HYPERLEDGER

Blockchain Integration Framework Whitepaper

Version 0.1 (Early Draft)

Photo by Pontus Wellgraf on Unsplash

Contributors

Contributors/Reviewers	Email
Hart Montgomery	hmontgomery@us.fujitsu.com
Hugo Borne-Pons	hugo.borne-pons@accenture.com
Jonathan Hamilton	jonathan.m.hamilton@accenture.com
Mic Bowman	mic.bowman@intel.com
Peter Somogyvari	peter.somogyvari@accenture.com
Shingo Fujimoto	shingo_fujimoto@fujitsu.com
Takeuchi Takuma	takeuchi.takuma@fujitsu.com
Tracy Kuhrt	tracy.a.kuhrt@accenture.com

Document Revisions

Date of Revision	Description of Changes Made
February 2020	Initial draft

1. Abstract

Blockchain technologies are growing in usage, but fragmentation is a big problem that may hinder reaching critical levels of adoption in the future.

We propose a protocol and it’s implementation to connect as many of them as possible in an attempt to solve the fragmentation problem by creating a heterogeneous system architecture [1].

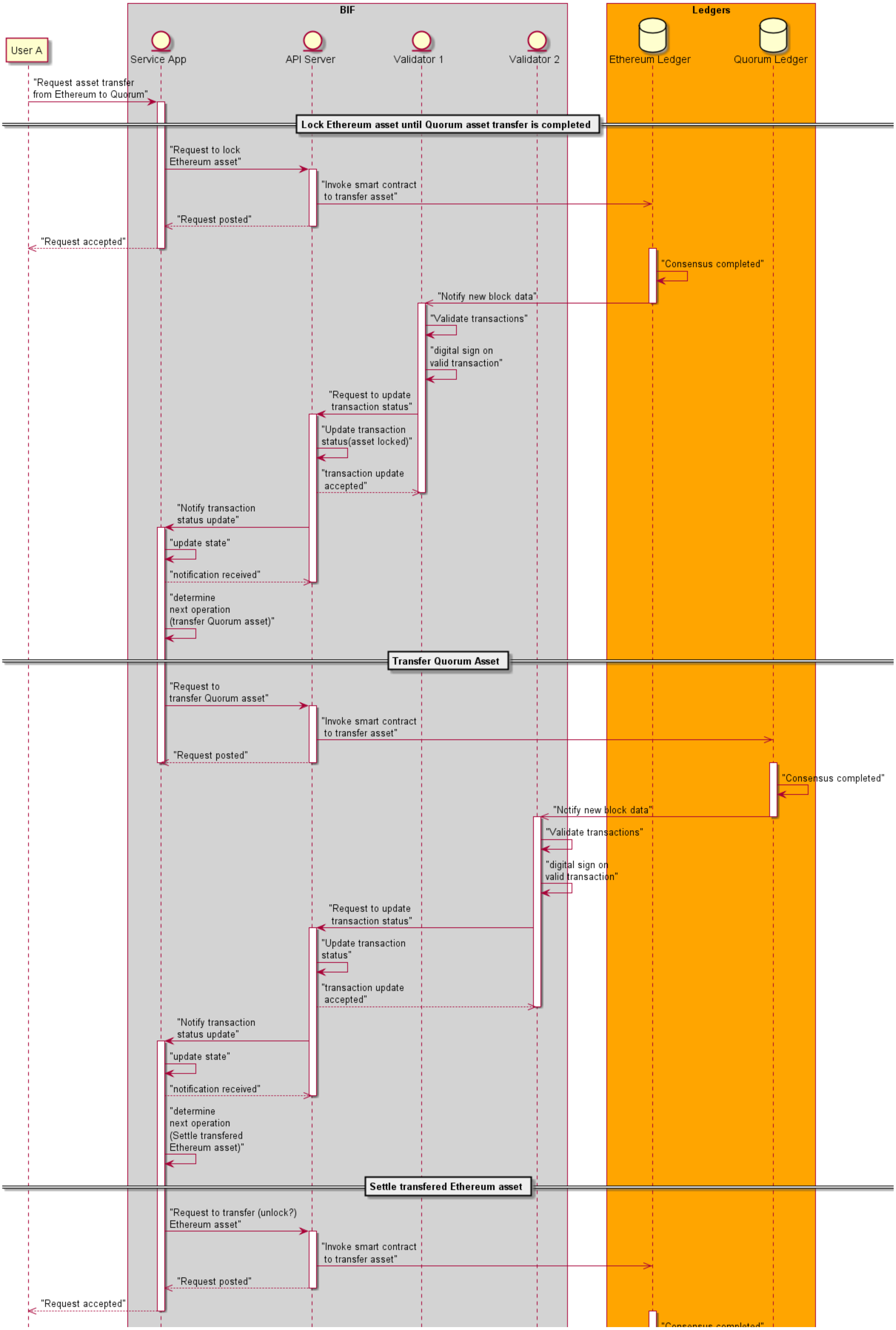
2. Example Use Cases

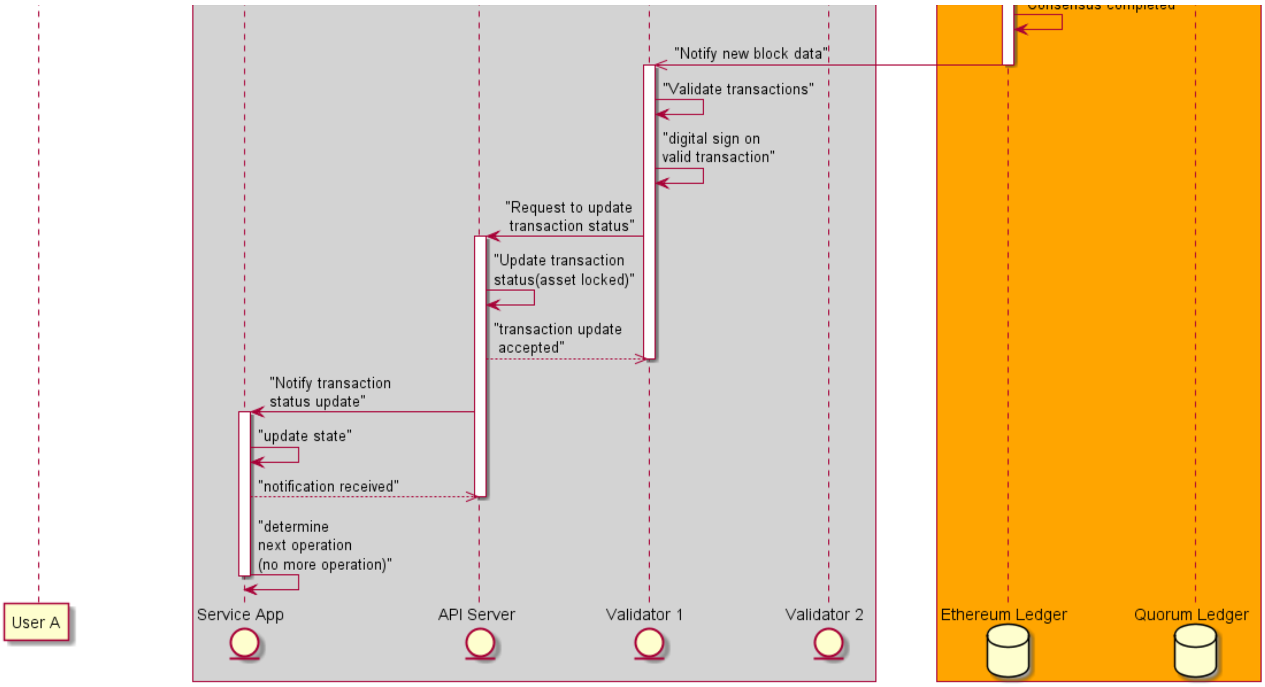
Specific use cases that we intend to support. The core idea is to support as many use-cases as possible by enabling interoperability between a large variety of ledgers specific to certain mainstream or exotic use cases.

