IQ 5 Confidential Data Store Pricing Idea:

- when you instantiate your SUAPP, it enters into a contract with the enshrined “StorageAllocator” contract
- contract terms are: *Number_of_Replicas
- *Duration
- Total_Data_Allocation
- Storage_Price

*

- initial price is dead simple: we use the price that we pay per mb in AWS and we only have 1 replica available.
- for recurring contracts the storage allocator is able to transfer money out of the SUAPP automatically to avoid the SUAPP deployer to have to manually re-trigger the contract. Could also expose API to update contract terms
- This does not take into account storage scarcity at all!

Goal

When a SUAPP attempts to call storageContract.StoreData(xyz)

, it will fail or succeed based on its set up and payment status/ policy.

Contract Storage Interface

interface IStorageContract { // Function to set a new storage policy function setStoragePolicy(StoragePolicy memory policy) external;

**** // Function to calculate the current price based on the storage policy function calculatePrice() external view returns (uint);

```
// Variables representing storage contract terms
function numberOfReplicas() external view returns (uint);
function duration() external view returns (uint); // Duration in days
function totalDataAllocation() external view returns (uint); // Total data allocation in GB
function storagePrice() external view returns (uint);

// Struct to hold storage policy details
struct StoragePolicy {
    uint numberOfReplicas;
    uint duration; // in days
    uint totalDataAllocation; // in GB
}

}
```

Sample Contract Storage Instantiation

contract StorageContract { uint public numberOfReplicas; uint public duration; // in days uint public totalDataAllocation; // in GB uint public storagePrice; address public suappContract;

```
// Struct to hold storage policy details
struct StoragePolicy {
    uint numberOfReplicas;
    uint duration;
    uint totalDataAllocation;
}

constructor(uint _storagePrice, address _suappContract) {
    storagePrice = _storagePrice;
    suappContract = _suappContract;
}

function setStoragePolicy(StoragePolicy memory policy) public {
    require(msg.sender == suappContract, "Only SUAPP can update the storage policy.");
    numberOfReplicas = policy.numberOfReplicas;
    duration = policy.duration;
    totalDataAllocation = policy.totalDataAllocation;
}

function calculatePrice() public view returns (uint) {
    return numberOfReplicas * duration * totalDataAllocation * storagePrice;
```

```
}

// Additional functions as required...

}
```

Sample SUAPP UX

contract SampleSUAPP { StorageContract public storageContract;

```
constructor(StorageContract.StoragePolicy policy) {
        setStoragePolicy(StorageContract.StoragePolicy policy);
    }

function deployStorageContract(uint storagePrice) public {
    storageContract = new StorageContract(storagePrice, address(this));
}

function updateStoragePolicy() public {
    // Example policy: 100GB for a duration of 1 day with 1 replica
    StorageContract.StoragePolicy memory policy = StorageContract.StoragePolicy({
        numberOfReplicas: 1,
        duration: 1, // 1 day
        totalDataAllocation: 100 // 100 GB
    });
    storageContract.setStoragePolicy(policy);
}

// Additional SUAPP specific functions...

}
```