A APPENDIX

A.1 Security Theorem of $\text{Prot}_{\text{HyperX}}$

In this section, we present the main theorem for our security protocol $\text{Prot}_{\text{HyperX}}$ described in § 3, and prove the theorem using the UC-framework [8].

A.1.1 *Ideal Functionality $\mathcal{F}_{HyperX}$*

We first present the cryptography abstraction of the $\text{Prot}_{\text{HyperX}}$ protocol in form of an ideal functionality $\mathcal{F}_{\text{HyperX}}$. The ideal functionality articulates the correctness and security properties that HyperX wishes to attain by assuming a trusted entity (as a result, $\mathcal{F}_{\text{HyperX}}$ is often drastically simplified compared with the actual real-world protocol $\text{Prot}_{\text{HyperX}}$). Then we prove that $\text{Prot}_{\text{HyperX}}$ UC-realizes $\mathcal{F}_{\text{HyperX}}$, implicating that $\text{Prot}_{\text{HyperX}}$, without assuming any trusted authorities, achieves the same security properties as $\mathcal{F}_{\text{HyperX}}$. The description of $\mathcal{F}_{\text{HyperX}}$ is given in Figure 11. We provide additional explanations below.

Session Creation. Through this interface, a data collector $\mathcal{P}_{\text{a}}$ requests $\mathcal{F}_{\text{HyperX}}$ to securely realize a data recruiting task $\mathcal{F}_{\text{Task}}$. The provided task $\mathcal{F}_{\text{Task}}$ must specify the essentials for starting a trading session, including the data quality assessment method (*e.g.,* the feature extractor $\text{M}_{\text{extractor}}$ or how it can be trained) and logistics (*e.g.,* similar to the $\text{SC}_{\text{config}}$ discussed in § 3.1.1). As a trusted entity, $\mathcal{F}_{\text{HyperX}}$ generates a utility key for $\mathcal{P}_{\text{a}}$, allowing $\mathcal{F}_{\text{HyperX}}$ to sign blockchain transactions and datastore service requests on behalf of $\mathcal{P}_{\text{a}}$. Next, $\mathcal{F}_{\text{HyperX}}$ deploys the *contract* on $\mathcal{F}_{\text{BC}}$. This is to emulate the real-world *side-effects*, which is necessary to prove $\text{Prot}_{\text{HyperX}}$ UC-realizes $\mathcal{F}_{\text{HyperX}}$. We provide further explanations below.

Provider Participation. A provider $\mathcal{P}_{\text{z}}$ joins the trading session via the ProviderParticipate interface. Due to the assumed trustiness, $\mathcal{P}_{\text{z}}$ can safely hands over its complete dataset $\mathcal{D}_{\text{whole}}$ to $\mathcal{F}_{\text{HyperX}}$. $\mathcal{F}_{\text{HyperX}}$ sends a storage request to $\mathcal{F}_{\text{DS}}$ on behalf of $\mathcal{P}_{\text{z}}$ to store $\mathcal{D}_{\text{whole}}$. Afterwards, $\mathcal{F}_{\text{HyperX}}$ evaluates the quality of $\mathcal{D}_{\text{whole}}$ using the method defined in $\mathcal{F}_{\text{Task}}$ (for instance the method discussed in § 3.1.1). Although $\mathcal{F}_{\text{HyperX}}$ has the full dataset from $\mathcal{P}_{\text{z}}$, its does not rely on this advantage for quality assessment. This is to ensure the parity of evaluation accuracies between $\mathcal{F}_{\text{HyperX}}$ and the real-world protocol $\text{Prot}_{\text{HyperX}}$.

Deal Closure. The collector $\mathcal{P}_{\text{a}}$ calls the DealClose interface to initiate the deal closure process. $\mathcal{F}_{\text{HyperX}}$ proceeds only if the best provider P_{opt}'s dataset is sufficiently good (as defined in $\mathcal{F}_{\text{Task}}$). We explicitly construct $\mathcal{F}_{\text{HyperX}}$ to handle Byzantine parties differently, which is again to achieve the parity between $\mathcal{F}_{\text{HyperX}}$ and $\text{Prot}_{\text{HyperX}}$ when handling the post-review abortion attack discussed in § 3.3. For non-Byzantine parties, the deal is closed atomically.

