Addressing Extreme Market Responses Using Secure Aggregation

Tucker Balch
tucker.balch@jpmorgan.com
J.P. Morgan AI Research
New York, NY, United States

Sahar Mazloom
sahar.mazloom@jpmorgan.com
J.P. Morgan AI Research
New York, NY, United States

Antigoni Polychroniadou
antigoni.polychroniadou@jpmorgan.com
J.P. Morgan AI Research
New York, NY, United States

ABSTRACT

An investor short sells when he/she borrows a security and sells it on the open market, planning to buy it back later at a lower price. That said, short-sellers profit from a drop in the price of the security. If the shares of the security instead increase in price, short sellers can bare large losses. Short interest stock market data, provide crucial information of short selling in the market for data mining by publishing the number of shares that have been sold short. Short interest reports are compiled and published by the regulators at a high cost. In particular, brokers and market participants must report their positions on a daily basis to Financial Industry Regulatory Authority (FINRA). Then, FINRA processes the data and provides aggregated feeds to potential clients at a high cost. Third party data providers offer the same service at a lower cost given that the brokers contribute their data to the aggregated data feeds. However, the aggregated feeds do not cover 100% of the market since the brokers are not willing to submit and trust their individual data with the data providers. Not to mention that brokers and market participants do not wish to reveal such information on a daily basis to a third party.

In this paper, we show how to publish short interest stock market data using Secure Multiparty Computation: In our process, brokers and market participants submit to a data provider their short selling information, including the symbol of the security and its volume in encrypted messages on a daily basis. The messages are encrypted in a way that the data provider cannot decrypt them and therefore cannot learn about individual participants input. Then, the data provider, can compute an aggregation on the encrypted data and publish the aggregation of the volume per security. It is important to note that the individual volumes are not revealed to the data provider, only the aggregated volume is published.

CCS CONCEPTS

• **Security and privacy** → *Privacy-preserving protocols.*

KEYWORDS

Short Interest, Secure Computation

ACM Reference Format:
Tucker Balch, Sahar Mazloom, and Antigoni Polychroniadou. 2022. Addressing Extreme Market Responses Using Secure Aggregation. In *3rd ACM International Conference on AI in Finance (ICAIF '22), November 2–4, 2022, New York, NY, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3533271.3561776

1 INTRODUCTION

Traders betting that a stock will go down in price borrow stock and sell it into the market. If they're right, and the stock price does go down, they're in a position to buy the shares they borrowed for cheaper than they sold the borrowed shares for, pay back the lender, and keep the difference. If they're wrong, and the stock goes up, they may have to buy stock to cover the loan at a higher price, and lose money. *Short squeeze* begins when the price jumps higher unexpectedly.

One of the biggest short squeezes involve the stock of European automaker Volkswagen. In 2008, the stock price jumped by more than 300% in a matter of days. At that time the company seemingly worth more than $400 billion! Short squeezes have dominated headlines this year as retail investors band together on internet platforms to spoil hedge fund shorts on certain stocks, such as the video game retailer GameStop. The share price jumped from less than $5 to $325 just in six months. A trading app suspended trading in the stock, giving short-sellers time to recover losses.

The Financial Industry Regulatory Authority (FINRA) requires firms to report SI positions on a daily basis. Market participants are highly interested in such data since they would like to predict and prevent short squeeze phenomena. FINRA offers aggregated data feeds to interested participants charging them at a significant price. Data providers such as Data Explorer, Bloomberg, to name just a few, offer the same service with aggregated feeds either by collecting data from FINRA or by the individual market participants. The dollar cost for the market participants to the date providers is lower since they are willing to sacrifice their data to receive the aggregated feeds. On the other hand, many market participants refuse to provide their individual data to the data providers due to security concerns and fear of revealing their trading strategies to third (untrusted) parties who can either leak the data or use them for their own benefit. That said, the aggregated feeds do not reflect 100% of the market and may not be so useful for data mining. Furthermore, the data collected from FINRA are published by the data providers with some lag. Hence when it's reported directly to the public that data is already stale.

