Cryptographically Secure Dark Pools for Anonymous Financial Transactions

Shreyan Jain, Ajinkya Nene, Josh Noel, Advaith Anand

May 16, 2018

Abstract

Dark pools are an essential component of the modern financial ecosystem that enable investors to hide information about their transactions from the investing public. However, transaction information is not kept anonymous from dark pool operators, who can thereby trade on that information and adversely affect clients. To secure against such information leakage, we design and implement a cryptographically secure dark pool that allows for fully anonymous financial transactions. We define a subscription protocol that allows users to join the dark pool and encrypt messages to each other, and we use homomorphic encryption along with multiparty computation to securely match orders. Assuming semi-honest adversaries, our system protects against transaction data leakage and provides strong anonymization guarantees to clients. Our implementation for the secure dark pool can be found on https://github.com/JoshNoel/Secure-Dark-Pool.

1 Motivation

In the current financial ecosystem, there are two main methods for trading securities: lit markets or dark pools. Lit markets are public exchanges like the NASDAQ or NYSE where all orders and transactions are issued to a publicly-available orderbook [1]. For most traders, revealing their positions such as "buy 100 shares of APPL" does not affect how the price on their positions are locked in. On the other hand, when an institutional trader such as a major hedge fund submits a large transaction such as "buy 100,000 shares of APPL", the size of their position adversely affects how their position is locked in. Public order books allow other traders to observe the large order, leaking the firm's alpha and potentially leading other firms to begin buying APPL as well. This may result in an increase in the price of APPL, making it incredibly expensive for the original investor to lock in their position.

To avoid this problem, institutional investors use dark pools, which are trading exchanges that are not accessible by the investing public and keep their order book hidden. These dark pools are operated by a few companies that perform

private order matching between a select group of clients. By making the order book inaccessible to the general public, dark pools allow investors to minimize the market impact of moving large volumes of securities in single transactions. This hiding property makes trading in these "dark" exchanges quite lucrative for investors, and as a result dark pools now account for approximately 10-20% of total equity trading volume [1].

Even though dark pools provide anonymity from the general public, they are still limited by the need to place full trust in the dark pool operators themselves. This can lead to major security concerns. In one instance, a major dark pool operator, ITG, used information from unusually large orders for certain securities along with data on returns for certain investors to make extremely profitable trades for themselves [2]. Such data exposure allows operators to adversely affect final execution prices for their own clients.

To address this issue, we propose a cryptographically secure dark pool that allows for truly anonymous financial transactions. With our proposed system, dark pool clients can be fully confident that information about their positions is hidden from everyone except for the entity they actually engaged in a trade with. In particular, our system was designed with the following security goals in mind:

- Allow investors to submit encrypted orders to a central server without leaking any information about the security being traded, the direction (buy/sell), or the transaction volume to the server.
- Match entities with complementary orders (same security, opposite directions) without leaking any information to non-matching entities.
- Securely compute transaction volume acceptable to both entities.
- Ensure client anonymity over the course of their trading history.

2 Threat Model

We assume a semi-honest adversary threat model. Semi-honest adversaries are entities that act according to system-defined protocols, but try to extract secret information from all received messages [3]. For the central dark pool server, this is a fair assumption to make since dark pool server code is publicly available and attested, and since a trusted third party is assumed to run this server side code. We also assume semi-honest clients that follow protocols defined by the server. This assumption is reasonable since all clients who subscribe to the dark pool are legitimate institutional investors who are verified by the server operator, and since client behavior must conform to a small number of acceptable actions delineated by the server.

Assuming semi-honest adversaries, we can form the following threat model regarding our dark pool server:

- The dark pool server sends correct data back to clients during order matching and volume computation but attempts to learn information about proposed trades.
- Clients try to learn information about orders placed by other clients, even when those orders do not match their own.
- Clients try to learn trading history of other anonymous clients.
- Clients try to associate entities in the dark pool with the actual institutional investors they represent.

3 Cryptographic Preliminaries

Our system uses the well known RSA-OAEP cryptosystem for secure communication between clients via the dark pool server [4]. In addition, our system also makes use of the following two cryptographic schemes.