Verbose Definition of $\mathcal{F}_{\text{HyperX}}$. We *intentionally define $\mathcal{F}_{HyperX}$ verbosely*. For instance, in the SessionCreate interface, $\mathcal{F}_{\text{HyperX}}$ signs a blockchain transaction on behalf of $\mathcal{P}_{\text{a}}$ to simulate the result of deploying a smart contract on $\mathcal{F}_{\text{SC}}$ in the real world. Other examples include the request sent to $\mathcal{F}_{\text{DS}}$ in the ProviderParticipate interface, additional messages sent to the participants to inform actions take by $\mathcal{F}_{\text{HyperX}}$, and the dedicated handling for Byzantine parties. These steps are not essential to ensure the trading correctness due to the assumed trustiness of $\mathcal{F}_{\text{HyperX}}$. However, they are crucial for $\mathcal{F}_{\text{HyperX}}$ to accurately emulate the external side effects of $\text{Prot}_{\text{HyperX}}$ in the real world. As we shall see in § A.1.5, the emulation is necessary to prove that $\text{Prot}_{\text{HyperX}}$ UC-realizes $\mathcal{F}_{\text{HyperX}}$.

Init: Data := $\emptyset$
Upon Receive SessionCreate($\mathcal{F}_{\text{Task}}$, $\mathcal{P}_{\text{a}}$):
  abort if $\mathcal{F}_{\text{Task}}$ lacks essentials to start a trading session
  generate the session ID sid $\leftarrow \{0,1\}^{\lambda}$ and a utility key for $\mathcal{P}_{\text{a}}$
  generate the trading *contract* based on $\mathcal{F}_{\text{Task}}.\text{SC}_{\text{config}}$
  compute a blockchain transaction $\mathcal{T}_{\text{BC}}$ on behalf of $\mathcal{P}_{\text{a}}$ to deploy *contract* on $\mathcal{F}_{\text{BC}}$
  halt until *contract* is intialized on $\mathcal{F}_{\text{BC}}$
  send $\mathcal{T}_{\text{BC}}$ to $\mathcal{P}_{\text{a}}$ to inform the action take by $\mathcal{F}_{\text{HyperX}}$
  set an expiration timeout *timer* for the session
  update Data[sid] = $\{\mathcal{F}_{\text{Task}}, \mathcal{P}_{\text{a}}, \mathcal{S}_{\text{pri}} := \emptyset\}$
Upon Receive ProviderParticipate(sid, $\mathcal{D}_{\text{whole}}$, *fund*, $\mathcal{P}_{\text{z}}$):
  $\{\mathcal{F}_{\text{Task}}, \_, \mathcal{S}_{\text{pri}}\}$:= Data[sid] and abort if not found
  abort if $\mathcal{P}_{\text{z}}$ is already in $\mathcal{S}_{\text{pri}}$ or *fund* is not sufficient
  generate a utility key for $\mathcal{P}_{\text{z}}$
  send a storage request $\mathcal{T}_{\text{BC}}(\mathcal{D}_{\text{whole}})$ on behalf of $\mathcal{P}_{\text{z}}$ to $\mathcal{F}_{\text{DS}}$
  obtain the data retrieval keys $\text{adr}^{\text{p}}_{\text{data}}$ for $\mathcal{D}_{\text{whole}}$
  send $\text{adr}^{\text{p}}_{\text{data}}$ and $\mathcal{T}_{\text{BC}}(\mathcal{D}_{\text{whole}})$ to $\mathcal{P}_{\text{z}}$ to inform action
  evaluate the quality *score* of $\mathcal{D}_{\text{whole}}$ based on the method in $\mathcal{F}_{\text{Task}}$
  update $\mathcal{S}_{\text{pri}}[\mathcal{P}_{\text{z}}] = \{\mathcal{D}_{\text{whole}}, \text{adr}^{\text{p}}_{\text{data}}, \textit{score}\}$
Upon Receive DealClose(sid, $\mathcal{P}$):
  $\{\mathcal{F}_{\text{Task}}, \mathcal{P}_{\text{a}}, \mathcal{S}_{\text{pri}}\}$:= Data[sid] and abort if not found
  assert $\mathcal{P} = \mathcal{P}_{\text{a}}$
  find the provider P_{opt} in $\mathcal{S}_{\text{pri}}$ with the highest *score*
  if $\mathcal{S}_{\text{pri}}[\text{P}_{\text{opt}}]$.*score* exceeds the quality threshold defined in $\mathcal{F}_{\text{Task}}$ **:**
    if $\mathcal{P}_{\text{a}}$ or P_{opt} is Byzantine failed **:**
      # An example post-review abortion policy
      disclose randomly sampled data from $\mathcal{S}_{\text{pri}}[\text{P}_{\text{opt}}].\mathcal{D}_{\text{whole}}$ to $\mathcal{P}_{\text{a}}$
      send the compensation defined in $\mathcal{F}_{\text{Task}}.\text{SC}_{\text{config}}$ to P_{opt}
    else : # close the deal atomically
      disclose the full dataset $\mathcal{S}_{\text{pri}}[\text{P}_{\text{opt}}].\mathcal{D}_{\text{whole}}$ to $\mathcal{P}_{\text{a}}$
      send the reward defined in $\mathcal{F}_{\text{Task}}.\text{SC}_{\text{config}}$ to P_{opt}
  close the session by Data.Erase(sid)
Timeout Callback DepositClaim(sid):
  abort if sid is not in Data
  return all deposits for participating providers Data[sid].$\mathcal{S}_{\text{pri}}$
  close the session by Data.Erase(sid)

