Anonymous Random Allocation and Its Applications

Azam Soleimanian

ConsenSys, ConsenSys R&D, Paris, France
{firstname.lastname}@consensys.net

Abstract. *Random Allocation* -the random assignment of the data to the parties- is a well-studied topic in the analysis of medical or judicial data, and the context of resource distribution. Random allocation reduces the chance of bias or corruption in the relevant applications, which makes the results more reliable. This is done by preventing a special or pre-planned assignment of the data to accommodate the assessment toward the desired results. This paper provides the first formal syntax and security notion of a random allocation scheme. Based on our new security notions of anonymity, confidentiality, and data-integrity, random allocation can cover more applications such as the *distributed audit system* where the confidentiality of data and the anonymity of auditors are of paramount importance. Our protocol allows the parties to stay anonymous during the concurrent executions of the protocol even if they have revealed themselves at a certain execution. The revelation property gives the possibility to the parties to claim certain advantages/faults at the end of a protocol-execution (without breaking the data-privacy or anonymity in other protocol-executions). We instantiate our syntax and prove the security based on simple cryptographic components and assumptions such as the Diffie-Hellman assumption, in the random oracle model.

Keywords: Random Allocation, Random Assignment, Mix-Net, Single Secret Election, Anonymity

1 Introduction

Random Allocation is the process of assigning the data to a set of parties based on a one-to-one random map. This concept provides a simple and efficient way for avoiding bias in clinical trials, or for preventing corruption in judicial or evaluation systems [Alt96, FD98]. This is done by preventing a special or pre-planned assignment of the data to accommodate the assessment toward the desired results. The solutions based on random allocations seem very practical, the group of parties agrees on a source of randomness (more precisely, on a random permutation) and respects the allocation.

For example, imagine a clinical trial where there are different research groups, each has its new intervention technique, and they want to know which of these techniques is more effective. They assign a random group of participants to

each intervention and check the results at the end. This process is known as the Randomized Controlled Trial (RCT).

Similarly, in judicial systems, to reduce the chance of corruption or biased judgment, the cases are randomly assigned to different judges or courts.

Random allocation can be used to address the scalability problem in the large-scale distributed systems where different tasks are assigned to different subsets of the system via a random allocation [GXC+17].

These examples and many others show the importance of random allocation. Despite all these applications [BCKM13], this topic has not been analyzed from a more precise security point of view. This paper puts forth a precise security notion for Random Allocation.

In the rest of the paper, we use the general terms “party” and “data” respectively for interventions and participants’ data in our RCT example. We may also use data-owner as a replacement for a participant. When it is clear from the text, we may use data and data-owner interchangeably.

The relevant communities are well aware of minimum security requirements such as "Uniqueness", "Allocation Concealment" and "Blinding" [SG05, SG02].

Uniqueness is a trivial requirement saying that the allocation should be a one-to-one map (more precisely, it is one-group-to-one-party). In our clinical trial example, the Allocation Concealment prevents the researcher groups from influencing the allocation of participants to the interventions. This property is known as the "Fairness" in cryptography.

In a (single)-Blind allocation (again consider RCT example), information that may influence the participants of the experiment is withheld until after the allocation is complete. Unblinding a participant can cause the observer bias or the confirmation bias, which arises from the expectation or the feeling of the participant towards the intervention. To blind the participants, the existing RCT techniques rely on a trusted third party who runs a random allocation and gives the result of the allocation only to the parties. Even though the data-owner may not have a direct interest in unblinding, still this kind of blindness is not realistic for many applications, where the adversary may try to attack the data allocated to a specific party just by colluding with the third party.

We suggest a strong notion of blindness called "Anonymity", which means nobody can identify the party for an allocated pair except the party itself. So, this is the blindness against all except the allocated party. More precisely, if the allocation outputs a pair party-data (i, j) the identity of the party behind the index i is unknown to everybody except the party-itself.

Moreover, we define the notion of "Confidentiality" that preserves the privacy of data against all except its allocated party. Satisfying anonymity and confidentiality, simultaneously, can be challenging since the data-owner needs to encrypt the data with the public key of its allocated party, while the party is anonymous and thus its public key is not known. Our solution allows transferring the data to the allocated parties privately and anonymously.

Note that when the allocation is complete, each party knows with which data it is paired. This may arise more attacks against the *accessibility of data*, where

a malicious party tries to replace the legitimate ciphertext with irrelevant data, without being detected. To go around this issue we also present the security notion of "Data-Integrity" which can detect the mentioned attack.

We emphasize that two notions confidentiality and data-integrity have not been studied in the literature of random allocation. Additionally, our definition of anonymity provides a stronger security notion than the mentioned blindness. These properties strengthen random allocation to support more applications such as *distributed audit systems* where auditors should check for the correct executions of different steps in the system. Our security notions allow an auditor to have access only to its allocated part, while the auditor is anonymous and therefore there is less chance for attacks or corruption (meanwhile, the fairness prevents biased judgment).

To summarize, in this paper, we present the first formal cryptographic ground to define and evaluate the security of Random Allocation. We present a protocol that satisfies uniqueness, fairness, and anonymity, and at the same time guarantees data-confidentiality and data-integrity. We call such a protocol as Anonymous Random Allocation (ARA). Our main idea is based on a Single Secret Leader Election (SSLE) scheme [BEHG20, CFG21] which was first motivated by the applications in the proof of stake for the blockchain [BPS16]. The SSLE scheme of [BEHG20] relies on shuffling [Dur64, Hås06, Hås16], re-randomization and commitment. Intuitively, re-randomization and shuffling are mainly used to achieve anonymity, while the commitment is used to reveal the allocations at the end of the protocol. The technique of Rerandomize-and-Shuffle (R&S) is a well-known concept in Mix-Nets [GJJS04,Wik04] which guarantees the anonymity of the senders. The intuition behind this technique is that it inserts the entropy into the relationship between the input and the output of a R&S phase, such that after enough steps of R&S, no one can link the last output to the initial input.

Here, we use a card-based example to explain the intuition behind SSLE and its extension to ARA protocol. Assume that there is a deck of cards and we agree that after shuffling the deck and distributing the cards among the players, the winner is the one who gets the “king of pick”. The winner can later prove that he/she is the winner, simply by revealing the target card. Here, there is an attack where one can prepare a copy of the target card in advance and put it in his/her hand. To prevent this attack, the cards should be distributed transparently. More precisely, instead of hiding the cards, we hide the players’ identities. First, in an *anonymity phase*, each player chooses a pseudonym, where this pseudonym is the commitment C_s to a selected (unique) secret s, pseudonyms are re-randomized and shuffled among the players such that at the end of this phase each player can link an output to its own pseudonym (while it can not find any link among other outputs and pseudonyms). Then in an *allocation phase*, the cards are shuffled and distributed transparently. The winner of the target card can later reveal itself by revealing its secret (which can be verified against the output of the anonymity phase).

While in SSLE a single data has a distinct value (namely, "king of pick"), in an ARA protocol data is treated equally and the aim is just to distribute the data among the parties (with the same anonymity property of SSLE). On the other hand, the main difference between SSLE and ARA is in the concept of data. While in SSLE data is only an abstraction, in ARA data is based on real-world information. ARA protocol not only extends the idea of SSLE to the distribution of data (rather than single selection), but also allows the existence of real data which can be transferred to the allocated party *privately* and *anonymously*. Such extensions make ARA a proper tool for applications, dealing with real data, demanding privacy of data and anonymity of parties (with revelation at the end) simultaneously. Furthermore, in our security model, we consider parties malicious and the data-owners honest-but-curious.

1.1 Our Technique

The intuition of our ARA protocol is as follows.

Each party P chooses a random secret s, commits to s through the commitment C_s, and adds C_s to the common list L. Then it re-randomizes and shuffles the list, where re-randomization is done as C_s^r for a randomly chosen r. The data-set would be shuffled and assigned to the elements of the list L in order. Intuitively, re-randomizations and shuffles guarantee the anonymity of the parties. While the correct shuffling and the binding property of commitment guarantee uniqueness and fairness. The commitment C_s (to the secret s) is generated such that the party can detect its associated entry in the list L, even after re-randomization. Let the commitment to the secret s be $C_s = (g^r, g^{sr}) = (u, v)$, then the relation $v = u^s$ is preserved through all the re-randomizations. This is important that the commitment is detectable by its owner, as the party needs to reveal itself at the end of the protocol. The above idea - *first commit, then re-randomize and shuffle*- was used in SSLE [BEHG20].[1]

We further extend this protocol to guarantee data-confidentiality and data-integrity. To do so, in the anonymity phase, each party appends its public key to the list L as well. Thus, at the end of this phase, the public key is re-randomized as $\mathsf{apk} = \mathsf{pk}^{r_a}$, where r_a is due to the re-randomization steps during the protocol, and we call it aggregate randomness. The data-owner can encrypt its message $M \in \mathbb{G}$ by a scheme similar to the El-Gamal encryption as $(\mathsf{ct}_0, \mathsf{ct}_1) = (R^r, M \cdot \mathsf{apk}^r)$, where $R = g^{r_a}$ is the (encoding of) aggregate randomness. Thus, to provide access to R, we add the generator g to the list L as well. Namely, in the set L, at first we have $l = (C_s, \mathsf{pk}, g)$ and after re-randomization steps, it changes to $l = (C_s^{r_a}, \mathsf{apk} = \mathsf{pk}^{r_a}, R = g^{r_a})$ which makes the mentioned encryption possible. For the aim of data-integrity, before allocating data to parties, we also add commitments C_m to a data-set DB. After the party has received its ciphertext, it decrypts and compares it against the commitment C_m.

