A System to Manage Digital Rights Tokens*

Co-Pierre Georg Bill Guo Alex Stewart

Abstract

We present and evaluate the design of a system that allows users to combine private data into data pools using trusted execution environments and manage these pools by issuing digital rights tokens (DRTs) to third-party data analysts. Digital rights token represent specific rights to a pool of data and are issued on the Algorand blockchain. Data analysts can purchase the right to execute open source code on the combined encrypted data and receive the result from this code execution, but not the underlying data. Different from other information infrastructures, the operators of the Nautilus platform cannot access the private data, which protects users' rights to their data. We discuss applications of the platform to financial and healthcare data analytics.

1 Background

1.1 Digital Privacy

For legal scholars, privacy is a contentious concept with different, overlapping and competing, schools of thought [1]. These include : (i) Privacy as the right to be let alone [2]; (ii) Privacy as secrecy [4]; and (iii) Privacy as control over personal information [5]. For this paper, we focus on digital privacy, i.e. privacy that pertains to *"social and economic activity conducted online"* [6]. Discussions of privacy have been shaped by technological and social innovations, ever since Warren and Brandeis' seminal article "The Right to Privacy", published in 1890 under the impression of a boom in photography and newspaper distribution.

Our goal is to find the most suitable definition of digital privacy without binding ourselves to the constraints of existing information systems. Towards this end, first note that all four notions of privacy contain an element of privacy as control over information.

*Support by the Algorand Foundation and Ripple's University Blockchain Research Initiative is gratefully acknowledged.

The Right to be let alone is, in essence, a right to conceal information about oneself, i.e. a notion of control over information and how it is shared with others. Once information is shared, the recipient can process the information in any way they see fit, including sharing it with a third party. This is precisely why the right to be let alone is a limited control right only, as a more general right would include the ability to restrict *how* a recipient processes the information. Furthermore, information about oneself is only a small subsection of the information a person is involved in creating and a more general right would include the ability to control *what* information is shared with others.

Obvious questions about attribution—who should have control over information that is not about oneself—arise, exacerbated by the fact that data is relational and subject to externalities [7]. More broadly, the question is who should control data. Even when it comes to personal information, like the name of your spouse, the answer is only seemingly clear. But sharing this information with anyone reveals the marital status not just of yourself, but also of your spouse, infringing on your spouse' ability to control this information.

In the privacy as secrecy view, privacy is violated by the disclosure of information. According to Richard Posner, *"The other privacy interest, concealment of information, is invaded whenever private information is obtained against the wishes of the person to whom the information pertains"* [4]. There are two problems with this view, both related to control over information. First, obtaining information, e.g. by intercepting communication and storing the intercepted information, says nothing about how the information is used once it is obtained. This idea implicitly assumes that we cannot control how the recipient of information will react to it, which might be true in the context of a human learning the information–we cannot restrict how the brain processes it–but is not true in the context of a machine receiving information. The second problem is that the privacy as secrecy view assumes that information, once shared, can be shared further, i.e. that it is non-rival. This can be seen in the argument by Kenneth L. Karst [8], summarized in Solove [1] as *"sometimes people do not want complete secrecy; rather, they desire confidentiality, which consists of sharing the information with a select group of trusted people."* But it is not a foregone conclusion that data is non-rival.

An alternative formulation to [8] could read: *"people do not want complete secrecy; rather, they desire control, which consists of allowing a select group of people to use the information in a specific way."*

Understanding privacy, and specifically digital privacy, in the context of a *desire to control* what happens with information, is closely aligned with the privacy-as-control over personal information view going back to Westin's original 1967 work [5], where he defines privacy as *"the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others."* Westin's focus on how information is *communicated* to others is warranted because his interest is in privacy, or rather in the harm a person suffers when their privacy is violated. In this regard, Solove adopts Westin's notion of information when he writes that *"Information can be easily transmitted and, once known by others, cannot be eradicated from their minds"* [1]. The underlying assumption is that infor-

mation is communicated and processed by humans, where *"what is once thought, cannot be taken back"*[1] However, the same is in no way true about information that is communicated, processed, and stored by computers.

In Westin's view, control over information is an expression of ownership: *"personal information, thought of as the right of decision over one's private personality, should be defined as a property right"* [5]. We adopt a notion of ownership as *control over a bundle of rights* in the sense of Demsetz [9]. Taken together, this leads to the following definition:

Definition 1. *Information ownership is the control over a bundle of rights attached to an information.*

Since data is the physical embodiment of information, this definition also naturally translates to data ownership.

The above definition of information ownership is protected from the critique of Westin's and Solove's implicit assumption that creator of an information cannot control how a third party processes and uses this information, should it be communicated to them. Rather, each right contained in the bundle can be spelled out explicitly, for example by specifying a piece of code that is allowed to process the information.

However, Definition 1.1 does not address the relational nature of data. To this end, consider an alternative definition where users jointly control the bundle of rights:

Definition 2. *Information co-ownership is the joint control over a bundle of rights attached to an information.*

The definition says nothing about how joint control is organized. Consequently, any system that allows users to own their data must provide a governance system within which joint control can be organized. With all of the above, we are now able to define digital privacy for the purpose of this paper:

Definition 3. *Digital privacy is a user's control over information they created.*

1.2 Digital Rights

Before outlining an information infrastructure that respects users' digital privacy in the next section, it is useful to specify how "rights attached to an information" can be encoded.

To this end, consider the rights granted to the creator of an information in a specific setting: when storing information in a file on a computer with a Unix-based operating system [13].

[1] From Friedrich Dürenmatt in Die Physiker (Physicists). In German: *"Was einmal gedacht wurde, kann nicht mehr zurückgenommen werden."*

Initially, Unix provided three file rights, read, write, and execute, to three types of users, the file's owner, members of the owner's group, and all other users. Assuming that user 1 created a file the Unix information infrastructure implies rights of the form:

$$\begin{array}{rcl} user_1 & :: & r_1 w_1 x_1 \\ user_2 & :: & r_1 w_1 x_1 \\ \vdots & \vdots & \vdots \\ user_N & :: & r_N w_N x_N, \end{array}$$

where $r_i \in \{r, -\}$, and similarly for w and x. In other words, each user i either has or has not the right to read, write or execute a file, i.e. only up to three rights. Consequently, each user only has a very limited bundle of rights and only limited options to control these rights (e.g. transfer them to another user). On Unix, the operating system itself provides some protection for users' rights, with the exception of root, who can always override the file permissions set by the file's owner.[2] While a discussion about the legal status and nature of digital rights is beyond the scope of this paper, our goal is to create an information infrastructure that recognizes digital rights as inalienable rights. The Universal Declaration of Human Rights [14] recognizes privacy as a human right in Article 12: *"No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence [...]"* and the right to private property in Article 17: *"1. Everyone has the right to own property alone as well as in association with others. 2. No one shall be arbitrarily deprived of his property."*

If taken seriously, this implies two criteria that an information infrastructure should satisfy: First, the owner or group of owners of an information must be able to define which digital rights they hold. As a consequence, they must also be able to decide which rights to grant to others. And second, the owner or group of owners of an information must not be deprived of their rights. To satisfy the first criterion, consider an information infrastructure that allows rights of the type

$$\begin{array}{rcl} user_1 & :: & r_1^1 \ldots r_1^K \\ user_2 & :: & r_1^1 \ldots r_1^K \\ \vdots & \vdots & \vdots \\ user_N & :: & r_N^1 \ldots r_N^K, \end{array}$$

where K is the total number of rights implemented by the infrastructure.

