HOMEBASE: AUTOMATING MULTI-PARTY WORKFLOWS VIA A ZERO-KNOWLEDGE VIRTUAL MACHINE

Ethan Gordon
Falls Technology, LLC
ethan.gordon@fallstechnology.com

ABSTRACT

Many workflows of a business, government, non-profit, or any organization, require digital interaction or interoperability with other similarly situated organizations. For instance, a medical provider must digitally interoperate with their patients as well as the patients' insurers. Data must move from each party while being subject to privacy concerns and regulation, as well as financial concerns put forth in the respective parties' contractual agreements. Moreover, there are many different steps of computation that occur in order for these workflows to properly be executed. The inordinate amount of time and money spent on verifying, reconciling, interpreting, and finally executing these workflows can be saved by automation made possible from zero-knowledge proof technology. In this paper we put forth a system, Homebase, that can accomplish such automation. While our first use-case is medical billing, Homebase's architecture has been generalized to handle all digitized multi-party workflows.

1 Introduction

This author has recently been dealing with a seemingly intractable problem of interoperability between a large bank, and multiple DMV's. The problem is that the author paid off the lien for his vehicle , officially ending his obligation to the bank/lien-holder, and subsequently requested the title of his vehicle. Ideally, all that solving this problem should involve is the bank, who owned the car and thus the title, simply sending him the title. However, no party, including the DMV's where the vehicle has been previously registered, know where the title is. In short, all these parties have separate databases that cannot speak to each other–which is unfortunate because each party must interact in order to execute the workflow involved in obtaining a title. It has been 9 months since the vehicle was paid off–the title is still in limbo. This problem acts a very clear example of multi-party workflows and their present state– highly inefficient and, admittedly, highly frustrating.

Similarly, and more importantly, the healthcare industry is especially poor at interoperability. This is no secret. The US Dept. of Health and Human Services (HHS) has charged the Office of the National Coordinator for Health Information Technology (ONC) with developing best standards for interoperability within the industry. For the ONC, in order to solve for true interoperability, an application must provide a seamless process for exchange of health information between two or more systems, so that each system may *use* the information once it is received [1]. With the developments of HL7 and FHIR, international standards of health data formatting and transmission, have helped drive interoperability, but systems and processes simply remained siloed–causing inefficient and vulnerable workflows that are frustrating for patients and all users involved.

In order for workflows as vital as those in healthcare to be automated, each party must be able to trust not only every party involved, but also each computation executed within the workflow. A Zero-Knowledge Virtual Machine provides the environment needed for trustable execution of code and generation of zero knowledge proofs (zkps). For the unitiated, a much more detailed explanation of the tech stack will follow, but for now, we will use Wikipedia's definition for zkps: "method by which one party (the prover) can prove to another party (the verifier) that a given statement is true while the prover avoids conveying any additional information apart from the fact that the statement is indeed true" [2]. For instance, one could theoretically prove that their credit score is above or within a certain range without revealing what their credit score actually is [3].

In this paper we put forth a design of a system meant to automate multi-party workflows that currently suffer from inefficiencies and insecurities. We call this system Homebase; symbolizing a particular party's data rounding all bases securely and efficiently, undergoing the requisite computations in order to come home, visible in their own system of record and user interface of choice. We first describe the architecture of homebase, diving into the details of the tech stack. We then examine related applications and research concerning healthcare workflows and secure data exchange specifically, detailing where Homebase is different and, we think, better. Lastly, we run through a medical-billing example–explaining how Homebase would be used in such an environment.

2 Design and Architecture

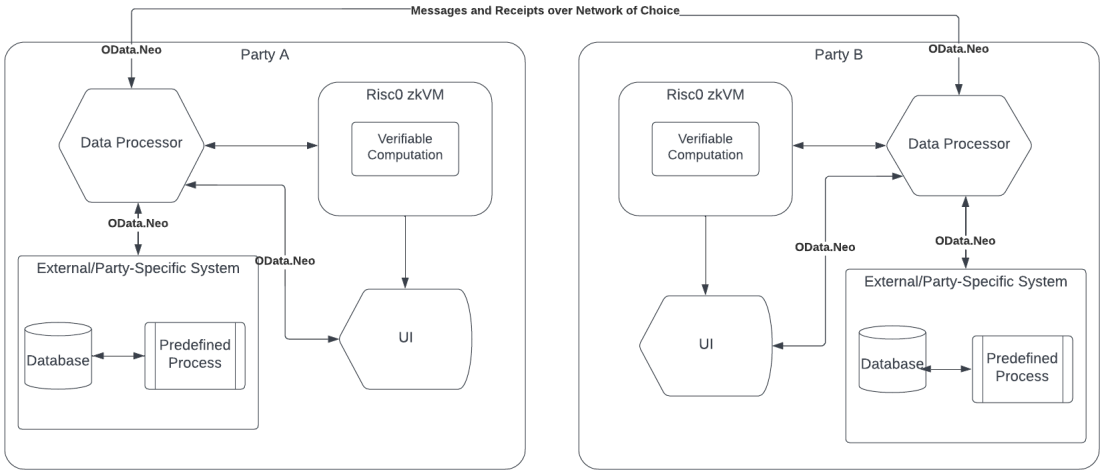

Figure 1: The general architecture of Homebase. The connection between a party's external system of record and the Homebase application via the Data Processor will be carried out via OData.Neo, a new version of OData [4], [5]. This connection will be explained in more detail below. The RISC Zero zkVM is a free-and-open-source general-purpose ZK computing platform [6]. Most of the time in this section will be spent on explaining the verifiable computation of a party's data.

2.1 Overview and Definitions

The Homebase system comprises a RISC Zero zkVM, a Data Processor, and a UI. The Data Processor will implement OData.Neo, a new, protocol-agnostic version of OData that can ideally query systems that may use non-REST based API's–such as gRPC or SOAP. The Data Processor will handle bi-directional activity that may either come from the external system or the UI. For instance, a patient may enter data into the Homebase UI–which triggers synchronization with their medical provider's database, requiring a sequence of verifiable computations processed by the zkVM. Or, an insurance provider may enter a claim within their external system which automatically becomes processed by the Data Processor, which passes the relevant data to the insurance provider's zkVM. Our approach to zk-technology primarily follows that of RISC Zero. The design of our Data Processor and implementation of OData will reference *The Standard*, a software design standard developed by Microsoft's Hassan Habib and company. Below are some definitions we hope to be helpful:

- Zero-Knowledge Virtual Machine (zkVM): "a virtual machine that runs trusted code and generates proofs" [7]
- RISC Zero zkVM: "RISC Zero's zkVM implementation based on the RISC-V architecture"
- RISC-V: "is an open standard instruction set architecture (ISA) based on established RISC principles" [8]
- RISC: a reduced instruction set computer
- Data Processor: A major component of the Homebase design that includes a broker component, services component, and an Exposer component–all subject to *The Standard*
- Brokers: "a [disposable] liaison between business logic and the outside world", i.e., a party's external system [9]

- Services: "containers of all the business logic in any given software - they are the core component of any system and the main component that makes one system different from another" [9]
- Exposers: disposable components that are responsible for exposing core business logic contained in Services, to the outside world [9]

We spend much of our time below explaining the ins-and-outs of the zkVM through an example produced by RISC Zero themselves–battleship! We also will spend time discussing what exactly the Data Processor looks like and why we design it to adhere to *The Standard*. Afterwards we will compare and contrast the Homebase system with other research and commercial applications. Finally, we walk through the medical billing example.

2.2 Understanding the RISC Zero Zero-Knowledge Virtual Machine through a game of battleship

What makes battleship the perfect example for explaining how a zkVM works is the its involvement of trust. Each player attempts to hide any information about the placement of their ships while also accurately reporting when their opponent's shot hit or missed. Similarly, in many multiparty workflows each party must trust the end result of another party's computation, whilst striving to hide any information they don't want revealed. Consider the setup of the game below (We assume that the reader is familiar with the rules of the game and how it is typically played).

