FHE Single Price Auction

Alexandre Belhoste

December 23, 2024

1 Single-price auction

1.1 Definitions

Bidder setup

- Let B_i denote the i-th bidder participating in a single-price auction with N bidders, where $i \in \{1, 2, \ldots, N\}$.
- Let $I_N = \{1, 2, \ldots, N\}$ denote the **index set** of bidders. We refer to I_N as the set of all bidder indices.
- Let $\mathcal{B} = \{B_1, B_2, B_3, \ldots, B_N\}$ denote the **bidder set**, which represents all participants in the auction. Each B_i corresponds to the i-th bidder.

Bid definitions

- Let q_i denote the **quantity of tokens** that bidder B_i wishes to purchase.
- Let p_i denote the **unit price** that bidder B_i is willing to pay for each token.

Auction definitions

- Let $\mathcal{P}_{all}$ denote the **set of all bid prices** in the auction, including repeated bids at the same price. Formally:

$$\mathcal{P}_{all} = \{p_i \mid i \in I_N\}$$

- Let S_p denote the **set of bidders** who bid at price p. This is formally defined as the **equivalence class** of p:

$$S_p = \{i \mid p_i = p,\, i \in I_N\}$$

- Let Q_p denote the **total quantity of tokens bid** at price p. It is defined as:

$$Q_p = \sum_{i \in S_p} q_i$$

Set of distinct pid prices

- Let $\mathcal{P}_{bids}$ denote the **set of distinct prices** bid in the auction (i.e., prices with one or more bidders). Formally:

$$\mathcal{P}_{bids} = \{p \mid \mathrm{Card}(S_p) > 0\}$$

- Let $K = \mathrm{Card}(\mathcal{P}_{bids})$ be the number of distinct bid prices.
- Let $\mathcal{P}_{bids}$ be represented as a **sorted list** of distinct prices in **decreasing order**:

$$\mathcal{P}_{bids} = \{p_1^{(b)}, p_2^{(b)}, \ldots, p_K^{(b)}\} \text{ where } p_1^{(b)} > p_2^{(b)} > \cdots > p_K^{(b)}$$

Cumulative quantity of tokens

- Let C_k denote the **cumulative quantity of tokens bid** up to price $p_k^{(b)}$. It is defined recursively as:

$$C_0 = 0,$$
$$C_k = \sum_{i=1}^{k} Q_{p_i^{(b)}} \quad \text{for } 1 \le k \le K$$

1.2 Bid validation

In the remainder of the problem, a bid is considered **valid** if and only if both its quantity and price are strictly positive. An **invalid** bid is equivalent to a bid with both price and quantity set to zero. Additionally, no bid quantity should exceed the total quantity of tokens available for sale, Q. This process is formally defined as follows:

$$\forall i \in I_N, \quad \begin{cases} p_i > 0 \text{ and } 0 < q_i \leq Q, & \text{if the bid is valid,} \\ p_i = 0 \text{ and } q_i = 0, & \text{if the bid is invalid.} \end{cases}$$

1.3 Uniform price

In a **uniform price auction**, the **uniform price** $p_u^{(b)}$ is the smallest price such that the cumulative quantity of tokens bid satisfies the total sold quantity Q. Formally:

$$p_u^{(b)} \text{ is the uniform price such that } 1 \leq u \leq K \quad \text{and} \quad C_{u-1} < Q \leq C_u$$

1.4 Allocation

Case 1: Exact match $C_u = Q$ Each winning bidder B_i receives exactly the quantity q_i they bid for because the total cumulative demand equals the supply. Formally:

$$\forall i \in I_N \quad q_i^* = \begin{cases} q_i & \text{if } \; p_i \geq p_u^{(b)} \\ 0 & \text{if } p_i < p_u^{(b)} \end{cases}$$

Case 2: $C_u > Q$ with a single bidder at $p_u^{(b)}$ Since only one bidder bids at $p_u^{(b)}$, this bidder's token quantity must be partially fulfilled to satisfy the total available token quantity Q.

$$\forall i \in I_N \quad q_i^* = \begin{cases} q_i & \text{if } \; p_i > p_u^{(b)} \\ Q - C_{u-1} & \text{if } \; p_i = p_u^{(b)} \\ 0 & \text{if } p_i < p_u^{(b)} \end{cases}$$