Because the number of possible rights is very large, we have decided to represent a right as a Digital Rights Token (DRT). A DRT is defined by a unique name, a short description, a unique data pool address to which the DRT refers, and a unique reference to a piece of code which implements the right. DRTs also include the token supply, and the unique address of a smart

[2] Various cryptographic approaches exist to provide users with a basic notion of digital privacy, the simplest of which is for the user to encrypt their data before storing it.

contract that manages the tokens. Specifically, the smart contract manages the issuance and ownership of tokens and their distribution in the primary market, as well as their redemption. Details of what is included in a DRT can be found in Appendix A.1.

1.3 Applications

Our system is applicable whenever users have private data to which they want to provide access to third parties for ex-ante agreed upon computations (and only those). The ability to ensure that only authorized code is executed is tightly linked to guaranteeing privacy. In fact, the problem of privacy is almost trivial one to solve once data owners can be sure that only authorized code is executed by the hosts without revealing their secret. The most obvious application of a system as described above is the medical sector. Other use cases can be found in cybersecurity and military applications, as well as in the financial sector.

1.3.1 Medical applications

The popularity of blockchain technology has brought forth multiple projects that want to create a distributed collection of medical records for each individual that can be accessed from everywhere, e.g. Medicalchain or Doc.ai. Since medical data is highly sensitive, it needs to be encrypted, at least on the personal level if not even more broadly. Due to the inefficiency of current approaches to compute over encrypted data, this procedures leaves millions of data points useless for not only medical research, but also for data driven diagnostics.

Using the protocol as described above, physicians can create data that they sign over to their patients, to utilize efficiently in data marketplaces. However, with the patients' permission (or if required by law), they are still able to analyze this data by being allowed access. The patients could also set it up such that their treating physician can access the data for free, while researchers, other physicians, and especially insurance companies are required to pay for access.

The system will foster medical research and can lead to fast advancements due to its possibility to compute over highly accurate and granular data. The viability of the code needs to be given to ensure the accuracy of research results that could otherwise lead to harmful conclusions.

1.3.2 Cybersecurity and military applications

Not only security-concerned institutions like the military or the intelligence agencies, but also government facilities need to make sure the software they run behaves as expected and does

not enable security breaches or contain malicious fragments. A code registry which ensures that only approved code is run on data, as proposed in our framework, allows these agencies to better control the code they run on their servers, in particular for cloud applications.

In 2017, President Obama expanded the National Security Agency's ability to share data with other agencies.[3] Raw data can now be accessed by 16 American intelligence groups, raising concerns of privacy advocates. Our system would pave the way for these governmental agencies to cooperate without decreasing their citizens' privacy. This cooperation could even be far more reaching than including just 16 intelligence groups, including national and state police, since no raw data but only computation results are shared. Our system can therefore fundamentally improve national security.

1.3.3 Applications in finance

Privacy is a particularly sensitive issue in the financial services industry. Not only are regulations particularly strict in this sector, but the very nature of financial intermediation is tied to the existence of private information. Three examples are pertinent for our purposes.

First, modern banks consist of sometimes hundreds of subsidiaries that need to coordinate their activities. Using public or even permissioned blockchains is only possible for a small subset of the activities subsidiaries have to carry out. For example, large bank holding companies manage their liquidity within the banking group and consolidate it before accessing the interbank market typically through a single designated subsidiary. If a bank's liquidity position was public knowledge (e.g. because transactions are recorded on a public blockchain), other banks would see when a bank has liquidity needs, which would fundamentally shift bargaining power and make access to liquidity more costly.

Second, credit bureaus collect creditor information from a large number of financial intermediaries, consolidate and clean it, and then make it available to other market participants. This data sharing is necessary for banks to have the full credit history of individuals or companies applying for new loans. Without this data, credit provision would be massively impaired due to asymmetric information, with substantially negative consequences for the real economy,.

Finally, supervisors mandated with maintaining financial stability crucially depend on the availability of accurate and up to date information for all regulated entities. This is one of the key lessons of the global financial crisis of 2007/2008 and the post-crisis regulatory framework mandates not only macroprudential oversight and supervision in the national context, but also internationally. This requires extensive data sharing agreements between different national and international regulators. Existing centralized data storage solutions provide only imperfect security, are costly to maintain, and have only limited functionality.

[3] See e.g. TechCrunch, accessed on 2017-01-13.

In all the cases mentioned above, the system outlined in this paper can provide significant efficiency gains.

1.3.4 Other applications

The current business model on the web is centered around user data. Even though our system returns data ownership to the user, it does not mean that services which are right now provided by siren servers will disappear. Talented machine learning engineers will register their learning algorithms that can then be trained using individuals' user data. These users may even decide to contribute their data for free in exchange for a better service product. It is more likely, however, that individual users are paid for the use of their data and that they use this income to pay for services that are currently "free".