3.1 Paillier Cryptosystem

The Paillier Cryptosystem is an asymmetric public key encryption scheme [5]. The scheme consists of the following three algorithms:

- **Key Generation.** Paillier keys are generated by choosing two large random primes p, q of equal length. Then the values $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$ are computed. Finally, an integer g is chosen from $\mathbb{Z}^*_{n^2}$ such that n divides the order of g. This can be guaranteed by ensuring that the inverse $\mu = L(g^\lambda \bmod n^2)^{-1} \bmod n$ exists, where L is the function defined as $L(u) = \frac{u-1}{n}$. Then the public key is given as $Pub = (n, g)$, and the secret private key is given as $Priv = (\lambda, \mu)$.
- **Encryption.** To encrypt some message $m \in \mathbb{Z}_n$, the sender chooses a value r uniformly at random from $\mathbb{Z}^*_n$. Then the ciphertext is computed as $Enc(m) = g^m \cdot r^n \bmod n^2$.
- **Decryption.** To decrypt some ciphertext $c \in \mathbb{Z}^*_n$, the recipient computes the message as $Dec(c) = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$.

We use the Paillier Cryptosystem because it has the additional property that it is additively homomorphic under encryption. In other words, Paillier satisfies the following two desired properties:

- **Additively Homomorphic.** Given any two messages $m_1, m_2 \in \mathbb{Z}_n$, $Enc(m_1) \cdot Enc(m_2) \bmod n^2 = Enc(m_1 + m_2 \bmod n)$.
- **Scalable.** Given any message $m \in \mathbb{Z}_n$ and any nonzero number $k \in \mathbb{Z}^*_n$, $Enc(m)^k \bmod n^2 = Enc(km \bmod n)$.

Note that these properties allow recipients of messages who don't possess the private key to compute on encrypted data without decrypting the ciphertexts. Our protocol uses homomorphic encryption to allow servers to compute on encrypted orders for order matching and to allow entities to compute on encrypted transaction volumes for volume computation [8].

3.2 Secure Multiparty Computation

Secure multiparty computation (MPC) refers to a wide range of closely related problems whose basic setting consists of n players $P_1, ..., P_n$ each holding some secret piece of data x_i. The central problem is for the players to compute some function $F(x_1, x_2, ..., x_n)$ on the data and receive the output without any player P_i learning any data share $x_j \neq x_i$ (i.e. any input that is not their own). This is typically achieved by some interactive protocol performed by the players that is provably equivalent to each P_i sending their share x_i to some trusted third party, who directly computes the function and returns the result to all players. MPC is a well-researched problem in the security literature, with several different published protocols for a wide range of basic functions F [6].

Our dark pool protocol takes advantage of one specific MPC protocol that was designed to solve a well-known cryptographic problem known as the Millionaire's Problem. First proposed by Andrew Yao in 1982, the problem involves two millionaires Alice and Bob who wish to determine which among them is the richer without divulging their own actual wealth to each other. As a MPC problem, this is equivalent to $n = 2, F(x_1, x_2) = \max(x_1, x_2)$ where max returns the larger of its two arguments [7]. Many protocols have been proposed to solve this problem, but we use one proposed by Lin due to its reliance on an additive homomorphic encryption scheme that can be easily supplied by the Paillier cryptosystem that we use [9]. Note that our particular system actually needs to compute the smaller of x_1 and x_2 for volume computation, but that is a quite simple modification. The actual protocol is described in the later section on Volume Computation.

4 System Design

We will consider our dark pool to run on an idealized central server, referred to as the *dark pool server*. Each entity that will participate in this dark pool (i.e. subscribe to the dark pool and request financial transactions) will be modeled as a single client referred to as an *entity*. We also assume the dark pool server maintains a global security parameter λ that it shares with all entities that subscribe to the dark pool. This parameter will be used to determine the lengths of primes used in all secure communication via the Paillier and RSA cryptosystems.

4.1 Ticker Embeddings

In order to encrypt and decrypt transactions, we need a suitable means to represent the security (e.g. stock, option, currency) being traded. For this purpose, we will maintain a unique mapping from the ticker or symbol for each security listed on the dark pool to a unique number from 1 to K, where K is some predetermined $(\lambda - 2)$-bit number. This mapping could be realized as a hash function or as a pseudo-random ordering of all possible tickers; our protocol is implementation-agnostic as long as the mapping is injective (no two tickers share the same embedding).

Moreover, we will add a sign to this mapped number to create a *transaction ID* for each security, with a positive sign representing a request to buy the corresponding security, and a negative sign indicating a request to sell the corresponding security. For any $order = (ticker, direction)$, we will use $order_{ID}$ to refer to the corresponding transaction ID. This mapping will be maintained by the dark pool server. Upon subscribing to the dark pool, any entity will also be able to download the mapping, allowing it to obtain the correct transaction ID for any order.