Case 3: $C_u > Q$ with multiple bidders at $p_u^{(b)}$ When the cumulative quantity C_u exceeds the total available quantity Q, the remaining quantity $Q - C_{u-1}$ must be allocated among multiple bidders tied at price $p_u^{(b)}$. We propose the following four tie-breaking rules:

- FHE-compliant rules:
  - **Price, quantity and bid placement**: Bidders at price $p_u^{(b)}$ are sorted based on their quantity and register ID (or timestamp)
  - **Price and bid placement**: Bidders at price $p_u^{(b)}$ are sorted based on their register ID (or timestamp).
  - **Price and randomization**: A unique winning bidder among those at price $p_u^{(b)}$ is randomly selected.
- Non-FHE-compliant rules:
  - **Pro-rata quantity allocation**: The remaining total token quantity $Q - C_{u-1}$ is allocated **proportionally** to the quantities requested by each bidder at $p_u^{(b)}$. This rule is **not FHE-compliant** since it requires the FHE computation of integer divisions.

2 FHE tie-breaking using a total strict order relation

In auction theory, it is essential to establish a tie-breaking rule to resolve situations where two or more bidders are tied (e.g., when they bid the same price). In the context of an FHE auction, the chosen tie-breaking rule must be FHE-compatible.

One way to achieve this is by using an FHE-compliant **total strict order** relation over the set of bidders $\mathcal{B}$, which eliminates any possible ties, preserves the final uniform price $p_u^{(b)}$, and transforms any situation where $C_u > Q$ with multiple bidders at $p_u^{(b)}$ into a solvable case with only a single bidder at $p_u^{(b)}$.

The final quantity allocation is performed according to the bid order induced by $>$ until all remaining tokens are sold. The last winning bidder's token quantity may be partially fulfilled to match the total available token quantity Q. This method ensures that the final uniform price is also $p_u^{(b)}$.

2.1 Bid placement order and uniqueness

At the start of the auction, each bidder B_i is assigned a **unique registration value** id_i that reflects the order in which they placed their bid. This value can be represented by a **register ID** or **timestamp**, ensuring each bidder has a unique and comparable placement value. The following properties hold for id_i:

Uniqueness: For any two bidders B_i and B_j:

$$id_i = id_j \iff i = j$$

This ensures that each bidder has a unique registration value.

Descending Order: The registration values id_i are assigned such that:

$$id_i > id_j \iff i < j$$

This means that bidder B_1 placed their bid first, and bidder B_N placed their bid last. As a result, we assume that the identity relation holds for all bidders in $\mathcal{B}$, expressed as:

$$B_i = B_j \iff i = j \iff id_i = id_j$$

2.2 Total strict order relations

Below, we introduce three different strict order relations such that:

1. The set of bidders $\mathcal{B}$ is totally ordered, meaning that:

$$\forall i, j \in I_N \quad i \neq j \iff B_i > B_j \text{ or } B_j > B_i$$

2. The final uniform price $p_u^{(b)}$ is preserved.

2.2.1 Order by price, quantity, and bid placement

$$\forall i, j \in I_N,\ B_i > B_j \iff \begin{cases} p_i > p_j, \\ \text{or} \\ p_i = p_j \text{ and } q_i > q_j, \\ \text{or} \\ p_i = p_j \text{ and } q_i = q_j \text{ and } id_i < id_j \end{cases}$$

This defines a **total strict order** on the set of bidders. Specifically:

- Bidders with higher prices are ranked higher.
- If the prices are equal, the bidder with the higher quantity is ranked higher.
- If both price and quantity are equal, the bidder with the earlier bid placement (lower id_i) is ranked higher.

2.2.2 Order by price and bid placement

$$\forall i,j \in I_N,\ B_i > B_j \iff \begin{cases} p_i > p_j, \\ \text{or} \\ p_i = p_j \text{ and } id_i < id_j \end{cases}$$

This defines a **total strict order** on the set of bidders. Specifically:

- Bidders with higher prices are ranked higher.
- If the prices are equal, the bidder with the earlier bid placement (lower id_i) is ranked higher.