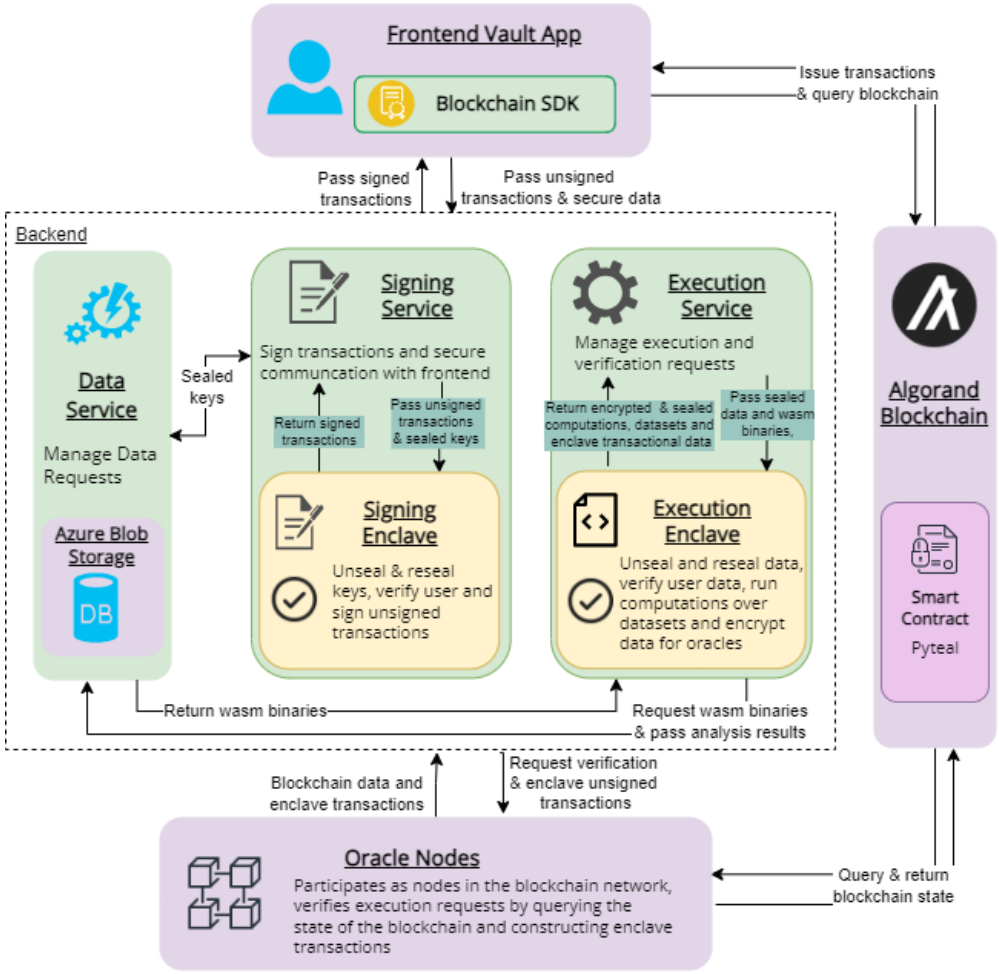
2 System Overview

How does an alternative information infrastructure that facilitates information co-ownership look like? Hanseth and Monteiro identify six aspects of information infrastructures [10]. Information infrastructures (i) have an enabling function; (ii) are shared by a collection of users and user groups; (iii) are open; (iv) are socio-technical systems; (v) are connected and interrelated ecologies of networks; (vi) develop through extending the installed base. Our goal is to outline an alternative information infrastructure which satisfies these six aspects and enables users to own information according to Definition 1.1. Because of this goal, our system is automatically aligned with the first aspect of information infrastructures.

Our system consists of four major components, shown in Figure 1. First, we use two trusted execution environments based on Intel's Software Guard Extension [11]-[12]. The execution enclave facilitates the execution of wasm binaries and provides critical measurements to validate users data contributions in a secure execution environment. The Signing Enclave allows users to delegate the signing of blockchain transactions from a user's wallet in specific, pre-defined circumstances. This is necessary in some of the user stories discussed in Section 2.2. These enclaves are managed by their associated services, the signing service and execution service, which are essentially acting as an API for their respective enclaves facilitating the exchange of data between components. Second, the rights associated with a dataset, i.e. the rights attached to an information, are represented by an Algorand Standard Asset and recorded on the Algorand mainnet.[4] For the purposes of a minimum viable product, Algorand possessed a uniquely suitable combination of smart contract capabilities, low transaction fees and configurable crypto assets. Third, to facilitate the interaction between the signing enclave and the

[4]The choice of blockchain is not paramount to our system design and we could have used other permissionless blockchains providing smart contract capabilities with only minor implications for our system design.

Figure 1: System's diagram for an information infrastructure that facilitates information co-ownership. A dashed line indicates that the components are logically related, even though they are individually implemented. Arrows indicate communication between components.



blockchain, we use a set of oracle nodes. These nodes have two functions: query the state of the blockchain and pass the result to the execution enclave, and create unsigned transactions from data received from the execution enclave to then be signed by the signing enclave and later forwarded to the blockchain. Fourth, the orchestration between the other three components is done via three microservices, the data service, signing service, and execution service. In addition to these four components, we also use a blob storage database, hosted on Microsoft Azure and all user interactions are facilitated by a Vault App.[5]

[5]Our source code is published under the GNU GPL v3 and available on github.

2.1 Threat Model and Limitations

Whoever controls the infrastructure providing the microservices that connect the other system components could, in principle, replace an existing service with a malicious version without users noticing. Consequently, user data must be encrypted with a key shared between the enclave and the user while it is processed by the service layer. In addition, data at rest is encrypted with the private key of the enclave. This ensures that only the enclave can process the raw user data within the protected environment, shielded from e.g. the enclave provider and database operator.

Open source code is used to create reproducible wasm binaries which are then sent to the execution enclave where the code is executed on the raw user data. Users can verify which code is being executed on their data using attestation. Our execution enclave uses a set of oracle nodes to retrieve the state of the blockchain. Once a user redeems a digital rights token by sending it to the smart contract associated with the data pool, the transaction is registered by the oracle nodes and passed to the execution enclave to be verified, which upon successful verification, triggers the execution of the binary code associated with the DRT.

Using a blockchain to record DRTs prevents the copying of authentic DRTs by a malicious third party, which would undermine a user's control over their data; While initially, there is a single oracle node, this can easily be adapted to include multiple oracle nodes and the requirement that k out of n total nodes agree on the state of the blockchain;

The remaining challenge to provide users with control over their data is to prove to users that only approved code is executed on their data. This is done via attestation, provided by the trusted execution environment.[6]. In a first step, our platform only allows the execution of open source code so that users are able to verify that a given wasm binary has been created with a specific piece of code using reproducible builds.

The goal of our platform is not to provide privacy, but control. When a third party executes code on user data, the result of this analysis is reported back to the third party. In that sense, users cede some privacy, since they allow the extraction of information from their underlying data. Crucially, though, our platform allows users to determine exactly how much privacy they want to cede and to whom.

There are various threats to user's control over their information with our information infrastructure. Notably, an extensive literature exists documenting potential attacks on Intel SGX (see, e.g., [15, 16, 17]). Since our current implementation only allows reference to open source code, users can detect whether the code they allow to be executed over their data includes one of the known attacks. However, users could also consent to the execution of malicious code without their knowledge of the malicious effects. This will remain a persistent threat.

[6] Eventually, we will provide a DCAP attestation library to facilitate user attestation queries.

2.2 User Stories

Before going into the exact sequence diagrams for the various user stories our platform caters for, we first give a high-level description of these. The start of any user story is that a user registers on our platform and creates an Algorand wallet.[7] The system caters for four types of users: data creators, data contributors, code creators, and data analysts. Data creators are users who create a new data pool, while data contributors join existing pools. Code creators provide the code referenced in digital rights tokens and data analysts buy DRTs for redemption. Notably, it is possible for a user to play several roles simultaneously, e.g. be a data and a code creator. In principle, our system also caters for a fifth type of user: a trader who buys and sells DRTs for profit.

Create data pool. A data pool consists of a data package and an associated smart contract that manages the issuance, distribution, ownership and redemption of DRTs. The data package includes the user data that is stored encrypted with the private key of the execution enclave and a reference to the data schema used by the pool.