2.1 Ethereum to Quorum Asset Transfer

Use Case Attribute Name	Use Case Attribute Value
Use Case Title	Ethereum to Quorum Asset Transfer
Use Case	1. User A owns an asset on a Ethereum ledger 2. User A transfers some asset on Ethereum ledger to a Quorum ledger
Interworking patterns	Value transfer
Type of Social Interaction	Generic Asset Transfer
Narrative	A person (User A) has multiple accounts on different ledgers (Ethereum, Quorum) and they wish to transfer some asset from Ethereum ledger to a Quorum ledger. The asset they are transferring is a generic asset meaning that it doesn’t have to be currency of any sort, but we assumed that User A agreed to release ownership of transferring asset on Ethereum ledger instead of getting ownership of transfered asset on Quorum ledger.
Actors	1. User A: The person or entity who has ownership of the transferred asset.
Goals of Actors	Some asset on Ethereum ledger had transfered to Quorum ledger.
Success Scenario	Transfer succeeds without issues. Asset is available on both Ethereum and Quorum ledgers.
Success Criteria	Presence of asset transfer across ledgers has cryptographic proof that is obtainable through BIF.
Failure Criteria	Asset transfer on both ledger is canceled.
Prerequisites	1. Ledgers are provisioned 2. User A identity established on both ledgers. 3. User A has access to BIF deployment
Comments	

Ethereum to Quorum Asset Transfer


BIF
Ledgers
User A
Service App
API Server
Validator 1
Validator 2
Ethereum Ledger
Quorum Ledger
"Request asset transfer from Ethereum to Quorum"
Lock Ethereum asset until Quorum asset transfer is completed
"Request to lock Ethereum asset"
"Invoke smart contract to transfer asset"
"Request posted"
"Request accepted"
"Consensus completed"
"Notify new block data"
"Validate transactions"
"digital sign on valid transaction"
"Request to update transaction status"
"Update transaction status(asset locked)"
"transaction update accepted"
"Notify transaction status update"
"update state"
"notification received"
"determine next operation (transfer Quorum asset)"
Transfer Quorum Asset
"Request to transfer Quorum asset"
"Invoke smart contract to transfer asset"
"Request posted"
"Consensus completed"
"Notify new block data"
"Validate transactions"
"digital sign on valid transaction"
"Request to update transaction status"
"Update transaction status"
"transaction update accepted"
"Notify transaction status update"
"update state"
"notification received"
"determine next operation (Settle transfered Ethereum asset)"
Settle transfered Ethereum asset
"Request to transfer (unlock?) Ethereum asset"
"Invoke smart contract to transfer asset"
"Request posted"
"Request accepted"
"Consensus completed"

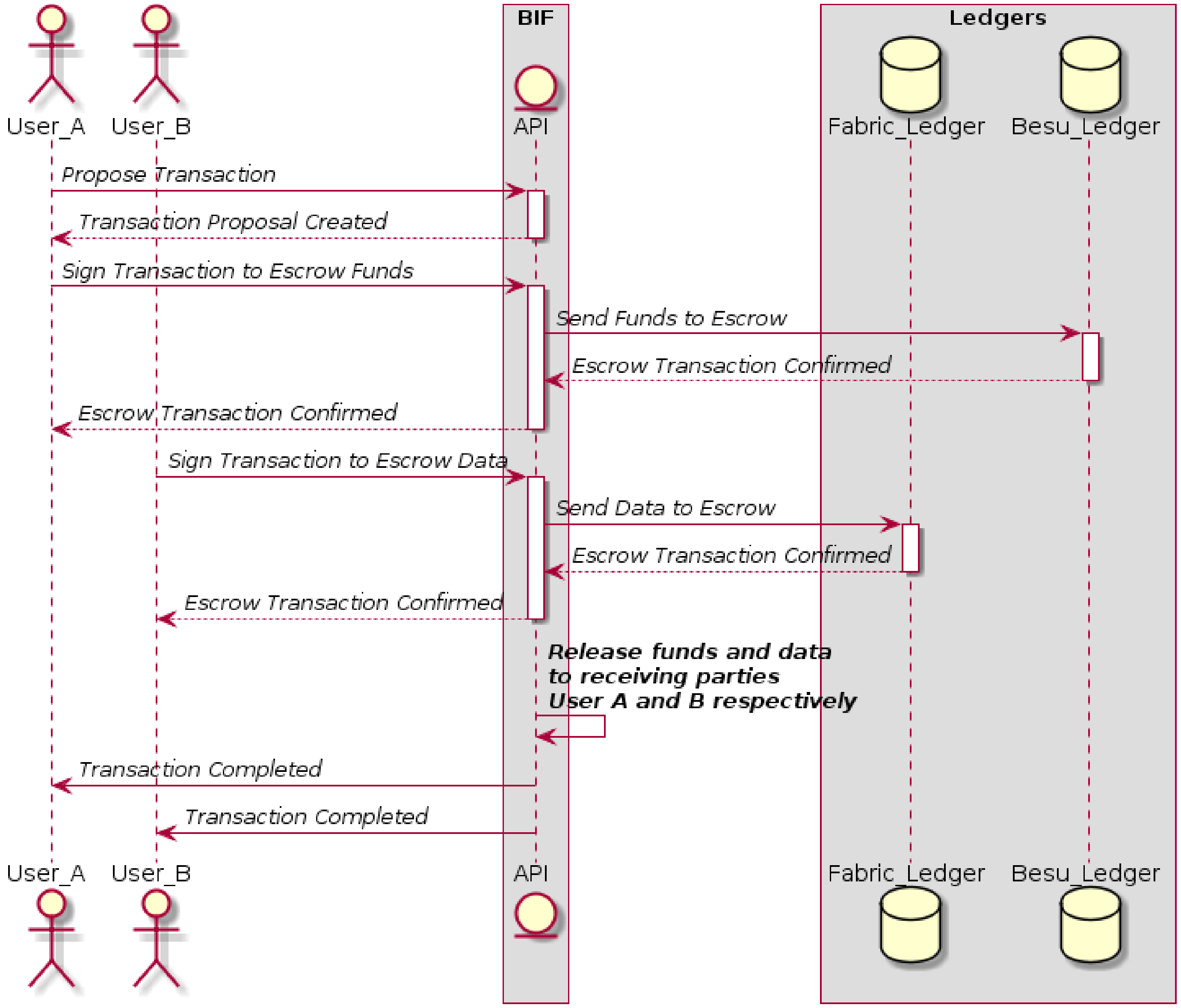

"Consensus completed"
"Notify new block data"
"Validate transactions"
"digital sign on
valid transaction"
"Request to update
transaction status"
"Update transaction
status(asset locked)"
"transaction update
accepted"
"Notify transaction
status update"
"update state"
"notification received"
"determine
next operation
(no more operation)"
User A
Service App
API Server
Validator 1
Validator 2
Ethereum Ledger
Quorum Ledger

2.2 Escrowed Sale of Data for Coins

W3C Use Case Attribute Name	W3C Use Case Attribute Value
Use Case Title	Escrowed Sale of Data for Coins
Use Case	1. User A initiates (proposes) an escrowed transaction with User B 2. User A places funds, User B places the data to a digital escrow service. 3. They both observe each other’s input to the escrow service and decide to proceed. 4. Escrow service releases the funds and the data to the parties in the exchange.
Type of Social Interaction	Peer to Peer Exchange
Narrative	Data in this context is any series of bits stored on a computer: * Machine learning model * ad-tech database * digital/digitized art * proprietary source code or binaries of software * etc. User A and B trade the data and the funds through a BIF transaction in an atomic swap with escrow securing both parties from fraud or unintended failures. Through the transaction protocol’s handshake mechanism, A and B can agree (in advance) upon * The delivery addresses (which ledger, which wallet) * the provider of escrow that they both trust * the price and currency Establishing trust (e.g. Is that art original or is that machine learning model has the advertised accuracy) can be facilitated through the participating DLTs if they support it. Note that User A has no way of knowing the quality of the dataset, they entirely rely on User B’s description of its quality (there are solutions to this problem, but it’s not within the scope of our use case to discuss these).
Actors	1. User A: A person or business organization with the intent to purchase data. 2. User B: A person or business entity with data to sell.
Goals of Actors	User A wants to have access to data for an arbitrary reason such as having a business process that can enhanced by it. User B: Is looking to generate income/profits from data they have obtained/created/etc.
Success Scenario	Both parties have signaled to proceed with escrow and the swap happened as specified in advance.
Success Criteria	User A has access to the data, User B has been provided with the funds.
Failure Criteria	Either party did not hold up their end of the exchange/trace.
Prerequisites	User A has the funds to make the purchase User B has the data that User A wishes to purchase. User A and B can agree on a suitable currency to denominate the deal in and there is also consensus on the provider of escrow.