[1] One can say that their protocol has two main phases: anonymity and *random single selection*, while our protocol has a similar anonymity phase pursuing a *random allocation* phase.

Note that unlike the secret s which is refreshed for each allocation, the secret/public key is fixed. Thus, we should make sure the adversary can not link two randomized public keys in two different allocations and break the anonymity. At first glance, the above idea may seem appropriate since the public key is re-randomized as well, and thus by the DDH assumption the re-randomized versions of the public key (in different allocations) all seem random. Unfortunately, the reasoning fails since, by the aim of the protocol, parties have to decrypt their associated ciphertext and check for the data-integrity. This means in the security proof, the simulator needs access to the secret key to simulate concurrent decryption from other allocation queries (while the secret key is not given through the DDH samples). To go around the mentioned issue, we use fresh random encryption keys for each allocation but bind the secret s to the identity of the party through another commitment $H'(\mathsf{pk}, g^s)$ such that after the revelation of s the owner of the pk would be recognized as the allocated party. Thus, the secret s is used for building a detectable-commitment that would be revealed later, the encryption keys $(\mathsf{esk}, \mathsf{epk})$ are used for data-encryption, and pk is used as the unique and fix identity of the party.

Putting together, our technique can be summarized as follows,

1. Each party chooses a fresh secret s and a fresh encryption key pair $(\mathsf{esk}, \mathsf{epk})$, it generates a detectable commitment C_s and appends (C_s, epk, g) to the list L. Then it re-randomizes and shuffles L. It also registers the commitment $H'(\mathsf{pk}, g^s)$.
2. For each data, we add a commitment C_m to the data-set DB. Then the elements of DB are randomly allocated to the element of L. Let us call the allocation as $A_{\mathsf{DB}} = \{(l, C_m) \in L \times \mathsf{DB}\}$.
3. Data is encrypted under its allocated encryption key.
4. The party can reveal its secret and its identity pk to claim an allocated pair.

To achieve confidentiality and data-integrity at the same time, the commitment C_m to the message $M \in \mathbb{G}$ should be hiding (meaning that it does not leak information about the message M). Here we use the hash function H (modeled as the random oracle [BR93, FS87]) to commit to M as $C_m = H(M, D)$ where D is chosen uniformly at random from the appropriate set. Then not only the encryption of M, but also the encryption of D is sent to the party making it possible to compare the received data against the commitment C_m. The opening D must belong to the group $\mathbb{G}$ since we are using the El-Gamal encryption scheme.

1.2 Related Work

Transfer of real data privately and anonymously is not a new problem, and is widely studied in the context of oblivious transfer [Rab05, IPS08], mix-nets [Cha81], onion routing systems [DMS04] and non-interactive anonymous router (NIRA) [SW21].

A 1-out-of-n oblivious transfer is a protocol where the sender holds several messages and the receiver chooses one of them such that the sender transfers the chosen message to the receiver whiteout knowing the choice of the receiver, and mutually the receiver does not get any information about other messages. In an ARA protocol, messages come from different senders and are allocated to different receivers who do not trust each other, this makes ARA a different problem.

Mix-nets and onion routing systems provide a way for anonymous communication where usually the sender of the ciphertext knows with who he/she is communicating and the ciphertext goes through a MPC (i.e., interactive messaging)[2], to hide this information from others. In ARA and non-interactive anonymous router, the assignment (of senders to receivers) is preprocessed which allows separating the (interactive) setup phase from the (non-interactive) messaging phase. Thus, ARA and non-interactive anonymous router are somehow orthogonal concepts to the mix-net and onion routing.

In a non-interactive anonymous router (NIRA), the router can allocate the senders to the receivers while neither the router nor the sender knows the assignment. More detailed, a trusted setup, which is realized by a trusted authority, provides n sender keys, n receiver keys and a token such that the private permutation is encoded inside the token. Each sender can send private messages to its allocated receiver by encrypting its message with its own key, then a router holding the token receives all the ciphertexts and navigates them to the allocated receivers (without knowing the allocation). Finally, the receiver uses its assigned key to decrypt the message.

Unlike NIRA which has an on-the-fly messaging, our ARA protocol has a predefined messaging phase[3] (due to the data-integrity), and the router is replaced with a pool. The ciphertexts are stored in a hash table and each party/receiver can get access to its assigned ciphertext through a flag (where the flag is its randomized public key associated with the ciphertext).

We emphasize that an ARA protocol supports not only anonymity and random allocation but also revelation. This property enables the parties to stay anonymous during the protocol but reveal themselves later, to claim some advantages or to give a proof (for example, to prove the end of the task and so claim their compensation).

Putting together, while ARA shares some similarities with the existing works, it follows a different aim and setting (Random allocation, private and anonymous predefined messaging, and revelation property).

[2] Indeed we are using the mix-net idea as a setup phase.

[3] More precisely, in NIRA a sender chooses its message at the moment, while in ARA the message is the one that was previously committed during the allocation phase.

2 Preliminaries

Notation. In this paper, the security parameter is denoted by κ. We say that a function $\mathrm{negl}(x)$ is negligible if $\mathrm{negl}(\kappa) \leq \kappa^{-\omega(1)}$. For a probabilistic polynomial time (p.p.t) algorithm A, the notation $y \leftarrow A(x)$ means: A receives x as the input and outputs y. The notation $x \xleftarrow{R} X$ stands for sampling the element x uniformly from the set X. We use $\mathcal{A}^{\mathcal{O}(\cdot)}$, to show that $\mathcal{A}$ has the oracle access to the algorithm or interactive protocol $\mathcal{O}$.

Definition 1 (DDH Assumption[DH76]). *For a cyclic group $\mathbb{G}$ of the prime order $p = \mathrm{poly}(\kappa)$ and with the generator g (described by $\mathcal{G} = (\mathbb{G}, p, g)$), we say that the DDH assumption holds in $\mathbb{G}$ if for any p.p.t. adversary $\mathcal{A}$ there is a negligible function* negl *such that for $a, b, c \xleftarrow{R} \mathbb{Z}_p$,*

$$|\Pr[\mathcal{A}(\mathcal{G}, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(\mathcal{G}, g^a, g^b, g^c) = 1]| \leq \mathrm{negl}(\kappa)$$

Definition 2 (Shuffle and Random Beacon (RB)). *A random shuffle is a random reordering of the elements of a set. A random beacon is a center or software that generates random values/shuffles. There are different techniques for generating the randomness: randomness from the financial data [BCG15, CH10], or based on cryptographic elements [NP, HMW18, BSL+20, SJK+17] which may provide more features like public verifiability.*

Note that our ARA protocol is already interactive, considering (black-box) interactions with RB improves the efficiency and is pretty realistic since many organizations are using RB for generating their required randomness.

In Appendix A, we also give the definition of other primitives that we will use in ARA, such as hash family, commitment scheme, public key encryption system, and their security notions.

2.1 Syntax and Security of ARA

An ARA protocol is formally defined as follows.