Let two players, Alice and Bob, engage in a game of battleship. If the game were played online today, Alice and Bob would likely need to connect to a game server which maintains the game state. However, with RISC Zero's zkVM, a game server becomes unnecessary. Both players can run the zkVM on their machines and play against each other without connecting to a central game server. In this case, each player maintains their own private game state, "yet every step of the game is cryptographically checked to prevent cheating" [10]. Keep in mind that theoretically the game can be played across any kind of network, so we are not tied to a blockchain network for carrying out this verifiable game of battleship.

Let a Battleship instance be represented as a tuple $Battleship = (GameState, Position, Digest)$. The game is started by Alice, initializing a $GameState$, where $GameState$ is an object containing Alice's set of ships, along with the x,y coordinates of each ship and its direction, i.e, Vertical or Horizontal. $GameState$ also contains a $salt$, random data used as additional input to the zkVM proof process.

After Alice chooses the placement of her ships, and the salt is added to the $GameState$, initialization is completed with an $InitMessage$ generated by Alice by calling the zkVM. See the figure below for an illustration of this process. An explanation of the illustration and its parts will follow.

The generation of the $InitMessage$ requires the zkVM proof system. We explain the zkVM and its proof system through walking the reader through steps 3 through 9 below.

3. Before $GameState.check()$ is run inside the zkVM, it is converted into a RISC-V ELF binary. Remember that RISC-V is an Instruction Set Architecture (ISA), and thus the $GameState.check()$ code is converted into an executable file with RISC-V instructions, which is to be executed by the zkVM. Please read through Appendix C for more details on RISC-V and the Binary ELF.
4. Upon execution of the binary, an execution trace of the full computation executed inside the zkVM is constructed and then each operation of the trace is compared to the RISC-V instructions used to build the ELF binary.
5. A Method Id is created from a hash of the ELF Binary. The cryptographic seal created in step 7 will be compared to the method ID.
6. Any value that the Binary ELF is instructed to make available to the computational receipt, which is to be sent to Bob, is committed to a Journal. So, a Journal contains any values that the Battleship logic creates that are meant to be viewed and verified.
7. The cryptographic seal allows third-party authentication of the journal. The seal is the zero knowledge proof that verifies that the journal was truly created via the RISC-V instructions and not altered in any way. The seal and the journal both combine to make a computational receipt.
8. The InitMessage, which is really just the computational receipt created by the zkVM, elucidated by the steps above, is then processed by Alice's host-code–which is really like the controller within a Model-View-Controller design.
9. From Alice's host-side code, her InitMessage is sent to Bob, to be processed by his host-side code. In the end, Bob will know that Alice's board was set up correctly–i.e., no ships were placed on top one another, no ships were placed outside the board coordinates, etc.

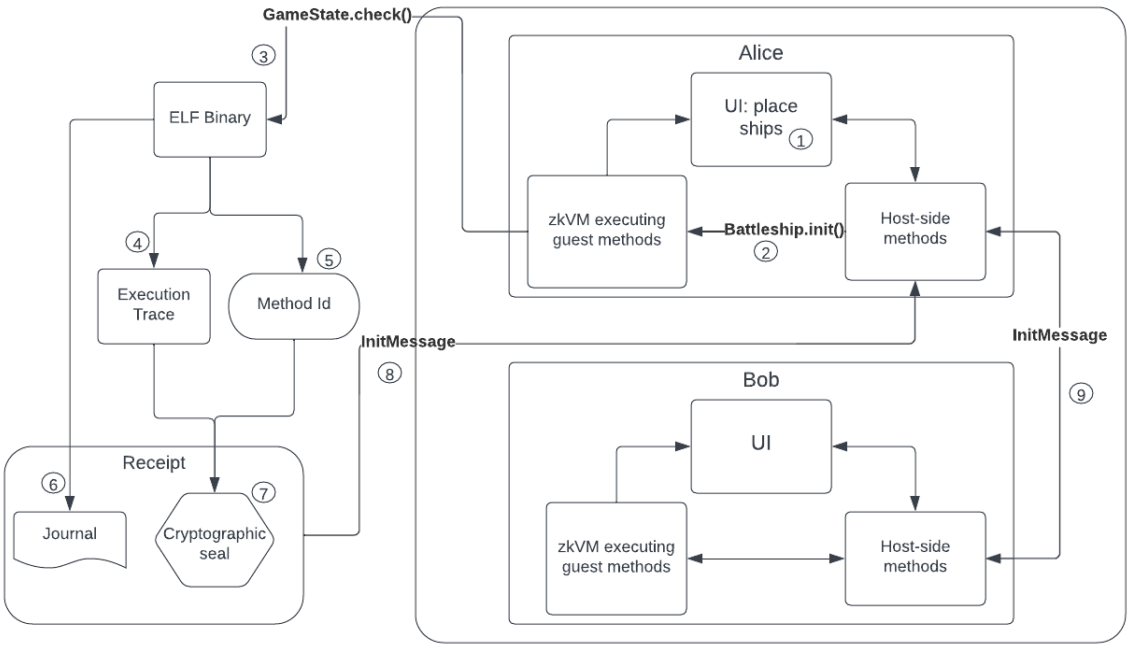

Figure 2: The initialization phase of the Battleship game. Steps 1 and 2 were explained above. Steps 3 through 9 showcase RISC Zero's zero-knowledge tech at work. They are described below. This diagram was constructed from details in [10].

As one can see, a lot is going on with the initialization of a battleship game board. The most important part of the initialization is the construction of the seal. As mentioned in step 7, the cryptographic seal involves the zero-knowledge proof system of RISCO Zero's zkVM. We discuss this step in more detail below, but first, a few things to know about zkps.

2.2.1 Structure of Zero Knowledge Proofs

Wikipedia has a wonderful article on zkps here [2], which contains many detailed examples and other helpful notes. However, the cited Wikipedia page involves *interactive* proofs of knowledge that satisfy zero knowledge constraints, while RISC Zero uses *non-interactive* proofs of knowledge. This distinction will be explained in more detail later in this paper, but we begin with the foundational definition of zero knowledge proofs found in Arora and Barak's *Computational Complexity: A Modern Approach*.[1] Please reference appendix A and B for primers on concepts contained in the following definition of zkps.

Definition 2.1 (Zero Knowledge Proofs)**.** Let L be an **NP**-language, and let M be a polynomial time Turing machine such that $x \in L \Leftrightarrow \exists_{u \in 0,1^{p(|x|)}}$ s.t. $M(x, y) = 1$ where $p()$ is a polynomial.

A pair P, V of interactive probabilistic polynomial-time algorithms is called a *zero knowledge proof* for L, if the following three conditions hold:

Completeness: For every $x \in L$ and u a certificate for this fact (i.e., $M(x, u) = 1$),

$$Pr[out_V \langle P(x, u), V(x) \rangle] \geq 2/3$$

where $langleP(x, u), V(x)\rangle$ denotes the interaction of P and V. P gets x, u as input and V gets x as input. Also, $out_V I$ denotes the output of V at the end of the interaction I.[2]

Soundness: If $x \notin L$, then for *every* strategy P^* and input u,

$$Pr[out_V \langle P^*(x, u), V(x) \rangle] \leq 1/3$$

[1] Note that this definition also assumes an interactive element, but it shouldn't cause much of an issue when we later discuss what makes a zkp non-interactive. Moreover, this definition treats P and V as probabilist polynomial-time algorithms, which is useful for us when describing STARKS.

[2] $Pr[out_V \langle P(x, u), V(x) \rangle] = out_V I$.

(The strategy P^* need not run in polynomial time).

Perfect Zero Knowledge: For every probabilistic polynomial-time interactive strategy V^*, there exists an expected probabilistic polynomial-time (stand-alone) algorithm S^* such that for every $x \in L$ and u a certificate for this fact,

$$out_{V^*}\langle P(x,u), V^*(x)\rangle \equiv S^*(x)$$

S^* is called the *simulator* for V^*, as it simulates the outcome of V^*'s interaction with the prover.