4.2 Subscription and Key Generation

Whenever a new entity decides it wants to trade on the dark pool, it will go through the following subscription procedure. First, it will be assigned a random unique entity ID i from the dark pool server. Next, it must generate a secure RSA public and private key pair for itself by randomly sampling two primes p and q each λ bits in size. It will then compute the public key (n, e) and publish it on the central dark pool server, which stores a list of all subscribed entities' public RSA keys. The entity will also compute its private RSA key d and store that locally for decryption of all incoming messages. Finally, the entity must compute a Paillier key pair for each other entity j in the dark pool. For each entity j, the new entity samples two new primes of size λ bits p' and q', derives a Paillier keypair $(Pub_{ij}, Priv_{ij})$ from the two primes, uses j's public RSA key (obtained from the dark pool server) to encrypt the private Paillier key, and sends Pub_{ij} along with $Enc_j(Priv_{ij})$ to the dark pool server, who stores Pub_{ij} and then forwards both to j.

Whenever an entity j gets a new key request of this format, it decrypts $Enc_j(Priv_{ij})$ using its private RSA key. If the key is valid (i.e. p' and q' are distinct primes of size λ bits), it accepts the key and $(Pub_{ij}, Priv_{ij})$ can henceforth be used to encrypt/decrypt messages between i and j. Otherwise, it rejects the key and i will have to repeat the process to communicate with j. Either way, j notifies i of its response via the server.

4.3 Executing Transactions

Suppose entity i now wishes to buy or sell some number V of shares of security S. In other words, it wants to perform the order $X = (S, dir)$ with volume V,

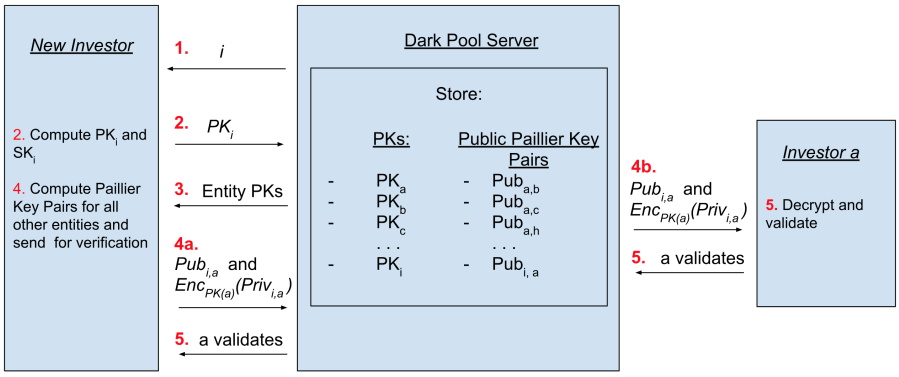

Figure 1: The Subscription and Key Generation Protocol

where *dir* is "buy" or "sell." This transaction is performed in two steps: first, a suitable partner is found, and next the actual trade volume is computed. Our protocol, thus, realizes each transactions as two steps: "order matching" and "volume computation."

4.3.1 Order Matching

First, i obtains the transaction ID X_{ID} for the proposed transaction $X = (S, dir)$ using the mapping downloaded upon subscription. Then, for each entity j that i has an active connection with, i calculates $X_{ID} \pmod{n_{ij}}$, where n_{ij} is taken from the public Paillier key Pub_{ij}. Note that ord_{ID} is a signed integer between $-K$ and K where K is of size $\lambda - 2$ bits, and each n_{ij} is an integer of size λ bits, so no two distinct transaction IDs will be mapped to the same value by this modulo operation. Finally, i encrypts the value $X_{ID} \pmod{n_{ij}}$ using the public Paillier key Pub_{ij} and sends $Enc_{ij}(X_{ID})$ to the dark pool server as a transaction proposed to entity j. (Note that we use Enc_{ij} to denote encryption using the Paillier key agreed upon by entities i and j and omit the modulus.)

Upon receiving proposed transactions from multiple entities, the dark pool server runs the following matching algorithm. The server considers all transactions proposed between each pair of entities i and j. Suppose i proposed some transaction $Enc_{ij}(X_{ID})$ to j, and that j proposed some transaction $Enc_{ij}(Y_{ID})$ to i. Then the server will choose some random nonzero number k from $\mathbb{Z}^*_{n_{ij}}$, compute the value $E' = (Enc_{ij}(X_{ID}) \cdot Enc_{ij}(Y_{ID}))^k \pmod{n_{ij}^2}$, and send this new value E' to both i and j. Each of i and j will then compute $Dec_{ij}(E') \pmod{n_{ij}}$ and if the decrypted value equals 0, both entities will be sure that their two proposed transactions "match" - in other words, they represent proposed transactions for the same security but in different directions.