Figure 11: The ideal functionality $\mathcal{F}_{\text{HyperX}}$.

A.1.2 *Security and Correctness Properties of $\mathcal{F}_{HyperX}$*

With the assumed trustiness and simplified construction, it is not difficult to conclude that $\mathcal{F}_{\text{HyperX}}$ offers the following correctness and security properties. First, the data quality assessment process is conducted confidentially so that each provider's dataset is kept secret from $\mathcal{P}_{\text{a}}$, and meanwhile models used for data quality assessment are opaque to providers. Second, the trading deal is closed atomically (*i.e.,* the collector $\mathcal{P}_{\text{a}}$ and the selected optimal P_{opt} exchange their goods simultaneously) if both parties are honest. Otherwise if they are Byzantine-failed, $\mathcal{F}_{\text{HyperX}}$ implements the post-review abortion or penalty policies according to the configuration of $\mathcal{F}_{\text{Task}}$. Detailed discussion on such policies are provided in § 3.3. We list the proportional compensation policy as an example in Figure 11. Finally, the accuracy for $\mathcal{F}_{\text{HyperX}}$ to select the best provider is fully driven by the dataset quality assessment method given by the $\mathcal{P}_{\text{a}}$, *i.e.,* $\mathcal{F}_{\text{HyperX}}$ focuses on ensuring the correctness of the underlying trading infrastructure.

A.1.3 Security Theorem

We now present our main security theorem.

THEOREM 2. *Assuming that the distributed algorithms used by the underlying blockchains and datastore services are provably secure, the hash function is pre-image resistant, and the digital signature is EU-CMA secure (*i.e., *existentially unforgeable under a chosen message attack), our decentralized real-world protocol $\mathsf{Prot}_{\mathsf{HyperX}}$ securely UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{HyperX}}$ against a malicious adversary in the Byzantine corruption model.*

Theorem 2 also holds for strictly weaker corruption models, such as the passive corruption model where the adversary is able to observe the complete internal state of a corrupted party (participant) whereas the corrupted party is still protocol compliant.

A.1.4 Proof Overview

We now present the detail proof of our main security Theorem 2. In the UC framework [8], the model of $\mathsf{Prot}_{\mathsf{HyperX}}$ execution is abstracted as a system of machines $(\mathcal{E}, \mathcal{A}, \pi_1, ..., \pi_n)$ where $\mathcal{E}$ is called the *environment*, $\mathcal{A}$ is the (real-world) adversary, and $(\pi_1, ..., \pi_n)$ are participants (referred to as *parties*) of $\mathsf{Prot}_{\mathsf{HyperX}}$ where each party may execute different parts of $\mathsf{Prot}_{\mathsf{HyperX}}$. Intuitively, the environment $\mathcal{E}$ represents the *external* system that contains other protocols, including ones that provide inputs to, and obtain outputs from, $\mathsf{Prot}_{\mathsf{HyperX}}$. The adversary $\mathcal{A}$ represents adversarial activity against the protocol execution, such as controlling communication channels and sending *corruption* messages to parties. $\mathcal{E}$ and $\mathcal{A}$ can communicate freely.

