( \sum_{i=1}^{2t-1} \lambda_i R_i \right) (x_j) \qquad \qquad R' := \sum_{i=1}^{2t-1} \lambda_i R_i$$

**Properties**:

- Re-sharing requires extra communication
- Security against t-1 corrupt parties. We also need  $2t-1 \le n$  to reconstruct R(0): honest majority. •

2t-1

Corrupt parties are still semi-honest here (imagine a malicious party that re-shares garbage coefficients)

### 3. MPC with Preprocessing: Beaver Triples

Beaver, 1991 [10]

- BGW multiplications are costly (in terms of interactions)
- We can save time by computing some things in advance
- MPC with preprocessing:
  - Offline phase: a trusted dealer generates input-independent cryptographic material
  - Online phase: parties use the material to save some time (less communication) when evaluating the circuit
- Beaver triples are secret-shared tuples for multiplication

### 3. Beaver Triples

#### Generation:

- 1. Take a random tuple (a,b,c) in  $F_p$  such that c = a\*b
- 2. Split it and distribute shares to the parties: [a], [b], [c]

#### Multiplication: we have [x], [y] and want [xy]

- 1. Each party reveals [x] [a], d := x a is now public
- 2. Each party reveals [y] [b], e: y b is now public
- 3. Each party computes locally [xy] = de + d[b] + e[a] + [c]

### 3. Beaver Triples

Security:

• x - a and y - b are one-time pad encryptions of x and y

Correctness:

 $\sum (de + d[b] + e[a] + [c])$ = (x-a)(y-b) + (x-a)b + (y-b)a + c = xy



### **Beaver Triples in a Circuit**

Computational and communication cost:

- Each party just needs to broadcast 2 values ([x] [a] and [y] [b])
- In BGW, each party generates a polynomial and sends n values (one for each other party)
- Triples don't depend on the input, and can't be reused, so we need to prepare enough to evaluate the whole circuit
- There are techniques to generate triples in batches

Applicability:

- Beaver triples work with other types of secret sharing, not just Shamir and BGW
- The trusted dealer can be emulated by the parties themselves (e.g. with HE [3])
- Information-theoretic security: no computational assumptions

#### 4. Implementation: Meta's Private Computation Framework

- General purpose library to build MPC systems
- Open-source: <u>https://github.com/facebookresearch/fbpcf</u>
- Architecture from the whitepaper [2]:



#### Cyptographic backend and scheduler

- Boolean circuits instead of arithmetic circuits
  - Inputs are secret-shared bits
  - AND and XOR instead of + and x
  - Easier to manipulate and compile programs
- Cryptographic primitives:
  - GMW secret sharing, a different scheme than BGW tailored for F<sub>2</sub> and resilient against up to n-1 corrupt parties (BGW needs a honest majority)
  - Beaver triples for AND gates
  - <u>https://github.com/facebookresearch/fbpcf/blob/main/fbpcf/engine/SecretShareEngine.cpp</u>

#### • Scheduler:

- Keep track of intermediate results
- Order gates and execute them
- Supports multithreading

#### C++ types and operators

- Frontend types: special C++ types for Bit, Int, BitString
- Everything is reduced to bitwise operations (gates)
- Gates are passed to the scheduler
- Example: integer comparison.

https://github.com/facebookresearch/fbpcf/blob/b38024cccc79dff74bbce3fbbf

9836caf80a4ce7/fbpcf/frontend/Int\_impl.h#L186

### **Example application**

#### • The millionaire game:

- Alice and Bob
- Each party has one (secret) input, corresponding to their wealth
- The output of the circuit is one bit, corresponding to who is the richest (but not their wealth)
- Parties shouldn't learn anything else than the output
- <u>https://github.com/facebookresearch/fbpcf/blob/main/example/millionaire/Millionaire/MillionaireGame.h</u>
- Deployment: TCP socket communication, parties can run in Docker

#### Conclusion

- Simple setting: honest-but-curious adversary and information-theoretic security
- Basic MPC techniques: Shamir secret sharing, BGW protocol, Beaver triples
- Local computations are lightweight (unlike FHE)
- But parties need to communicate more often

#### Conclusion

There are many other important concepts we didn't cover. Some keywords:

- Malicious security: we can adapt protocols with MACs, ZK proofs and other techniques (e.g. see the SPDZ family of protocols and its modern implementations [4]).
- **Oblivious transfer (OT)**: a useful primitive where a receiver privately picks one of two secrets offered by a sender.
- **Garbled circuits**: evaluate circuits in constant number of rounds (BGW's number of rounds is proportional to the depth of the circuit).
- FHE and Homomorphic Secret Sharing: other ways of achieving MPC.
- **Oblivious RAM (ORAM):** hide data access patterns efficiently.

### Conclusion

State-of-the-art MPC protocols can be practical:

- Usually with 2 or 3 *active* parties (think non-colluding cloud providers)
- But can handle large numbers of *passives* parties (think browsers) who share their input once and let the active parties compute the output
- Primitives tailored for different use cases

Examples:

- AES evaluation on a secret-shared secret key [5]
- Distributed aggregation for contact tracing or telemetry [7]
- Training ML models on secret-shared data [6]

#### References

[1] D. Evans, V. Kolesnikov, and M. Rosulek, "A Pragmatic Introduction to Secure Multi-Party Computation," SEC, vol. 2, no. 2–3, pp. 70–246, Dec. 2018, doi: 10.1561/3300000019.

[2] "Private Computation Framework 2.0 - Meta Research," Meta Research. https://research.facebook.com/publications/private-computation-framework-2-0/ (accessed Mar. 08, 2023).

[3] N. P. Smart and T. Tanguy, "TaaS: Commodity MPC via Triples-as-a-Service," in Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, New York, NY, USA, Nov. 2019, pp. 105–116. doi: 10.1145/3338466.3358918.

[4] M. Keller, "MP-SPDZ: A Versatile Framework for Multi-Party Computation," in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, Nov. 2020, pp. 1575–1590. doi: 10.1145/3372297.3417872.

[5] I. Damgård and M. Keller, "Secure Multiparty AES: (Short Paper)," in Financial Cryptography and Data Security, vol. 6052, R. Sion, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 367–374. doi: 10.1007/978-3-642-14577-3\_31.

[6] P. Mohassel and P. Rindal, "ABY3: A Mixed Protocol Framework for Machine Learning," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, Oct. 2018, pp. 35–52. doi: 10.1145/3243734.3243760.

[7] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics," presented at the 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17), 2017, pp. 259–282. Accessed: Dec. 15, 2020. [Online]. Available: <a href="https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs">https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs</a>

[8] A. Shamir, "How to share a secret," Commun. ACM, vol. 22, no. 11, pp. 612–613, Nov. 1979, doi: 10.1145/359168.359176.

[9] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in Proceedings of the twentieth annual ACM symposium on Theory of computing, New York, NY, USA, Jan. 1988, pp. 1–10. doi: 10.1145/62212.62213.

[10] D. Beaver, "Efficient Multiparty Protocols Using Circuit Randomization," in Advances in Cryptology — CRYPTO '91, vol. 576, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 420–432. doi: 10.1007/3-540-46766-1\_34.

37

MPC Details and Demo

### The End