1.1 Our contributions

Using cryptographic techniques, we address the above problem and allow a data provider to publish SI data without revealing any information about the short selling data of individual market participants. We provide a suite of cryptographic protocols for aggregation of encrypted data:

- *Secure Aggregation of Short Positions*: We propose a secure multiparty aggregation protocol, based on techniques from multiparty secure computation (MPC) [Chaum et al. 1987; Goldreich et al. 1987; Yao 1982, 1986], such that market participants provide their short positions (volume) per security in an encrypted way to a service provider.
- *Secure Aggregation of Long Positions*: The protocol can also be used for aggregation of encrypted/masked long positions.

In particular, we construct protocols that allow a service provider and the market participants to jointly compute the aggregation of all of the short and long positions in a security, without the market participants revealing their own short and long positions. Only after the aggregation on the encrypted data, the service provider reveals the aggregated results to the market participants. This type of secure calculation on a daily basis can help prevent the consequences of short squeezes. The protocols are based on the secure aggregation protocols used in federated learning [Bonawitz et al. 2017]. An example for the aggregation of the positions of three clients can be found in Table 1. We also provide a full implementation of our system in Section 5.

There are some other techniques to preserve the privacy of the participants such as Differential Privacy [Dwork et al. 2014]. However, Differential Privacy is solving an orthogonal question to Secure Computation. In this work, the goal is to publish the aggregated volume of Short Sell positions to the market participants, while preserving the privacy of individual's private positions during the computation. Therefore, Secure Computation which allows us to aggregate on the encrypted/masked position data of the participants, is a suitable solution for this problem. In contrast, Differential Privacy does not protect the data during the computation, and only by adding bounded noise to the result of the computation, aims to protect the output of the computation and hence protect the privacy of participants. However, adding deferentially private noise to the results of the computation reduces the accuracy of the output and consequently affects the utility of the result.

2 PRELIMINARIES

2.1 Secure Multiparty Computation

Secure Multi-Party Computation (MPC) allows a set of mutually distrustful parties to compute a function jointly over their inputs while maintaining the privacy of the inputs and ensuring the correctness of the outputs. Secure computation is beneficial only in places where parties cannot trust each other, and cannot trust some external entity. In particular, in case such incorruptible trusted party can be found, computing the joint function can be performed easily: The parties send their inputs to the trusted party, which compute the function on the inputs and send the parties the output of the computation. However, in many realistic scenarios the parties cannot rely or trust such an external entity, and secure computation comes to eliminate the need for such trusted party. Seminal results established in the '80s [Ben-Or et al. 1988; Chaum et al. 1987; Goldreich et al. 1987; Yao 1986] show that *any* computation can be emulated by a secure protocol.

More concretely, consider n parties $P_1, \ldots, P_n$ that hold private inputs $x_1, \ldots, x_n$ and wish to compute some arbitrary function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, where the output of P_i is y_i. MPC enables the parties to compute the function using an interactive protocol, where each party P_i learns exactly y_i, and nothing else. The function f can also be reactive, namely, it might maintain states between different invocations.

Security of MPC. The security of the protocol should be preserved even in the presence of some adversarial entity that corrupts some of the participating parties, combines their transcripts and coordinates their behaviors. Several different types of adversaries have been studied, where the complexity and the efficiency of the protocols depend on the level of security:

A semi-honest adversary (also known as "honest-but-curious" or "passive"), follows the protocol specification but may attempt to learn secret information about the private information of the honest parties from the messages it receives. The most efficient protocols can be constructed assuming that the adversary is semi-honest. *A malicious adversary* (also known as "active") may, in addition, deviate from the protocol specification and follow any arbitrary behavior. This is the strongest level of security possible, and protocols that achieve these level of security are often much slower than protocols that are secure in the presence of a semi-honest adversary. Such protocol usually work by adding many checks throughout the protocol execution that parties follow the protocol as as specified.

In this paper, for simplicity of exposition, we focus on semi-honest security and computational security. We provide a formal security definition of security in this model in the Appendix A, which follows the standard security definition [Canetti 2000; Goldreich 2004].

2.2 Network Topology & Threat Model

We opt for a star network topology, where there is one service provider who is connected to all other parties. This central server can be distinct from the n original parties.