To create a new data pool, a registered user must follow four steps. First, she defines a data schema for the pool according to an open standard understood by our platform. Second, she selects the file containing one or multiple rows of data conforming to the schema. And third, the data creator selects from a list of available digital rights. Each right is represented as a DRT. They are assigned to the pool by calling the smart contract. Once this is done, data is uploaded and the data pool is created. The user then receives a contributor token representing their initial data contribution–which is a DRT in itself.

Join data pool. Users can join existing data pools. To do so, the user must first purchase an AppendDRT either directly from the smart contract managing the data pool, or from another user. Once the user has purchased an AppendDRT, they can transfer it back to the smart contract managing the data pool. This step is necessary so the smart contract has proof that the AppendDRT was sent by its new contributor. Fourth, once the smart contract has received the AppendDRT, the user can start to upload the data. To do so, the user needs to upload data that matches the pool's data schema. Once the data is uploaded, the user receives a contributor token for their contribution which is used to implement revenue sharing among data contributors.

Execute Digital Rights Tokens. Our system recognizes DRTs in four states: defined, created, issued, and redeemed. All DRTs must first be defined before they can be created or issued. Data creators can choose among pre-defined DRTs during the data pool creation or define a new DRT according to their own specification. We do not impose any restrictions on the code that is represented in a DRT as long as it is open source and can be compiled into a wasm binary in a replicable build. Once the pool has been created, the selected DRTs are created as an Algorand Standard Asset, as well as the associated smart contract. Users can browse created DRTs on our

[7] Later, we automate this process so that all users automatically have an Algorand wallet address.

platform and decide to purchase these, either to trade them on or to redeem them. A DRT is redeemed by sending it back to the issuing smart contract.

Users can purchase available DRTs from the smart contracts managing the data pools. Each DRT makes reference to a specific code repository, which implements the data creators' and -contributors' control over their data. DRTs can be either traded or executed (redeemed). DRTs are executed by sending them back to the smart contract managing the data pool. Once this blockchain transaction is recognized by the oracle nodes, they will inform the execution enclave which then pulls the referenced piece of code and uses its own private key to unseal the pool data and execute the code. Once the code execution is completed, the result is encrypted with the private key of the redeeming user for future use.[8]

Claim royalties If an analyst purchases a DRT for the data pool, the smart contract managing the pool receives these funds and allocates them internally to all data contributors in proportion to their contribution to the overall pool. By holding a contributor token to the smart contract, users can claim royalties accrued for their contribution so far; Upon instruction to claim royalties from the user, the smart contract detects the ownership of the contributor token and distributes the royalties. Royalties can be claimed as often as the user wants, although blockchain transaction fees must be paid by the user to do so.

3 Sequence Diagrams

In our sample implementation, we use the Algorand blockchain as distributed ledger for the recording of Digital Rights Tokens. When a user creates a data pool, the system also instantiates a corresponding smart contract to handle the creation, issuance, and execution of digital rights tokens. This section details the sequence diagrams that implement the user stories our platform caters for, summarized in the previous section. The sequence diagrams are shown in Figures 3–6 in Appendix A.2.

3.1 Create Data Pool

The first important functionality provided by our information infrastructure is for a user to create a new data pool. There are seven steps in this process, shown in Figure 3. Before the process starts, the user logs in to the frontend [1–1.6] and establishes a secure communication channel with the signing and execution enclaves by initiating a Diffie-Hellman key exchange and by requesting an attestation from the signing and execution enclaves. These requests are managed by the signing and execution services which both return their attestation reports, enclave public

[8] Other integrations are possible, e.g. with Power BI or simply writing the data to a virtual file system so that the user can integrate it in their own data processing pipelines.

keys, and the shared secrets.

Then, in step one [2–6], the user starts the create data pool flow and selects a data file and a data schema file. These are uploaded and the system verifies that data file and schema are consistent. The user then enters a description for the pool and selects which Digital Rights Tokens to issue.

In step two [7–10], the system creates a deploy smart contract transaction on the Algorand blockchain. The transaction signing process [8.1–8.6] is used several times, so it makes sense to explain it in some detail. To sign a transaction, the user requests the public key of the signing enclave and a shared secret, and encrypts the transaction details using this. The encrypted transaction is then sent to the signing service which in turn sends it to the signing enclave. The signing enclave then uses its key to decrypt the transaction, then sign it by accessing the signing key of the user's Algorand wallet which is accessed by the enclave, and re-encrypt the now-signed transaction with the remote attestation encryption key. With this, the transaction signing process ends. The reason for signing the transaction within the enclave rather than directly by the user is so that users are able to delegate signing transactions. The signed transaction is then sent back to the frontend from where it is issued to the blockchain using the Algorand SDK, which returns the application ID.

Step three [11–13], using the application ID, the frontend creates a payment transaction to fund the smart contract. This is necessary because the smart contract requires a minimal balance to conduct the necessary operations of a data pool. The transaction is signed by the signing enclave and issued to the blockchain using the transaction signing process.

Once this is done, in step four, the frontend requests the remote attestation encryption key from the signing enclave and encrypts the data file once it receives the key. The frontend then passes the encrypted data file and the transaction ID of the smart contract deployment to the execution enclave via the execution service. The execution service triggers [16–17] the creation of a secure communication channel between the execution enclave and the oracle nodes. The nodes request the enclave attestation and share their public key. The enclave returns the attestation report encrypted with the oracle node's public key. Next, the execution service passes the encrypted data and transaction ID to the execution enclave, which decrypts it using the oracle node's public key and a shared secret. The execution enclave then [21] passes the encrypted transaction ID to verify to the oracle nodes via the execution service. The oracle nodes decrypt the transaction ID and query the state of the blockchain for this transaction ID. They encrypt the result with the execution enclave's key and return it to the execution enclave via the execution service [26].

In step five, the execution enclave decrypts the oracle node data and validates that the smart contract has been deployed and that there is enough funds to meet the minimal balance required. This is done via Byzantine fault tolerance, requiring that K out of N nodes report the same state of the blockchain. Upon successful validation, the data is decrypted. The execution enclave then counts the rows of data and computes the hash of the sealed data set. The

execution enclave then counts the rows of data and computes the hash of the sealed data set. Next, the execution enclave parses the oracle node data and gathers the transaction details that would later initialize the smart contract [28–29].

Step six, the execution enclave passes the sealed data to the data service [30], which stores it. It also sends the unsigned transaction data to the oracle nodes which decrypt it and construct the unsigned transaction to initialize the smart contract. This is done via the oracle nodes to prevent a central entity from blocking the smart contract creation and thereby infringing on a user's digital rights. The oracle nodes establish a secure communication channel with the signing enclave [34–35] and encrypt the transaction data with its public key and send it via the signing service to the signing enclave. The signing enclave in turn decrypts the unsigned transaction data, performs the same K out of N Byzantine fault tolerance as before to establish the true state of the blockchain. It then signs the transaction using the wallet key and passes it to the oracle nodes who deploy the smart contract to the blockchain [43].

Initialization of the smart contract includes creating the unique Append DRT for the data pool, the contributor token representing the initial contribution of the pool, and adding the hash of the first data. In the transaction response from the blockchain, the frontend parses and stores the contributor token ID and append DRT ID [44–46].