To see this, note that since the Paillier scheme is additively homomorphic, we have that $E' = (Enc_{ij}(X_{ID}) \cdot Enc_{ij}(Y_{ID}))^k \pmod{n_{ij}^2} = (Enc_{ij}(X_{ID} + Y_{ID}))^k \pmod{n_{ij}^2} = Enc_{ij}(k(X_{ID} + Y_{ID}))$. Thus $Dec_{ij}(E') = 0 \pmod{n_{ij}}$ implies

$Dec_{ij}(Enc_{ij}(k(X_{ID} + Y_{ID}))) = 0 \pmod{n_{ij}}$, or $k(X_{ID} + Y_{ID})) = 0 \pmod{n_{ij}}$. But because of how k was chosen, we must have $k \neq 0 \pmod{n_{ij}}$, so the two proposed transactions X_{ID} and Y_{ID} will match if and only if they add to 0. But by construction of the ticker embedding, this is true if and only if the two transactions X_{ID} and Y_{ID} represent different directions on the same security. Thus our protocol performs matching correctly. At this point, the two entities i and j can move on to the next stage, volume computation.

4.3.2 Volume Computation

Note that the matching stage only included computations involving the security and direction. In this stage, the two paired entities i and j complete the transaction by agreeing upon an order volume and submitting the transaction to their brokers. Note that this problem is essentially equivalent to the millionaire's problem: assuming i and j each have some upper limit on the desired transaction volume, the actual transaction volume is computed as the lower of these two values without leaking the actual values to the other entity.

First both entities individually decide their desired maximum transaction volumes. Call i's desired volume x and j's y. Note that both x and y are positive integers. The entities then compute a length-n bit encoding of these values. In other words, i calculates the binary representation $x = x_n x_{n-1} \cdots x_1$ adding leading 0's if necessary, and similarly j calculates the binary representation $y = y_n y_{n-1} \cdots y_1$. Here n is a globally defined parameter decided by the dark pool server such that the maximum single transaction volume permitted is 2^n. Since typical transaction volumes never exceed 1 million, setting $n = 20$ would be sufficient for this system.

Next, i generates a fresh Paillier key pair $(Pub, Priv)$ of size λ (call the associated encryption and decryption functions Enc' and Dec'). It then creates a 2 by n table A by setting, for all $1 \leq i \leq n$, $A[x_i][i]$ to $Enc'(0)$ and $A[\tilde{x}_i][i]$ to $Enc'(r)$ where r is some randomly chosen number and $\tilde{x}_i$ denotes the bit complement of x_i. The table is then sent by i to j via the dark pool server. Upon receiving A, j creates n values c_i using the following process: for each $1 \leq i \leq n$, if $y_i = 0$, set c_i to $(A[y_n][n] \times A[y_{n-1}][n-1] \times \cdots A[y_{i+1}][i+1] \times A[1][i])^k$ where k is some randomly chosen number; otherwise, set c_i to some randomly chosen nonzero number. Then j randomly permutes the n values $c_1, ..., c_n$ and sends $\pi(c_1, ..., c_n)$ to i via the dark pool server. Finally, i decrypts each $c_{\pi(i)}$ as $m_i = Dec'(c_{\pi(i)})$ and checks if any $m_i = 0$. If some decrypted value equals 0, i concludes that $x > y$ and notifies j that y is the agreed upon trade volume; otherwise i notifies j that x is the agreed upon trade volume and sends $Enc_j(x)$ to j. At this point, the two entities can carry out the actual transaction via their brokerages.

Note that i could lie in two scenarios. First, it turns out that $x > y$ but i tells j that x was the lesser value. But this would fail immediately as j could instantly detect that $x > y$ upon decrypting $Enc_j(x)$. Second, it turns out that $x < y$ but i tells j that y was the lesser value. This lie would go undetected but ultimately causes i to submit a larger transaction then it initially proposed,

which doesn't adversely affect j in any way.