W3C Use Case Attribute Name	W3C Use Case Attribute Value
Comments	Hyperledger Private Data: https://hyperledger-fabric.readthedocs.io/en/release-1.4/private_data_tutorial.html Besu Privacy Groups: https://besu.hyperledger.org/en/stable/Concepts/Privacy/Privacy-Groups/

Hyperledger Blockchain Integration Framework
Sequence Diagram - Escrowed Sale of Data for Coins



2.3 Money Exchanges

Enabling the trading of fiat and virtual currencies in any permutation of possible pairs.

> On the technical level, this use case is the same as the one above and therefore the specific details were omitted.

2.4 Stable Coin Pegged to Other Currency

W3C Use Case Attribute Name	W3C Use Case Attribute Value
Use Case Title	Stable Coin Pegged to Other Currency
Use Case	1. User A creates their own ledger 2. User A deploys BIF in an environment set up by them. 3. User A implements necessary plugins for BIF to interface with their ledger for transactions, token minting and burning.
Type of Social Interaction	Software Implementation Project
Narrative	Someone launches a highly scalable ledger with their own coin called ExampleCoin that can consistently sustain throughput levels of a million transactions per second reliably, but they struggle with adoption because nobody wants to buy into their coin fearing that it will lose its value. They choose to put in place a two-way peg with Bitcoin which guarantees to holders of their coin that it can always be redeemed for a fixed number of Bitcoins/USDs.
Actors	User A: Owner and/or operator of a ledger and currency that they wish to stabilize (peg) to other currencies
Goals of Actors	1. Achieve credibility for their currency by backing funds. 2. Implement necessary software with minimal boilerplate code (most of which should be provided by BIF)
Success Scenario	User A stood up a BIF deployment with their self-authored plugins and it is possible for end user application development to start by leveraging the BIF REST APIs which now expose the functionalities provided by the plugin authored by User A
Success Criteria	Success scenario was achieved without significant extra development effort apart from creating the BIF plugins.
Failure Criteria	Implementation complexity was high enough that it would've been easier to write something from scratch without the framework
Prerequisites	* Operational ledger and currency *Technical knowledge for plugin implementation (software engineering)
Comments	

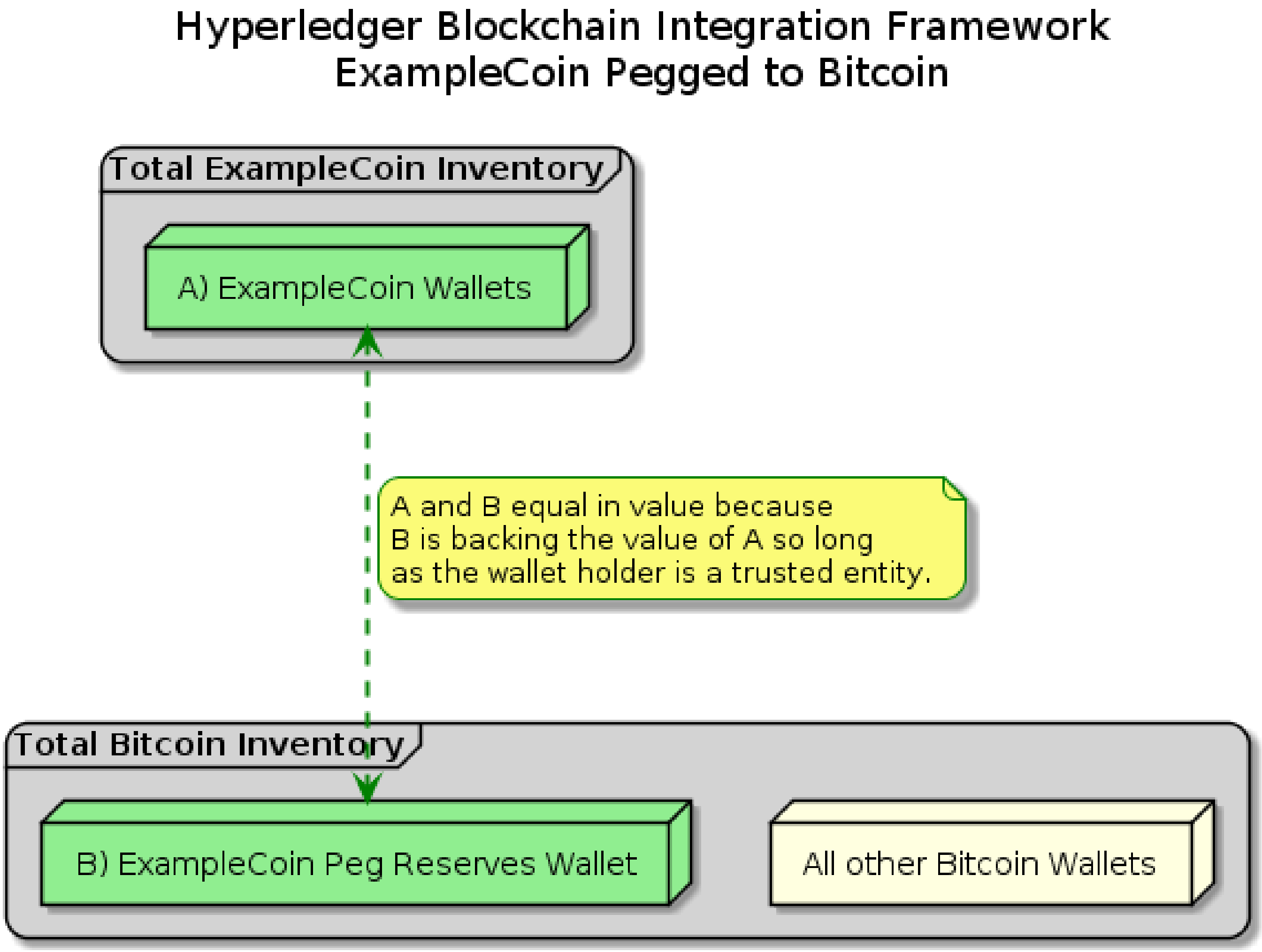
> Sequence diagram omitted as use case does not pertain to end users of BIF itself.

2.4.1 With Permissionless Ledgers (BTC)

A BTC holder can exchange their BTC for ExampleCoins by sending their BTC to ExampleCoin Reserve Wallet and the equivalent amount of coins get minted for them onto their ExampleCoin wallet on the other network.

An ExampleCoin holder can redeem their funds to BTC by receiving a Proof of Burn on the ExampleCoin ledger and getting sent the matching amount of BTC from the ExampleCoin Reserve

Wallet to their BTC wallet.


Hyperledger Blockchain Integration Framework
ExampleCoin Pegged to Bitcoin
Total ExampleCoin Inventory
A) ExampleCoin Wallets
A and B equal in value because
B is backing the value of A so long
as the wallet holder is a trusted entity.
Total Bitcoin Inventory
B) ExampleCoin Peg Reserves Wallet
All other Bitcoin Wallets