2.2.3 Order by price and randomization

Let $rand(i)$ be a random value uniquely assigned to each bidder B_i, used for tie-breaking in the order relation.

$$\forall i,j \in I_N,\ B_i > B_j \iff \begin{cases} p_i > p_j, \\ \text{or} \\ p_i = p_j \text{ and } rand(i) > rand(j) \end{cases}$$

This defines a **total strict order** on the set of bidders. Specifically:

- Bidders with higher prices are ranked higher.
- If the prices are equal, the bidder with the highest random value is ranked higher.

3 FHE precomputations

3.1 Bid validation

3.1.1 Upper bounds

To ensure that no operation results in arithmetic overflow, the following conditions must be satisfied:

$$N < 2^{16}$$
$$\sum_{i=1}^{N} q_i < 2^{256}$$

which can be simplified into the stricter condition:

$$N < 2^{16} \quad \text{and} \quad Q < 2^{240} \quad \text{and} \quad \forall i \in I_N,\ q_i \leq Q$$

By imposing this condition on each bidder, we can handle the worst-case scenario without arithmetic overflow:

$$\begin{cases} N = 2^{16} - 1 & \text{unique bidders participating in the auction,} \\ Q = 2^{240} - 1 & \text{tokens available for sale in the auction,} \\ q = 2^{240} - 1 & \text{quantity of tokens bid by each bidder.} \end{cases}$$

3.1.2 Quantity clamping

Ensure that bid quantities do not exceed the total available quantity Q:

$$\forall i \in I_N, \quad q_i := \min(Q, q_i)$$

3.1.3 Validation

Update bids to reflect validity conditions:

$$(p_i, q_i) := \begin{cases} (0,0), & \text{if } p_i = 0 \text{ or } q_i = 0, \\ (p_i, q_i), & \text{otherwise.} \end{cases}$$

3.2 Price Matrices $\mathbf{P_{eq}}$, $\mathbf{P_{ge}}$, $\mathbf{P_{gt}}$

3.2.1 Definitions

- Let $\mathbf{P_{eq}} = (\mathbf{P_{eq}}[i,\ j])_{\ 1\leq i,j\leq N}$ denote the price equality matrix on $\mathcal{B}$, whose entries are defined as follows:

$$\forall i, j \in I_N, \quad \mathbf{P_{eq}}[i,j] = \begin{cases} 1 & \text{if } \ p_i = p_j \\ 0 & \text{otherwise} \end{cases}$$

- Let $\mathbf{P_{ge}} = (\mathbf{P_{ge}}[i,\ j])_{\ 1\leq i,j\leq N}$ denote the price comparison matrix on $\mathcal{B}$, whose entries are defined as follows:

$$\forall i, j \in I_N, \quad \mathbf{P_{ge}}[i,j] = \begin{cases} 1 & \text{if } \ p_i \geq p_j \\ 0 & \text{otherwise} \end{cases}$$

- Let $\mathbf{P_{gt}} = (\mathbf{P_{gt}}[i,\ j])_{\ 1\leq i,j\leq N}$ denote the price comparison matrix on $\mathcal{B}$, whose entries are defined as follows:

$$\forall i, j \in I_N, \quad \mathbf{P_{gt}}[i,j] = \begin{cases} 1 & \text{if } \ p_i > p_j \\ 0 & \text{otherwise} \end{cases}$$

Which can be simplified:

$$\mathbf{P_{eq}}[i,j] = \begin{cases} 1 & \text{if } \ i = j \\ p_i = p_j & \text{if } \ i < j \\ \mathbf{P_{eq}}[j,i] & \text{if } \ i > j \end{cases}$$

$$\mathbf{P_{gt}}[i,j] = \begin{cases} 0 & \text{if } \ i = j \\ p_i > p_j & \text{if } \ i < j \\ \neg(\mathbf{P_{gt}}[j,i] \vee \mathbf{P_{eq}}[i,j]) & \text{if } \ i > j \end{cases}$$

$$\mathbf{P_{ge}}[i,j] = \begin{cases} 1 & \text{if } \ i = j \\ \mathbf{P_{gt}}[j,i] \vee \mathbf{P_{eq}}[j,i] & \text{if } \ i > j \end{cases}$$

