Random Number Sales Contract: Parameter Table

Let’s cover the following, **for each scenario**:

- **pricePerByte** (set by contract owner/admin)
- **priceDepositDivisor** (set by contract owner/admin)
- **minMinerDeposit** (set by buyer for their call)
- **numberOfBytes** (set by buyer for their call)
- **What do miners typically deposit?**
- **How can a buyer construct their buy call?**
- **What is the final price they pay in QU?**

Miner flow:

- Miners choose a security deposit (must be one of the “validDepositAmounts”).
- Only miners who have revealed entropy with at least `minMinerDeposit` in the recent pool are considered "eligible" for a BuyEntropy request.
- **Higher deposits mean more buyers can select them (if they want higher security).**

Buyer flow:

Buyers pick:

- `numberOfBytes` (between 1 and 32)
- `minMinerDeposit` (how strong/secure they want: e.g. only buy from a miner who locked at least X QU for their entropy)
- Calls `BuyEntropy` (contract checks pool, applies price formula, requires enough QU sent).

---

price/byte	depDiv	MinerDeposits	#Bytes	minDep	div(minDep/depDiv)	Price(qu)	Notes
10	1,000	100, 1k, 10k	8	100	0	80	Any miner ≥100 QU
10	1,000	100, 1k, 10k	8	1,000	1	160	Any miner ≥1k QU
10	1,000	100, 1k, 10k	8	10,000	10	880	Only miner ≥10k QU
10	10,000	100, 1k, 10k	8	10,000	1	160	Less sensitive to deposit
100	1,000	100, 1k, 10k	8	10,000	10	8,800	Expensive premium random
10	100	10k, 50k, 100k	32	50,000	500	160,320	Huge premium for security
5	1,000	1k, 10k	32	10,000	10	1,760	Low base cost
100	5,000	50k, 100k	16	50,000	10	17,600	Flat curve, big random

---

How this works in practice:

Miner Example:

- Suppose a miner wants to maximize the number of buyers who can select their entropy.
- The miner makes a commitment with a deposit of 10,000 QU (one of the valid powers of ten, as in your contract).
- This miner will be eligible for any buyer call with `minMinerDeposit` ≤ 10,000 QU.

Buyer Example 1:

- Buyer wants **cheap random**; doesn't care about miner having a high deposit:
  - Sets `minMinerDeposit = 100`
  - With default settings (`pricePerByte=10` , `priceDepositDivisor=1000` , `numberOfBytes=8`):
    - They pay: `10 * 8 * (div(100,1000) + 1) = 10 * 8 * (0 + 1) = 80 QU`
- Buyer wants **secure random**; only accepts entropy from high-security miners:
  - Sets `minMinerDeposit = 10,000`

    - Pays: `10 * 8 * (div(10,000,1000) + 1) = 10 * 8 * (10 + 1) = 880 QU`

Buyer Example 2:

- Buyer wants **bulk random** (32 bytes, high deposit security):
    - `minMinerDeposit = 50,000`
    - Using `pricePerByte=10`, `priceDepositDivisor=1,000`:
        - Price: `10 * 32 * (50 + 1) = 10 * 32 * 51 = 16,320 QU`

Buyer Example 3:

- Admin wants to "flatten" cost curve (make deposit matter less):
    - Increases `priceDepositDivisor` to 10,000 (from 1,000).
    - Buy with `minMinerDeposit=10,000`: `div(10,000,10,000)=1`, so Price: `10 * 8 * 2 = 160 QU`.
    - Makes high security not much more expensive.

Summary of Control and Choices

- **As the contract owner/admin:**
  Set `pricePerByte` for base cost of every byte.
- **Set `priceDepositDivisor`** to control how much higher-minDeposit requests cost buyers.
    - **Lower divisor:** Big cost jump for higher miner security
    - **Higher divisor:** Flatter cost curve—high-security random is still affordable
- **Miners** select their security/eligibility via their deposit (higher = eligible for more buyers, get more rewards).
- **Buyers** choose:
    - `numberOfBytes` (how much random)
    - `minMinerDeposit` (how "secure", i.e., only accept entropy from miners who have revealed with at least this much deposit)

- Contract formula combines these for the final price, using only state-configured parameters.