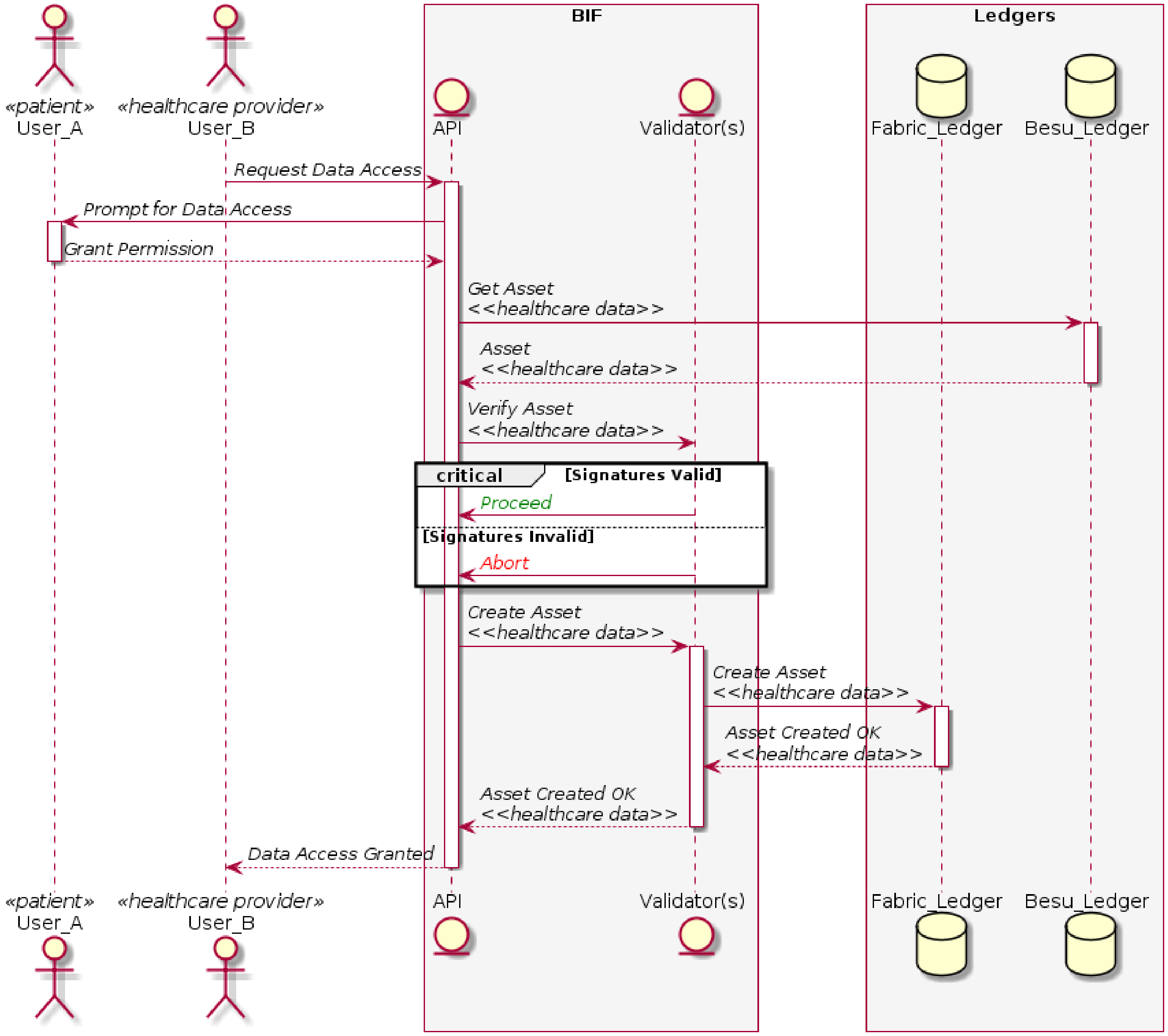
2.4.2 With Fiat Money (USD)

Very similar idea as with pegging against BTC, but the BTC wallet used for reserves gets replaced by a traditional bank account holding USD.

2.5 Healthcare Data Sharing with Access Control Lists

W3C Use Case Attribute Name	W3C Use Case Attribute Value
Use Case Title	Healthcare Data Sharing with Access Control Lists
Use Case	1. User A (patient) engages in business with User B (healthcare provider) 2. User B requests permission to have read access to digitally stored medical history of User A and write access to log new entries in said medical history. 3.User A receives a prompt to grant access and allows it. 4. User B is granted permission through ledger specific access control/privacy features to the data of User A.
Type of Social Interaction	Peer to Peer Data Sharing
Narrative	Let's say that two healthcare providers have both implemented their own blockchain based patient data management systems and are looking to integrate with each other to provide patients with a seamless experience when being directed from one to another for certain treatments. The user is in control over their data on both platforms separately and with a BIF backed integration they could also define fine grained access control lists consenting to the two healthcare providers to access each other's data that they collected about the patient.
Actors	* User A: Patient engaging in business with a healthcare provider * User B: Healthcare provider offering services to User A. Some of said services depend on having access to prior medical history of User A.
Goals of Actors	* User A: Wants to have fine grained access control in place when it comes to sharing their data to ensure that it does not end up in the hands of hackers or on a grey data market place. User B
Success Scenario	User B (healthcare provider) has access to exactly as much information as they need to and nothing more.
Success Criteria	There's cryptographic proof for the integrity of the data. Data hasn't been compromised during the sharing process, e.g. other actors did not gain unauthorized access to the data by accident or through malicious actions.
Failure Criteria	User B (healthcare provider) either does not have access to the required data or they have access to data that they are not supposed to.
Prerequisites	User A and User B are registered on a ledger or two separate ledgers that support the concept of individual data ownership, access controls and sharing.
Comments	It makes most sense for best privacy if User A and User B are both present with an identity on the same permissioned, privacy-enabled ledger rather than on two separate ones. This gives User A an additional layer of security since they can know that their data is still only stored on one ledger instead of two (albeit both being privacy-enabled)

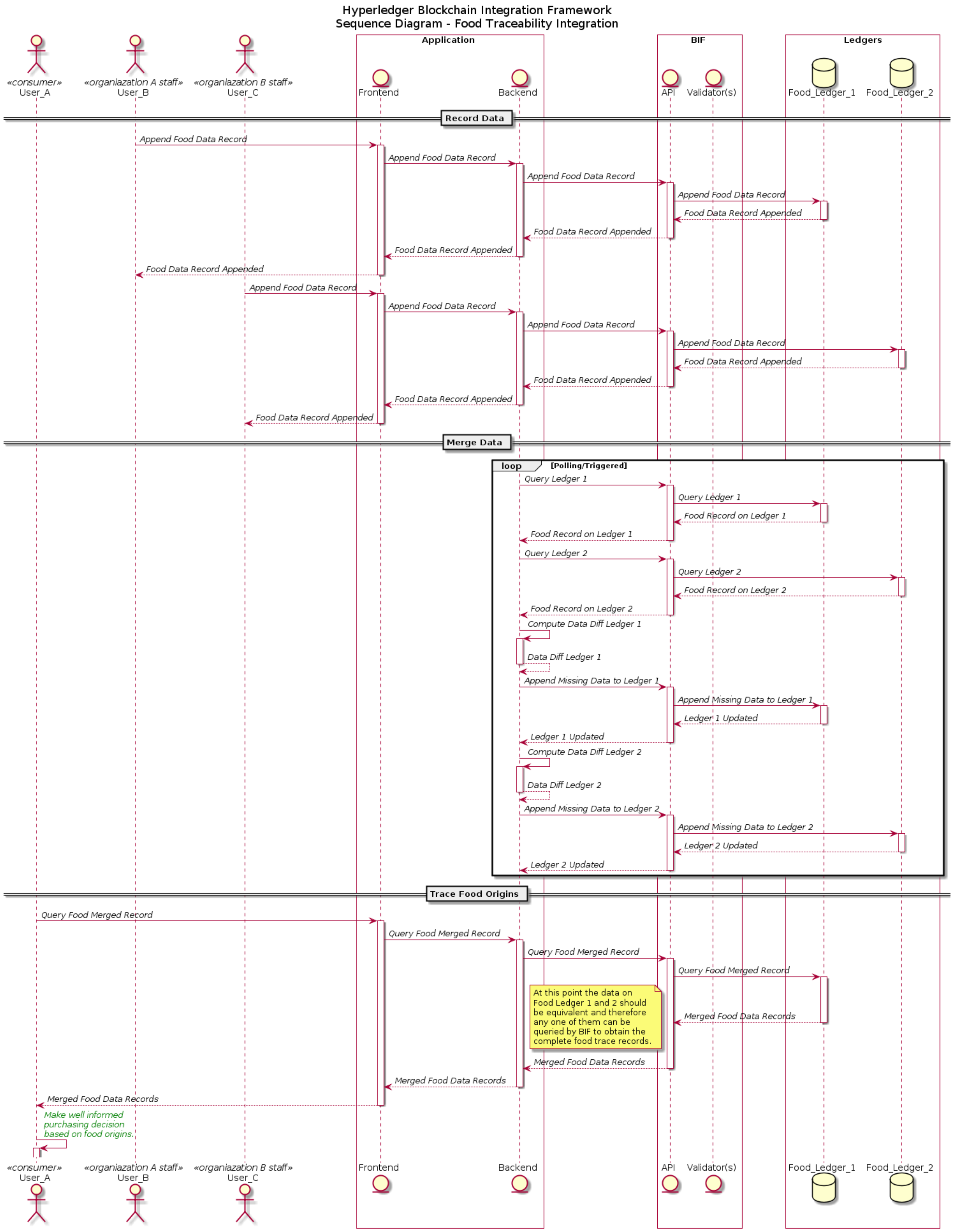
Hyperledger Blockchain Integration Framework
Sequence Diagram - Healthcare Data Sharing with Access Control Lists