3.2.2 FHE Cost

Operations	$\mathbf{P_{eq}}$	$\mathbf{P_{ge}}$	$\mathbf{P_{gt}}$
fheEq(U256)	$N(N-1)/2$	0	0
fheGt(U256)	0	0	$N(N-1)/2$
fheOr(Bool)	0	$N(N-1)$	$N(N-1)/2$
fheNot(Bool)	0	0	$N(N-1)/2$
FHE Units	$2N(N-1)$	$N(N-1)$	$11N(N-1)/2$

3.3 Quantity Matrices $\mathbf{Q_{eq}}$, $\mathbf{Q_{ge}}$, $\mathbf{Q_{gt}}$

Similarly, we define the quantity matrices $\mathbf{Q_{eq}}$, $\mathbf{Q_{gt}}$ and $\mathbf{Q_{ge}}$ in the same manner.

3.4 Random Matrix $\mathbf{Rand_{gt}}$

To perform the auction allocation using price and randomization, we assign each bidder B_i a unique random value to serve as a tie-breaking rule between bidders.

3.4.1 Definition

- Let $rand(i)$ denote a bijective function that assigns a unique random value to each bidder B_i. It is formally defined as follows:
$$rand : I_N \to I_N \forall i, j \in I_N, \quad i = j \iff rand(i) = rand(j)$$

- Let $\mathbf{Rand_{gt}} = (\mathbf{Rand_{gt}}[i,\ j])_{\ 1 \leq i,j \leq N}$ denote random value comparison matrix on $\mathcal{B}$. The entries $\mathbf{Rand_{gt}}[i, j]$ are defined as follows:
$$\forall i, j \in I_N, \quad \mathbf{Rand_{gt}}[i, j] = \begin{cases} 1 & \text{if } \ rand(i) > rand(j) \\ 0 & \text{otherwise} \end{cases}$$

3.4.2 FHE Fisher-Yates Shuffle Algorithm

One way to compute the matrix $\mathbf{Rand_{gt}}$ is by shuffling the set $\{1, 2, \ldots, N\}$ using the $O(N)$ Fisher-Yates shuffle algorithm.

3.4.3 Solidity sample code

```
// returns true if i is a power of 2
function isPowerOfTwo(uint16 i) returns (bool);

function swap(uint16 a, euint16 b, euint16[] memory arr) {
    for(uint16 i = 0; i < N; ++i) {
        ebool b_eq_i = TFHE.eq(b, i);
        // arr[b] = arr[a];
        arr[i] = TFHE.ifThenElse(b_eq_i, arr[a], arr[i]);
        // arr[a] = arr[b]
        arr[a] = TFHE.ifThenElse(b_eq_i, arr[i], arr[a]);
    }
}

function shuffle(euint16[] memory arr) {
    for(uint16 i = N-1; i >= 1; --i) {
        euint16 j;
        // j = random integer such that 0 <= j <= i
        if (isPowerOfTwo(i+1)) {
            // returns [0, i+1) = [0, i]
            j = TFHE.randEuint16(i+1);
        } else {
            euint16 rnd = TFHE.randEuint16();
            // returns [0, i+1) = [0, i]
            j = TFHE.rem(rnd, i+1);
        }
        swap(i, j, arr);
    }
}

function rand(uint16 i) returns(euint16) {
    return shuffledArray[i];
}
```

3.4.4 FHE Cost

Operations	swap	shuffle	$\mathbf{Rand_{gt}}$
fheEq(U16)	N	0	0
fheIfThenElse(U16)	$2N$	0	0
fheRand(U16)	0	$N-1$	0
fheRem(U16)	0	$N-1$	0
fheGt(U16)	0	0	N^2
FHE Units	$6N$	$(6N+28)(N-1)$	$2N^2$

3.5 Bid Order Comparison Matrix $\mathbf{B_{gt}}$

Given a strict order relation $>$ on $\mathcal{B}$, we define $\mathbf{B_{gt}} = (\mathbf{B_{gt}}[i,j])_{1\leq i,j\leq N}$ as the **binary comparison matrix** associated with $>$. The entries $\mathbf{B_{gt}}[i,j]$ are defined as follows:

$$\forall i,j \in I_N, \quad \mathbf{B_{gt}}[i,j] = \begin{cases} 1 & \text{if } B_i > B_j, \\ 0 & \text{otherwise.} \end{cases}$$