Definition 3 (Anonymous Random Allocation). *An Anonymous Random Allocation is a tuple of six main algorithms/protocols $\Pi = (\mathsf{Setup}, \mathsf{Anonymity}, \mathsf{Alloc}, \mathsf{Enc}, \mathsf{Reveal}, \mathsf{Verif})$ among a set of parties P_i and data-owners D_i as follows (here N is the total number of parties in the system),*

- $\mathsf{Setup}(1^\kappa, N)$*: it generates and returns the public parameters* pp*, a state* st_0*, and public/private keys* $(\mathsf{sk}_i, \mathsf{pk}_i)_{i\in[N]}$*. All the other algorithms implicitly use* pp *in their inputs.*
- $\mathsf{Anonymity}(\mathsf{st}_c)$*: it is a protocol among a subset of parties (indexed by c), each party who joins, updates the public state* st_c *(set as* st_0 *at the beginning) by adding some public information. Each party keeps track of its corresponding secret information through a private state* pst_c*.*

- $\mathsf{Alloc}(\mathsf{st}_c, \mathsf{DB}, m_1, \ldots, m_n, \mathsf{RB})$*: it is a protocol between parties and data-owners that starts after the anonymity phase. The data-set* DB *is set to the empty at the beginning, and* n *is the number of parties participating in the allocation* c*. Assuming* n *data-owner, each adds its data* C_m *to* DB *where the data corresponds with a message* m*. Parties receive the data-set* DB*, the random beacon* RB*, and* st_c *as their input, and return an allocation between* st_c *and* DB*. This consequently implies an allocation between the set of parties and the data-set called* $A_{c,\mathsf{DB}}$ *(we can denote the allocation as* $A_{c,\mathsf{DB}} = T \cup \{\mathsf{st}_c, \mathsf{DB}\}$ *where* $T = \{(i,j)\} \subseteq \mathsf{st}_c \times \mathsf{DB}$*). They output the allocation* $A_{c,\mathsf{DB}}$*.*
- $\mathsf{Enc}(A_{c,\mathsf{DB}}, m)$*: it returns a ciphertext* ct *associated with a public key* $\mathsf{epk} \in \mathsf{st}_c$ *and the message* m*. Where* m *and* epk *are an allocated pair through* $A_{c,\mathsf{DB}}$*. The pair* $(\mathsf{epk}, \mathsf{ct})$ *is added to a pool.*
- $\mathsf{Reveal}(A_{c,\mathsf{DB}}, \mathsf{pst}_c, c, \mathsf{ct}, \mathsf{pk})$*: Upon receiving* $A_{c,\mathsf{DB}}$*, the party* P *reveals itself as the owner of* pk *and claims a specific pair party-data* $(i,j) \in A_{c,\mathsf{DB}}$ *by sending a proof* $\pi_{i,j}$*. From the pool, it reads the pair* $(\mathsf{epk}, \mathsf{ct})$ *generated by its allocated data-owner* j*. It runs the decryption over the ciphertext* ct *and may send an invalidity request after (if the decrypted data is not compatible with* DB*). Thus the output is* $(\mathsf{pk}, \pi_{i,j}, (i,j), Result)$ *where* $Result = \mathsf{esk}$ *as an invalidity request, and otherwise* $Result = valid$*.*
- $\mathsf{Verif}(A_{c,\mathsf{DB}}, \mathsf{pk}, \pi_{i,j}, (i,j), c, Result, \mathsf{ct})$*: for the allocation* c*, it verifies the correctness of a claim (or an invalidity request) that the data* j *is allocated to the party* i *who owns the public key* pk*.*

Note that while we have a fixed setup algorithm, we may have several allocations w.r.t different subsets of parties. Thus, when we say an allocation is indexed by c, it means that all other algorithms are working with the same state st_c (or $A_{c,\mathsf{DB}}$) and are indexed by c.

Security Notion. For all the following security requirements we consider concurrent security which means: several allocations may happen where honest and corrupted parties participate in any number of allocations together. The corruptions are static meaning that at the beginning of each game the adversary declares which parties are corrupted, we denote the set of corrupted parties by $\mathcal{CP}$. We use the term "user" to refer to both parties or data-owners. The threat model assumes that parties are malicious while data-owners are honest-but-curious (HBC).[4]

The uniqueness guarantees that each allocation is a one-to-one map between the set of parties and the data-set (for some applications this might be one group of data to one party[5]). In other words, the adversary tries to find an allocation (indexed by c) such that it is not a one-to-one map.

[4] The definitions of uniqueness, fairness, and anonymity are the generalization of counterpart definitions from the SSLE to the ARA context [BEHG20] (These extend the definitions- from the single selection- to the distribution of data through a 1-to-1 map. Moreover, we also consider malicious parties and HBC data-owners).

[5] For this case, one can randomly distributes the data among n batches and then runs ARA among the parties and batches-of-data.

Definition 4 (Uniqueness). *We say that an ARA protocol Π is unique, if for any p.p.t. adversary $\mathcal{A}$ there is a negligible function* $\mathrm{negl}(\kappa)$ *such that* $\Pr\left[\mathsf{Unique}_{\mathcal{A}}(1^\kappa) = 1\right] \leq \mathrm{negl}(\kappa)$*, for the experiment* $\mathsf{Unique}_{\mathcal{A}}(1^\kappa)$ *given in Fig. 1. In this experiment, after many oracle queries to* QARA*, the adversary $\mathcal{A}$ outputs an allocation c allocating two pairs party-data (i, j) and (k, t) (for honest parties) such that they breach the 1-to-1 relation.*

Here $\mathcal{A}^{\mathsf{QARA}}$ *means the adversary has the oracle access to the interactive algorithms* $(\mathsf{QAnonymity}_m, \mathsf{QAlloc}_m, \mathsf{QEnc}_m, \mathsf{QReveal}_m)$ *where $\mathcal{A}$ plays the role of corrupted users and challenger C plays the role of all the honest users. We use the same index m for a set of algorithms to show they are related to the same allocation m. For the index c output by the adversary, we have the condition that the adversary has already queried* $\mathsf{QARA}_c(\cdot)$.

$\underline{\mathsf{Unique}_{\mathcal{A}}(1^\kappa)}$:
$(\mathsf{pp}, \mathsf{st}_0, \{\mathsf{sk}_i, \mathsf{pk}_i\}_{i\in[N]}) \leftarrow \mathsf{Setup}(1^\kappa, N)$
$in := (\mathsf{pp}, \mathsf{st}_0, \{\mathsf{pk}_i\}_{i\in[N]}, \{\mathsf{sk}_i\}_{i\in\mathcal{CP}})$
$(c; (i, j), (k, t)) \leftarrow \mathcal{A}^{\mathsf{QARA}(\cdot)}(1^\kappa, in)$
such that $(i, j), (k, t) \in A_{c,\mathsf{DB}}$.
output 1 iff
1. $\mathsf{Verif}(A_{c,\mathsf{DB}}, \mathsf{pk}_i, \pi_{i,j}, (i, j), c, \mathit{Result}, \mathsf{ct}_{i,j}) = 1$ and $\mathsf{Verif}(A_{c,\mathsf{DB}}, \mathsf{pk}_k, \pi_{k,t}, (k, t), c, \mathit{Result}, \mathsf{ct}_{i,j}) = 1$.
2. case I. $(i = k \wedge j \neq t)$ for the honest party i or case II. $(i \neq k \wedge j = t)$ for i or k honest.

Fig. 1: The experiment for Uniqueness.

Remark 1. We emphasize that malicious parties always can exchange their claims since they just can share their secrets among themselves and generate proper proofs. This is the reason why in the experiment $\mathsf{Unique}_{\mathcal{A}}(1^\kappa)$ we conditioned the output on honest parties. Though, the mentioned issue can be prevented by the design, where the verification algorithm ignores all the claims deviating from a one-to-one map (which then by the uniqueness, does not include the honest parties, with overwhelming probability).

Fairness is about the uniform distribution of the allocations and the possibility that the honest party can claim its allocated data. By the uniform distribution, the probability that a specific data is allocated to a specific party is equal for all the data and parties. Thus, the adversary aims to decrease the chance that a specific pair happens, for the honest parties, either by changing the distribution or by disturbing the party in the generation of its proof.

Definition 5 (Fairness). *In the experiment* $\mathsf{Fair}_{\mathcal{A}}(1^\kappa)$*, Fig. 2, after many oracle queries to* QARA*, the adversary $\mathcal{A}$ chooses a data-index j hoping that the chosen index would not be allocated to an honest party in the next (i.e., the last) allocation (or no honest party can generate a valid proof to claim otherwise).*

In Fig. 2, let $\mathcal{A}$ play the role of corrupted users and challenger C play the role of honest users. After a challenge request "one more, n", the adversary has

the interactive oracle access just to one allocation indexed by c and including n parties (denoted by $\mathcal{A}^{\mathsf{QARA}_{c,n}}$*), where* $\perp \leftarrow \mathcal{A}^{\mathsf{QARA}_{c,n}(\cdot)}$ *means* $\mathcal{A}$ *does not have any output apart from the ones in the interactive oracle* $\mathsf{QARA}_{c,n}$*. We say that an ARA protocol is fair if for any p.p.t adversary* $\mathcal{A}$ *there is a negligible function* negl *such that* $\left|\Pr[\mathsf{Fair}_{\mathcal{A}}(1^{\kappa}) = 1] - \frac{t}{n}\right| \leq \mathrm{negl}(\kappa)$*, where t is the number of the corrupted parties in the challenge allocation c of the game* $\mathsf{Fair}_{\mathcal{A}}(1^{\kappa})$*, and n is the total number of parties in this allocation.*

$\underline{\mathsf{Fair}_{\mathcal{A}}(1^{\kappa})}$:
$\mathsf{pp}, \mathsf{st}_0, \{\mathsf{sk}_i \mathsf{pk}_i\}_{i\in[N]} \leftarrow \mathsf{Setup}(1^{\kappa}, N)$
$in := (\mathsf{pp}, \mathsf{st}_0, \{\mathsf{pk}_i\}_{i\in[N]}, \{\mathsf{sk}_i\}_{i\in\mathcal{CP}})$
$(\text{one more}, n; j) \leftarrow \mathcal{A}^{\mathsf{QARA}(\cdot)}(1^{\kappa}, in)$
where $\mathcal{A}$ chooses an index $j \leq n$ (a data-owner).
$\perp \leftarrow \mathcal{A}^{\mathsf{QARA}_{c,n}(\cdot)}$
output 1 if for any honest party i in the allocation c:
$\mathsf{Verif}(A_{c,\mathsf{DB}}, \mathsf{pk}_i, \pi_{i,j}, (i,j), c, \mathit{Result}, \mathsf{ct}_{i,j}) = 0$

Fig. 2: The experiment for Fairness.