By applying the above definition to the RISC Zero zkVM, the P in battleship is an instantiation of an Object $Prover$. The $Prover$ runs the $GameState.check()$ computation with Alice's placement of ships as input. Steps 4-7 described above are then executed to ensure that the conditions of completeness, soundness, and perfect zero knowledge are satisfied by the zkVM. In the case of battleship, the Verifier, AKA Bob, processes the computational receipt by sending it to his zkVM via the zkVM API to verify that the computational receipt is the output of a valid execution of the intialization method for $GameState$.

2.2.2 The Cryptographic Seal

In step 4 the execution trace is developed, which is a table in which a row is one step in the computation, from the initialization of the $GameState$ to the termination of the $BattleShip$ instance. The columns of the execution trace contain the current state of the RISC-V architecture as well as markers for initialization and termination points. We call these data columns and control columns respectively.

The first part of the construction of the seal involves appending rule checks to the execution trace to validate that the computation was executed according to the rules set forth by the RISC-V ELF binary. For instance, let the following table represent the initialization of the game, where the data columns represent the location of the placement of ships. See table 1 below for what this looks like.

Data Column1	Data Column2	Initialization?	Transition?	Is Data Column1 and Column2 $\in$ (0,10)?
2	3	1	0	0
3	1	0	1	0
6	2	0	1	0

Table 1: The data columns in this table hold the x, and y placements of a ship respectively. The "1" in the Initialization column marks that this cycle of the execution trace is initialization, the "1" in the Transition column indicates that the $GameState$ is in a state of transition, and the last column acts as a rule check to ensure that the ships were placed according to the bounds of the game board–the rules set forth. These column values should equal zero when a rule holds.

After padding the execution trace with random noise that will not be detected by the rule checking columns, *trace polynomials* are constructed from the execution trace columns–the data columns and the control columns. The data and control columns become data and trace polynomials.[3] The RISC Zero zkVM runs an inverse number theoretic transform (iNTT) to obtain the coefficients of the trace polynomial.[4] Using Python's intt package, we can compute the coefficient of the data polynomials:

```
from sympy import intt

data_cols = [2, 3, 1]
prime_mod = 15*2**27 + 1
coefficients = intt(data_cols, prime_mod)

print("Result: ", coefficients)

Result:  [503316483, 213646055, 1509949442, 1799619864] // Output
```

Listing 1: Inverse Number Theoretic Transform with Python

So, for instance the first data column from Table 1 is transformed into the following data polynomial (using the coefficients obtained in Listing 1):

[3]Missing from this explanation are *PLONK* columns and polynomials. PLONK stands for Permutations over Lagrange-bases for Oecumenical Noninteractive Arguments of Knowledge. Yes, I know. PLONK and the respective columns and polynomials will be somewhat elucidated in Appendix B.

[4]Please see Appendix A for a primer on iNTTs and the math involved in this paper.

$$d1 = 503316483 + 213646055x + 1509949442x^2 + 1799619864x^3$$

Evaluating these trace polynomials on a particular set produces a Reed-Solomon encoded *trace block*.[5] As stated in [11], the introduction of Reed-Solomon codes enhances the soundness of of the proof system. Once the trace polynomials are constructed, they are committed into two separate Merkle Trees[6], one for the data polynomials and one for the control polynomials. The column for checking the rules of the ELF binary, the third column from table 1, is also converted into polynomials deemed *constraint polynomials*.

Using a parameter called the *constraint mixing parameter*, the constraint polynomials are combined to form a *mixed constraint polynomial* $C(x)$, where x is input. And it is here where we will re-introduce the $Prover$ and $Verifier$ of zero knowledge proofs, referring to these as P and V respectively. Remember, P and V are algorithms run within the zkVM that can receive input via the zkVM API. So, using the mixed constraint polynomial $C(x)$, P computes another polynomial, the *High Degree Validity Polynomial* $V(x)$. Lastly, P splits V into four lower degree polynomials to each be evaluated over a specific field, encodes them in a Merkle Tree, and then sends the Merkle root to V, the Verifier.

2.2.3 Notes on the *non-interactive* in non-interactive zero knowledge proofs

The zero-knowledge proof system of RISC Zero's zkVM is non-interactive, in the sense that the Prover and the Verifier do not need to execute a series of interactions for the Prover to suffieciently prove the veracity of the claim being made. The description of the proof-system put forth so far does not explicitly explain how it is non-interactive. Enter the *Fiat-Shamir Transform*.[7]

In the proof system described in section 2.2.1, an interaction between Prover and Verifier was assumed. This meant that the Verifier was allowed to send unpredictable messages to the Prover. The Fiat-Shamir Transform (FST) makes these messages unnecessary. The FST constructs a proof stream in between the Prover and Verifier, simulating an interaction that's not actually occuring. The Verifier receives the serialized proof of the Prover's claim. It may seem like magic now, but please read Appendix B, where it will still seem like magic but a little more tangible.

2.2.4 Wrapping Up the zkVM

As one can see, there are many complicated and deep operations occurring during a simple initialization of an even simpler version of a battleship game. And trust us, we haven't even scratched the surface of it all. The RISC Zero zkVM is highly powerful software that has the potential to change computation as we know it today. We are lucky to be able to use it within our tech stack so that unnecessarily frustrating and complex multiparty workflows can be streamlined using the fascinating technology of zero knowledge. Please look through Appendices B and C for a continuation of the details of constructing a seal and the mathematical tools used.

2.3 The Data Processor and OData.Neo

While the zkVM is vital to Homebase delivering verifiable computation, it would be completely useless if Homebase couldn't then share messages and receipts to sync it's connected parties' databases and systems of record. Because, again, the goal of Homebase is to ensure that Alice's system shows what Bob wants to reveal to it, and vice versa. We have determined that OData.Neo is the best technology to help carry out this goal. First, some history on the OData protocol and its evolution.

OData, or Open Data Protocol, is an OASIS Standard that purports to "simplify the querying and sharing of data across disparate applications and multiple stakeholders for re-use in the enterprise, Cloud, and mobile devices" [12]. OData's tagline is "the Best Way to Rest", as it is a protocol for creating "REST-based data services which allow resources, identified using Uniform Resource Locators (URLs) and defined in a data model, to be published and edited by Web clients using simple HTTP messages" [13].

OData is especially useful for integration of enterprise systems. A run-of-the-mill example may be a health insurance company that must integrate their Oracle DBMS, which contains most of their customer, claims, and medical data, and their customer relationship management (CRM) system, such as Salesforce (SF). OData enables the company to expose the data from their Oracle DBMS in SF, without storing it in SF. This data virtualization occurs through a

[5] For an explanation of Reed-Solomon codes and more cryptography techniques, please see Appendix B.

[6] See Appendix B

[7] Please see Alan Szepieniec's excellent explanation of the Fiat-Shamir Transform and STARKs in general here: https://aszepieniec.github.io/stark-anatomy/.

representation of the DBMS's data model, denoted as the OData Metadata Document, being consumed by SF so that the DBMS's data can be queried each time a specific record is needed in SF. So, this allows for execution of workflows in SF using data in the company's DBMS. Hopefully, it is becoming clear why we are choosing OData as the route for sharing zkVM messages and computational receipts.

Let's consider a very simple data model. See Figure 3 for reference. The goal of OData is to generate a Metadata document based on this data model, expose it as a queryable OData service, so that SF can execute a member enrollment workflow for incoming Member/Patient, for instance. OData relies on an *entity data model* (EDM) that consists of a Schema, Entity Types, Entity Properties, and Entity Keys.

See Listing 2 below for an example of what an OData client will see by calling for the metadata of the health insurance company's OData service–i.e., *https://hostname/ODataExample/V1.0/*$metadata$. Salesforce has an OData adapter, that acts as an OData client, for their platform which can read an OData service's EDM and then query the OData service using *system query options*.