3.5.1 Order by Price, Quantity, and Bid Placement

$$\forall i,j \in I_N, \quad \mathbf{B_{gt}}[i,j] = \begin{cases} 0 & \text{if } \ i=j \\ \mathbf{P_{ge}}[i,j] \ \wedge \ [\ \mathbf{P_{gt}}[i,j] \ \vee \ \mathbf{Q_{ge}}[i,j]\ ] & \text{if } \ i<j \\ \neg\mathbf{B_{gt}}[j,i] & \text{if } \ i>j \end{cases}$$

Operations	$\mathbf{B_{gt}}$
fheOr(bool)	$N(N-1)/2$
fheAnd(bool)	$N(N-1)/2$
fheNot(bool)	$N(N-1)/2$
FHE Units	$3N(N-1)/2$

3.5.2 Order by Price and Bid Placement

$$\forall i,j \in I_N, \quad \mathbf{B_{gt}}[i,j] = \begin{cases} 0 & \text{if } \ i=j \\ \mathbf{P_{ge}}[i,j] & \text{if } \ i<j \\ \mathbf{P_{gt}}[i,j] & \text{if } \ i>j \end{cases}$$

The additionnal FHE-cost is null when bidders are sorted using price and bid placement.

3.5.3 Order by Price and Randomization

Using the $\mathbf{Rand_{gt}}$ matrix defined above, the comparison matrix can be computed as follows:

$$\forall i,j \in I_N, \quad \mathbf{B_{gt}}[i,j] = \begin{cases} 0 & \text{if } \ i=j \\ \mathbf{P_{ge}}[i,j] \wedge [\ \mathbf{P_{gt}}[i,j] \vee \mathbf{Rand_{gt}}[i,j]\ ] & \text{if } \ i<j \\ \neg\mathbf{B_{gt}}[j,i] & \text{if } \ i>j \end{cases}$$

Operations	$\mathbf{B_{gt}}$
fheOr(bool)	$N(N-1)/2$
fheAnd(bool)	$N(N-1)/2$
fheNot(bool)	$N(N-1)/2$
FHE Units	$3N(N-1)/2$

4 FHE sort

4.1 Rank function

4.1.1 Definition

Let $rank : \{1, 2, \ldots, N\} \to \{0, 1, \ldots, N^* - 1\}$ denote the **rank function** under the strict order relation $>$ defined on $\mathcal{B}$, which assigns a rank value to each bidder index i. Specifically, $rank$ represents the position of B_i in the descending order of the set $\mathcal{B}$. Formally, the rank of bidder B_i is given by:

$$rank(i) = |\{B_j \in \mathcal{B} \mid B_j > B_i\}|$$

Where:

- $|\cdot|$ denotes the cardinality of the set.
- $rank(i) = 0$ if B_i is the largest element under $>$,
- $rank(i) = N^* - 1$ if B_i is the smallest element under $>$.
- If $N^* = N$ then each bidder has a unique rank and the auction has no tie under $>$.
- If $N^* < N$ then two or more distinct bidders are sharing the same rank, the auction has one or more ties.

Thus, the $rank$ function can also be written as follows using the comparison matrix $\mathbf{B_{gt}}$:

$$rank(i) = \sum_{\substack{j=1 \\ j \neq i}}^{N} \mathbf{B_{gt}}[j, i]$$

4.1.2 Unique rankings

When $>$ produces a set of bidders with unique rankings (i.e. no ties), the rank function rank(i) becomes bijective.

4.2 Rank vector rank

4.2.1 Definition

Let $\mathbf{rank} = (\mathbf{rank}[i])_{1 \leq i \leq N}$ denote the vector whose entries $\mathbf{rank}[i]$ are defined as:

$$\forall i, k \in \{1, 2, 3, \ldots, N\}, \quad \mathbf{rank}[i] = rank(i)$$

4.2.2 FHE cost

Operations	$rank(i)$	**rank**
fheAdd(U16)	$N-1$	$N(N-1)$
FHE Units	$5(N-1)$	$5N(N-1)$

4.2.3 Solidity sample code

```
function rankAt(uint16 i) returns(euint16 rank) {
    rank = TFHE.asEuint16(0);
    for(uint16 j = 0; j < N; ++j) {
        if (i != j) {
            rank = TFHE.add(r, TFHE.asEuint16(Bgt(j,i)));
        }
    }
}
```