The anonymity guarantees that for a target data, as long as its allocated party does not reveal itself, no one can predict the allocated party.

Definition 6 (Anonymity). *In the experiment* $\mathsf{Predict}_{\mathcal{A}}(1^{\kappa})$*, Fig. 3, after many oracle queries to* QARA*, the adversary* $\mathcal{A}$ *chooses a data-index j hoping that in the next (i.e., last) allocation the chosen index would be allocated to an honest party and the adversary can detect the party (before the parties reveal themselves).*

In Fig. 3, let $\mathcal{A}$ *play the role of corrupted users and challenger C play the role of honest users. After a challenge request "one more, n", the adversary has the interactive oracle access just to one allocation, indexed by c and including n parties, as* $\mathsf{QAnonymity}_{c,n}$*,* $\mathsf{QAlloc}_{c,n}$ *and* $\mathsf{QEnc}(c, n)$ *(but not to* $\mathsf{QReveal}_{c,n}$*). We say an ARA protocol* Π *is Anonymous if for any p.p.t adversary* $\mathcal{A}$ *there is a negligible function* negl *such that* $\Pr[\mathsf{Predict}_{\mathcal{A}}(1^{\kappa}) = 1] \leq \frac{1}{n-t} + \mathrm{negl}(\kappa)$ *where t is the number of corrupted parties in the allocation c. This means, for the honest parties, the adversary can not guess their allocated data, with the probability of more than* $1/(n-t)$*.*

Confidentiality protects the data against all but the allocated party.

Definition 7 (Confidentiality). *In the experiment* $\mathsf{IND}_{\mathcal{A}}(1^{\kappa})$*, Fig. 5, after many oracle queries to* QARA*, the adversary* $\mathcal{A}$ *outputs two messages* (m_0, m_1) *, the challenger chooses one of them randomly and embeds it in the data-set* DB *of the last allocation (for the honest users), they participate in the last allocation with the mentioned* DB*. The adversary wins if it can guess the chosen message.*

In Fig. 5, let $\mathcal{A}$ *play the role of corrupted users and challenger C play the role of honest users. The adversary can send queries to* QAlloc *for messages m (to be embedded in the data-set). It would send a challenge pair* (m_0, m_1) *for a*

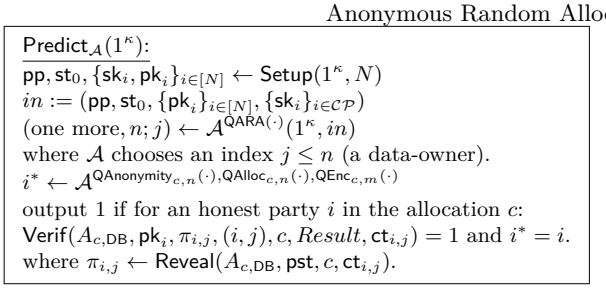

Fig. 3: The experiment for Anonymity.

challenge allocation c (to be used in Alloc_c for the honest data-owners, denoted as $\mathsf{Alloc}(\mathsf{st}_c, \mathsf{DB}(m_b))$). We say that an ARA protocol is confidential if in the experiment $\mathsf{IND}_{\mathcal{A}}(1^\kappa)$, for any p.p.t adversary $\mathcal{A}$ there is a negligible function $\mathrm{negl}(\kappa)$ *such that* $\Pr[\mathsf{IND}^b_{\mathcal{A}}(1^\kappa) = 1,\ b \xleftarrow{R} \{0,1\}] \leq 1/2 + \mathrm{negl}(\kappa)$.

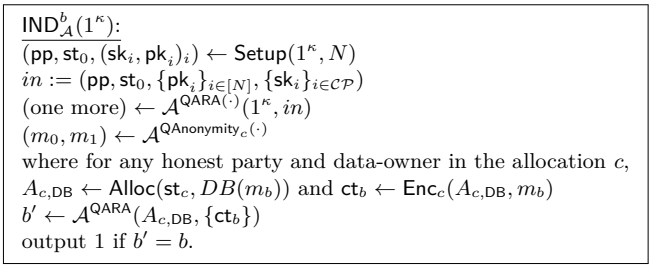

Fig. 4: The experiment for Confidentiality.

Remark 2. Note that we can not preserve the confidentiality of data that is allocated to the adversary. Simply, because data is revealed to its allocated party, and the adversary already knows the data. That is why the challenger gives ct_b only for honest users.

The Data-integrity guarantees that an honest party would get access to the correct allocated data stored in DB. More precisely, the adversary tries to find a pair party-data (i, j) for an allocation c such that the embedded message m in the data-set is different from the embedded message in the ciphertext, while the verification still passes. Data-integrity can prevent man-in-the-middle attacks where the adversary tries to replace the legitimate ciphertext with a malicious one.

Definition 8 (Data-Integrity). *We say that an ARA protocol Π has Data-Integrity, if for any p.p.t. adversary $\mathcal{A}$ there is a negligible function* negl *such that* $\Pr[\mathsf{Integrity}_{\mathcal{A}}(1^\kappa) = 1] \leq \mathrm{negl}(\kappa)$, *where in the experiment* $\mathsf{Integrity}_{\mathcal{A}}(1^\kappa)$ *the adversary has access to the oracle* QARA, *then it outputs an allocation c which she has already sent a query for that, such that the allocation includes an allocated pair party-data (i, j) and a ciphertext $\mathsf{ct}_{i,j}$ (associated with the allocation c), where the party i is honest and $\mathsf{ct}_{i,j}$ is the corresponding ciphertext from the data-owner j to the party i.*

$\mathsf{Integrity}_{\mathcal{A}}(1^{\kappa})$:
$(\mathsf{pp}, \mathsf{st}_0, (\mathsf{sk}_i, \mathsf{pk}_i)_i) \leftarrow \mathsf{Setup}(1^{\kappa}, N)$
$in := (\mathsf{pp}, \mathsf{st}_0, \{\mathsf{pk}_i\}_{i\in[N]}, \{\mathsf{sk}_i\}_{i\in\mathcal{CP}})$
$(c, (i,j), m_j, \mathsf{ct}_{i,j}) \leftarrow \mathcal{A}^{\mathsf{QARA}(\cdot)}(1^{\kappa}, in)$
The challenger sets $m'_j = \mathsf{Dec}(\mathsf{pst}_c, ct_{i,j})$ (where pst_c is the private state of party i).
It outputs 1 if $m'_j \neq m_j$ and m'_j and m_j are both compatible with DB.

Fig. 5: The experiment for Data-Integrity.

3 Our ARA protocol

In this section, we present our ARA scheme based on three main building blocks; R&S, commitment, and encryption system where all are instantiated based on the DDH assumption or collision-resistant hash.

The general idea is that each party, who joins the system, adds its encryption key $(g, g^{\mathsf{esk}}) \in \mathbb{G}^2$ and a detectable-commitment $C_s = (g^{r'}, g^{r's}) \in \mathbb{G}^2$ to the public state and then re-randomizes and shuffles the state. On the other hand, each data-owner generates a hiding commitment C_m to its message m and adds the commitment to DB. Then the elements in the data-set DB are allocated to the elements of the public state.

After the allocation phase, each data owner encrypts its data (message m) and the opening of its commitment. The encryption is done via a randomized public key allocated to its data, via the allocation phase. The allocated party uses its secret key esk to decrypt the ciphertext and if the result is not compatible with the commitment in the DB, it arises an invalidity request. It also outputs the opening of its detectable-commitment C_s, to reveal itself as the allocated party.

There are some details to make the encryption and decryption possible. In particular, we use the El-Gamal encryption. Note that the decryption algorithm of the El-Gamal system only works on messages which are either in the group $\mathbb{G}$ or have a small size (to make the discrete-logarithm operation possible). On the other hand, in our ARA protocol, a data-owner should send the opening (of its committed data) to its allocated party by the El-Gamal encryption. Therefore, we use a commitment scheme where the opening factor belongs to the group $\mathbb{G}$, to be compatible with the decryption of El-Gamal.