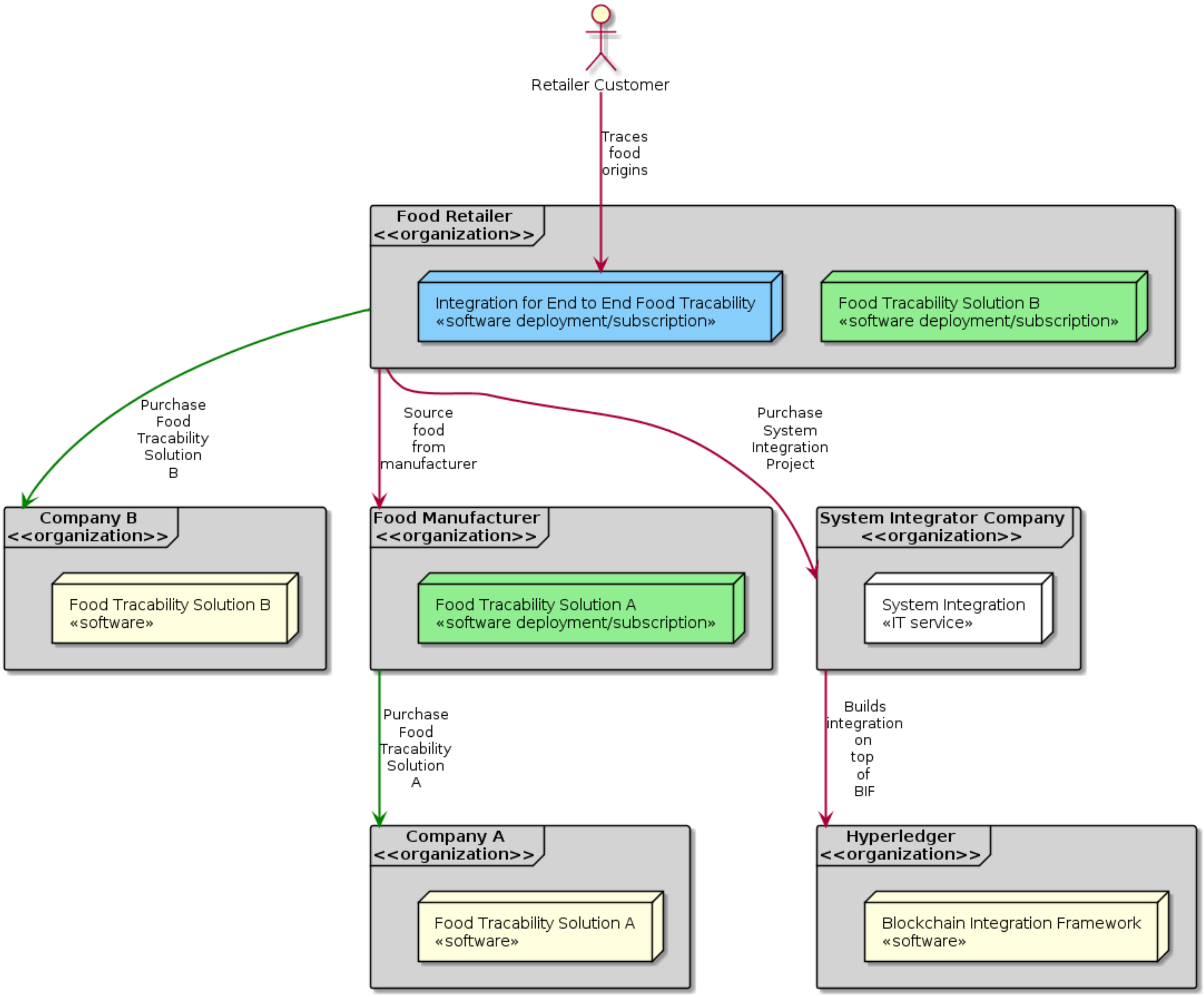
2.6 Integrate Existing Food Traceability Solutions

W3C Use Case Attribute Name	W3C Use Case Attribute Value
Use Case Title	Food Traceability Integration
Use Case	1. Consumer is evaluating a food item in a physical retail store. 2. Consumer queries the designated end user application designed to provide food traces. 3. Consumer makes purchasing decision based on food trace.
Type of Social Interaction	Software Implementation Project
Narrative	Both Organization A and Organization B have separate products/services for solving the problem of verifying the source of food products sold by retailers. A retailer has purchased the food traceability solution from Organization A while a food manufacturer (whom the retailer is a customer of) has purchased their food traceability solution from Organization B. The retailer wants to provide end to end food traceability to their customers, but this is not possible since the chain of traceability breaks down at the manufacturer who uses a different service or solution. BIF is used as an architectural component to build an integration for the retailer which ensures that consumers have access to food tracing data regardless of the originating system for it being the product/service of Organization A or Organization B.
Actors	Organization A, Organization B entities whose business has to do with food somewhere along the global chain from growing/manufacturing to the consumer retail shelves. Consumer: Private citizen who makes food purchases in a consumer retail goods store and wishes to trace the food end to end before purchasing decisions are finalized.
Goals of Actors	Organization A, Organization B: Provide Consumer with a way to trace food items back to the source. Consumer: Consume food that’s been ethically sourced, treated and transported.
Success Scenario	Consumer satisfaction increases on account of the ability to verify food origins.
Success Criteria	Consumer is able to verify food items’ origins before making a purchasing decision.
Failure Criteria	Consumer is unable to verify food items’ origins partially or completely.

W3C Use Case Attribute Name	W3C Use Case Attribute Value
Prerequisites	1. Organization A and Organization B are both signed up for blockchain enabled software services that provide end to end food traceability solutions on their own but require all participants in the chain to use a single solution in order to work. 2. Both solutions of Organization A and B have terms and conditions such that it is possible technically and legally to integrate the software with each other and BIF.
Comments	

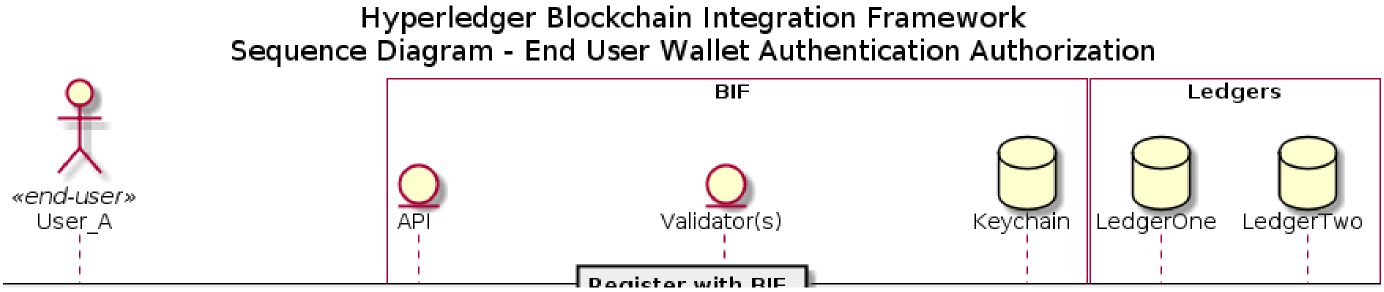

Hyperledger Blockchain Integration Framework
Sequence Diagram - Food Traceability Integration
Application
BIF
Ledgers
«consumer» User_A
«organiazation A staff» User_B
«organiazation B staff» User_C
Frontend
Backend
API
Validator(s)
Food_Ledger_1
Food_Ledger_2
Record Data
Append Food Data Record
Append Food Data Record
Append Food Data Record
Append Food Data Record
Food Data Record Appended
Food Data Record Appended
Food Data Record Appended
Food Data Record Appended
Append Food Data Record
Append Food Data Record
Append Food Data Record
Append Food Data Record
Food Data Record Appended
Food Data Record Appended
Food Data Record Appended
Food Data Record Appended
Merge Data
loop
[Polling/Triggered]
Query Ledger 1
Query Ledger 1
Food Record on Ledger 1
Food Record on Ledger 1
Query Ledger 2
Query Ledger 2
Food Record on Ledger 2
Food Record on Ledger 2
Compute Data Diff Ledger 1
Data Diff Ledger 1
Append Missing Data to Ledger 1
Append Missing Data to Ledger 1
Ledger 1 Updated
Ledger 1 Updated
Compute Data Diff Ledger 2
Data Diff Ledger 2
Append Missing Data to Ledger 2
Append Missing Data to Ledger 2
Ledger 2 Updated
Ledger 2 Updated
Trace Food Origins
Query Food Merged Record
Query Food Merged Record
Query Food Merged Record
Query Food Merged Record
At this point the data on Food Ledger 1 and 2 should be equivalent and therefore any one of them can be queried by BIF to obtain the complete food trace records.
Merged Food Data Records
Merged Food Data Records
Merged Food Data Records
Merged Food Data Records
Make well informed purchasing decision based on food origins.
«consumer» User_A
«organiazation A staff» User_B
«organiazation B staff» User_C
Frontend
Backend
API
Validator(s)
Food_Ledger_1
Food_Ledger_2