The protocols that we describe and compare against are secure in the semi-honest model. A semi-honest adversary follows the protocol correctly but tries to learn as much as possible about the inputs of the uncorrupted parties from the messages it receives. Furthermore, if there are multiple semi-honest corruptions, we allow the adversary to combine the views of the corrupted parties to potentially learn more information.

For a protocol of n parties, we offer security against any $n-1$ semi-honest corruptions or collusion of the server with $n-1$ semi-honest corruptions.

3 SECURE AGGREGATION OF POSITIONS

3.1 The Functionality

As discussed in the introduction, we are interested in a platform in which clients provide their encrypted positions/volumes and the platform will publish the aggregation of volumes per symbol.

We define the AggregationPlatform functionality. The participating parties are authorized brokers or market participants that were invited by the service provider S to engage in the system. While the service provider runs the aggregation algorithm only at some designated times throughout the day, clients can keep submitting their encrypted volumes. We proceed with a formal description of the functionality:

Functionality 1: $F_{\mathsf{AggregationPlatform}}$

- **Participating parties:** A set of authorized clients $\mathcal{P} = \{P_i\}_{i=1}^n$ and the service provider S.
- **Public parameters:** An ordered list of symbols that can be traded $\mathsf{Symb} = (\sigma_1, \dots, \sigma_m)$.
- **The functionality:**
  1. **The command** Initialization**().** Upon receiving the command from the service provider S, the functionality initializes the two values per symbol $\{\mathcal{L}^\sigma, \mathcal{S}^\sigma\}_{\sigma\in\mathsf{Symb}}$ to 0, as well as two vectors $L^\sigma[1,\dots,n]$, $S^\sigma[1,\dots,n]$ for every symbol $\sigma \in \mathsf{Symb}$, and a list of indices $\mathsf{Q} = \emptyset$.
  2. **The command** $\mathsf{Register}(i, (L_i^\sigma, S_i^\sigma)_{\sigma\in\mathsf{Symb}})$**.** Upon receiving the command from client P_i, the functionality sets $L^\sigma[i] = L_i^\sigma$ and $S^\sigma[i] = S_i^\sigma$ for every symbol $\sigma \in \mathsf{Symb}$. Moreover, it appends the index i to Q, and sends to J the message Client i has registered.
     If client P_i already provided an input in this round, then its previous occurrence in Q is removed.
  3. **The command** Aggregation()**.** Upon receiving the command from the service provider S, the functionality proceeds as follows:
     (a) Aggregates $\mathcal{L}^\sigma = \sum_{i=1}^n L^\sigma[i]$ and $\mathcal{S}^\sigma = \sum_{i=1}^n S^\sigma[i]$ for every symbol $\sigma \in \mathsf{Symb}$, and sends the aggregated values $\{\mathcal{L}^\sigma, \mathcal{S}^\sigma\}_{\sigma\in\mathsf{Symb}}$ to the service provider S and the participants.

While there are many clients in the system, no client learns anything about the inputs of other clients. The only information revealed is the aggregation of all the Long and Short positions per symbol.

4 SECURE AGGREGATION PROTOCOL

In this section, we realize Functionality **??** using a secure aggregation n-party protocol. We provide this sub-protocol Π_{Aggr} in Section 4.1.

The protocol of our platform $\Pi_{\mathsf{AggrPlatform}}$ proceeds as follows: The clients first run the $\Pi_{\mathsf{Aggr}}.\mathsf{Setup}(1^\lambda)$ protocol in which each client P_i gets the domain Symb.

Then, in order to aggregate all values of all clients to some variable v (some symbol $\sigma \in \mathsf{Symb}$ and side in long/short):

1. RegisterInput: In this stage, client P_i register its input L_i (or S_i).
2. Aggregation: Upon invoking this command, the service provider checks that all clients provided inputs. If not, it outputs $\perp$. Otherwise, it learns $\sum_{i=1}^n L_i$.

We assume that each client has a secure channel with the service provider S. The clients do not know each other, but communicate to each other through S.

Protocol 1 ($\Pi_{\mathsf{AggrPlatform}}$: Implementing $F_{\mathsf{AggregationPlatform}}$).