As mentioned before, a necessary property is the anonymity which cuts any link between the identity of the party (i.e., its public key) and its inputs to the protocol (through different allocations). For this aim, in each new allocation, parties use fresh inputs (including the secret s for the detectable-commitment and the encryption keys $(\mathsf{esk}, \mathsf{epk})$), while the connection between the inputs and the fixed public key is revealed at the end when the allocation is done. This allows the party to claim an allocation associated with its public key while the anonymity is preserved.

In the following construction $H_1 : \mathbb{G} \rightarrow \{0,1\}^{\mathrm{poly}(\kappa)}$, $H_2 : \mathbb{G} \times \mathbb{G} \rightarrow \{0,1\}^{\mathrm{poly}(\kappa)}$ and $H : \mathbb{G} \times \mathbb{G} \rightarrow \{0,1\}^{\mathrm{poly}(\kappa)}$ are hash functions where H is modeled as the random oracle.

3.1 Construction

Setup: as the input, it receives the security parameters κ and the number of parties N. It creates an empty list L, for the cyclic group $\mathbb{G}$ of prime order p it chooses a generator g, and sets $\mathsf{pp} = (\mathbb{G}, g, p)$ and $\mathsf{st}_0 = L$. It outputs $(\mathsf{pp}, \mathsf{st}_0, \{\mathsf{pk}_i\}_{i \in N})$ where $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ and $\mathsf{sk}_i \xleftarrow{R} \mathbb{Z}_p$ is the secret key of party P_i.

Anonymity Phase: For the allocation c, let the state be $\mathsf{st}_c = (L, t_1, \ldots, t_{i-1})$. Let parties join in order. If P_i is the newly joint party:

- It checks that there are no two same t_j in the state st_c.
- It chooses a secret $s \xleftarrow{R} \mathbb{Z}_p$ and an encryption secret key esk, then it generates an identifiable commitment as $C_s = (u, v) = (g^{r'}, g^{r's})$ for $r' \xleftarrow{R} \mathbb{Z}_p$.
- It appends C_s, the encryption public key $\mathsf{epk} = g^{\mathsf{esk}}$ and the generator g to the list L, i.e., $L \leftarrow L \cup (C_s, \mathsf{epk}, g)$. It stores s and esk in its private state pst_c.
- It re-randomizes and shuffles the elements of L as $l_j^{r_j}$ (where l_j stands for the j-th entry of L) for a randomly chosen $r_j \xleftarrow{R} \mathbb{Z}_p$.
- Finally, it updates the state as $\mathsf{st}_c = (L, t_1, \ldots, t_i)$ where $t_i = (H_1(g^s), H_2(\mathsf{pk}_i, g^s))$ and different from other values t_j. It outputs the updated state st_c.
- **Generating Proof Of R&S**: If P has joined previously (i.e., $P \in \{P_1, \ldots, P_{i-1}\}$): let its identifiable commitment be as $C = (u, u^s)$, P verifies the output of the party P_i by finding the randomized version of C (i.e., it searches for a pair $C' = (u', v')$ from the output such that $v' = u'^s$).

Allocation Phase: This protocol receives the last state st_c and the set $\mathsf{DB} := \emptyset$.

- Each data-owner generates the commitment $C_m = H(M, g^d)$ to its data $M \in \mathbb{G}$, for a unique value $d \xleftarrow{R} \mathbb{Z}_p$ and locally stores $m = (M, g^d)$.
- They add the commitments C_m to the data-set DB.[6]
- Parties receive DB and a public random shuffle (from RB) and apply it over the data-set DB.
- Finally, the elements of the resulting set are allocated one-by-one to the elements of the list L in order, where the resulting allocation is called $A_{c,\mathsf{DB}}$. We will show an allocated pair as $(l, C_m) \in A_{c,\mathsf{DB}}$ where $l = (C_s^{r_a}, \mathsf{aepk} = \mathsf{epk}^{r_a}, R_a = g^{r_a})$ is an element of L and r_a is due to the re-randomization steps (we also can assume that $\mathsf{st}_c, \mathsf{DB} \in A_{c,\mathsf{DB}}$).

Encryption Phase: Upon receiving the allocation $A_{c,\mathsf{DB}}$, such that $(l, C_m) \in A_{c,\mathsf{DB}}$ (and $C_m = H(M, g^d)$ is associated with the data-owner D), the data-owner D encrypts its data $M \in \mathbb{G}$ and its chosen revealing factor g^d by El-Gamal encryption as,

$$\mathsf{ct}_0 = R_a^r, \mathsf{ct}_0' = R_a^{r'},\ \mathsf{ct}_1 = M \cdot \mathsf{aepk}^r,\ \mathsf{ct}_1' = g^d \cdot \mathsf{aepk}^{r'}$$

where aepk is the anonymous encryption public key of its allocated party, $r, r' \xleftarrow{R} \mathbb{Z}_p$, and R_a is the (encoding of) aggregate randomness (note that $l = (C_s^{r_a}, \mathsf{aepk} =$

[6] Since the data-owner is not anonymous, we always can append a unique identifier to C_m to be sure the elements of DB are unique.

$\mathsf{epk}^{r_a}, R_a = g^{r_a}$)). The ciphertext is sent to a pool that can be stored in a key-value hash table (i.e., key=$H(\mathsf{aepk})$, value= $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1; \mathsf{ct}'_0, \mathsf{ct}'_1)$).

Reveal: Upon receiving the allocation $A_{c,\mathsf{DB}}$ such that $(l, C_m) \in A_{c,\mathsf{DB}}$ and reading the ciphertext ct (through the pool), the party P as the owner of esk (where esk is the secret key associated with aepk):

- Decrypts $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1; \mathsf{ct}'_0, \mathsf{ct}'_1)$ by its encryption secret key esk as $M' = \frac{\mathsf{ct}_1}{\mathsf{ct}_0{}^{\mathsf{esk}}}$ and $g^{d'} = \frac{\mathsf{ct}'_1}{\mathsf{ct}'_0{}^{\mathsf{esk}}}$.
- From its local storage, it recovers its secret s for the commitment C_s (as the proof for the allocated pair $(l, C_m) \in A_{c,\mathsf{DB}}$.
- It sends an invalidity request (for data-integrity) by setting $\mathit{Result} = \mathsf{esk}$ if the decrypted data is not compatible with the commitments C_m in the data-set. More precisely, let $M', g^{d'}$ be the decrypted data:

$$M' = \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_0, \mathsf{ct}_1),\ g^{d'} = \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}'_0, \mathsf{ct}'_1)$$

  It sends an invalidity request (i.e, $\mathit{Result} = \mathsf{esk}$) if $C_m \neq H(M', g^{d'})$, Otherwise $\mathit{Result} = \mathit{valid}$.
- Finally, it outputs $(\mathsf{pk}, \pi = s, (l, C_m) \in A_{c,\mathsf{DB}}, \mathit{Result})$.

Verification: upon receiving the allocation $A_{c,\mathsf{DB}}$, the allocated pair $(l, C_m) \in A_{c,\mathsf{DB}}$, the public key pk, the proof $\pi = s$, the request Result and the ciphertext ct (associated with $\mathsf{aepk} \in l$), the verifier:

- Checks if $t = (H_1(g^s), H_2(\mathsf{pk}, g^s)) \in \mathsf{st}_c$.
- For $C_s^{r_a} = (u, v) \in l$, it checks the relation $v = u^s$.
- If the above checks pass, it accepts pk as the owner of the allocated pair $(l, C_m) \in A_{c,\mathsf{DB}}$.
- For $\mathit{Result} = \mathsf{esk}$, first, it checks if $R_a^{\mathsf{esk}} = \mathsf{aepk}$, then it decrypts the associated ct as $(M', g^{d'})$ (through esk), and if $C_m \neq H(M', g^{d'})$, it confirms the invalidity request.

4 Security Analysis

In this section, we analyze the security properties of our ARA protocol.

To prove the uniqueness, we need to show that there is a one-to-one map between the set of registered parties and the data. To do so, first, we show that there is a one-to-one map between the set of registered parties and the set L, from there evidently there is a one-to-one map between L and DB by the construction.