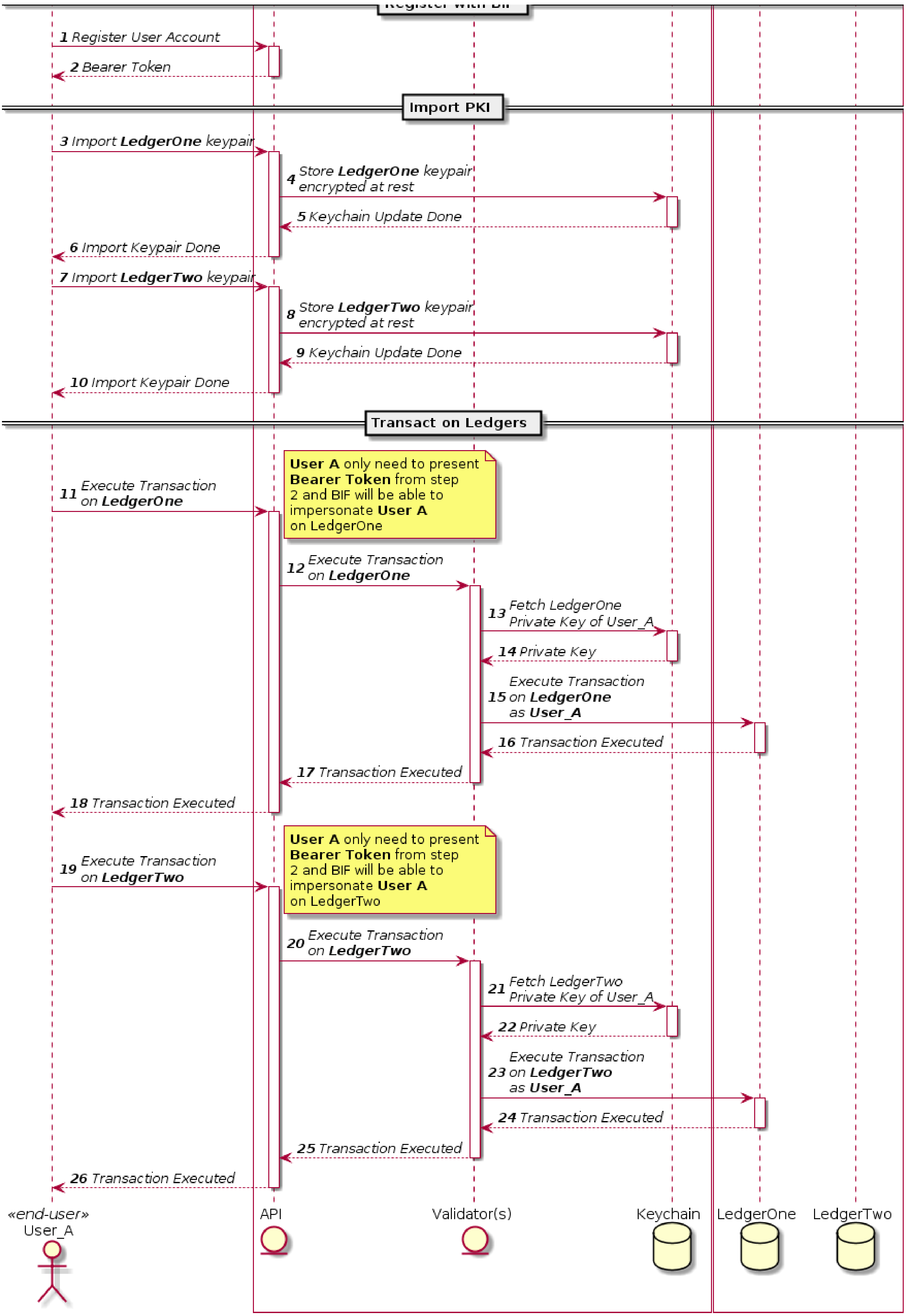
Hyperledger Blockchain Integration Framework
Food Tracability Integration


Retailer Customer
Traces food origins
Food Retailer <<organization>>
Integration for End to End Food Tracability «software deployment/subscription»
Food Tracability Solution B «software deployment/subscription»
Purchase Food Tracability Solution B
Source food from manufacturer
Purchase System Integration Project
Company B <<organization>>
Food Tracability Solution B «software»
Food Manufacturer <<organization>>
Food Tracability Solution A «software deployment/subscription»
System Integrator Company <<organization>>
System Integration «IT service»
Purchase Food Tracability Solution A
Builds integration on top of BIF
Company A <<organization>>
Food Tracability Solution A «software»
Hyperledger <<organization>>
Blockchain Integration Framework «software»

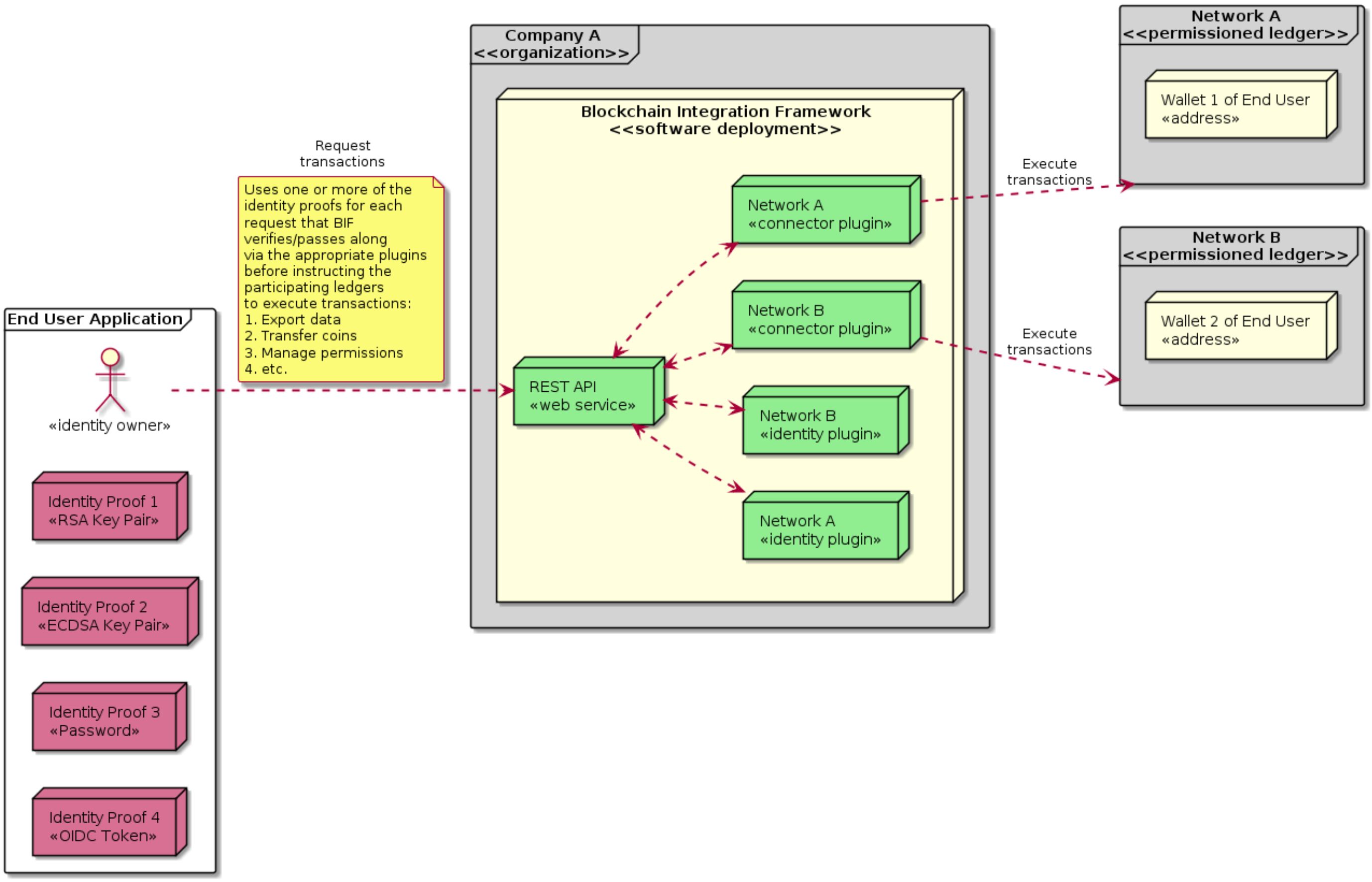
2.7 End User Wallet Authentication/Authorization