In step seven, the frontend uses the ID of the contributor token and Append DRT, which it received in step six and then creates an optin transaction to the contributor token. This transaction is signed by the signing enclave using the transaction signing process [48.1–48.6] similar to step two, and issued from the frontend to the blockchain [49]. The smart contract registers the ownership of the contributor token and transfers the contributor token to the user's wallet. The frontend then passes the information of the contributor token to the data service which stores it in blob storage as the last step of the process [55].

3.2 Join Data Pool

The second important functionality is for a user to be able to join an existing data pool. This is also the most complicated functionality because we require that both the pool's existing data as well as the user's new data remain encrypted in transit and at rest.

There are ten steps. Step one [1–1.6], the user logs in and the Frontend establishes a secure communication channel with the execution enclave and requests the Enclave's attestation. Once this is returned, the Frontend queries the data pools from the data service and receives them. The user then selects a data pool to join from a list of available data pools [5]. Next, the user selects data to upload [6] and the platform then verifies the schema [7] to confirm that the new data complies with the schema of the data pool. The data is then uploaded, encrypted with a secret shared between the user and the execution enclave [8].

Step two, the user first must optin to the `AppendDRT` [9–10] because it is an Algorand Stan-

dard Asset in our reference implementation. This is done via the transaction signing process [8.1–8.6 of the Create Data Pool sequence diagram]. The signed transaction is returned to the Frontend from where it is finally sent to the blockchain.

Step three, the user buys an `AppendDRT` in an atomic transfer [11–13], implemented as a group transaction on Algorand.[9] An atomic transfer is a way to solve the escrow problem. The user sends the payment for the `AppendDRT` to the smart contract managing the pool, and receives the DRT in return. Importantly, either both transactions are successful or neither, removing any counterparty risk. The group buy transaction process is also done via the transaction signing process as before. This step ends with the Frontend issuing the group transaction to the blockchain.

Step four, once the `AppendDRT` has been transferred to the user, a second group transaction takes place [14–16]. First, The user transfers the `AppendDRT` back to the smart contract. This step is necessary to ensure that the smart contract keeps an up-to-date list of all contributors for royalty disbursement later. The sending address uniquely identifies the contributor. Second, the user must send a fee for executing the append operation to the smart contract. This fee is fixed in our reference implementation and covers infrastructure costs. And third, the user sends an application call transaction to the smart contract to request being added as data contributor. This group transaction is also signed via the transaction signing process as before. Once the group transaction has been submitted to the blockchain, the smart contract adds the user as a pending contributor and returns the transaction ID to the frontend [17].

Step five, the user retrieves the execution enclave's public key and secret key and encrypts their raw data and the transaction ID [18]. The new encrypted data is then passed to the execution service which issues a request for the current sealed pool data to the data service. Once the execution service receives the current sealed pool data [21], it passes the transaction ID to the execution enclave to commence verification of the transaction. Before this, a secure communication channel is established between the oracle nodes and execution enclave [22-23].

Step six [25–28], the execution enclave requests a verification of the transaction ID from the oracle node via the execution service. The oracle nodes decrypt the transaction ID with the shared secret between the exeuction enclave and the oracle node[29] and queries the state of the blockchain for this transaction ID [30-31]. The oracle nodes then encrypts the transaction information with the execution enclave's public key and shared secret key and returns the encrypted oracle node data to the execution service. The execution service then passes the encrypted oracle node data, the encrypted frontend data and the current sealed pool data to the execution enclave [32–34]. The execution enclave then decrypts and verifies the oracle node data, requiring that K out of N oracle nodes are in agreement about the state of the blockchain [35].

Step seven. In this crucial step, the execution enclave then unseals the pool data and the

[9] For details on group transactions on Algorand, see here.

newly submitted data. It then merges the two data sets and counts the rows in the user's data set. It then seals the joint dataset, computes the hash of the new dataset and parses the transaction details from the oracle nodes [36]. Then, the execution enclave encrypts the number of rows contributed, the new hash, and the unsigned transaction details with the shared secret of the oracle nodes [37]. Next, the execution enclave passes the sealed data to the data service and the unsigned transaction data to the oracle node, both via the execution service. The data service stores the sealed data [39–40] and the oracle nodes decrypt the transaction data and constructs an unsigned transaction to approve the data contributor [41–43].

In step eight, the oracle nodes establish a secure communication channel with the signing enclave by requesting the enclave's attestation and retrieving its public key [44–45]. With this, the oracle nodes encrypt the data with the signing enclave's public key and a secret key. Then, they pass the encrypted unsigned transactions to the signing enclave via the signing service [47–48] which decrypts the unsigned oracle transactions [49]. The signing enclave then performs a K out of N verification of the transaction content from N oracle nodes, ensuring the truthfulness of the transaction. The unsigned contributor approval transaction is then signed with the secret key of the signing enclave. Then, the signed contributor approval transaction is passed via the signing service to the oracle nodes [51-52] and from there to the blockchain.

Step nine, once the smart contract receives the approval transaction from the signing enclave to add a contributor [53], it creates a contributor token, which includes the reference to the smart contract, representing the data contribution and registers the user's ownership of the token, and updates the hash of the pool data.

Step ten, to claim the contributor token, the user first must optin to it [57]. This is done analogous to the optin to the `AppendDRT` in step two above. The user then issues a transaction to claim the newly created contributor token [60] that sends an instruction to the smart contract to transfer the contributor token to the user's wallet [62-63]. Lastly, because rights represented by a contributor token is public information, we can store this information in blob storage for later ease of use [64–66].

3.3 Execute Digital Rights Tokens

This flow starts with a user who has already purchased a Digital Rights Token and wishes to execute the code linked in the DRT on the issuing data pool. Step one, after logging in and establishing a secure communication channel [1.1–1.6], the frontend queries a list of the user's DRTs directly from the blockchain and allows the user to select which DRT to execute [2–6].

Step two, the frontend initializes an execute DRT group transaction [7], consisting of: i) The DRT asset transfer transaction; ii) The payment transaction of the code execution fee (a flat fee initially, later it can depend on the actual runtime of the code); and (iii) An application call transaction to execute the DRT.

In step three, the execute DRT group transaction is signed using the same transaction signing process as before ([8.1–8.6] in the Create Data Pool use case). The signed transaction is issued to the blockchain by the frontend [8], which receives the transaction ID of the group transaction in return [9].

Next, in step four, the frontend obtains the public key of the execution enclave and uses this key to encrypt the transaction ID received in step three.

Step five, the frontend sends the encrypted transaction ID to the execution service, which passes it to the execution enclave to commence verification of the transaction. Before this, a secure communication channel is established between the oracle nodes and execution enclave [12-13]. The verification process of the transaction ID is performed as explained in Step 6 of the Join Data Pool Sequence diagram [14–25].