Theorem 1. *If the values t are unique, and H_2 is collision-resistant, then our ARA scheme has the uniqueness property.*

Proof. By the definition of uniqueness (Definition 4), we have,

$$\Pr[\mathsf{Unique}_{\mathcal{A}}(1^\kappa) = 1] \leq \Pr[\mathsf{Unique}_{\mathcal{A}}(1^\kappa) = 1|\ \text{case I}] + \Pr[\mathsf{Unique}_{\mathcal{A}}(1^\kappa) = 1|\ \text{case II}]$$

Since the values t_j are unique, and by the correctness of public shuffle, an honest party would not get assigned to two different data. Thus, case I can not happen. The only situation that case II may happen (i.e., an honest and a malicious party are allocated to the same data) is,

– If $C_s = C_{s'}$ for $s \neq s'$. This is prevented by the binding of $C_s = (g^r, g^{rs})$.[7]
– During the reveal phase the adversary $\mathcal{A}$ can copy the secret s as its own claim. This is prevented by the collision-resistance of hash function H_2 since $H_2(\mathsf{pk}, g^s) \neq H(\mathsf{pk}_{\mathcal{A}}, g^s)$ with overwhelming probability. ■

Theorem 2. *If Proofs of R&S are correct and RB outputs uniformly random shuffles, then our ARA protocol satisfies the fairness property.*

Proof. For fairness, two main points help:
– The shuffle on the data-side is uniformly random; this is true since it is a public shuffle generated by RB.
– The pseudonyms of the parties (i.e, C_s) are not removed and are still detectable by the party-itself; this point is guaranteed by the proof of R&S (see the construction, the last step in the anonymity phase or Definitions 13 and 14).

The former guarantees that the probability that a chosen data gets allocated to an honest party is $(n-t)/n + \mathrm{negl}(\kappa)$. The latter guarantees that the honest party can claim its allocated pair successfully (note that in Definition 5, if an honest party can not claim its allocated data, the adversary wins). ■

Intuitively, anonymity is guaranteed since, on one hand, the adversary can not detect the randomized version of values associated with honest parties. On the other hand, the honest shuffles are uniform. Putting these two together, the adversary knows which values (after R& S) are associated with the set of honest parties but it can not specifically associate a value to an honest party. Or said in the other words, the adversary faces an entropy $n - t$ out of n (where t is the number of corrupted parties). In the following theorem, we formalized this intuition. We say a shuffle is honest if it is executed by an honest party.

Theorem 3. *If the DDH assumption holds in the group $\mathbb{G}$ and honest shuffles are uniformly random, then our ARA protocol is anonymous.*

Proof. The proof proceeds through a sequence of games.

$\boxed{\mathbf{G}_0}$: is the real game of anonymity (Definition 6).

Let x be the last honest party in the challenge allocation c, and we show the list L with L_x when it reaches x (for the allocation c).

$\boxed{\mathbf{G}_1}$: is similar to $\mathbf{G}_0$, except that, for each honest party, the associate element $l = ((u, u^s), \mathsf{aepk}, R_a)$ from the list L_x, is replaced with $l = ((u, u^{s'}), \mathsf{aepk}, R_a)$ for $s' \xleftarrow{R} \mathbb{Z}_p$, and from then on s' is used as the secret of the honest user.

$\boxed{\mathbf{G}_2}$: is similar to the game $\mathbf{G}_1$, except that, for each honest party the associated element $l = ((u, u^{s'}), \mathsf{aepk} = R_a^{\mathsf{esk}}, R_a)$ from L_x is replaced with $l = ((u, u^{s'}), R_a^{\mathsf{esk}'}, R_a)$ where $\mathsf{esk}' \leftarrow \mathbb{Z}_p$.

[7] It is a binding commitment because for $s \neq s'$ we have $C_s \neq C_{s'}$ (they are different either in the first entry or in the second entry).

The probability that the adversary wins in the game $\mathbf{G}_2$ is $1/(n-t)$ since s' and esk' are random (and so are not related to any honest party), and particularly that the honest shuffle by x is uniformly random (i.e., neither the values nor the positions do not carry information about the honest parties).

We prove the indistinguishability of adjacent games in Lemmas 1 and 2. ■

Lemma 1. *In Theorem 3, two games $\boldsymbol{G}_0$ and $\boldsymbol{G}_1$ are computationally indistinguishable under the DDH assumption.*

Proof. The proof proceeds through a hybrid of games as follows (where $\mathbf{G}_{0,0} := \mathbf{G}_0$).

$\boxed{\mathbf{G}_{0,\gamma}}$: is similar to the game $\mathbf{G}_{0,\gamma-1}$, except that, the entry $l_\gamma = ((u, u^s), \mathsf{aepk}, R_a) \in L_x$, as the γ-th element of L_x and associated with an honest party, is replaced with $l_\gamma = ((u, v), \mathsf{aepk}, R_a)$ for $v \xleftarrow{R} \mathbb{G}$.

$\boxed{\mathbf{G}_{0-1}}$: is similar to the game $\mathbf{G}_{0,n}$ (where n is the number of parties in the challenge allocation), except that, each randomness v is replaced with $u^{s'}$ for $s' \xleftarrow{R} \mathbb{Z}_p$.

Note that $\mathbf{G}_{0-1} = \mathbf{G}_1$ and $\mathbf{G}_{0,0} = \mathbf{G}_0$. If l_γ corresponds with a corrupted party, clearly two games $\mathbf{G}_{0,\gamma-1}$ and $\mathbf{G}_{0,\gamma}$ are identical. Thus, in the following, we assume that l_γ is associated with an honest party.

We now are ready to show the indistinguishability of games $\mathbf{G}_{0,\gamma-1}$ and $\mathbf{G}_{0,\gamma}$.

Let $\mathcal{A}$ be the attacker to the DDH problem, and the attacker $\mathcal{B}$ tries to distinguish between $\mathbf{G}_{0,\gamma-1}$ and $\mathbf{G}_{0,\gamma}$. The adversary $\mathcal{A}$ (the simulator) simulates the game for $\mathcal{B}$ as follows:

- $\mathcal{A}$ receives the challenge $(\mathcal{G}, A = g^a, B = g^b, C)$ from its challenger where $C = g^{ab}$ or $C \xleftarrow{R} \mathbb{G}$. It generates the public parameters pp and st_0 via $\mathcal{G}$. It generates the secret/public keys by the real algorithm. Finally, it sends all the public keys, pp, st_0, and the secret keys of the corrupted parties to $\mathcal{B}$. It guesses the index y for the honest party corresponding with l_γ. Where l_γ is the γ-th element of list L_x.
- For the honest party y (its guess), it sets the secret $s_y = a$ and so at the anonymity phase $\mathsf{Anonymity}_c$, the party y inserts the commitment $C_s = (g^r, A^r)$ to the list L.
- When the list L reaches the honest party x (i.e., in L_x), if the simulator knows the secret s, associated with l_γ, it aborts (this means l_γ corresponds with an honest party other than y). Otherwise, l_γ is associated with party y and has the form $l_\gamma = ((g^{r_a r}, A^{r r_a}), (g^{r_a \mathsf{esk}}, g^{r_a}))$ where g^{r_a} is due to the previous re-randomizations. It re-randomizes l_γ by considering the randomness $r_\gamma = b r_a^{-1}$ and so l_γ is replaced with $((B^r, C^r), (B^{\mathsf{esk}}, B))$ while the simulator knows r and esk for the honest party. For the other elements of L_x, it behaves like in the real protocol[8].
- The allocation phase QAlloc_c is done like in the real algorithm.

[8] Note that here is why in the construction of ARA we needed to use separate randomnesses r_j to re-randomize the elements $l_j \in L$ (i.e., $l_j^{r_j}$ rather than l_j^r).

– When $\mathcal{B}$ outputs its guess for the winner, if the winner is an honest party and $\mathcal{B}$ has guessed it correctly (experiment $\mathsf{Predict}$ outputs 1), $\mathcal{A}$ outputs 1. Otherwise, it outputs 0.

In the above simulation, if $C = g^{ab}$, the simulator $\mathcal{A}$ has simulated the game $\mathbf{G}_{0,\gamma-1}$ for $\mathcal{B}$, and if C is random, it has simulated the game $\mathbf{G}_{0,\gamma}$. Thus, if the simulation does not abort and $\mathcal{B}$ distinguishes the games by the probability ϵ, the adversary $\mathcal{A}$ solves the DDH problem with the probability $\frac{1}{n-t} \cdot \epsilon$ (where $(n-t)$ is the number of honest parties in the challenge allocation and is appearing here since y is chosen randomly from the set of honest parties).