- **Implementing the command** Initialization**.** The parties invoke $\Pi_{\mathsf{Aggr}}.\mathsf{Setup}(1^\lambda)$.
  The service provider initializes a list Q.
- **Implementing the command** $\mathsf{RegisterInput}(i, \{L_i^\sigma, S_i^\sigma\}_{\sigma\in\mathsf{Symb}})$
  When a client P_i wishes to register positions, it engages with the service provider in protocols $y_i^L = \Pi_{\mathsf{Aggr}}.\mathsf{RegisterInput}(L_i^\sigma)$ and $y_i^S = \Pi_{\mathsf{Aggr}}.\mathsf{RegisterInput}(S_i^\sigma)$. The service provider appends i to the list Q.
- **Implementing the command** $\mathsf{Aggregation}(\{y_i^L, y_i^S\}_{i\in[n]})$**.**
  1. After all parties registered their inputs, the service provider runs $\mathcal{L}^\sigma = \Pi_{\mathsf{Aggr}}.\mathsf{Output}(y_i^L)$ and $\mathcal{S}^\sigma = \Pi_{\mathsf{Aggr}}.\mathsf{Output}(y_i^S)$ for every $\sigma \in \mathsf{Symb}$.

4.1 Secure Aggregation Protocol

In what follows we present the secure aggregation protocol Π_{Aggr}, depicted in Protocol 1, performed by a set of clients $(P_1, \dots, P_n)$ and a service provider S. All operations are performed modulo some bound p.

During setup, every pair of parties P_i and P_j will share some common randomness $r_{i,j} = r_{j,i}$. In the online aggregation phase, client P_i sends its weights masked with these common random strings, adding all $r_{i,j}$ for $j > i$ and subtracting all $r_{k,i}$ for $k < i$. That is, P_i sends to server S the following message for its data L_i or S_i:

$$y_i := (L_i + \sum_{j=i+1}^{n} r_{ij} - \sum_{k=1}^{i-1} r_{ki}) \bmod p \qquad (1)$$

To establish common randomness, each pair of client parties run the Diffie-Hellman Key agreement protocol [Diffie and Hellman 1976] communicating via the server. The cryptographic primitives used in Protocol 1 include:

- An algorithm $\mathcal{G}(1^\lambda)$, where λ is the security parameter, that outputs a representation of a cyclic group $\mathbb{G}$ of order q (with $||q|| = \lambda$) for which the discrete logarithm problem is believed to be hard. Recall that a group $\mathbb{G}$ is cyclic if there exists a generator g such that $\{g^0, g^1, \dots, g^{q-1}\} = \mathbb{G}$. Moreover, the discrete logarithm problem is believed to be hard if for every probabilistic polynomial time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:
  $$\Pr_{x\leftarrow\mathbb{Z}_q}\left[\mathcal{A}(\mathbb{G}, g, q, g^x) = x\right] = \mathsf{negl}(\lambda)$$
  In other words, it is hard to guess x given g^x for particular groups $\mathbb{G}$.
- A key derivation function $H : \mathbb{G} \to \{0,1\}^\lambda$. It is assumed that if h is distributed uniformly in $\mathbb{G}$, then $H(h)$ is distributed uniformly in $\{0,1\}^\lambda$.
- A pseudorandom generator with double expansion, i.e., $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. It is assumed that for every distinguisher D there exists a negligible function $\mathsf{negl}(\cdot)$ such that:
  $$\left|\Pr_{s\leftarrow\{0,1\}^\lambda}[D(G(s)) = 1] - \Pr_{r\leftarrow\{0,1\}^{2\lambda}}[D(r) = 1]\right| = \mathsf{negl}(\lambda)$$

Protocol 1 Secure Aggregation Protocol Π_{Aggr} for Short positions

The protocol Π_{Aggr} runs with parties $P_1, \ldots, P_n$ and a service provider S. It proceeds as follows:
Inputs: For $i \in [n]$, party P_i holds input L_i.
Public Parameters: $(\mathbb{G}, g, q)$ generated by $\mathcal{G}(1^\lambda)$ and modulo p.
$\mathbf{\Pi_{Aggr}}$.Setup(1^λ):

Round 1: Each party P_i for $i \in [n]$ proceeds as follows:

- Choose n secrets $a_{i,1}, \ldots, a_{i,n}$ uniformly and independently at random from $\mathbb{Z}_q$ and compute $(pk_{i,1}, \ldots, pk_{i,n}) = (g^{a_{i,1}} \bmod p, \ldots, g^{a_{i,n}} \bmod p)$.
- Each party P_i sends $pk_{i,j}$ to the service provider who forwards $pk_{i,j}$ to party P_j.