W3C Use Case Attribute Name	W3C Use Case Attribute Value
Use Case Title	Wallet Authentication/Authorization
Use Case	1. User A has separate identities on different permissioned and permissionless ledgers in the form of private/public key pairs (Public Key Infrastructure). 2. User A wishes to access/manage these identities through a single API or user interface and opts to on-board the identities to a BIF deployment. 3. User A performs the on-boarding of identities and is now able to interact with wallets attached to said identities through BIF or end user applications that leverage BIF under the hood (e.g. either by directly issuing API requests or using an application that does so.
Type of Social Interaction	Identity Management
Narrative	End user facing applications can provide a seamless experience connecting multiple permissioned (or permissionless) networks for an end user who has a set of different identity proofs for wallets on different ledgers.
Actors	User A: The person or entity whose identities get consolidated within a single BIF deployment
Goals of Actors	User A: Convenient way to manage an array of distinct identities with the trade-off that a BIF deployment must be trusted with the private keys of the identities involved (an educated decision on the user's part).
Success Scenario	User A is able to interact with their wallets without having to access each private key individually.
Success Criteria	User A's credentials are safely stored in the BIF keychain component where it is the least likely that they will be compromised (note that it is never impossible, but least unlikely, definitely)
Failure Criteria	User A is unable to import identities to BIF for a number of different reasons such as key format incompatibilities.
Prerequisites	1. User A has to have the identities on the various ledgers set up prior to importing them and must have access to the private
Comments	


Hyperledger Blockchain Integration Framework
Sequence Diagram - End User Wallet Authentication Authorization
BIF
Ledgers
«end-user»
User_A
API
Validator(s)
Keychain
LedgerOne
LedgerTwo
Register with BIF


Register with BIF
1 Register User Account
2 Bearer Token
Import PKI
3 Import LedgerOne keypair
4 Store LedgerOne keypair encrypted at rest
5 Keychain Update Done
6 Import Keypair Done
7 Import LedgerTwo keypair
8 Store LedgerTwo keypair encrypted at rest
9 Keychain Update Done
10 Import Keypair Done
Transact on Ledgers
User A only need to present Bearer Token from step 2 and BIF will be able to impersonate User A on LedgerOne
11 Execute Transaction on LedgerOne
12 Execute Transaction on LedgerOne
13 Fetch LedgerOne Private Key of User_A
14 Private Key
15 Execute Transaction on LedgerOne as User_A
16 Transaction Executed
17 Transaction Executed
18 Transaction Executed
User A only need to present Bearer Token from step 2 and BIF will be able to impersonate User A on LedgerTwo
19 Execute Transaction on LedgerTwo
20 Execute Transaction on LedgerTwo
21 Fetch LedgerTwo Private Key of User_A
22 Private Key
23 Execute Transaction on LedgerTwo as User_A
24 Transaction Executed
25 Transaction Executed
26 Transaction Executed
«end-user»
User_A
API
Validator(s)
Keychain
LedgerOne
LedgerTwo

Hyperledger Blockchain Integration Framework
Authentication, Authorization for Permissioned Chains



3. Software Design

3.1. Principles

3.1.1. Wide support

Interconnect as many ecosystems as possible regardless of technology limitations

3.1.2. Plugin Architecture from all possible aspects

Identities, DLTs, service discovery. Minimize how opinionated we are to really embrace interoperability rather than silos and lock-in. Closely monitor community feedback/PRs to determine points of contention where core BIF code could be lifted into plugins. Limit friction to adding future use cases and protocols.

3.1.3. Prevent Double spending Where Possible

Two representations of the same asset do not exist across the ecosystems at the same time unless clearly labelled as such [As of Oct 30 limited to specific combinations of DLTs; e.g. not yet possible with Fabric + Bitcoin]

3.1.4 DLT Feature Inclusivity

Each DLT has certain unique features that are partially or completely missing from other DLTs. BIF - where possible - should be designed in a way so that these unique features are accessible even when interacting with a DLT through BIF. A good example of this principle in practice would be Kubernetes CRDs and operators that allow the community to extend the Kubernetes core APIs in a reusable way.

3.1.5 Low impact

Interoperability does not redefine ecosystems but adapts to them. Governance, trust model and workflows are preserved in each ecosystem Trust model and consensus must be a mandatory part of the protocol handshake so that any possible incompatibilities are revealed up front and in a transparent way and both parties can “walk away” without unintended loss of assets/data. The idea comes from how the traditional online payment processing APIs allow merchants to specify the acceptable level of guarantees before the transaction can be finalized (e.g. need pin, signed receipt, etc.). Following the same logic, we shall allow transacting parties to specify what sort of consensus, transaction finality, they require. Consensus requirements must support predicates, e.g. “I am on Fabric, but will accept Bitcoin so long X number of blocks were confirmed post-transaction” Requiring KYC (Know Your Customer) compliance could also be added to help foster adoption as much as possible.

3.1.6 Transparency

Cross-ecosystem transfer participants are made aware of the local and global implications of the transfer. Rejection and errors are communicated in a timely fashion to all participants. Such transparency should be visible as trustworthy evidence.

3.1.7 Automated workflows

Logic exists in each ecosystem to enable complex interoperability use-cases. Cross-ecosystem transfers can be automatically triggered in response to a previous one. Automated procedure, which is regarding error recovery and exception handling, should be executed without any interruption.

3.1.8 Default to Highest Security

Support less secure options, but strictly as opt-in, never opt-out.

3.1.9 Transaction Protocol Negotiation

Participants in the transaction must have a handshake mechanism where they agree on one of the supported protocols to use to execute the transaction. The algorithm looks an intersection in the list of supported algorithms by the participants.

3.1.10 Avoid modifying the total amount of digital assets on any blockchain whenever possible

We believe that increasing or decreasing the total amount of digital assets might weaken the security of blockchain, since adding or deleting assets will be complicated. Instead, intermediate entities (e.g. exchanger) can pool and/or send the transfer.

3.1.11 Provide abstraction for common operations

Our communal modularity should extend to common mechanisms to operate and/or observe transactions on blockchains.

3.1.12 Integration with Identity Frameworks (Moonshot)

Do not expend opinions on identity frameworks just allow users of BIF to leverage the most common ones and allow for future expansion of the list of supported identity frameworks through the plugin architecture. Allow consumers of BIF to perform authentication, authorization and reading/writing of credentials.

Identity Frameworks to support/consider initially:

- Hyperledger Indy (Sovrin)
- DIF
- DID

3.2 Feature Requirements

3.2.1 New Protocol Integration

Adding new protocols must be possible as part of the plugin architecture allowing the community to propose, develop, test and release their own implementations at will.

3.2.2 Proxy/Firewall/NAT Compatibility

Means for establishing bidirectional communication channels through proxies/firewalls/NAT wherever possible

3.2.3 Bi-directional Communications Layer

Using a blockchain agnostic bidirectional communication channel for controlling and monitoring transactions on blockchains through proxies/firewalls/NAT wherever possible. * Blockchains vary on their P2P communication protocols. It is better to build a modular method for sending/receiving generic transactions between trustworthy entities on blockchains.

3.2.4 Consortium Management

Consortiums can be formed by cooperating entities (person, organization, etc.) who wish to all contribute hardware/network resources to the operation of a BIF cluster (set of validator nodes, API servers, etc.).

After the forming of the consortium with it’s initial set of members (one or more) it is possible to enroll or remove certain new or existing members.

BIF does not prescribe any specific consensus algorithm for the addition or removal of consortium members, but rather focuses on the technical side of making it possible to operate a cluster of nodes under the ownership of separate entities without downtime while also keeping it possible to add/remove members.

A newly joined consortium member does not have to participate in every component of BIF: Running a validator node is the only required action to participate, etcd, API server can remain the same as prior to the new member joining.

3.3 Working Policies

1. Participants can insist on a specific protocol by pretending that they only support said protocol only.
2. Protocols can be versioned as the specifications mature
3. The two initially supported protocols shall be the ones that can satisfy the requirements for Fujitsu’s and Accenture’s implementations respectively

4. Architecture

4.1 Interworking patterns

4.1.1 Interworking patterns list

The Blockchain Integration Framework has several interworking patterns as the following.