For the indistinguishability of $\mathbf{G}_{0,n}$ and $\mathbf{G}_{0-1}$: Note that the game $\mathbf{G}_{0,\gamma}$ was equivalent with the case that the DDH challenge C (used in l_γ) is random. Thus in the game $\mathbf{G}_{0,n}$, the entries associated with honest parties all are replaced with random values C. Consequently, we can replace each (random) challenge C with $B^{s'}$ for $s' \leftarrow \mathbb{Z}_p$ (a fresh s' per each C). Note that all the mentioned changes are possible thanks to the fact that for the challenge allocation c, the adversary does not have access to $\mathsf{QReveal}_c$ (and does not verify the commitments C_s, t). ■

Lemma 2. *In Theorem 3, two games $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ are computationally indistinguishable under the DDH assumption.*

Proof. The proof is similar to the proof of Lemma 1, with the difference that, this time the challenge A is used as the encryption secret key esk of the honest party y, and for the re-randomization x uses the randomness $r_\gamma = br_a^{-1}$. This means that before x applies re-randomization, l_γ is $((g^{rr_a}, g^{rsr_a}), (A^{r_a}, g^{r_a}))$ and after it is $l_\gamma = ((B^r, B^{rs}), (C, B))$ (where the simulator knows r and s for the honest parties). All the other steps are similar to the proof of Lemma 1. ■

In the following theorem, we discuss the confidentiality of our ARA scheme. We use the hiding property of the commitment $C_m = H(M, g^d)$ where $d \xleftarrow{R} \mathbb{Z}_p$, and also the CPA-security of El-Gamal encryption. The formal proof is more tricky since the randomness d which guarantees the hiding of the commitment is also embedded in the El-Gamal ciphertext. Thus, we should make sure that no information regarding d is leaking through the ciphertexts.

Theorem 4. *If the DDH assumption holds in the group $\mathbb{G}$, our ARA protocol has confidentiality in the random oracle model.*

Proof. We proceed through a sequence of games (summarized in Fig. 6).

$\boxed{\mathbf{G}_0}$: is the real game $\mathsf{IND}^0_{\mathcal{A}}(1^\kappa)$ (Definition 7).

$\boxed{\mathbf{G}_1}$: is similar to the game $\mathbf{G}_0$, except that, in the simulation of ct and ct', the anonymous encryption public key aepk is replaced with epk. Two games $\mathbf{G}_0$ and $\mathbf{G}_1$ are indistinguishable since the challenger (having the role of the honest party and honest data-owner) knows the key epk and does not need $R_a = g^{r_a}$ to generate ct (remember that in the ARA construction, the key epk was unknown to the encryptor and so we used R_a to make the encryption possible). On the

other hand, two distributions (aepk^r, R_a^r) and $(\mathsf{epk}^{r'}, g^{r'})$ are identical.

$\boxed{\mathbf{G}_2}$: is similar to the game $\mathbf{G}_1$, except that, the message M_0 in the ciphertext is replaced with M_1, for all the honest parties.

This game is indistinguishable from $\mathbf{G}_1$ thanks to the security of El-Gamal encryption. To see this, let $\mathcal{A}$ be the attacker to the CPA-security of El-Gamal encryption and $\mathcal{B}$ be the adversary trying to distinguish two games $\mathbf{G}_1$ and $\mathbf{G}_2$. The adversary $\mathcal{A}$ simulates the game for the adversary $\mathcal{B}$ as follows:

- The adversary $\mathcal{A}$ receives the set of corrupted parties from $\mathcal{B}$. It chooses an honest party P^* randomly, and receives the public key epk^* from its challenger (where it sets epk^* as the encryption public key of its chosen honest party P^* for the challenge allocation c). It runs the real algorithm Setup to generate $(\mathsf{pp}, \mathsf{st}_0, \{\mathsf{pk}_i\}_i)$. Finally, it sends $(\mathsf{pp}, \mathsf{st}_0, \{\mathsf{pk}_i\}_{i\in N})$ and $\{\mathsf{sk}_i\}_{i\in\mathcal{CP}}$ to $\mathcal{B}$.
- For the challenge allocation c, it uses epk^* in $\mathsf{Anonymity}_c$ associated with P^*.
- The adversary $\mathcal{B}$ sends its chosen challenge (M_0, M_1), where $\mathcal{A}$ outputs the challenge M_0, M_1 as well.
- They run $\mathsf{Alloc}(\mathsf{st}_c, \mathsf{DB}(M_0))$, i.e., for the honest data-owners, $\mathcal{A}$ embeds M_0 in the data-set.
- When the adversary $\mathcal{A}$ receives $\mathsf{ct} = \mathsf{Enc}(\mathsf{epk}^*, M_b) = (g^r, M_b \cdot (\mathsf{epk}^*)^r)$ it forwards it to $\mathcal{B}$. It chooses $d \xleftarrow{R} \mathbb{Z}_p$ to simulate $\mathsf{ct}' = \mathsf{Enc}(\mathsf{epk}^*, g^d)$ and $C_m = H(M_0, g^d)$.
- When $\mathcal{B}$ outputs b', the adversary $\mathcal{A}$ also outputs b'.

$\boxed{\mathbf{G}_3}$: is similar to the game $\mathbf{G}_2$, except that, the simulator replaces the randomness d with d' in the ciphertext. The simulation is as before except that the adversary $\mathcal{A}$ sends the challenge $(D_0, D_1) = (g^d, g^{d'})$ for its chosen $d, d' \xleftarrow{R} \mathbb{Z}_p$. It relays $(\mathsf{ct}, \mathsf{ct}') = \mathsf{Enc}(\mathsf{epk}, M_1), \mathsf{Enc}(\mathsf{epk}, D_b)$ and $C_m = H(M_0, D_0)$ to $\mathcal{B}$. The indistinguishability of $\mathbf{G}_3$ and $\mathbf{G}_2$, is similar to the one for $\mathbf{G}_2$ and $\mathbf{G}_1$.

$\boxed{\mathbf{G}_4}$: is similar to the game $\mathbf{G}_3$, except that, for the challenge allocation QAlloc_c, the simulator replaces C_m with $H(M_1, D_0)$. These two games are indistinguishable because D_0 is chosen randomly from a large space and is unknown to $\mathcal{B}$, and thus $\mathcal{B}$ can not efficiently find $H(M_0, D_0)$ or $H(M_1, D_0)$ (through the RO queries).[9]

$\boxed{\mathbf{G}_5}$: is similar to the game $\mathbf{G}_4$, except that, we simulate the ciphertext ct' as $\mathsf{Enc}(\mathsf{epk}, D_b)$. These two games are indistinguishable by the security of El-Gamal, similar to games $\mathbf{G}_2$ and $\mathbf{G}_3$.

$\boxed{\mathbf{G}_6}$: is similar to the game $\mathbf{G}_5$, except that, the key epk is replaced with aepk. The proof of indistinguishability is similar to the one for games $\mathbf{G}_0$ and $\mathbf{G}_1$. Note that this is the real game IND^1 (i.e., IND^b for $b = 1$). ■

Theorem 5. *If the commitment C_m is binding (or equivalently, H is collision-resistant), then our ARA protocol satisfies data-integrity.*

[9] In fact, this is the hiding property of commitment C_m.

Proof. In the experiment $\mathsf{Integrity}_{\mathcal{A}}(1^{\kappa})$, assume that the adversary outputs an allocated pair $(l, C_m) \in A_{c,\mathsf{DB}}$ and also $(m = (M, d), (\mathsf{ct}, \mathsf{ct}'))$, where $l = (C_s^{r_a}, \mathsf{aepk}, R_a)$ and aepk is associated with esk. From the received ciphertexts we have,

$$M' = \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_0, \mathsf{ct}_1),\ g^{d'} = \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}'_0, \mathsf{ct}'_1)$$

The adversary wins if $C_m = H(M, g^d)$ while $m \neq m'$ for $m' = (M', g^{d'})$. Which is equivalent to breaking the binding property of the commitment scheme C_m. This completes the proof. ■

References

Alt96. DG Altman. Better reporting of randomised controlled trials: the consort statement. *BMJ*, 313:570, 1996.

BCG15. Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. *IACR Cryptol. ePrint Arch.*, 2015:1015, 2015.

BCKM13. Eric Budish, Yeon-Koo Che, Fuhito Kojima, and Paul Milgrom. Designing random allocation mechanisms: Theory and applications. *American Economic Review*, 103(2):585–623, 2013.

BEHG20. Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 12–24. ACM, 2020.

BPS16. Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptol. ePrint Arch.*, 2016:919, 2016.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

BSL+20. Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. Randpiper - reconfiguration-friendly random beacons with quadratic communication. *IACR Cryptol. ePrint Arch.*, 2020:1590, 2020.

CFG21. Dario Catalano, Dario Fiore, and Emanuele Giunta. Efficient and universally composable single secret leader election from pairings. *IACR Cryptol. ePrint Arch.*, 2021:344, 2021.

CH10. Jeremy Clark and Urs Hengartner. On the use of financial data as a random beacon. In Douglas W. Jones, Jean-Jacques Quisquater, and Eric Rescorla, editors, *2010 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '10, Washington, D.C., USA, August 9-10, 2010*. USENIX Association, 2010.

Cha81. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

DH76. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

DMS04. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320. USENIX, 2004.

Dur64. Richard Durstenfeld. Algorithm 235: Random permutation. *Commun. ACM*, 7(7):420, 1964.