Finally, we prove that the above protocol correctly solves for $\min(x, y)$. More rigorously, we prove that $x > y$ if and only if some $c_{\pi(i)}$ decrypts to 0. First, assume there exists some $c_{\pi(i)}$ such that $Dec'(c_{\pi(i)}) = 0$. With all but negligible probability, this implies that $c_{\pi(i)}$ was computed as

$$c_{\pi(i)} = (A[y_n][n] \times A[y_{n-1}][n-1] \times \cdots A[y_{i+1}][i+1] \times A[1][i])^k \tag{1}$$

(a $c_{\pi(i)}$ that resulted instead by choosing some random value would have very low probability of decrypting to 0). This also implies, by the way j assigns c_i's, that $y_i = 0$. Next, note that, for $k > i$, $A[y_k][k] = Enc'(0)$ whenever $y_k = x_k$, and otherwise equals $Enc'(r)$ where r is some random value. Then, for $k > i$, let r_k be a random variable that equals 0 whenever $y_k = x_k$ and otherwise equals some random r. Similarly, $A[1][i] = Enc'(0)$ if and only if $x_i = 1$, so define r_i to be 0 if $x_i = 1$ and otherwise be some random r. Then we can write

$$Dec'((A[y_n][n] \times A[y_{n-1}][n-1] \times \cdots \times A[y_{i+1}][i+1] \times A[1][i])^k) = 0 \tag{2}$$

$$Dec'((Enc'(r_n) \times Enc'(r_{n-1}) \times \cdots \times Enc'(r_{i+1}) \times Enc'(r_i))^k) = 0 \tag{3}$$

$$Dec'((Enc'(k(r_n + r_{n-1} + ... + r_{i+1} + r_i))) = 0 \tag{4}$$

$$k(r_n + r_{n-1} + ... + r_{i+1} + r_i) = 0 \tag{5}$$

by the homomorphic properties of Paillier encryption. Since k is nonzero, with very high probability the last equation holds if and only if each $r_i = 0$. But this is true if and only if $x_k = y_k$ for all $i < k \leq n$ and $x_i = 1$. Together, these facts along with $y_i = 0$ imply $x > y$ as desired.

Next we prove the reverse direction. Suppose $x > y$. Let i be the first (i.e. largest) index at which they differ, i.e. $x_j = y_j$ for all $i < j \leq n$. Since $y < x$, we must have $y_i = 0$ and $x_i = 1$. Then note that, since $y_i = 0$, j will compute c_i as

$$(A[y_n][n] \times A[y_{n-1}][n-1] \times \cdots A[y_{i+1}][i+1] \times A[1][i])^k \tag{6}$$

But, for all $i < j \leq n$, $x_j = y_j$ so we have $A[y_j][j] = A[x_j][j] = Enc'(0)$. And since $x_i = 1$, we have $A[1][i] = A[x_i][i] = Enc'(0)$ as well. Then we have

$$c_i = (Enc'(0) \times Enc'(0) \times \cdots \times Enc'(0))^k \tag{7}$$

By additive homomorphism, this becomes $c_i = Enc'(k(0+0+...+0)) = Enc'(0)$. Then i will decrypt $c_{\pi(i)}$ to 0 as desired.

4.4 Partial Order Filling

Note that our current protocol doesn't allow the dark pool to fulfill large volume orders by matching it with multiple orders. In other words, if i wishes to sell 100,000 shares of some stock but the entity it matches with only wishes to buy 100 shares, only part of i's order will be filled. In this case, entity i can simply propose the transaction again, repeating this process until the

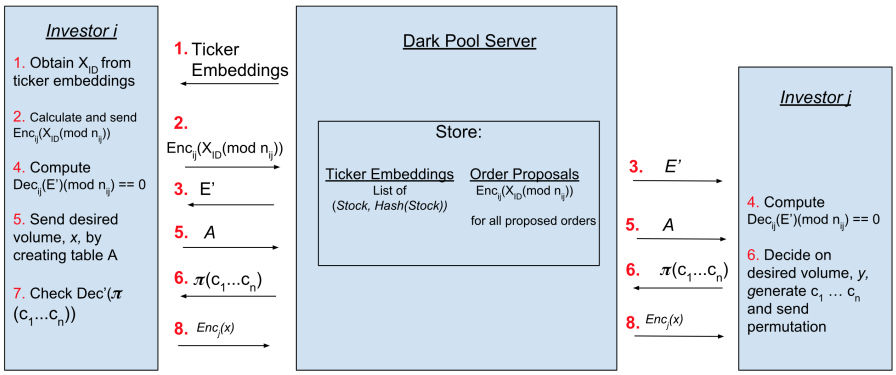

Figure 2: Order Matching and Volume Computation. Note that in this example, Investor i is sending volume and j is receiving, but the opposite could also happen where j sends and i receives volume. In this example, $x < y$, so i sends $Enc_j(x)$ in Step 8.

order is completely filled as a series of individual transactions. For performance gains, the dark pool server will pair any proposed transaction with all other transactions that "match" it instead of just one. This enables larger orders to be split up into multiple transactions in parallel. Once i completes its full order, it can simply withdraw its request for that order.