To prove that $\mathsf{Prot}_{\mathsf{HyperX}}$ UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{HyperX}}$, we need to prove that $\mathsf{Prot}_{\mathsf{HyperX}}$ *UC-emulates* $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$, which is the *ideal protocol* (defined below) of the ideal functionality $\mathcal{F}_{\mathsf{HyperX}}$. That is, for any adversary $\mathcal{A}$, there exists an adversary (often referred to as a *simulator*) $\mathcal{S}$ such that $\mathcal{E}$ cannot distinguish between the ideal world, featured by $(\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}, \mathcal{S})$, and the real world, featured by $(\mathsf{Prot}_{\mathsf{HyperX}}, \mathcal{A})$. Mathematically, on any input, the probability that $\mathcal{E}$ outputs $\vec{1}$ after interacting with $(\mathsf{Prot}_{\mathsf{HyperX}}, \mathcal{A})$ in the real world differs by at most a negligible amount from the probability that $\mathcal{E}$ outputs $\vec{1}$ after interacting with $(\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}, \mathcal{S})$ in the ideal world.

The ideal protocol $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$ is a wrapper around $\mathcal{F}_{\mathsf{HyperX}}$ by a set of dummy parties that have the same interfaces as those of the parties in $\mathsf{Prot}_{\mathsf{HyperX}}$. As a result, $\mathcal{E}$ is able to interact with $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$ in the ideal world the same way it interacts with $\mathsf{Prot}_{\mathsf{HyperX}}$ in the real world. These dummy parties simply pass received inputs from $\mathcal{E}$ to $\mathcal{F}_{\mathsf{HyperX}}$ and relay outputs of $\mathcal{F}_{\mathsf{HyperX}}$ to $\mathcal{E}$, without implementing any other logic. $\mathcal{F}_{\mathsf{HyperX}}$ controls the keys of these dummy parties. For the sake of clear presentation, we abstract the real-world participants of $\mathsf{Prot}_{\mathsf{HyperX}}$ as three types of parties $\{\mathcal{P}_{\mathsf{COL}}, \mathcal{P}_{\mathsf{PRI}}, \mathcal{P}_{\mathsf{SC}}\}$, representing the collector, provider and trading contract. In the ideal world, the corresponding dummy party for $\mathcal{P}_{\mathsf{COL}}$ is denoted as $\mathcal{P}^{\mathcal{I}}_{\mathsf{COL}}$. This annotation mechanism applies for other parties as well.

Based on [8], to prove that $\mathsf{Prot}_{\mathsf{HyperX}}$ UC-emulates $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$ for any adversaries, it is sufficient to construct a simulator $\mathcal{S}$ only for the *dummy adversary* $\mathcal{A}$ that simply relays messages between $\mathcal{E}$ and the real-world parties. The overall proof procedure is that the simulator $\mathcal{S}$ observes the *side effects* of $\mathsf{Prot}_{\mathsf{HyperX}}$ in the real world, such as transitions on the blockchain and requests to the datastore service, and then accurately emulates these effects in the ideal world, with the help from $\mathcal{F}_{\mathsf{HyperX}}$. As a result, $\mathcal{E}$ cannot distinguish the ideal and real worlds.

A.1.5 Indistinguishability of Real and Ideal Worlds

To prove indistinguishability of the real and ideal worlds from the perspective of $\mathcal{E}$, we will go through a sequence of *hybrid arguments*, where each argument is a hybrid construction of $\mathcal{F}_{\mathsf{HyperX}}$, a subset of dummy parties of $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$, and a subset of real-world parties of $\mathsf{Prot}_{\mathsf{HyperX}}$, except that the first argument that is $\mathsf{Prot}_{\mathsf{HyperX}}$ without any ideal parties and the last argument is $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$ without any real world parties. We prove that $\mathcal{E}$ cannot distinguish any two consecutive hybrid arguments. Then based on the transitivity of protocol emulation [8], we prove that the first argument (*i.e.,* $\mathsf{Prot}_{\mathsf{HyperX}}$) UC-emulates the last argument (*i.e.,* $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$).

During the proof process, we will also specify how the simulator $\mathcal{S}$ should be constructed by specifying what actions $\mathcal{S}$ should take upon observing instructions from $\mathcal{E}$. As a distinguisher, $\mathcal{E}$ sends the same instructions to the ideal world dummy parities and real world parties.

Real World. We start with the real world $\mathsf{Prot}_{\mathsf{HyperX}}$ with a dummy adversary that simply passes messages to and from $\mathcal{E}$ to these real-world parties.