Step six, once the execution enclave can be sure of the transaction ID of the execute DRT group transaction, it parses the oracle node and triggers the execution service to request the sealed dataset of the data pool that issued the DRT, and the wasm binary referenced in the DRT [25-26]. The execution service fetches the wasm binary code from the referenced public repository. Because we use replicable builds, data creators can easily verify what code exactly is included in the wasm binary, which in turn is what guarantees their control over their data. Once the execution service receives the sealed data and the wasm binary, it passes both to the execution enclave [29].

Step seven, the execution enclave unseals the data using the enclave's sealing key and executes the wasm binary on this data [30]. It then seals and stores the data again and encrypts the result of the computation using the secret shared between the execution enclave and the frontend.

In step eight, the execution enclave passes the sealed and encrypted result and sealed data to the execution service [31], which then sends both to the data service which in turn stores the sealed data and the sealed and encrypted result of the computation [32–33]. Lastly, the data service sends the encrypted result to the frontend [34] where it is decrypted and displayed to the user, who can also download it [35–37].

3.4 Create Digital Rights Tokens

Lastly, we turn to the creation of a Digital Rights Token, before discussing the issuance and trading of DRTs. Figure 6 shows how to create a DRT. To do this, a data creator selects a data pool from the list of pools they have created in step one [2-3]. Then, the user selects which code is referenced in the DRT from a list of available wasm binaries [4]. Next, the user enters a short description of the DRT [5].

In step two, the users creates the Create DRT transaction by following the sign transaction

process [8.1–8.6 of the Create Data Pool sequence diagram] Once the user receives the signed Create DRT transaction from the signing enclave, the frontend issues it to the blockchain [8]. This instructs the smart contract, representing the data pool, to create an Algorand Standard Asset that represents the DRT [9]. The newly created Asset ID is then returned to the frontend [10]. In step three, the user issues a instruction to the smart contract to store the DRT in in the Algorand Box Storage of the smart contract with this transaction following again the sign transaction process from before [11]. Once the user receives the signed transaction, it is sent to the blockchain by the frontend where the DRT is registered in the box storage [13-14]. Since DRTs are traded on a blockchain, there is a continuous record of ownership, this ownership is recorded in the box storage of the smart contract. This provides a convenient way to obtain information about the current owner of a DRT without having to query the blockchain.

A Appendix

A.1 Digital Rights Tokens

Below is sample code for the AppendDRT which allows other users to append their information to the data pool managed by the smart contract with address `KAHE4HQEWEUDEUTUBSVU5E2VHMBC BGEY46X7TGJ2EE7V4DCEYDUCLWCCA4`. Each DRT includes a link to a URL where the code for the DRT can be pulled from.

Figure 2: Sample code for the AppendDRT.


```
{
"index":239815614,
"params":
    {
    "creator":"KAHE4...",
    "decimals":0,
    "default-frozen":false,
    "manager":"KAHE4...WCCA4",
    "metadata-hash":"QUFBQ...",
    "name":"Append",
    "name-b64":"QXBwZW5k",
    "reserve": "KAHE4...WCCA4",
    "total":15,
    "unit-name":"DRT",
    "unit-name-b64":"RFJU",
    "url":"https://github.com/ntls-io/ntc/append",
    "url-b64":"aHR0cHM6..."
    }
}
```

A.2 Sequence Diagrams

Figure 3: Sequence diagram for the Create Data Pool user story. Each lane indicates a system component, each arrow a communication between two components.

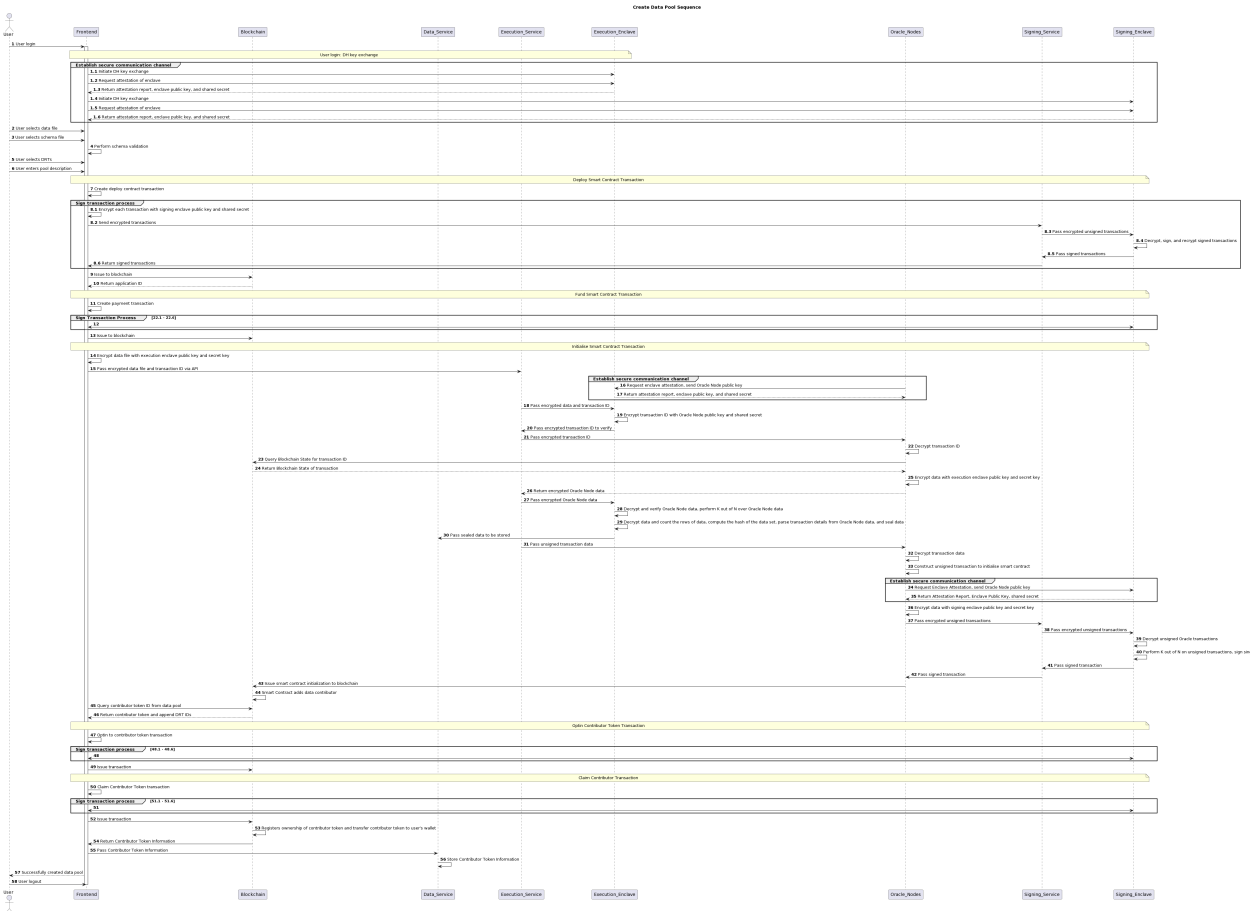


Figure 4: Sequence diagram for the Join Data Pool user story. Each lane indicates a system component, each arrow a communication between two components.