4.5 Reanonymization Protocol

In the current protocol, entities can associate certain information (i.e. which entities are engaging in particular transactions) to the entity IDs. In order to avoid any information leakage of this sort, the dark pool server will undergo a re-anonymization protocol periodically (i.e. at the end of every day). In this protocol, every entity will be assigned a new, unique, random entity ID, and entities will undergo the subscription protocol all over again to generate fresh RSA and Paillier keys.

5 Proof of Security

In the following subsections, we show that our protocol is provably secure in the semi-honest adversary model.

5.1 Subscription

In the subscription protocol, each entity generates fresh RSA and Paillier keys for secure communication. We prove that all keys generated this way are secure in the semi-honest adversary model. First, note that RSA private keys are generated locally by entities and kept on their own machines. Thus, RSA keys

are never compromised. Moreover, assuming a semi-honest server, we can be confident that all entities will share the correct public RSA keys.

In the second stage of subscription, new entities generate fresh Paillier keypairs and send the public key along with the RSA-encrypted private key to the server, which forwards it to the corresponding entities. Assuming the RSA hypothesis, since RSA private keys are never compromised the server or any unintended recipients will not be able to recover the private Paillier keys transmitted in this way. Moreover, in the semi-honest adversary model, the server does not manipulate data sent by entities or send fake data, so all entities that receive encrypted Paillier keys can be sure the data originated from a valid entity. Thus, the only Paillier private keys used by our system will be genuine keys that are stored only by the two entities who use them to communicate.

5.2 Order Matching

In the first stage of transaction execution, entities send $Enc_{ij}(X_{id})$ to the dark pool server, where X_{id} is the unique embedding for transaction $X = (S, dir)$. Note that if any actor can recover X_{id}, they can use the universal ticker embedding to invert X_{id} and learn both the security and direction. First, we show that the dark pool server cannot recover X_{id} from $Enc_{ij}(X_{id})$ except with negligible advantage. This is true by the fact that the dark pool server never obtains the private Paillier key $Priv_{ij}$, and by the fact that Paillier encryption is IND-CPA secure, so the server cannot distinguish between $Enc_{ij}(X_{id})$ and $Enc_{ij}(X')$ except with negligible advantage, where X' is any other valid transaction ID.

Next, we show that, except with negligible probability, an entity j cannot recover X_{id} from $(Enc_{ij}(X_{id}) \cdot Enc_{ij}(Y_{id}))^k$ unless $X_{id} + Y_{id} = 0$. We refer to this property as *multiplicative hiding*. Suppose $X_{id} + Y_{id} \neq 0$. Then j receives $(Enc_{ij}(X_{id}) \cdot Enc_{ij}(Y_{id}))^k = Enc_{ij}(k(X_{id} + Y_{id}))$ and thus can obtain $k(X_{id}+Y_{id}) \bmod n_{ij}$. Note that k is a value randomly chosen by the server and thus not accessible to j. Observe that $k(X_{id}+Y_{id}) \bmod n_{ij}$ is indistinguishable from $k'(X'+Y_{id}) \bmod n_{ij}$, where $k' = k(X_{id}+Y_{id})(X'+Y_{id})^{-1} \bmod n_{ij}$ and X' is any valid transaction ID not equal to X_{id}, whenever $(X' + Y_{id})$ is invertible mod n_{ij}. Thus the value that j receives can be opened to any valid transaction ID X' such that $(X'+Y_{id})$ is invertible mod n_{ij}. This property holds whenever $(X' + Y_{id})$ is relatively prime to n_{ij}. But $n_{ij} = pq$ where p and q are both very large primes, so the value j receives can be opened to about $\frac{p-1}{p} \cdot \frac{q-1}{q}$ of all possible transaction IDs, or nearly all of them. Thus j cannot distinguish between X_{id} and X' except with negligible probability, which shows that the order matching scheme hides all transaction information whenever orders fail to match.