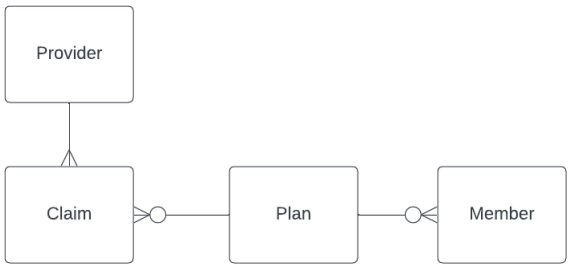

Figure 3: This simple data model involves a health insurance claims process where a prover/doctor may have many claims, one health insurance plan may covers many particular claims, as well as many members/patients.

The most advanced OData library around is Microsoft's ASP.NET OData library [14]. This library has included all of OData's latest version's system query options, such as $filter, $search, $count, along with $compute and$search. These system queries will be important for how our DataProcessor operates. But we will need more complex operations and points of integration. OData.Neo will provide us with these operations and points of integration.

```
<edmx:Edmx Version="4.0">
    <edmx:DataServices>
        <Schema Namespace="OData.InsuranceExample">
        <EntityType Name="Plan"></EntityType>
        <EntityType Name="Provider"></EntityType>
        <EntityType Name="Claim"></EntityType>
        <EntityType Name="Member"></EntityType>
        <EntityContainer Name="Container"></EntityContainer>
        </Schema>
    </edmx:DataServices>
</edmx:Edmx>
```

Listing 2: OData Entities

2.3.1 OData.Neo and *The Standard*

As shown in Figure 1, our DataProcessor must be able to integrate with any party's existing system, and communicate with another party's Homebase host. OData.Neo will provide the capability to interoperate with a party's system of record, process the system's data with the zkVM, and pass messages and receipts to the other party.

OData.Neo is protocol agnostic, meaning that it can interoperate with systems that may use RESTful, gRPC, SOAP, or GraphQL. This is important because an insurance company can interoperate with a hospital system that uses SOAP, and a physical therapy office that uses a simple RESTful API. We will briefly introduce an example that will carry us through the rest of this section and be revisited in the last section of the paper. As stated in the introduction, OData.Neo is an implementation of *The Standard*, which we will explain through the following example.

Consider a scenario where a patient, John, who is insured by Acme Insurance, visits his local hospital for an endoscopy as he has been having a hard time swallowing food. To relate it to the data model from Figure 3 and Listing

2, John is an instance of the *Member* entity and his local hospital is an instance of the *Provider* entity. So, John as a member will have a one-to-one relationship with the *Plan* entity, which has a one-to-many relationship with the *Claim* entity.

Once John's endoscopy is completed, the hospital submits a health insurance claim to Acme Insurance that contains details such as ID's, Current Procedural Terminology (CPT) codes, diagnoses, and other information.[8] OData.Neo will be sitting inside both of the Hospital and Insurance systems' instance of Homebase, receiving incoming queries and messages and exposing outgoing queries and messages via an *Exposer* component. Please refer to Figure 4 for a diagram of this process.

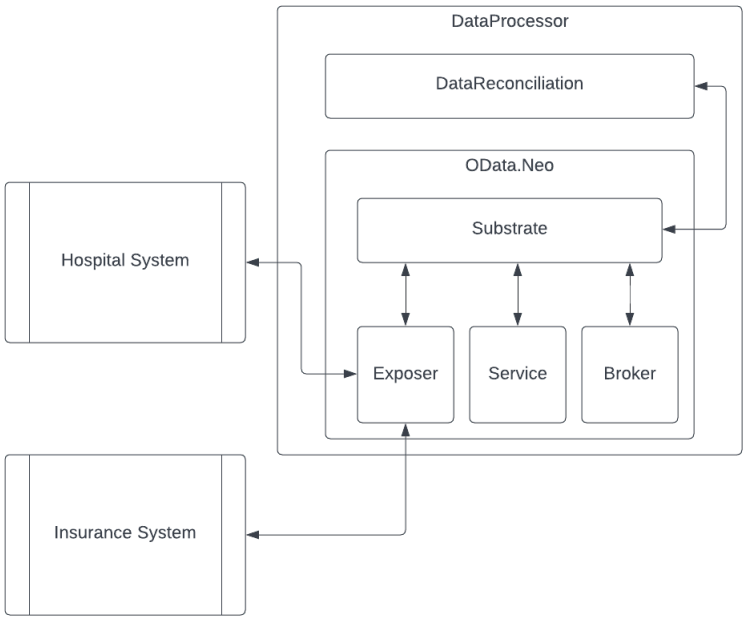

Figure 4: This diagram shows how OData.Neo is situated within the DataProcessor. The DataProcessor will be sitting in both parties' instances of Homebase, but it is pulled out in this diagram for this example. As you can see, OData.Neo contains four components; the *Exposer* component, *Service* component, *Broker* component, and *Substrate* component. These components are very important to the software architecture prescribed in *The Standard*. Incoming and outgoing queries and requests are first met by the Exposer. The *DataReconciliation* component is another important component to the DataProcessor which also adheres to the Standard.

Again, the goal of utilizing OData.Neo is streamlining integration, which is obviously a vital part of multi-party workflows. Many inefficient workflows feature "integration" that entails disorganized email chains. With OData.Neo, one party's processing can be translated into OData.Neo expressions, which can then be translated into whatever the targeted party relies upon for executing workflows. See below for a step-by-step explanation of how OData.Neo and the DataProcessor's components are utilized.

1. *Exposer*: The Exposer component in OData.Neo will ingest the hospital's claim as input and prepare the input to be passed to the Substrate component. For instance, say that the hospital relies on a SOAP Web Service, and a claim is sent in XML form like the following:

```
<Patient>
     <id value="patient-1"/>
     <name>
       <use value="official"/>
       <family value="Ashcroft"/>
       <given value="John"/>
     </name>
```

[8]The insurance claim process is fairly complex and will not be followed to every detail in this paper. We will assume that the claim will include only a handful of fields.

```
      <gender value="male"/>
      <birthDate value="1954-06-11"/>
      ...
  <encounter>
        <encounterDate value="2022-09-12",
        ...
    </encounter>
  <coding>
      <system value="http://terminology.hl7.org/CodeSystem/v3-ActCode"/>
      <code value="PPO"/>
      <display value="Acme Platinum PPO"/>
      ...
    <payor>
        <identifier>
          <system value="http://www.bindb.com/bin"/>
          <value value="123456"/>
        </identifier>
        <display value="Acme Insurance Inc."/>
      </payor>
    ...
    <coding>
      <system value="https://www.cms.gov/codes/billtype"/>
      <code value="831"/>
      <display value="Hospital Outpatient Endoscopy"/>
    </coding>
    ...
    <diagnosis>
    <sequence value="1"/>
    <diagnosisCodeableConcept>
      <coding>
        <system value="http://hl7.org/fhir/sid/icd-10-cm"/>
        <code value="M96.1"/>
        <display value="Diverticulitis"/>
      </coding>
    </diagnosisCodeableConcept>
  </diagnosis>
```

Listing 3: XML Insurance Claim

A requirement of the Standard is that an Exposer must not communicate with Brokers, components that will be described below. Moreover, Exposers cannot contain any business logic that is crucial to the core operation of OData.Neo. The Standard prescribes these rules in order to promote principles of simplicity, rewritability, autonomy of components, and more. In simple terms, the Exposer will be the component responsible for receiving input from consumers such as the hospital and insurance system, as well as sending responses confirming receipt.

2. *Substrate*: Once the Exposer component ingests the XML and correctly prepares it for future processing, it is passed to the Substrate component. The Substrate acts as an event bus that components subscribe to. So, the hospital system can send the claim and a subscription can be passed back to the system which notifies the system of changes that are made to the Query being processed within OData.Neo. So, if the hospital system controller executes a POST request with the above XML as the body in Listing 3, and the Exposer passes this request to the Substrate, a "fire-n-observe" relationship now exists between the hospital and OData.Neo.[15].
3. *Service*: The Service component, which interacts with the Substrate, is responsible for executing the core business logic of OData.Neo. This is where the primary contracts are formed for generic OData.Neo entities and OData expressions are formed. Ideally, and according to Standard requirements, the Service component will be isolated from external dependencies as much as possible. OData.Neo's Service component is complicated and a full explanation would require another five pages.
4. *Broker*: If a particular process requires an external library or particular API, the Broker component comes in to help. Assisting in the isolation of the Service component from external dependencies, the Broker responds to events in the Substrate and passes functionality into the Substrate to be used in the Service's computations. For instance, assume that the Substrate passed the hospital's insurance claim query to the Service component, which has begun mapping the query into an OData expression. However, this insurance claim requires adherence to HL7 FHIR formatting. In order to ensure compliance to HL7 FHIR, the Broker

makes a call out to *http://terminology.hl7.org/CodeSystem/v3-ActCode*, for instance, to confirm the Service-constructed expression does so.