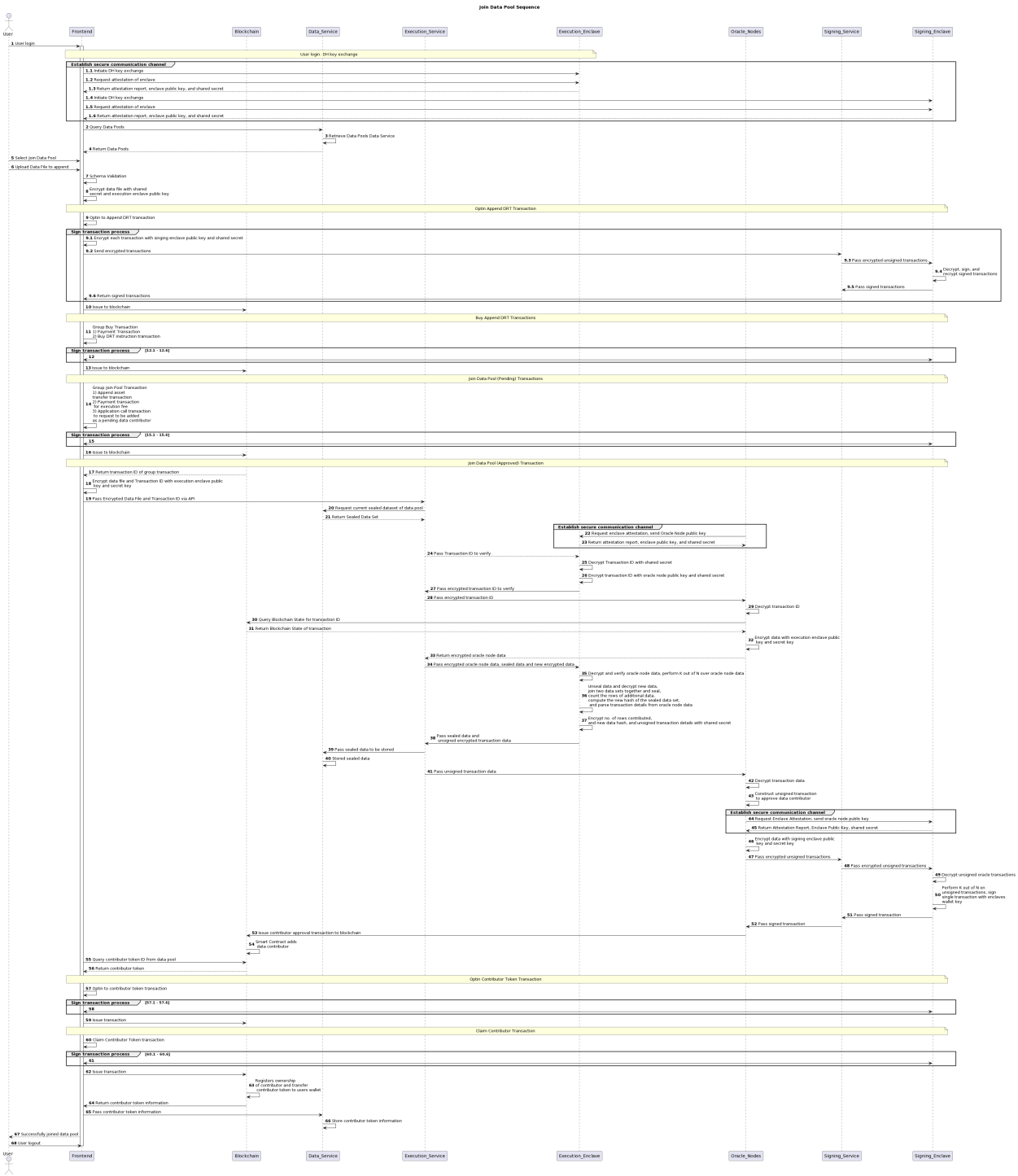


Figure 5: Sequence diagram for the Execute DRT user story. Each lane indicates a system component, each arrow a communication between two components.