Hybrid $\mathbf{A_1}$. Hybrid $\mathbf{A_1}$ is the same as the real world, except that the $\mathcal{P}_{\mathsf{COL}}$ is replaced by the dummy $\mathcal{P}^{\mathcal{I}}_{\mathsf{COL}}$. Upon $\mathcal{E}$ gives an instruction to $\mathcal{P}^{\mathcal{I}}_{\mathsf{COL}}$ to start a data recruiting task $\mathcal{F}_{\mathsf{Task}}$, $\mathcal{S}$ extracts $\mathcal{F}_{\mathsf{Task}}$ from the instruction and constructs a SessionCreate call to $\mathcal{F}_{\mathsf{HyperX}}$ with parameter $(\mathcal{F}_{\mathsf{Task}}, \mathcal{P}^{\mathcal{I}}_{\mathsf{COL}})$. $\mathcal{F}_{\mathsf{HyperX}}$ will then output a blockchain transaction to $\mathcal{P}^{\mathcal{I}}_{\mathsf{COL}}$ to emulate the actions taken by $\mathcal{P}_{\mathsf{COL}}$ in the real world. $\mathcal{S}$ then dispatch the blockchain transaction on $\mathcal{F}_{\mathsf{BC}}$ to deploy the trading contract $\mathcal{P}_{\mathsf{SC}}$ in the Hybrid $\mathbf{A_1}$.

Upon observing an instruction from $\mathcal{E}$ to execute a branch defined in the Watching service of $\mathcal{P}_{\mathsf{COL}}$ (or $\mathcal{P}^{\mathcal{I}}_{\mathsf{COL}}$) (see definitions in Figure 6) , $\mathcal{S}$ is able to collect any necessary information from $\mathcal{P}_{\mathsf{SC}}$ in the Hybrid $\mathbf{A_1}$ to handle the instruction. For instance, if $\mathcal{E}$ instructs $\mathcal{P}^{\mathcal{I}}_{\mathsf{COL}}$ to sample addresses for a provider pid, $\mathcal{E}$ retrieves the $\mathsf{adr}^{\mathsf{h}}_{\mathsf{data}}$ for pid from $\mathcal{P}_{\mathsf{SC}}$ (abort if not found) and randomly samples a set of addresses from $\mathsf{adr}^{\mathsf{h}}_{\mathsf{data}}$. Then with the help of $\mathcal{F}_{\mathsf{HyperX}}$, $\mathcal{S}$ creates a blockchain transaction on behalf of $\mathcal{P}^{\mathcal{I}}_{\mathsf{COL}}$ to call the SampleData interface of $\mathcal{P}_{\mathsf{SC}}$ with the sampled address. For other instructions, such as reviewing a provider's submitted features, $\mathcal{S}$ may also rely on $\mathcal{F}_{\mathsf{HyperX}}$ to evaluate these features.

If $\mathcal{P}_{\mathsf{COL}}$ is corrupted by any Byzantine corruption messages from $\mathcal{E}$, it may not follow the predefined protocol. However, this does not prevent $\mathcal{S}$ from emulating a corrupted $\mathcal{P}_{\mathsf{COL}}$ in Hybrid $\mathbf{A_1}$ since every protocol execution by $\mathcal{P}_{\mathsf{COL}}$ in the real world is publicly visible on the trading contract $\mathcal{P}_{\mathsf{SC}}$. Therefore, $\mathcal{S}$ can reproduce these executions in Hybrid $\mathbf{A_1}$.

Finally, in the real world, the trading session is automatic in the sense that it can continuously proceed even without additional instructions from $\mathcal{E}$ after successful session setup. In the Hybrid $\mathbf{A_1}$, although $\mathcal{P}_{\mathsf{COL}}$ has been replaced by the dummy party $\mathcal{P}^{\mathcal{I}}_{\mathsf{COL}}$ without any internal logic, $\mathcal{S}$, with the public information from

$\mathcal{P}_{\mathsf{SC}}$ (in the Hybrid $\mathbf{A_1}$) and the help of $\mathcal{F}_{\mathsf{HyperX}}$, is still able to drive the process so that from $\mathcal{E}$'s perspective, the trading session is executed automatically. Finally, since $\mathcal{P}_{\mathsf{SC}}$ still lives in the Hybrid $\mathbf{A_1}$, $\mathcal{S}$ should not trigger the DealClose interface of $\mathcal{F}_{\mathsf{HyperX}}$ to avoid double execution on the same contract terms.