5. *DataReconciliation*: Once the OData.Neo expression is constructed, it is passed to the DataProcessor's DataReconciliation component to interface with the rest of the Homebase application, especially the zkVM functionality. The details of this process will be explained in depth in the last section of this paper. For now, we take for granted that the DataReconciliation component triggers the execution of a process similar to the initialization of Battleship. However, instead of the computational receipt authenticating the correct placement of ships, the receipt authenticates the correct bill amount attributed to the insurance claim as set forth by the contract between provider and insurer. For instance, a digital agreement between the insurer and provider will already be in place, determining the amount John, a *Member*, receives under his *Plan* (Acme Platinum PPO), for an endoscopy operation. The OData.Neo claim query will be processed and checked against this digital agreement, and then sent back to OData.Neo through the DataReconciliation component. Finally, OData.Neo's Exposer component then determines the protocol used by the insurance system, and processes the computational receipt, along with the claim, according to the system used by the insurance company.

OData.Neo is still under construction and we are constructing OData.Neo.Java–the Java implementation of OData.Neo. OData.Neo's architecture is deeper and more complex than what is described above, but we hope that the reader has gained some understanding of how OData.Neo will be a vital component to the operation of Homebase. In the next section, we introduce some related applications and areas of research, explaining how we think Homebase augments and changes existing applications and research avenues. And in the final section, we present Homebase through a medical billing example.

3 Related Work: Similarities and Contributions

Much of the research and work surrounding multi-party workflow automation involves an intersection of blockchain and business process automation (BPA) [16], [17]. The existing applications and research around the technology seem to not rely on zero-knowledge proof technology, except for another OASIS standard called the *Baseline Protocol* [18]. However, Baseline's use of zero knowledge proof technology is not as mature as RISC Zero and does not utilize a zkVM within its architecture.

There are however some examples of utilizing zero knowledge proof technology for healthcare applications [19], [20]. We will spend some time describing the techniques used in [19] as it aligns closely to Homebase's goal. As mentioned above about RISC Zero's zkVM, it does not assume any network-specific protocol, so we are not tied to blockchain technology like the applications and research mentioned in this section. However, we will still briefly discuss important technology used in [16] and [17], like authenticated data structures (ADS) and smart contracts.

3.0.1 Healthcare Examples Utilizing Zero-Knowledge Proof Technology

Zheng et al develop a blockchain-based insurance claim system for the exact use-case Homebase will be applied to in this paper [19]. The authors define their system model as comprising patients, hospitals, insurance companies, private key generation (PKG), blockchain, and certificate authorities (CA)s. The system is comprised of three phases: a feeding authentic data (FAD) phase, a privacy-preserving transactions (PPT) phase, and an identity privacy-preserving (IPP) phase. Below is a quick breakdown of the process the authors propose.

1. An insurance company *B* publishes requirements c_i for purchasing insurance contracts by the smart contract. The smart contract will be released in the blockchain.
2. A patient *A* will obtain medical data *X* from the hospital which is designated by *B*, think of an "in-network" vs "out-of-network" designation. The hospital will generate a unique *ID* for *A*
3. The hospital then builds an arithmetic circuit $\mathbf{C}$ that is constructed according to requirements c_i, which is then used to generate a ZKP, the price of the insurance contract *M*, a hash value h_x, based on *X* and *ID*.[9] The hospital then encrypts *M* with a public key of the insurance company pk_B and a digital signature σ_{h_x} on the hash value h_x, resulting in h_M. The hospital submits the results to a smart contract, and the transaction will be recorded on-chain.
4. The encrypted value h_M will be decrypted as *M* with *B*'s private key sk_B, and the smart contract executes an algorithm outputting whether the transaction is valid or not.

[9]The zero knowledge proof technology used in this paper is a zk-SNARK, or a *Zero Knowledge Succint Non-Interactive Argument of Knowledge.*

5. If the transaction is valid, the smart contract notifies *A* to transfer *M* to *B*. (This step involves the verification of whether the services provided by the hospital fall within the scope of the contract. If it does, then *M* represents a co-pay, for instance.)
6. The last step involves the authentication of *A*'s identity so that *B* can ensure that they provide the rest of the insurance coverage (i.e., the bill minus *A*'s co-pay) M_{claim}.
7. Lastly, *B* transfers M_{claim} to the hospital–ending the claim process.

While the process used by the authors in [19] is interesting and should be successful, we find that it will not be a scalable. First and foremost, the process includes storing a smart contract containing business logic on a blockchain network. This practice is inadvisable as storing requirements for a business contract on-chain is insecure. Moreover, the zkp technology utilizes zk-SNARKS, which are much slower than zk-STARKS, the method used by RISC Zero's zkVM. The paper also does not mention the use of a zero-knowledge virtual machine, which negates the need for a more complicated tech stack involving multiple layers.

However, we do see many similarities between the process outlined above and Homebase. There is a need for the requirements of a business interaction to be executable in code, and used to verify the computation made by another party. However, Homebase allows for storage of such requirements on the host's system and and a virtualization of the resulting computation and the computational receipt over a network of the parties' choice.

The MediBloc project claims to provide a "patient-centered health data ecosystem" via its Panacea blockchain [20]. Panacea was built on the Cosmos SDK and Tendermint.[10] Sitting on top of Panacea's Tendermint implementation is a Decentralized Identifier (DID) component and a Data Exchange Coordination component. See the image below for reference.

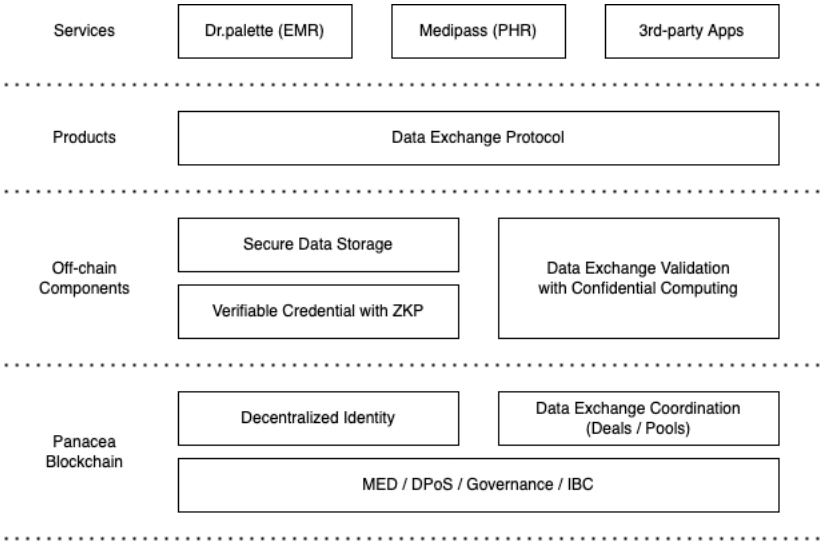

Figure 5: The MediBloc Tech Stack. Taken from there website here [20].