FD98. Katie Featherstone and Jenny L Donovan. Random allocation or allocation at random? patients' perspectives of participation in a randomised controlled trial. *BMJ*, 317:1177, 1998.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

GJJS04. Philippe Golle, Markus Jakobsson, Ari Juels, and Paul F. Syverson. Universal re-encryption for mixnets. In Tatsuaki Okamoto, editor, *CT-RSA 2004*, volume 2964 of *LNCS*, pages 163–178. Springer, Heidelberg, February 2004.

GXC+17. Zhimin Gao, Lei Xu, Lin Chen, Nolan Shah, Yang Lu, and Weidong Shi. Scalable blockchain based smart contract execution. In *23rd IEEE International Conference on Parallel and Distributed Systems, ICPADS 2017, Shenzhen, China, December 15-17, 2017*, pages 352–359. IEEE Computer Society, 2017.

Hås06. Johan Håstad. The square lattice shuffle. *Random Struct. Algorithms*, 29(4):466–474, 2006.

Hås16. Johan Håstad. The square lattice shuffle, correction. *Random Struct. Algorithms*, 48(1):213, 2016.

HMW18. Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

KL14. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

NP. NIST-Project. NIST randomness beacon. *https://beacon.nist.gov/home*.

Rab05. Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. https://eprint.iacr.org/2005/187.

SG02. Kenneth F Schulz and David A Grimes. Blinding in randomised trials: hiding who got what. *Lancet*, 359 (9307):696–700, 2002.

SG05. Kenneth F Schulz and David A Grimes. Allocation concealment in randomised trials: defending against deciphering. *EPIDEMIOLOGY SERIES*, 359:614, 2005.

SJK+17. Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 444–460. IEEE Computer Society, 2017.

SW21. Elaine Shi and Ke Wu. Non-interactive anonymous router. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 489–520. Springer, Heidelberg, October 2021.

Wik04. Douglas Wikström. A universally composable mix-net. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 317–335. Springer, Heidelberg, February 2004.

A Appendix : Missing Materials

Definition 9 (Collision-Resistant Hash Function). $\mathcal{H} = \{H_i : \mathcal{X} \to \mathcal{Y}\}_{i \in \mathcal{I}}$ *is a family of collision-resistant hash functions if:*

Generation. *There is a p.p.t. algorithm* $\mathsf{Gen}(1^\kappa)$ *which outputs* $i \in \mathcal{I}$.
Efficient evaluation. *Given* x *and* i*, one can compute* $H_i(x)$ *in time polynomial in* κ.
Collision-resistance (CR). *For every p.p.t. adversary* $\mathcal{A}$ *there is a negligible function* negl *such that for* $i \leftarrow \mathsf{Gen}(1^\kappa)$

$$\Pr\left[(x, x') \leftarrow \mathcal{A}(1^\kappa, i) \ : \ \begin{array}{l} x \neq x' \wedge \\ H_i(x) = H_i(x') \end{array}\right] \leq \mathrm{negl}(\kappa).$$

Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption system, in the following definition pp stands for the public parameters (e.g., for the El-Gamal encryption it is the group description).

Definition 10 (IND-CPA encryption [KL14]). *A public key encryption* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CPA, if for any p.p.t adversary* $\mathcal{A}$*, arbitrary messages* m_0, m_1 *(such that* $|m_0| = |m_1|$*), there is a negligible function* negl *such that:*

$$|\Pr[\mathcal{A}(1^\kappa, \mathsf{pp}, \mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_0)) = 1] - \Pr[\mathcal{A}(1^\kappa, \mathsf{pp}, \mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_1)) = 1]| \leq \mathrm{negl}(\kappa)$$

where $(\mathsf{pp}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$.

Definition 11 (Commitment[KL14]). *A commitment scheme is a tuple of four algorithms* $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verif})$ *as follows:*

- $\mathsf{Setup}(1^\kappa)$*: it returns the public parameters* pp*, which is the implicit input of all the following algorithms.*
- $\mathsf{Commit}(m)$*: it returns a commitment* c *to the message* m.
- $\mathsf{Open}(c, m)$*: it returns an opening* π*, for the commitment* c *to the message* m.
- $\mathsf{Verif}(c, m, \pi)$*: it returns* 1*, if the opening* π *and the message* m *are compatible with the commitment* c*, otherwise it returns* 0.

Definition 12 (Security of Commitment: Binding and Hiding).

- **Binding.** *we say that a commitment scheme* $\mathcal{C}$ *is (computationally) binding, if for any p.p.t. adversary* $\mathcal{A}$ *there is a negligible function* negl *such that:*

$$\Pr\left[(c, m, m', \pi, \pi') \leftarrow \mathcal{A}(1^\kappa, \mathsf{pp}) \ : \ \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa) \\ m \neq m' \\ \mathsf{Verif}(c, m, \pi) = 1 \\ \mathsf{Verif}(c, m', \pi') = 1 \end{array}\right] \leq \mathrm{negl}(\kappa)$$

– **Hiding.** *We say that a commitment scheme $\mathcal{C}$ is (computationally) hiding, if for any p.p.t. adversary $\mathcal{A}$, arbitrary messages m_0, m_1 (such that $|m_0| = |m_1|$), and* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$*, there is a negligible function* negl *such that:*

$$|\Pr[\mathcal{A}(1^\kappa, \mathsf{pp}, \mathsf{Commit}(m_0)) = 1] - |Pr[\mathcal{A}(1^\kappa, \mathsf{pp}, \mathsf{Commit}(m_1)) = 1]| \leq \mathrm{negl}(\kappa)$$

A Randomize-and-shuffle (R& S) scheme is a scheme that receives a list $L = \{\ell_i\}_i$ and updates it to L^* such that L^* and L are equal up to a randomization and a shuffle.

Definition 13 (a R& S scheme). *A R& S scheme associated with the list L is a tuple of p.p.t. algorithms as follows,*

– $\mathsf{Randomize}(L)$*: it receives the list $L = \{\ell_i\}_i$, it chooses the randomnesses r_i and randomizes elements of L to $L' = \{\ell'_i\}_i$ such that $\ell'_i = \ell_i^{r_i}$, then it outputs L'.*
– $\mathsf{Shuffle}(L')$*: it received a randomized list L' and shuffle its elements. It outputs the result to L^*. It outputs L^*.*

Definition 14 (Proof of R& S). *A proof of R& S guarantees that the R& S has been executed correctly and L and L^* are equal up to a randomization and a shuffle of their elements. Indeed, it is a simple check comparing L with L^* as follows,*

– *for each element $\ell_i \in L$, there is an element ℓ_j^* in L^* such that $\ell_j^* = \ell_i^{r_i}$ for a randomness r_i.*

Game	Ciphertext & Commitment	Justification
$\mathbf{G}_0$	$\mathsf{ct} = \mathsf{Enc}(\mathsf{aepk}, M_0)$ $\mathsf{ct}' = \mathsf{Enc}(\mathsf{aepk}, g^d; g^{r_a})$ $C_m = H(M_0, g^d)$	Real Game
$\mathbf{G}_1$	$\mathsf{ct} = \mathsf{Enc}(\boxed{\mathsf{epk}}, M_0)$ $\mathsf{ct}' = \mathsf{Enc}(\boxed{\mathsf{epk}}, g^d)$ $C_m = H(M_0, g^d)$	info.the.
$\mathbf{G}_2$	$\mathsf{ct} = \mathsf{Enc}(\mathsf{epk}, \boxed{M_1})$ $\mathsf{ct}' = \mathsf{Enc}(\mathsf{epk}, g^d)$ $C_m = H(M_0, g^d)$	El-Gamal
$\mathbf{G}_3$	$\mathsf{ct} = \mathsf{Enc}(\mathsf{epk}, M_1)$ $\mathsf{ct}' = \mathsf{Enc}(\mathsf{epk}, \boxed{g^{d'}})$ $C_m = H(M_0, g^d)$	El-Gamal
$\mathbf{G}_4$	$\mathsf{ct} = \mathsf{Enc}(\mathsf{epk}, M_1)$ $\mathsf{ct}' = \mathsf{Enc}(\mathsf{epk}, g^{d'})$ $C_m = H(\boxed{M_1}, g^d)$	RO
$\mathbf{G}_5$	$\mathsf{ct} = \mathsf{Enc}(\mathsf{epk}, M_1)$ $\mathsf{ct}' = \mathsf{Enc}(\mathsf{epk}, \boxed{g^d})$ $C_m = H(M_1, g^d)$	El-Gamal
$\mathbf{G}_6$	$\mathsf{ct} = \mathsf{Enc}(\boxed{\mathsf{aepk}}, M_1)$ $\mathsf{ct}' = \mathsf{Enc}(\boxed{\mathsf{aepk}}, g^d)$ $C_m = H(M_1, g^d)$	info.the.

Fig. 6: Overview of games for Confidentiality