5.3 Volume Computation

Finally we show that the volume computation scheme is secure as well. Note that for this scheme, i generates a fresh Paillier keypair and keeps the private key secret. Now note that, under the semi-honest adversary model, both the dark

pool server and j receive the correct, unmodified table A from i. Since Paillier encryption is IND-CPA secure, for any i neither the server nor j will be able to distinguish between $A[x_i][i] = Enc'(0)$ and $A[\tilde{x}_i][i] = Enc'(r)$ except with negligible advantage, so neither will be able to guess x_i with probability greater than $1/2 + negl$. Then neither will be able to guess x with probability greater than $(1/2+negl)^n \approx (1/2)^n$, which is no better than guessing x randomly. Thus i's desired volume is secure under the semi-honest adversary model.

Next, we show that i learns nothing more about y except for whether or not it is less than x assuming a semi-honest j and server that correctly transmit all values. Suppose i concludes $x > y$. Then i received k values that decrypted to 0, where $1 \leq k \leq n$. These k values correspond to the fact that there are exactly k values of i such that $x_j = y_j$ for all $i < j \leq n$ and $y_i = 0$. The only way this is true is if the first $k-1$ bits of both x and y are 0, $x_{n-k} = 1$, and $y_{n-k} = 0$. Note that entity i learns nothing new about y from this: since it already knew $y < x$ and that the first $k-1$ bits of x were 0, it was necessarily true that the first k bits of y were 0. Moreover, the other $n-k$ values that i receives are indistinguishable from random values under the IND-CPA property of Paillier, since j uses exponentiation with a random exponent k. Now suppose i concludes $x < y$. Then i receives no values that decrypt to 0 and instead receives n values that are indistinguishable from random values, telling it nothing about y.

6 Evaluation

6.1 Security Goals

As stated in our threat model, we are assuming a semi-honest adversary. With this in mind, we succeed in achieving each of our stated security goals from Section 1. First, through the use of Paillier and RSA-OAEP encryption, investors can securely submit orders to the central server without leaking information about their position. In addition, through multiplicative hiding, the system hides all order information from non-matching entities, and through multiparty computation, acceptable transaction volumes are computed without revealing the desired volumes upfront. Furthermore, clients are only identified by their randomly assigned entity ID assigned by the server and their public RSA keys, guaranteeing them true anonymity when interacting with the system. Moreover, through the reanonymization protocol, our system gives dark pool clients anonymity over the course of their trading history, as it is impossible to track who belongs to which RSA public key after a key refresh.

6.2 Vulnerabilities

Although our system is provably secure in the semi-honest adversary model, we now analyze some remaining security vulnerabilities in the system and possible mitigations.

- **DoS Attacks.** Our system is certainly vulnerable to DoS or Denial of

Service attacks, in which one or more clients could spam the central dark pool server with a series of order requests, increasing the load on the server until it is finally brought down. This can be mitigated in two ways. First, although we described the dark pool as a single server, in reality the system would be practically realized as a distributed system running on several servers, allowing the system to handle additional load. Second, the system could set an upper limit on the number of allowed order proposals per entity each day, limiting the total load on the system to an acceptable amount.

- **Fraudulent Order Requests.** Even in the semi-honest adversary model, clients can can obtain side channel information from the dark pool by submitting many trade requests and seeing which ones match to learn what other entities are actively trading. One mitigation would be to have the server flag an entity whenever it proposes a trade that matches but doesn't carry out the actual transaction. Since the server associates entities with their true identities, they can penalize adversarial behavior by, for example, banning the entity from participating. Note that entities could work around this by proposing extremely small volume trades that cost them very little, thus bypassing the flag and succeeding in the goal of identifying other entities' trades. Nevertheless, setting an upper limit on proposed orders and a lower limit on transaction volume would make such adversarial behavior less feasible.

- **Malicious Server.** Note that if we relaxed the semi-honest adversary assumption, a malicious server could encrypt fake Paillier keys that it generates itself and forward it to the entities. This would allow it to listen to any orders sent by those entities, decrypt them, and obtain information about those entities' desired positions. Moreover, a malicious server could also spawn fake clients to act as entities and perform fake transactions with extremely high transaction volumes to learn everything about other clients' trading behavior. Our underlying assumption of a trustworthy server makes such behavior highly unlikely.

6.3 Proof of Concept

Our implementation can be found at: https://github.com/JoshNoel/Secure-Dark-Pool. Our current implementation supports an arbitrary number of clients attempting to complete an arbitrary number of trades. We tested our proof of concept implementation in a simulated environment in which fake clients communicate with the dark pool server. The server and clients all run on the same local machine, so network overhead is not tested. The current proof of concept completely implements the system described thus far.

During the start of every key-refresh period, clients are allowed to register for a set amount of time. When a client registers with their public key the client assigns the server their UID along with a port number to which they

should establish a socket connection to the server through which all future communication occurs. Once the registration period is over, the server launches a subprocess for every client and assigns a client in every pair of clients to generate the pair's Paillier key pair. Once the key exchange is over the server begins accepting and processing trades.