MediBloc's decentralized applications (dApps), such as Medipass, "allows individual patients to conveniently utlize medical information stored and used by different hospitals and institutuions for treatment and vaccination history management, insurance claims, etc." [20]. MediBloc tackles the problem of sharing medical data by classifying and converting healthcare data into verifiable credentials (VCs). VC technology is based on the World Wide Web Consortium's (W3C) published standard [23]. It is interesting to consider medical data as VCs because VCs are more often equated to a driver's license or a diploma–data that verifies one's identity, training, or education. For instance, a common example presented is usually that of a University Degree. W3C's VC trust model involves (i) an Issuer, the University, (ii) a Holder, the degree holder, and (iii) a Verifier, the hiring manager, for instance. MediBloc's VC component uses ZKP's to mask certain data fields that the Holder of the medical data does not want revealed, while still verifying data integrity. The specific technology used for this functionality are BBS+ signatures.

[10]The Cosmos SDK is a framework for building blockchains [21]. Tendermint Core is a "Byzantine Fault Tolerant (BFT) middleware that takes a state transition machine - written in any programming language - and securely replicates it on many machines" [22].

Because medical data cannot be transacted with on-chain, MediBloc created an "off-chain decentralized oracle powered by confidential computing" [20].[11] Utilizing Intel's Software Guard Extensions, MediBloc set up a system where data can only be decrypted by oracle nodes that are running in a secure enclave. If the oracle node successfully verifies the data, it is then re-encrypted for the data consumer.

Lastly, MediBloc must present an option for where all the medical data will be stored. In the documents, the IPFS is considered but eventually passed on because of it essentially being a public network. It seems that MediBloc has not yet figured out the data storage problem, but hopefully HBH can.

Similar to our feelings about [19]'s implementation, feel that Medibloc has an interesting approach yet lacks scalability. It suffers from having too many external dependencies, such as reliance upon an off-chain decentralized oracle node. Its use is also tied to a blockchain network, which will be hard for many firms to switch to in the near future. Also, transforming medical data into verifiable credentials is quite interesting and novel, but does not appear to be useful for the near future; most already existing players in the healthcare industry have either heavily structured Electronic Health Record (EHR) data, which cannot be easily transformed into VC's, or heavily unstructured data, which also cannot be easily transformed into VC's. Most troubling, Medibloc does not currently have a place to store all of its medical data, which must be subject to very strict security requirements.

Our approach to solving multi-party workflows in healthcare strongly varies from Medibloc's approach. For instance, one weakness of Homebase is a missing Identity and Access Management (IAM) component to handle identity authentication. Medibloc has an interesting decentralized approach to handling digital identities. However, Medibloc does not appear to have any verifiable computation functionality. This makes the execution of multi-party workflows less synchronized and more complex. Moreover, we believe Homebase has a stronger approach to zkp technology than Medibloc's use of BBS+ signatures.

3.0.2 Examples at the intersection of BPA and Blockchain

Simm et al in [17] put forth a "Verifiable business process" (VBP), which they define as "a transaction-based server-centric solution for adding trust to a registry or a service in order to achieve verifiable multi-party business process automation." A common goal between this VBP and Homebase is made clear by the definition of VBP, but the authors have a very different approach to this idea of a verifiable multi-party workflow. Instead of a zero-knowledge virtual machine, and a proof mechanism based in zero knowledge proof technology in general, and STARKs in particular, the VBP consists of authenticated data structures (ADS), verifiable state machines (VSM), and verifiable log of registry changes (VLRC) [17].

Authenticated data structures "are a model of computation where untrusted responders answer queries on a data structure on behalf of a trusted source and provide a proof of the validity of the answer to the user" [25]. For instance, the VBP uses an authenticated map of key-value pairs as their ADS. This authenticated map can be queried by untrusted servers and its values can be proven to its users. A VSM is then used to track each business process which is stored in the ADS. For instance, if a multiparty business process exists between a trucking company and a manufacturer, such as executing a contract, each subprocess will be tracked by the VSM. One subprocess may include verification that 90% of the manufactured goods arrived intact at the destination. The digitalized process of inputting the state of the trucked goods can be stored within the VSM. The changes to the VSM are logged in the VLRC so that the VSM can be audited by stakeholders.

Simm et al's approach in [17] is interesting yet the applicability of the approach is unclear. Their paper does not provide enough details for how the VBP would be implemented and how the relevant data will be transmitted between parties. We feel that our choice of a zero knowledge virtual machine is superior to ADS and VSMs because it is well defined and has reference implementations to draw inspiration from. Moreover, the problem of storage is more easily solved by Homebase–there is no change to each party's current storage strategy. However, with a VBP, each party must know the current state of the VSM, which requires some consensus mechanism and blockchain network, which inreases complexity and makes implementation more difficult.

In [16], Flavio Corradini and authors architect a system that executes multi-party business processes on multiple blockchains. The authors approach multi-party business processes with a model-driven angle–each process is first modeled by "choreography's." A choreography might be a representation of a business process between a retailer and a manufacturer, such as a payment process. The choreography would typically include the participants and the goals of each participants within the process. Next, a translator is used to convert this choreography into a smart contract.

A smart contract is really just a program that executes on a blockchain. The idea in [16] is that a relationship between two parties and a workflow they both share can be modeled and automatically translated into a program that

[11] Oracle nodes are network nodes that connect blockchains to external systems [24]

executes based on the immediate fulfillment of conditions necessary for the completion of their workflow. So, if a manufacturer agrees to only ship to the retailer once a certain payment is received, this can be represented in a smart contract that automatically initiates a shipment as soon as the program/smart contract receives notification of payment.

The issue with putting all one's eggs in the smart contract, on-chain basket is the immutability, traceability, and transparency of storing any business logic on-chain. The way that organizations and businesses transact evolves, and storing these smart contracts on an immutable distributed ledger like blockchain, makes altering a multiparty business process more challenging. With Homebase, we don't need to worry about utilizing a blockchain network or figuring out distributed storage mechanisms–two parties can simply use a pre-existing agreement that is represented in a RISC-V executable file.

For the reasons stated above, we feel that Homebase presents some similar approaches to multi-party workflows as other current research applications. However, for the most part, Homebase is a wholly unique endeavor to automate the inter-organizational workflows that trouble so much of us today. The research that has been conducted and currently undergoing provide a helpful path for understanding the difficulties involved with developing applications using still-nascent technology. We are confident that the technology stack put forth by Homebase avoids the scalability pitfulls that seem to plague similar applications. In the next section, we walk the reader through Homebase's solution to medical billing, hopefully showcasing Homebase's potential usability.

4 Medical Billing: A Homebase Example

At last we have arrived at the medical billing walk-through, Homebase's first use-case and implementation. We revisit the case of John Ashcroft and his endoscopy. The basic idea of this walk-through is to use the battleship example described in the beginning of this paper as a guide to understanding how a medical billing workflow will be implemented. Instead of two parties, Alice and Bob, we introduce three parties, John Ashcroft as Patient and Member, Desert Springs General Hospital, and Acme Insurance. If needed, refer to Figure 1 for a refresher on the overall architecture–simply imagine a third party added to the diagram. Similar to the $GameState$ in battleship there must be a shared state of agreement between each party. For instance, let $InsurerMemberAgreement$ hold a set of relations between an insurer and member similar to the $GameState$ holding a set of ships and each ship's position. Both John and Acme Insurance will store this agreement in their respective systems–John could simply store it in his phone or laptop. Desert Springs General Hospital and Acme Insurance will similarly share a $InsurerProviderAgreement$. See Listing 4 below for reference.

```
Stored in John and Acme's systems ->

        "InsurerMemberAgreement": {
                "MemberId": "12AFW234",
                "MemberName": "John Ashcroft",
                "MemberDOB": "1954-06-11",
                "Plan": {
                    "Name": "Platinum PPO",
                    "Premium": "$0",
                    "co-pay": "$20"
                }
                "DatesOfCoverage": {
                    "StartDate": "2022-01-01",
                    "EndDate": "2022-12-31"
                }
                ...
                ...
        }


Stored in the Provider and Acme's systems ->

        "InsurerProviderAgreement": {
            "ProviderId": "2341ADC",
            "ProviderName": "Desert Springs General Hospital",
            "Plans": [
                {
                "Platinum PPO": [
                    {
                    "Codes": [
```

```
                        ...
                        ...
                        {"code": 831,
                        "display": "Hospital Outpatient Endoscopy",
                        "coverage": 100}
                    ]
                    }
                ]
                }
            ]
        }
```

Listing 4: Agreements

As stated in the $InsurerProviderAgreement$, under $Plan$ "Acme Platinum PPO", John should receive 100% coverage of the endoscopy. Each party will have all their necessary agreements stored in their respective systems–Homebase's DataProcessor is responsible for converting these agreements into digestable formats that the RISC Zero zkVM can handle.[12] For simplicity, we assume that the respective parties' agreements' states have been initialized and checked for validity. The most important piece of this example, is the execution of a medical billing process, which uses the state of the agreements to check for a complete and valid health insurance claim and billing process. As each shot that Alice and Bob made in battleship was verified using the zkVM, each step of the billing process will undergo the same. In the enumerated steps below, we walk the reader through how Homebase automates this process and makes health insurance claims more accurate at the same time.