Fact 1. *With the aforementioned construction of $\mathcal{S}$ and $\mathcal{F}_{HyperX}$, it is immediately clear that the outputs of the dummy $\mathcal{P}^{\mathcal{I}}_{COL}$ in the Hybrid A_1 are exactly the same as the outputs of the actual $\mathcal{P}_{COL}$ in the real world, and all side effects (*i.e.*, blockchain transactions and datastore requests) in the real world are accurately emulated by $\mathcal{S}$ in the Hybrid A_1. Thus, $\mathcal{E}$ cannot distinguish with the real world and the Hybrid A_1.*

Hybrid $\mathbf{A_2}$. Hybrid A_2 is the same as the Hybrid A_1, expect that $\mathcal{P}_{\mathsf{PRI}}$ is further replaced by the dummy $\mathcal{P}^{\mathcal{I}}_{\mathsf{PRI}}$. When $\mathcal{E}$ instructs $\mathcal{P}^{\mathcal{I}}_{\mathsf{PRI}}$ to participate a trading session with a dataset $\mathcal{D}_{\mathsf{whole}}$, $\mathcal{S}$ extracts $\mathcal{D}_{\mathsf{whole}}$ and constructs a call to the ProviderParticipate interface of $\mathcal{F}_{\mathsf{HyperX}}$ with parameters $(\mathcal{P}^{\mathcal{I}}_{\mathsf{PRI}}, \mathcal{D}_{\mathsf{whole}})$. Then $\mathcal{S}$ creates a blockchain transaction on behalf of $\mathcal{P}^{\mathcal{I}}_{\mathsf{PRI}}$ to register the provider on $\mathcal{P}_{\mathsf{SC}}$ in Hybrid $\mathbf{A_2}$. Afterwards, for any instruction from $\mathcal{E}$ to execute a branch in Watching interface of $\mathcal{P}_{\mathsf{PRI}}$, $\mathcal{S}$ has all required information from $\mathcal{P}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{HyperX}}$ to handle the instruction on behalf of $\mathcal{P}^{\mathcal{I}}_{\mathsf{PRI}}$. Thus, Hybrid $\mathbf{A_2}$ is identically distributed as Hybrid $\mathbf{A_1}$ from the view of $\mathcal{E}$.

Hybrid $\mathbf{A_3}$, *i.e.,* the ideal world. Hybrid $\mathbf{A_3}$ is the same as the Hybrid $\mathbf{A_2}$, expect that $\mathcal{P}_{\mathsf{SC}}$ (the last real-world party) is further replaced by the dummy $\mathcal{P}^{\mathcal{I}}_{\mathsf{SC}}$. Thus, the Hybrid $\mathbf{A_3}$ is essentially $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$. In $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$, $\mathcal{S}$ is required to resume the same responsibility of $\mathcal{P}_{\mathsf{SC}}$ in the Hybrid $\mathbf{A_2}$. Emulating a public smart contract is trivial. In particular, for any instruction from $\mathcal{E}$ to invoke *contract*, $\mathcal{S}$ locally executes *contract* with the same input and then publishes the updated *contract* to $\mathcal{P}^{\mathcal{I}}_{\mathsf{SC}}$ via $\mathcal{F}_{\mathsf{HyperX}}$. Therefore, $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$ is indistinguishable with the Hybrid $\mathbf{A_2}$ from $\mathcal{E}$'s perspective.

Then given the transitivity of protocol emulation, we show that $\mathsf{Prot}_{\mathsf{HyperX}}$ UC-emulates $\mathcal{I}_{\mathcal{F}_{\mathsf{HyperX}}}$, and therefore prove that $\mathsf{Prot}_{\mathsf{HyperX}}$ UC-realizes $\mathcal{F}_{\mathsf{HyperX}}$, which implies that $\mathsf{Prot}_{\mathsf{HyperX}}$ achieves the same security properties as $\mathcal{F}_{\mathsf{HyperX}}$. Throughout the simulation, we maintain a key invariant: $\mathcal{S}$ and $\mathcal{F}_{\mathsf{HyperX}}$ together can always accurately simulate the desired outputs and side effects on all (dummy and real) parties in all Hybrid worlds. Thus, from $\mathcal{E}$'s view, the indistinguishability between the real and ideal worlds naturally follows. This concludes our proof for Theorem 2.