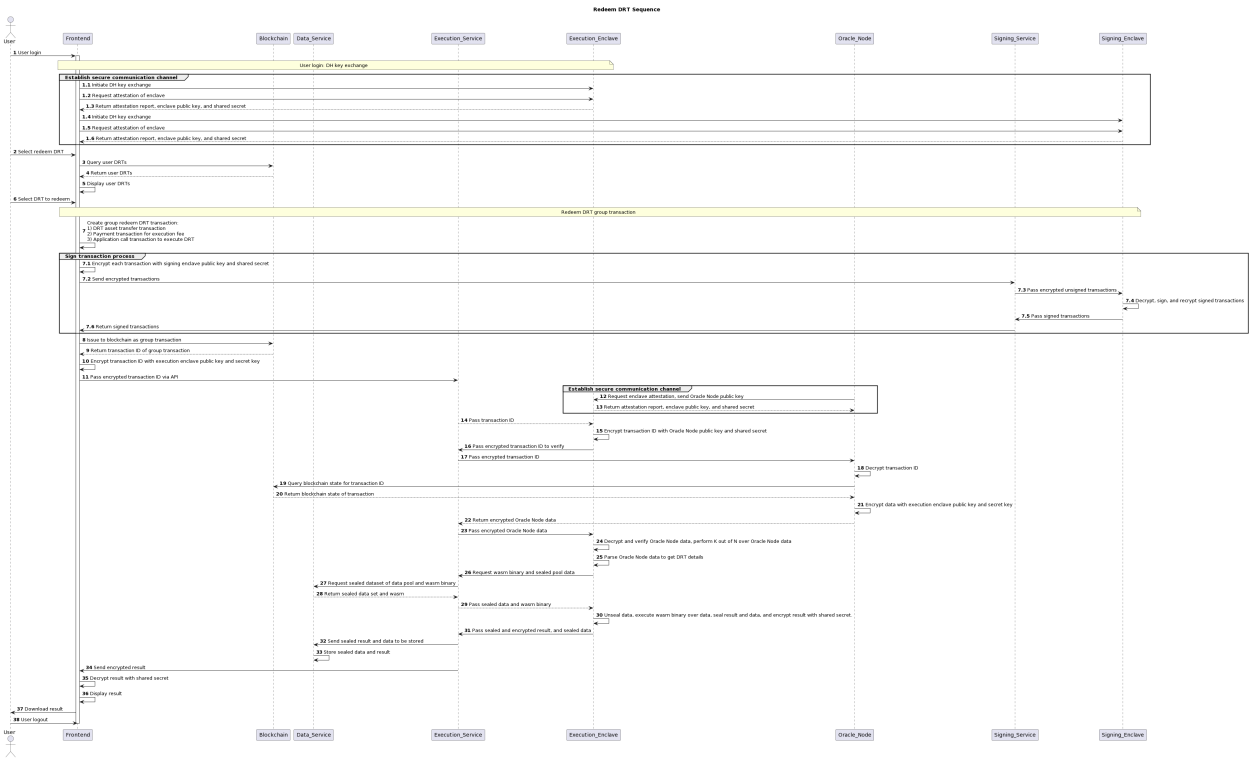
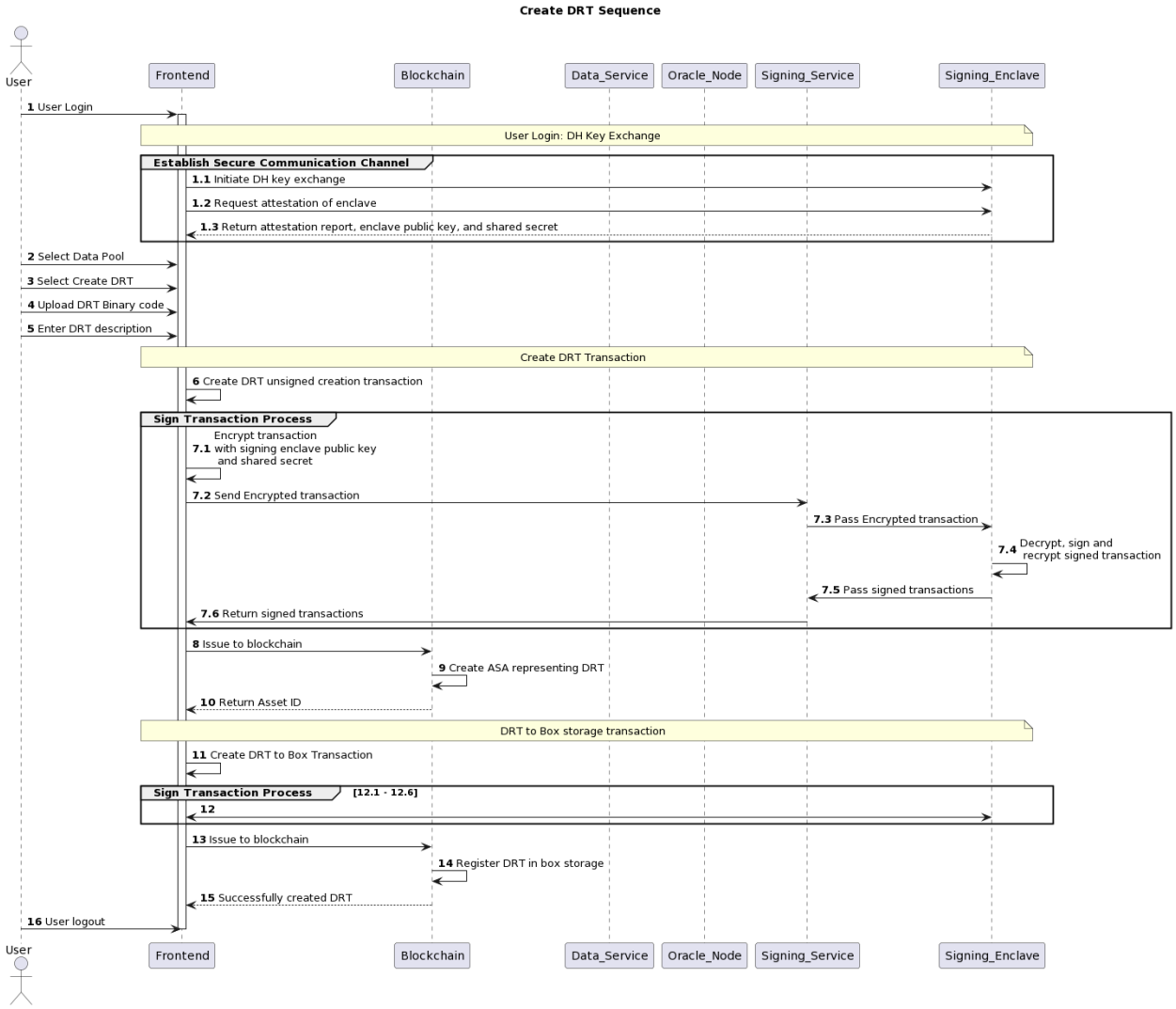


Figure 6: Sequence diagram for the Create DRT user story. Each lane indicates a system component, each arrow a communication between two components.



References

[1] Daniel J. Solove, *"Understanding Privacy"*, Harvard University Press, Cambridge, Massachusetts, (**2008**)

[2] Samuel D. Warren and Louis D. Brandeis, *"The Right to Privacy"*, 4 Harvard Law Review 193, (**1890**)

[3] Ruth Gavison, *"Privacy and the Limits of the Law"*, Yale Law Review 89(3), (**1980**)

[4] , *"The Economics of Justice"*, Harvard University Press, Cambridge, Massachusetts, (**1983**)

[5] Alan Westin, *"Privacy and Freedom"*, Atheneum, New York, (**1967**), p. 7

[6] Itay P. Fainmesser, Andrea Galeotti, and Ruslan Momot, *"Digital Privacy"*, Management Science 0(0) (**2022**)

[7] Daron Acemoglu, Ali Makhdoumi, Azarakhsh Malekian, and Asu Ozdaglar, *"Too Much Data: Prices and Inefficiencies in Data Markets"*, American Economic Journal: Microeconomics, 14(4) (**2022**)

[8] Kenneth L. Karst, *"The Files: Legal Controls over the Accuracy and Accessibility of Stored Personal Data"*, 31 Law and Contemporary Problems 342, 344, (**1966**)

[9] Harold Demsetz, *"Toward a Theory of Property Rights"*, The American Economic Review, 57(2), (**1967**)

[10] Ole Hanseth and Eric Monteiro, *"Understanding Information Infrastructure"*, University of Oslo Press, Oslo, (**2000**)

[11] Ittai Anati, Shay Gueron, Simon P Johnson, and Vincent R. Scarlata, *"Innovative technology for cpu based attestation and sealing"*, In Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP, Vol. 13, (**2013**)

[12] Victor Costan and Srinivas Devadas, *"Intel SGX Explained"*, IACR Cryptology ePrint Archive, Vol. 2016, pp. 86ff. (**2016**)

[13] Dennis M. Ritchie and Ken Thompson, *"The UNIX Time-Sharing System"*, Communications of the ACM, 17 (7), pp. 365–375, (**1974**)

[14] United Nations General Assembly, *"Universal declaration of human rights"*, 217 [III] A, Paris, (**1948**)

[15] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yaarom Yuval, and Raoul Strackx, *"Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient out-of-Order Execution"*, Proceedings of the 27th USENIX Conference on Security Symposium, pp. 991-1008, (**2018**)

[16] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin and T. H. Lai, *"SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution"* 2019 IEEE European Symposium on Security and Privacy, pp. 142-157, (**2019**)

[17] , K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss and F. Piessens, *"Plundervolt: Software-based Fault Injection Attacks against Intel SGX"*, 2020 IEEE Symposium on Security and Privacy, pp. 1466-1482, (**2020**)