Round 2: Each party P_j for $j \in [n]$ proceeds as follows:
Upon receiving all values $(pk_{1,j}, \ldots, pk_{n,j})$, compute the shared common keys $r_{i,j}$ for all $i \in [n]$ as follows:

(a) Using the secret $a_{j,i}$ compute $c_{i,j} = c_{j,i} = \left(pk_{i,j}\right)^{a_{j,i}} = (g^{a_{i,j}})^{a_{j,i}} \bmod p$.
(b) Let $c_{1,j}, \ldots, c_{n,j}$ be the set of all common keys. Use a key-derivation function and set $r_{i,j} = r_{j,i} = H(c_{i,j})$

Given the above setup, we can perform aggregation as follows:
$\mathbf{\Pi_{Aggr}}$.RegisterInput$(S_i, \{r_{i,j}\}_{j\in[n]})$:
The next steps are repeated per symbol. Without loss of generality we describe the algorithm for a single symbol for the Short positions. Each party P_i proceeds as follows: Compute and send y_i to the service provider.

$$y_i := S_i + \sum_{j=i+1}^{n} r_{i,j} - \sum_{k=1}^{i-1} r_{k,i} \bmod p.$$

$\mathbf{\Pi_{Aggr}}$.Output$(\{y_i\}_{i\in[n]})$:
The service provider computes $Y = \left(\sum_{i=1}^{n} y_i \bmod p\right)/n$ and sends Y to all parties.

4.2 Invalid Inputs

In our current protocol, clients could submit encryptions of "invalid" inputs, e.g. a very large number of shares to distorted the aggregated sum. These types of attacks are generally considered outside the scope of cryptography. However, to address these types of attacks, we would need a rigorous, mathematical definition of an invalid input, e.g. every input range should be a value between 0 and 10^7. Armed with such a definition, our protocol could be augmented with special-purpose Zero-Knowledge Proofs (e.g. efficient lattice-based range proofs [Esgin et al. 2019; Libert et al. 2018; Yang et al. 2019]) to ensure that participants' inputs satisfied these criteria.

Moreover, submitting false number of shares undermines the correctness of the responses, and such a behavior can be detected by the audit process from regulators. A potential "auditing" functionality to detect this type of behavior will require cryptographic commitments on the number of shares which will be opened during the audit process. If we assume participating clients derive some benefit from participating in this system, submitting false values has direct negative consequences for the attackers (beyond the risk of detection and punishment).

4.3 Dropout Clients

The above protocols operate only with clients who are online throughout the whole execution of the protocol. The recent work of [Guo et al. 2022] provides the most efficient secure protocol for the case where clients stop responding (they either go offline or their computer get disconnected) during the execution of the protocol.

5 IMPLEMENTATION AND EVALUATION

We implemented our framework in Python language, and use NaCl library (https://pypi.org/project/PyNaCl) for networking and cryptographic functions. To evaluate the performance of our method, we conduct several experiments on a machine with 2.6 GHz 6-Core Intel Core i7 and 16 GB of RAM. We run the experiments with different number of clients $\in \{3, 5, 10, 20, 50, 100, 200\}$ participating into the platform. We also vary the size of the Symbol list to different values as $\in \{10, 100, 500, 1000, 3000, \text{All}(3417)\}$; 'All' refers to the full list of US Security Symbols used for Axe Inventory which contains 3417 symbols. We measure the running time of the process for each phase, Setup and Aggregation, separately. Setup phase includes the time for parties to perform key exchange and establish a secure channel, whereas Aggregation phase includes the time that parties send their encrypted/masked volumes to the Server and then the Server aggregates the encrypted/masked values.

Table 1 provides an example for the Secure Aggregation platform for three parties that each has short positions in four different symbols. It shows the volume of short positions for each client per symbol, and the random mask value it uses to encrypt/mask its short position. The last column is the aggregated volume across all users for each symbol. Note that even though clients A and B have the same volume (0) for two different symbols (GME and TSLA), their mask values are different, as they are generated randomly.