4.3 Rank Matrix $\mathbf{R_{eq}}$

4.3.1 Definition

Let $\mathbf{R_{eq}} = (\mathbf{R_{eq}}[i,k])_{1\leq i,k\leq N}$ denote the matrix whose entries $\mathbf{R_{eq}}[i,k]$ are defined as:

$$\forall i,k \in \{1,2,3,\ldots,N\}, \quad \mathbf{R_{eq}}[i,k] = \begin{cases} 1, & \text{if } \mathbf{rank}[i] = k-1, \\ 0, & \text{otherwise.} \end{cases}$$

4.3.2 FHE cost

Operations	$\mathbf{R_{eq}}$
fheEq(U16)	N^2
FHE Units	$2N^2$

4.3.3 Solidity sample code

```
function Req(uint16 i, uint16 k) returns(ebool eq) {
    eq = TFHE.eq(rankAt(i), k);
}
```

4.4 Quantity vector $\mathbf{q_{rank}}$

4.4.1 Definition

Let $\mathbf{q_{rank}} = (\mathbf{q_{rank}}[k])_{1\leq k\leq N}$ denote the vector where the k-th entry represents the total quantity of tokens bid by all the bidders ranked at the k-th position upon completion of the auction. The entries of $\mathbf{q_{rank}}$ are defined as follows:

$$\forall k \in \{1,2,\ldots,N\}, \quad \mathbf{q_{rank}}[k] = \sum_{i\in\{1,2,\ldots,N\}} \mathbf{R_{eq}}[i,k].q_i$$

4.4.2 Case with unique rankings

When $>$ produces a set of bidders with unique rankings (i.e. no ties), the rank function rank(i) becomes bijective. In this case the vector $\mathbf{q_{rank}}$ can be expressed using bitwise operations resulting in a more efficient formula in terms of FHE cost:

$$\forall k \in \{1,2,\ldots,N\}, \quad \mathbf{q_{rank}}[k] = \bigvee_{i\in\{1,2,\ldots,N\}} \mathbf{R_{eq}}[i,k] \wedge q_i$$

4.4.3 FHE cost

Operations	$\mathbf{q_{rank}}[k](unique)$	$\mathbf{q_{rank}}(unique)$	$\mathbf{q_{rank}}[k](ties)$	$\mathbf{q_{rank}}(ties)$
fheAnd(U256)	N	N^2	0	0
fheOr(U256)	$N-1$	$N(N-1)$	0	0
fheAdd(U256)	0	0	$N-1$	$N(N-1)$
fheIfThenElse(U256)	0	0	N	N^2
FHE Units	$4N-2$	$N(4N-2)$	$14N-10$	$N(14N-10)$

4.4.4 Solidity sample code

```
function Req(uint16 bi, uint16 k) returns(ebool);
function quantity(uint16 bi) returns (euint256);

// If bidders can have identical rankings (i.e., ties exist)
function qRankWithTies(uint16 k) returns(euint256 qRank) {
    qRank = TFHE.ifThenElse(Req(0,k), quantity(0), TFHE.asEuint256(0));
    for(uint16 i = 1; i < N; ++i) {
        euint256 q = TFHE.ifThenElse(Req(i,k), quantity(i), TFHE.asEuint256(0));
        qRank = TFHE.add(qRank, q);
    }
}
```

```
// returns 0 or 2^256-1
function Req(uint16 bi, uint16 k) returns(euint256);
function quantity(uint16 bi) returns (euint256);

// If bidders have unique rankings (i.e., there are no ties)
function qRankUnique(uint16 k) returns(euint256 qRank) {
    qRank = TFHE.and(Req(0,k), quantity(0));
    for(uint16 i = 1; i < N; ++i) {
        euint256 q = TFHE.and(Req(i,k), q(i));
        qRank = TFHE.or(qRank, q);
    }
}
```