In regards to performance, registration through key exchange scales linearly with the number of clients registering as one client in every pair of clients must generate a Paillier key pair. The generation and distribution of the Paillier key pairs is the slowest part of this process. With two clients this takes 2.1 seconds and with four it takes 3 seconds. Though slow, this process only occurs once per key-refresh and can therefore be amortized across a long time-period.

The submission and matching of trades happens very quickly with 0.15ms round-trip time from submission to match when complementary trades are submitted simultaneously. However, the performance and bandwidth requirements of this step scale as n^2 where n is the number of outstanding trades since, for a given client, the client process on the server must multiply every pair of trades proposed by all other clients.

Volume calculation between matched entities is by far the slowest step during the trading process, though it does take constant time. Within this step both matched clients must calculate and send four rather large messages to the other, two of which involve large homomorphic operations (specifically calculating the table A and values c_i). In the benchmarks we ran, this took 3.2 seconds to complete, though this could be overlapped with the submission of new trades. Moreover, in the future the server could be extended to support clients computing the volumes for multiple trades at once.

7 Future Work

In the situation that our secure dark pool is deployed into the financial markets, there may be a need to disclose trades after a period time as per SEC regulations. To accomplish this, we would add a commitment scheme using Pederson commitments that allows for the secure logging of transactions on the dark pool server. Whenever the SEC requires logs of transactions that went through the dark pool, the server would ask for all entities to reveal their commitment keys to decrypt the log and submit it to the SEC for filing.

Next, our current implementation has large bandwidth requirements to transmit ciphertexts and has performance bottlenecks due to slow encryption processes like homomorphic encryption. In the future, we will focus on making our system more performant by reducing unneeded complexity in our encryption protocols and reducing bandwith requirements by reducing the amount of data required for the system to operate.

Finally, our threat model assumes semi-honest adversaries. To make our security guarantees stronger, we could allow for clients that deviate from client specifications, such as clients that submit high numbers of erroneous requests to deny access to the system. We will do this by adding more checks in our system

against clients to validate that they act according to correct and predetermined behavior.

8 Conclusion

In this project, we developed a dark pool system which allows for truly anonymous transactions. Current dark pool protocols allow the operator of a dark pool to see all of the order information of clients using the exchange, which can result in the operators trading on this private information. Our system allows for secure communication through encryption, complementary order matching through homomorphic encryption, and transaction volume computation through a secure multi-party computation protocol. Using our suite of protocols, we are able to guarantee information security for dark pool clients against semi-honest adversaries. We were able to successfully implement a proof of work of our system and simulate active trades with multiple clients, showing that such a system could be practically feasible with some performance improvements.

9 Acknowledgments

We would like to thank our instructors, Prof. Ronald Rivest and Prof. Yael Kalai for running a highly instructive and interesting class as well as for their guidance on our project. In addition, we would like to thank the 6.857 TA's for their useful advice on how to approach our final project.

References

[1] Picardo, CFA Elvis. "An Introduction to Dark Pools." Investopedia, Investopedia, 5 May 2017

[2] Levine, Matt. "ITG Hid a Secret Trading Desk in Its Dark Pool." Bloomberg.com, Bloomberg, 12 Aug. 2015

[3] C. Hazay, Y. Lindell: A Note on the Relation between the Definitions of Security for Semi-Honest and Malicious Adversaries. IACR Cryptology ePrint Archive 2010/551

[4] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. Journal of Cryptology, 17(2):81– 104, 2004.

[5] Paillier, Pascal (1999). "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". EUROCRYPT. Springer. pp. 223–238. doi:10.1007/3-540-48910-X_16.

[6] P. Bogetoft, D. L. Christensen, I. Damgard, M. Geisler, T. Jakobsen, M. Kroigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M.

Schwartzbach, and T. Toft, "Secure multiparty computation goes live," Cryptology ePrint Archive, Report 2008/068, February 2008.

[7] M. Jakobsson and A. Juels, "Mix and match: secure function evaluation via ciphertexts," in Asiacrypt '00: proceedings of the 6th international conference on the theory and application of cryptology and information security, London, UK, 2000, pp. 162-177.

[8] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. CoRR, abs/1704.03578, 2017.

[9] Y. Lin and W.-G. Tzeng. An efficient solution to the millionaires' problem based on homomorphic encryption. In ACNS, 2005.