Table 2 contains the running time of the protocol during the setup phase, in which the key establishment between the parties happens. The running time of the setup phase depends only on the number of parties and it is independent of the size of the Symbol list.

		Client A's Positions		**Client B's Positions**		**Client C's Positions**		**Aggregated**
Symbol	Security Name	Exact Value	Masked Value	Exact Value	Masked Value	Exact Value	Masked Value	**Values**
AMZ	Amazon	1000	51430274	200	25792475	200	76841733	1400
GME	GameStop	0	71796890	100	54854547	6000	80575124	6100
TSLA	Tesla	700	61302206	0	25792475	2200	19197321	2900
VRSN	VeriSign	4300	48744874	1200	47154557	500	35074570	6000

Table 1: Demonstrating the volume of short positions for each client per symbol (the exact and masked values are in the same column). Last column shows the aggregated value of Short positions across all clients per symbol, which can be published.

Number of Parties	3	5	10	20	50	100	200
Run Time (s)	0.0039	0.0051	0.0157	0.0314	0.1274	0.4103	1.4145

Table 2: Running Time (seconds) of Key Setup phase, for different number of clients.

	Number of Parties						
Number of Symbols	3	5	10	20	50	100	200
10	0.0181	0.0239	0.1099	0.1276	0.3699	0.6527	1.7459
100	0.0204	0.0334	0.0855	0.2008	0.8046	2.6124	11.1628
500	0.0472	0.0754	0.1929	0.5129	3.1220	12.5978	52.0158
1000	0.0762	0.1208	0.2957	0.9851	6.2835	25.3394	102.9129
3000	0.1613	0.2698	0.7653	2.4802	17.9473	76.9245	307.8306
All (3417)	0.1846	0.3063	0.8396	2.9964	20.3990	86.0329	353.8592

Table 3: Running Time (sec) of Secure Aggregation phase, with different number of symbols and different number of clients. 'All' refers to the full list of US Security Symbols used for Axe Inventory which contains 3417 symbols.

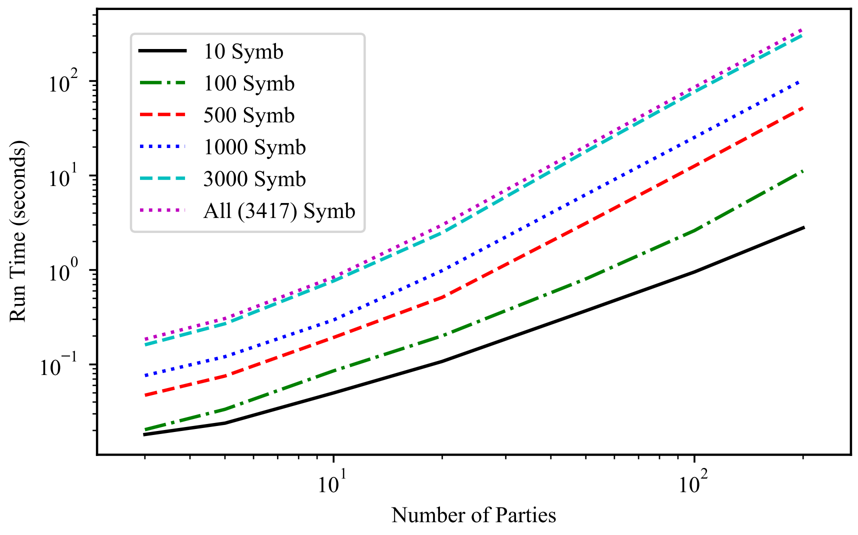


Figure 1: Effect of number of parties and number of symbols on run time

The running time of the protocol in the aggregation phase is shown in Table 3. As the number of parties or the size of the symbol list increases, consequently the running time of the protocol increases. Figure 1 demonstrates the effect of the number of parties or the size of symbol list on the running time. The graph is in *loglog* format.