4.5 Price vector $\mathbf{p_{rank}}$

4.5.1 Definition

Let $\mathbf{p_{rank}} = (\mathbf{p_{rank}}[k])_{1\leq k\leq N}$ denote the vector whose k-th entry represents the common token unit price bid by each of k-th highest-ranked bidders upon completion of the auction.
The entries of $\mathbf{p_{rank}}$ can be computed as follows:

$$\forall k \in \{1,2,\ldots,N\}, \quad \mathbf{p_{rank}}[k] = \bigvee_{i\in\{1,2,\ldots,N\}} \mathbf{R_{eq}}[i,k] \wedge p_i$$

4.5.2 FHE cost

Operations	$\mathbf{p_{rank}}[k](bitwise)$	$\mathbf{p_{rank}}(bitwise)$	$\mathbf{p_{rank}}[k](if/then/else)$	$\mathbf{p_{rank}}(if/then/else)$
fheAnd(U256)	N	N^2	0	0
fheOr(U256)	$N-1$	$N(N-1)$	0	0
fheIfThenElse(U256)	0	0	N	N^2
FHE Units	$4N-2$	$N(4N-2)$	$4N$	$4N^2$

4.5.3 Solidity sample code

```
function Req(uint16 bi, uint16 k) returns(ebool);
function price(uint16 bi) returns (euint256);

function pRank(uint16 k) returns(euint256) {
    euint256 p = TFHE.ifThenElse(Req(0,k), price(0), TFHE.asEuint256(0));
    for(uint16 i = 1; i < N; ++i) {
        p = TFHE.ifThenElse(Req(i,k), price(i), p);
    }
    return p;
}
```

```
// returns 0 or 2^256-1
function Req(uint16 bi, uint16 k) returns(euint256);
function price(uint16 bi) returns (euint256);

function pRank(uint16 k) returns(euint256 p) {
    euint256 p = TFHE.and(Req(0,k), price(0));
    for(uint16 i = 1; i < N; ++i) {
        euint256 _p = TFHE.and(Req(i,k), q(i));
        p = TFHE.or(p, _p);
    }
    return p;
}
```

5 FHE auction

5.1 Cumulative quantity vector $\mathbf{c}_{\mathbf{rank}}$ and validity vectory $\mathbf{1}_{\mathbf{valid}}$

5.1.1 Definitions

- Let $\mathbf{c}_{\mathbf{rank}}[k]$, where $k \in \{0, 1, 2, \ldots, N\}$, denote the cumulative quantity of tokens bid by the top k-th highest-ranked bidders when the auction concludes. Specifically, it is given by:

$$\mathbf{c}_{\mathbf{rank}}[k] = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=1}^{k} \mathbf{q}_{\mathbf{rank}}[i] & \text{if } k > 0 \end{cases}$$

- Let $\mathbf{1}_{\mathbf{valid}}[k]$, where $k \in \{1, 2, \ldots, N\}$, denote the binary valid indicator for the cumulative bid quantity, defined as follows:

$$\mathbf{1}_{\mathbf{valid}}[k] = \begin{cases} 1 & \text{if } \ \mathbf{c}_{\mathbf{rank}}[k-1] < Q \\ 0 & \text{otherwise} \end{cases}$$

  Here:

  - Q denotes the total offered quantity of tokens.
  - $\mathbf{1}_{\mathbf{valid}}[k] = 1$ indicates that the cumulative bid quantity up to the $(k-1)$-th rank is **valid** meaning there are still tokens left to be distributed.
  - $\mathbf{1}_{\mathbf{valid}}[k] = 0$ indicates that an overflow occurred at rank $k-1$, so no tokens remain to distribute at rank k.

5.1.2 FHE cost

Operations	$\mathbf{c}_{\mathbf{rank}}$	$\mathbf{1}_{\mathbf{valid}}$
fheAdd(U256)	$N-1$	0
fheLt(U256)	0	N
FHE Units	$10(N-1)$	$9N$

5.2 Final Quantity Vector $\mathbf{q}^*_{\mathbf{rank}}$

5.2.1 Definition

Let $\mathbf{q}^*_{\mathbf{rank}} = (\mathbf{q}^*_{\mathbf{rank}}[k])_{1\leq k\leq N}$ denote the vector whose k-th entry represents the maximum theoretical cumulative quantity of tokens won by all the k-th highest-ranked bidders upon completion of the auction.