4.0.1 Provider Creates Claim

An advantage of Homebase is that it doesn't require an organization to change anything about their current system and process.[13] The healthcare industry has struggled to modernize because there has been industry-wide hope for infrastructure change–which requires many mountains to move. We hope that Homebase provides the middleware that helps healthcare modernize without requiring such an insurmountable effort.

1. John Ashcroft completed his endoscopy at Desert Springs General–it turns out that he has Diverticulitis. Desert Springs is a mid-sized hospital that utilizes a mid-tier but well known Electronic Health Record (EHR) system that has a built-in insurance claim module. Using this system, an employee in Desert Spring's billing office codes John's encounter with his doctor and submits his health insurance claim for processing.
2. A SOAP Web Service interfaces with Desert Spring's EHR system, which exposes John's health insurance claim as the XML in Listing 3. Homebase's DataProcessor will be responsible for polling the web service to learn when a claim is submitted for processing.
3. Once the DataProcessor learns of the new claim, OData.Neo's Exposer component will be leveraged to query the SOAP Web Service and pull John's XML formatted insurance claim, so that it may be passed into the Substrate component. As a reminder, the DataProcessor architecture is placed below so that the reader may have a better understanding of the components as we walk through this example.
4. The SOAP Web Service is notified of all operations that occur on this health insurance claim while it is processed by OData.Neo. Once the Service and Broker component convert the healthcare claim into an OData.Neo JSON format, it is passed to the DataReconciliation component.
5. The DataReconciliation component is the last step for this initial, health insurance-claim-creation phase. Similar to the OData.Neo components, the DataReconciliation component will be modular and more complex than this description may paint it. But, it is primarily responsible for reconciling the state of the $InsurerProviderAgreement$, and identifying the *critical values* of the health insurance claim. The task of verifying these values will then be RISC Zero's responsibility. See the pseudo-code in Listing 5 below for an example implementation.

```
    class ClaimVerification
        method pullAgreementState() is
            agreementState = Odata.Neo.get(agreementState)
            hospital = agreementState.getHospital()
```

[12] It is likely that the zkVM will be able to handle a wide range of formats. For instance, see the following github repositories showcasing RISC Zero's handling of json and png files: https://github.com/risc0/risc0-rust-examples/tree/tzerrell/add-json/json, https://github.com/brianretford/risc0-playground/blob/main/methods/guest/src/bin/proof_of_png.rs

[13] We are, however, assuming that the current system has an API for interacting with it.

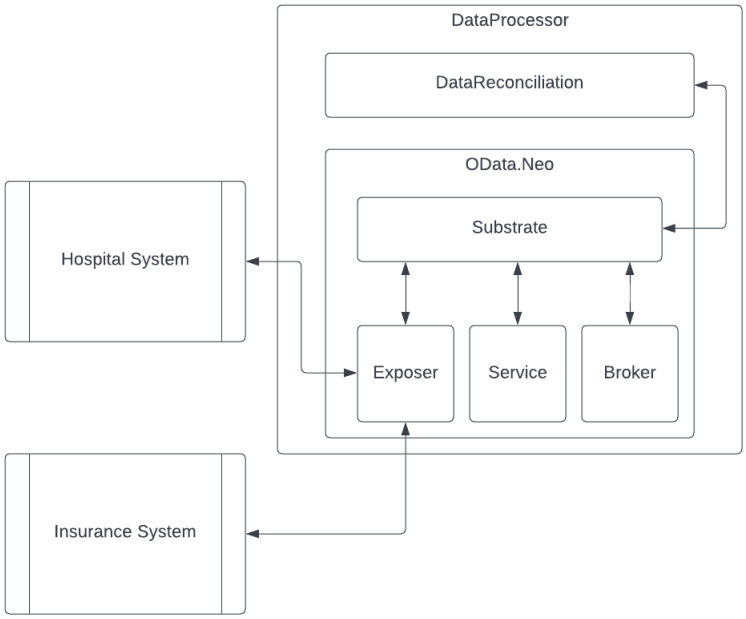

Figure 6: DataProcessor and its components.

```
            plan = agreementState.getPlan()
            codes = agreementState.getCodes()

        method pullClaim() is
            claimState = DataReconciliation.getClaim()
            claimCode = claimState.getCode()

            for (code in codes)
                if code = claimCode
                    grabCoverage()
                    ...
                    ...
                    ...

```

Listing 5: DataReconciliation processing the claim

6. Which values of the healthcare claim are considered *critical values* will vary. In the listing above, the claim's coverage of the code–which, in John's case, denotes the endoscopy procedure–is considered the critical value. Luckily for John, it is %100 coverage of the procedure. Once that figure is obtained, it is shared with the zkVM sitting in the Provider's Homebase instance.

4.0.2 The zkVM Processes and Verifiably Computes Critical Values of Insurance Claim

1. Remember that battleship required the verification that the placement of ships was within a certain range, (10, 0), (0,10) for instance. This verification was computed, and the first primary step for verification is the construction of an execution trace. See Table 1 for reference. While, our insurance claim is more complicated, the same process applies. For simplicity, let's assume that we merely have to prove the following:

$$c.X_{PlanA} == c.Y_{ClaimB}$$

Where c is coverage and X and Y denote varying values. For John, we are hoping that $X = Y = 100$. The actual computation will be more complicated, since we will have to provide logic for verifying that John is a member and his policy is what the claim provides, etc. But these conditions could similarly be represented as logical statements that require verification themeselves. Nevertheless, we resume with the task of building an execution trace, determining the equation above.

2. As we discussed in section 2.2.2, *The Cryptographic Seal*, the constructed execution trace holds Data Columns which hold the state of the RISC-V processor, and all other properties necessary to a complete verifiable computation. While oversimplified, the following table shows what an execution trace for verifying John's healthcare claim may look like:

Clock Cycle	Data Column1	Data Column2	Does entry i from Data Column1 == entry j from Data Column2?
0	"0"	"75"	1
1	"0"	"75"	1
2	"100"	"100"	0

Table 2: A simplified example of an iteration through code coverages, where Data Column1 and Data Column2 are iterating through procedure codes–Data Column2 refers to the agreement coverage, and Data Column1 refers to the claim coverage.

3. Once the execution trace is constructed, it is combined with its Method Id to form the cryptographic seal. We omit the juicy details of the entire construction and direct the reader to previous sections and Appendix B. Also developed alongside the seal, is the Journal which will include whatever values the Hospital and John want to reveal about the procedure. For instance, it simply could reiterate the critical values included in the healthcare claim–using Homebase as a tool for automation more than a tool for privacy.