Finally, Figure 2 illustrates the cost of the setup phase relative to the cost of the secure aggregation phase. The bar graph shows that as the number of parties increases, the cost of the secure aggregation phase becomes more dominant, affecting the performance of the protocol. For example, when the number of parties is 200, the cost of the setup phase is 1.4 seconds, regardless of the number of symbols in the list. However, the cost of secure aggregation phase increases from 1.7459 to 52.0159 seconds when the number of symbols increases from 10 to 500 symbols, still with 200 parties. The protocol supports more number of parties, however the real dataset of FINRA does not have high number of participants which is almost less than 1000 participants. For large number of parties, we can run the Diffie Hellman key exchange protocol not across all parties but by creating neighborhoods which consist of $\log N$ participants, where N is the total number of parties [Bell et al. 2020].

6 CONCLUSIONS AND FUTURE WORK

In this work, we show how to publish short interest stock market data, while protecting the private data of the participants, using Secure Multiparty Computation. We propose a solution to securely compute the aggregate of short sell position of participants on their encrypted input. This securely aggregated value, now can be published on a daily basis, such that traders will be better informed of short selling in the market, track short interest in stocks, and gain from the advantages of this powerful market data which is not available today at low price.

Future work: We can use our system to compute several other aggregated metrics which can provide useful trading tools and indicators to the investors. Furthermore, we can combine the aggregated data with other sentiment data. As mentioned in 4.3, we can also expand our protocol to support dropout users.

ACKNOWLEDGMENTS

This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co. and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

REFERENCES

James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. *Secure Single-Server Aggregation with (Poly)Logarithmic Overhead.* Association for Computing Machinery, New York, NY, USA, 1253–1269. https://doi.org/10.1145/3372297.3417885

Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, Janos Simon (Ed.). ACM, 1–10.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 1175–1191.

Ran Canetti. 2000. Security and Composition of Multiparty Cryptographic Protocols. *J. Cryptol.* 13, 1 (2000), 143–202.

David Chaum, Claude Crépeau, and Ivan Damgård. 1987. Multiparty Unconditionally Secure Protocols (Abstract). In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings.* 462. https://doi.org/10.1007/3-540-48184-2_43

Whitfield Diffie and Martin E. Hellman. 1976. New directions in cryptography. *IEEE Trans. Information Theory* 22, 6 (1976), 644–654. https://doi.org/10.1109/TIT.1976.1055638

Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

Muhammed F Esgin, Ron Steinfeld, Joseph K Liu, and Dongxi Liu. 2019. Lattice-based zero-knowledge proofs: new techniques for shorter and faster constructions and applications. In *Annual International Cryptology Conference.* Springer, 115–146.

Oded Goldreich. 2004. *The Foundations of Cryptography - Volume 2: Basic Applications.* Cambridge University Press. https://doi.org/10.1017/CBO9780511721656

Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA.* 218–229. https://doi.org/10.1145/28395.28420

Yue Guo, Antigoni Polychroniadou, Elaine Shi, David Byrd, and Tucker Balch. 2022. MicroFedML: Privacy Preserving Federated Learning for Small Weights. *IACR Cryptol. ePrint Arch.* (2022), 714. https://eprint.iacr.org/2022/714

Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. 2018. Lattice-based zero-knowledge arguments for integer relations. In *Annual International Cryptology Conference.* Springer, 700–732.

Rupeng Yang, Man Ho Au, Zhenfei Zhang, Qiuliang Xu, Zuoxia Yu, and William Whyte. 2019. Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications. In *Annual International Cryptology Conference.* Springer, 147–175.

Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982.* IEEE Computer Society, 160–164.

Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986.* IEEE Computer Society, 162–167.

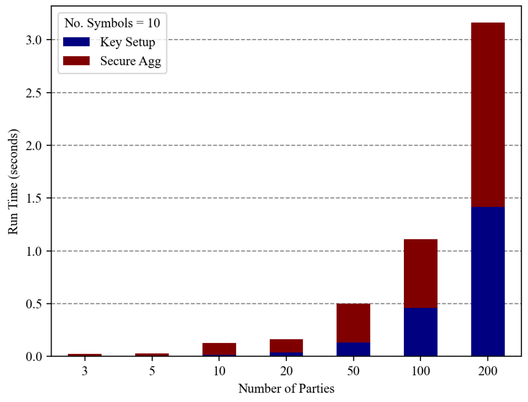
A SECURITY DEFINITIONS

Our protocol protects the privacy of honest users in the semi-honest setting given the topology in Section 2.2. Following the standard idea-real paradigm, we will show that when executing the protocol with a set of parties $\mathcal{U}$ of size n with threshold $t < n$, the joint view of the server S and any set of less than t users does not leak any information about the other users' inputs except what can be inferred from the output of the computation.