$$\mathbf{q}^*_{\mathbf{rank}}[k] = \begin{cases} \mathbf{q}_{\mathbf{rank}}[k] & \text{if } \ \mathbf{1}_{\mathbf{valid}}[k] \\ 0 & \text{otherwise} \end{cases}$$

5.2.2 Case with unique rankings

When $>$ produces a set of bidders with unique rankings (i.e. no ties), the value of $\mathbf{q}^*_{\mathbf{rank}}[k]$ must be clamped to $Q - \mathbf{c}_{\mathbf{rank}}[k-1]$ and the above formula is modified as follows:

$$\mathbf{q}^*_{\mathbf{rank}}[k] = \begin{cases} \min(\mathbf{q}_{\mathbf{rank}}[k],\ Q - \mathbf{c}_{\mathbf{rank}}[k-1]) & \text{if } \ \mathbf{1}_{\mathbf{valid}}[k] \\ 0 & \text{otherwise} \end{cases}$$

Here:

- The condition if $\mathbf{1}_{\mathbf{valid}}[k]$ ensures that no arithmetic overflow occurs.
- In case of unique rankings, $\mathbf{q}^*_{\mathbf{rank}}[k]$ represents the **final quantity** of tokens won by the bidder ranked in the k-th position.

5.2.3 FHE cost

Operations	$\mathbf{q}^*_{\mathbf{rank}}$ (unique)	$\mathbf{q}^*_{\mathbf{rank}}$ (ties)
fheMin(U256)	N	0
fheSub(U256)	N	0
fheIfThenElse(U256)	N	N
FHE Units	$25N$	$4N$

5.2.4 Solidity sample code

```
// If bidders can have identical rankings (i.e., ties exist)
function qRankStarWithTies(uint16 k) returns(euint256) {
    return TFHE.ifThenElse(valid(k), qRank(k), TFHE.asEuint256(0));
}
```

```
// If bidders have unique rankings (i.e., there are no ties)
function qRankStarUnique(uint16 k) returns(euint256) {
    euint256 q_max = TFHE.sub(Q, crank(k-1));
    euint256 q_clamped = TFHE.min(qRank(k), q_max);
    return TFHE.ifThenElse(valid(k), q_clamped, TFHE.asEuint256(0));
}
```

5.3 Final Uniform Price $p_u^{(b)}$

5.3.1 Definition

$p_u^{(b)}$, the final uniform price for each token sold when the auction concludes, is determined by the following recurrence relation:

$$\forall k \in \{0, 1, 2, \ldots, N\} \quad p^*_k = \begin{cases} 0 & \text{if } \ k = 0 \\ \mathbf{p}_{\mathbf{rank}}[k] & \text{if } \ \mathbf{1}_{\mathbf{valid}}[k] \ \text{ and } \ k > 0 \\ p^*_{k-1} & \text{otherwise} \end{cases}$$

$$\text{final uniform price} = p_u^{(b)} = p^*_N$$

The final uniform price value can interpreted as follows:

$$\begin{aligned} p_u^{(b)} &= 0 \quad \text{if there are no winning bidders} \\ p_u^{(b)} &> 0 \quad \text{if the auction concludes with at least one winning bidder} \end{aligned}$$

5.3.2 FHE cost

Operations	$p_u^{(b)}$
fheIfThenElse(U256)	N
FHE Units	$4N$

5.3.3 Solidity sample code

```
function uniformPrice() returns(euint256 pu) {
    pu = TFHE.asEuint256(0);
    for(uint16 i = 0; i < N; ++i) {
        pu = TFHE.ifThenElse(valid(k), pRank(k), pu);
    }
}
```

6 FHE operations unit cost

Operations	Cost	Unit Cost (approx.)
fheBitShl(U16)	35000	1
fheBitAnd(Bool)	26000	1
fheBitAnd(U256)	44000	2
fheEq(U16)	54000	2
fheEq(U256)	100000	4
fheGt(U16)	105000	4
fheGt(U256)	231000	9
fheGe(U16)	105000	4
fheGe(U256)	231000	9
fheAdd(U16)	133000	5
fheAdd(U256)	253000	10
fheSub(U256)	253000	10
fheIfThenElse(U16)	47000	2
fheIfThenElse(U256)	90000	4
fheMin(U256)	277000	11
fheRand(U16)	100000	4
fheRem(U16)	622000	24