4.0.3 Sending the Verified Claim Via OData.Neo

1. Within the Hospital's instance of Homebase, the constructed Journal and Cryptographic seal will be passed to the DataProcessor and re-enter a DataReconciliation process. The DataReconciliation component will reconfigure the message and computational receipt into a digestible format so OData.Neo's Service component can convert the data into a transferable format.
2. The OData.Neo formatted computational receipt and message is then passed into the Substrate component, which triggers a notice to be sent to the Insurer's Homebase instance–ending up in the Insurer's external system. This step is meant to learn what interface the Insurer system uses–a RESTful API, for instance.
3. The Insurer then is able to view the message, computational receipt, and journal values in their desired format using their already existing system.[14] Typically, using today's technology, a claims processor would be responsible for looking through the claim, reconciling differences themselves, while having multiple other responsibilities. If a healthcare provider opts to use existing software for adjudicating healthcare claims, they may experience even more time dealing with inaccuracies and inability to detect fraudulent claims.[15]

4.0.4 Patient User Interface

As patients, it would be nice to have a transparent understanding of how our medical data is processed, our medical bills are constructed, and how all of it interacts with our health insurance coverage. Many patients, and I'm sure the reader as well, has experienced opening the mail and discovering that they owe their doctor $254.56, struggling to remember when they actually received the medical services being charged for. Homebase intends to change this by exposing the medical billing process to a patient's user interface, notifying them when a data transaction is executed involving the patient's medical and billing data.[16]

1. The subscription feature of OData.Neo's Substrate component allows for constant updates to the patient when computation involving their data are executed. For instance, when the claim containing the patient's data is queried by Homebase, a notification with the data can be exposed in the patient's UI–which could merely be a RESTful API as the back-end with a React UI as the front-end.
2. Most importantly, the patient could have an understanding of when their claim is submitted, and how much their insurance company is being billed. Moreover, instead of worrying about a surprise bill coming in the mail 6 months in the future, the patient can be notified when the insurer has received proof of the claim's correctness.

[14] The reader may rightfully think that exposing new data types like computational receipts and journals in a proprietary healthcare IT system to be a tricky endeavor–but this is the promise of OData.Neo–protocol/system agnosticism.

[15] See https://www.ahealthcarez.com/health-insurance-claims-adjudication

[16] In the future, we hope to introduce new features such as one-time medical forms where a patient has their medical history stored in their Homebase instance locally, and can port this information to their medical provider's systems.

3. Ideally, a patient will have an understanding of what exactly is being done with their data at every step of the workflow. An organized a simple user interface that keeps patients in the loop, allowing for a timely execution of their healthcare services, will keep patients happy and confident that they are not being taken advantage of.
4. As mentioned before, in the future we would like to provide read/write capabilities for the patient so that a patient can fill out their medical information once and only once. Because OData.Neo is protocol agnostic, a patient's Homebase implementation and existing medical information should be able to be ported to a medical providers system once it has its own Homebase implementation.

5 Conclusion

The technology that Homebase is building upon is very new and exciting. Everything Homebase is involved in is under active development. Our hope is to bring a technology to businesses and individuals that makes working with one-another easier and reduces friction. Our hope is to optimize healthcare services, as well as make inter-governmental processes less cumbersome. In short, we want to slice through areas of intractable bureaucratic gridlock and provide the world with more opportunities to innovate and solve harder problems rather than stay stuck working on repetitive workflows.

6 Bibliography

References

[1] The Office of the National Coordinator for Health Information Technology. The path to interoperability. 2013.

[2] Zero-knowledge proof. https://en.wikipedia.org/wiki/Zero-knowledge_proof. Accessed: 11-5-2022.

[3] zkdocs: Zero-knowledge information sharing. https://a16zcrypto.com/zkdocs-zero-knowledge-information-sharing/. Accessed: 11-5-2022.

[4] The future of odata nxt (neo). https://devblogs.microsoft.com/odata/the-future-of-odata-odata-nxt/. Accessed: 11-6-2022.

[5] Odata neo. https://github.com/OData/OData.Neo. Accessed: 11-6-2022.

[6] Introducing risc zero. https://www.risczero.com/blog/announce. Accessed: 11-6-2022.

[7] Key terminology. https://www.risczero.com/docs/terminology. Accessed: 11-6-2022.

[8] Risc-v. https://en.wikipedia.org/wiki/RISC-V. Accessed: 11-6-2022.

[9] The standard. https://github.com/hassanhabib/The-Standard. Accessed: 11-6-2022.

[10] Battleship on risc zero. https://www.risczero.com/docs/examples/battleship_rust. Accessed: 11-6-2022.

[11] Constructing a seal. https://www.risczero.com/docs/explainers/proof-system/constructing-a-seal. Accessed: 11-14-2022.

[12] Oasis open data protocol (odata) tc. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata. Accessed: 11-20-2022.

[13] Ralf Handl Michael Pizzo and Martin Zurmuehl. Odata version 4.01. part 1: Protocol. Standard, OASIS, April 2020.

[14] Odata in asp.net web api. https://learn.microsoft.com/en-us/aspnet/web-api/overview/odata-support-in-aspnet-web-api/. Accessed: 11-25-2022.

[15] The future of odata nxt (neo). https://devblogs.microsoft.com/odata/the-future-of-odata-odata-nxt/. Accessed: 11-26-2022.

[16] Flavio Corradini et al. Model-driven engineering for multi-party business processes on multiple blockchains. *Blockchain: Research and Applications*, page 100018, 2021.

[17] Jamie Steiner Joosep Simm and Ahto Truu. Verifiable multi-party business process automation. *Lecture Notes in Business Information Processing*, 397, 2020.

[18] Baseline protocol. https://docs.baseline-protocol.org/. Accessed: 11-27-2022.

[19] Houyu Zheng et al. A novel insurance claim blockchain scheme based on zero-knowledge proof technology. *Computer Communications*, pages 207–216, 2022.

[20] Medibloc. https://medibloc.gitbook.io/panacea-core/. Accessed: 10-9-2022.

[21] Cosmos sdk. https://docs.cosmos.network/main/intro/overview. Accessed: 10-19-2022.

[22] Tendermint core. https://github.com/tendermint/tendermint. Accessed: 10-19-2022.

[23] Verifiable credentials data model v1.1. https://www.w3.org/TR/vc-data-model/. Accessed: 10-19-2022.

[24] What is a blockchain oracle. https://chain.link/education/blockchain-oracles. Accessed: 10-19-2022.

[25] Robert Tamassia. Authenticated data structures. https://cs.brown.edu/research/pubs/pdfs/2003/Tamassia-2003-ADS.pdf. Accessed: 11-27-2022.

[26] Peng Zhang et al. Fhirchain: Applying blockchain to securely and scalably share clinical data. *Computational and Structural Biotechnology Journal*, 16(3):267–278, 2018.

[27] Ariel C. Ekblaw. Medrec: Blockchain for medical data access, permission management and trend analysis. 2017.

[28] Healthcare fraud. https://www.fbi.gov/how-we-can-help-you/safety-resources/scams-and-safety/common-scams-and-crimes/health-care-fraud. Accessed: 9-29-2022.

[29] Dinh C. Nguyen et al. Blockchain for secure ehrs sharing of mobile cloud based e-health systems. *IEEE Access*, 7:66792–66806, 2019.

[30] Xueping Liang et al. Integrating blockchain for data sharing and collaboration in mobile healthcare applications. *IEEE Xplore*, 2017.

[31] Sharma et al. Blockchain-based interoperable healthcare using zero-knowledge proofs and proxy re-encryption.

[32] Abdellatif et al. Medge-chain: Leveraging edge computing and blockchain for efficient medical data exchange. *IEEE Internet of Things Journal*, 2021.

[33] Xia et al. Medshare: Trust-less medical data sharing among cloud service providers via blockchain. *IEEE Access*, 2017.

[34] Guo et al. Secure attribute-based signature scheme with multiple authorities for blockchain in electronic health records systems. *IEEE Access*, 2018.

[35] Centers for Medicare and Medicaid Services. No surprises: Understand your rights against surprise medical bills. https://www.cms.gov/newsroom/fact-sheets/no-surprises-understand-your-rights-against-surprise-medical-bills. Accessed: 10-15-2022.

[36] Turing machine. https://en.wikipedia.org/wiki/Turing_machine. Accessed: 11-17-2022.

Appendices

A Mathematical Primer

A.0.1 Number Theoretic Transform

B Zero-knowledge Proof Techniques

B.0.1 Permutations over Lagrange-bases for Oecumenical Noninteractive Arguments of Knowledge, or PLONK

C RISC-V