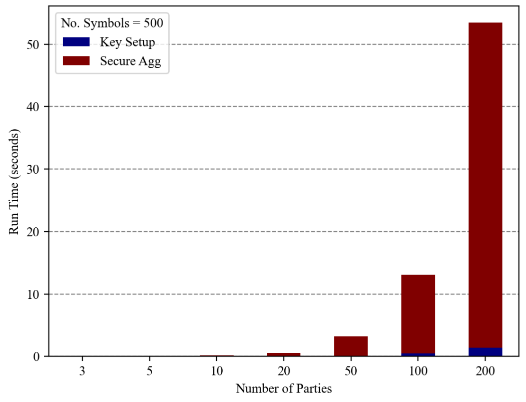
The view of a party P_i consists of its internal state (including its input w_i and randomness) and all messages this party received from other parties. The messages *sent* by this party do not need to be part of the view because they can be determined using the other elements of its view. Given any subset C of corrupted parties out of the n parties in $\mathcal{U}$, let $\mathsf{REAL}_C^{\mathcal{U},t,k}(w_1, \ldots, w_n)$ be a random variable representing the combined views of all parties in C in the above protocol execution, where the randomness is over the internal randomness of all parties. We are going to show that there exists an efficient simulator that, for every choice of the honest clients' inputs, outputs a simulation of the adversarial participants' view of the protocol run whose distribution is computationally indistinguishable from the distribution of the adversaries' view of the real protocol run.

The following theorem shows that the joint view of any subset of less than t users and the server can be simulated without knowing the secret input of any other users. In other words, the adversary controlling less than t users cannot learn anything other than the output of the computation.

Theorem A.1 (Semi-Honest Security, against t clients). *For security parameter λ, an n-party protocol for an aggregation function f is secure if there exists a PPT simulator* SIM *such that for all $k, t, \mathcal{U}$*



(a) No. Symbols = 10



(b) No. Symbols = 500

Figure 2: Cost of each phase of, Setup and Secure Aggregation, separately. (a) Cost for 100 participants (b) Cost for 500 participants.

where $t < |\mathcal{U}|$, all corrupted parties $C \subseteq \mathcal{U}$ and all $w_{\mathcal{U}}$, which denote all secret inputs of all users in all iterations k, letting $n = |\mathcal{U}|$, then the output of SIM *is computationally indistinguishable from the output of* $\mathsf{REAL}^{\mathcal{U},t,k}$:

$$\mathsf{REAL}_C^{\mathcal{U},t,k}(\lambda, w_{\mathcal{U}}) \approx_c \mathsf{SIM}^{\mathcal{U},t,k}(\lambda, w_C)$$

Security with a curious server. We also consider security against an honest-but-curious server, who can additionally combine knowledge with some honest-but-curious clients. We show that any such group of corrupted users can be simulated given the inputs of all users in this group, and only the sum of the values of the honest users. That said, the corrupted clients and the server learn nothing more than their own inputs and the sum of the inputs of the honest users.

Theorem A.2 (Semi-honest security with curious server). *For security parameter λ, an n-party protocol for an aggregation function f, is secure if there exists a PPT simulator* SIM *such that for all $k, t, \mathcal{U}$ where $t < |\mathcal{U}|$ and Server $\mathcal{S}$, all corrupted parties $C \subseteq \mathcal{U} \cup \mathcal{S}$ and all $w_{\mathcal{U}}$, which denote all secret inputs of all users in all iterations k, letting $n = |\mathcal{U}|$, then the output of* SIM *is computationally indistinguishable from the output of* $\mathsf{REAL}^{\mathcal{U},t,k}$:

$$\mathsf{REAL}_C^{\mathcal{U},t,k}(\lambda, w_{\mathcal{U}}) \approx_c \mathsf{SIM}_C^{\mathcal{U},t,k}(\lambda, w_C, z)$$

where $z = \sum_{w_u \in \mathcal{U} \setminus C}$