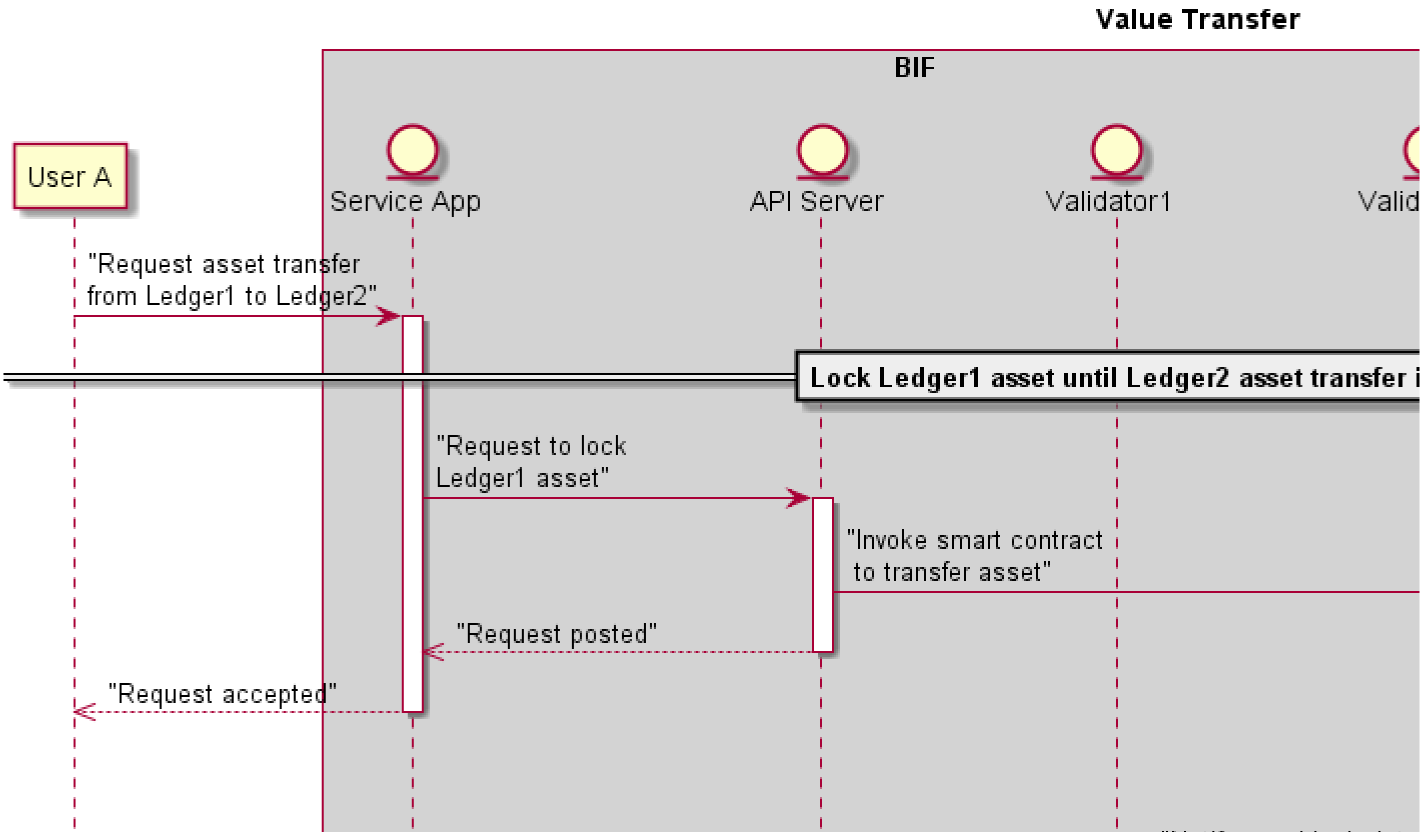
- Note: In the following description, **Value (V)** means numerical assets (e.g. money). **Data (D)** means non-numerical assets (e.g. ownership proof). Ledger 1 is source ledger, Ledger 2 is destination ledger.

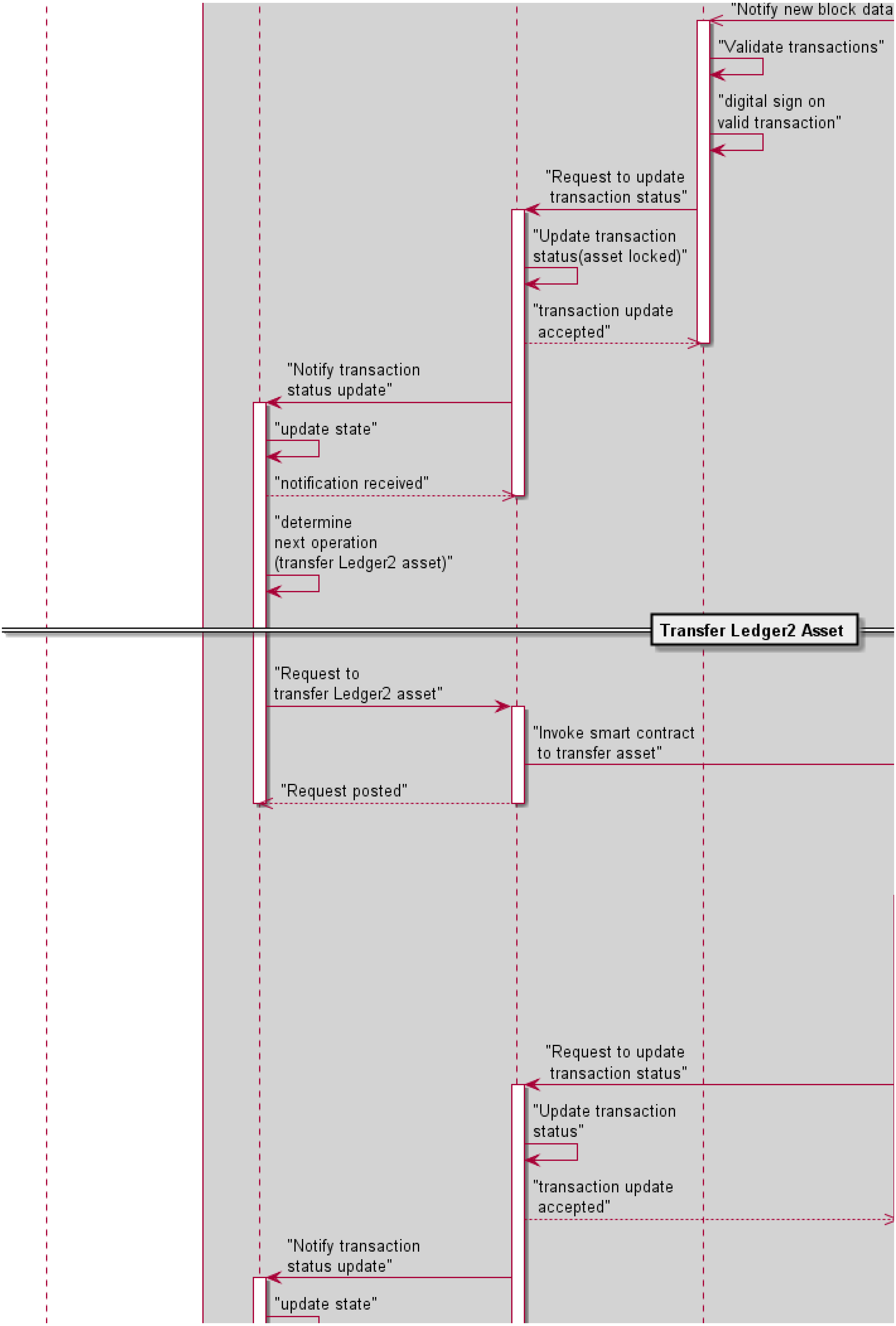
No.	Name	Pattern	Consistency
1.	value transfer	V -> V	check if V1 = V2 (as V1 is value on ledger 1, V2 is value on ledger 2)
2.	value-data transfer	V -> D	check if data transfer is successful when value is transferred
3.	data-value transfer	D -> V	check if value transfer is successful when data is transferred
4.	data transfer	D -> D	check if all D1 is copied on ledger 2 (as D1 is data on ledger 1, D2 is data on ledger 2)
5.	data merge	D <-> D	check if D1 = D2 as a result (as D1 is data on ledger 1, D2 is data on ledger 2)

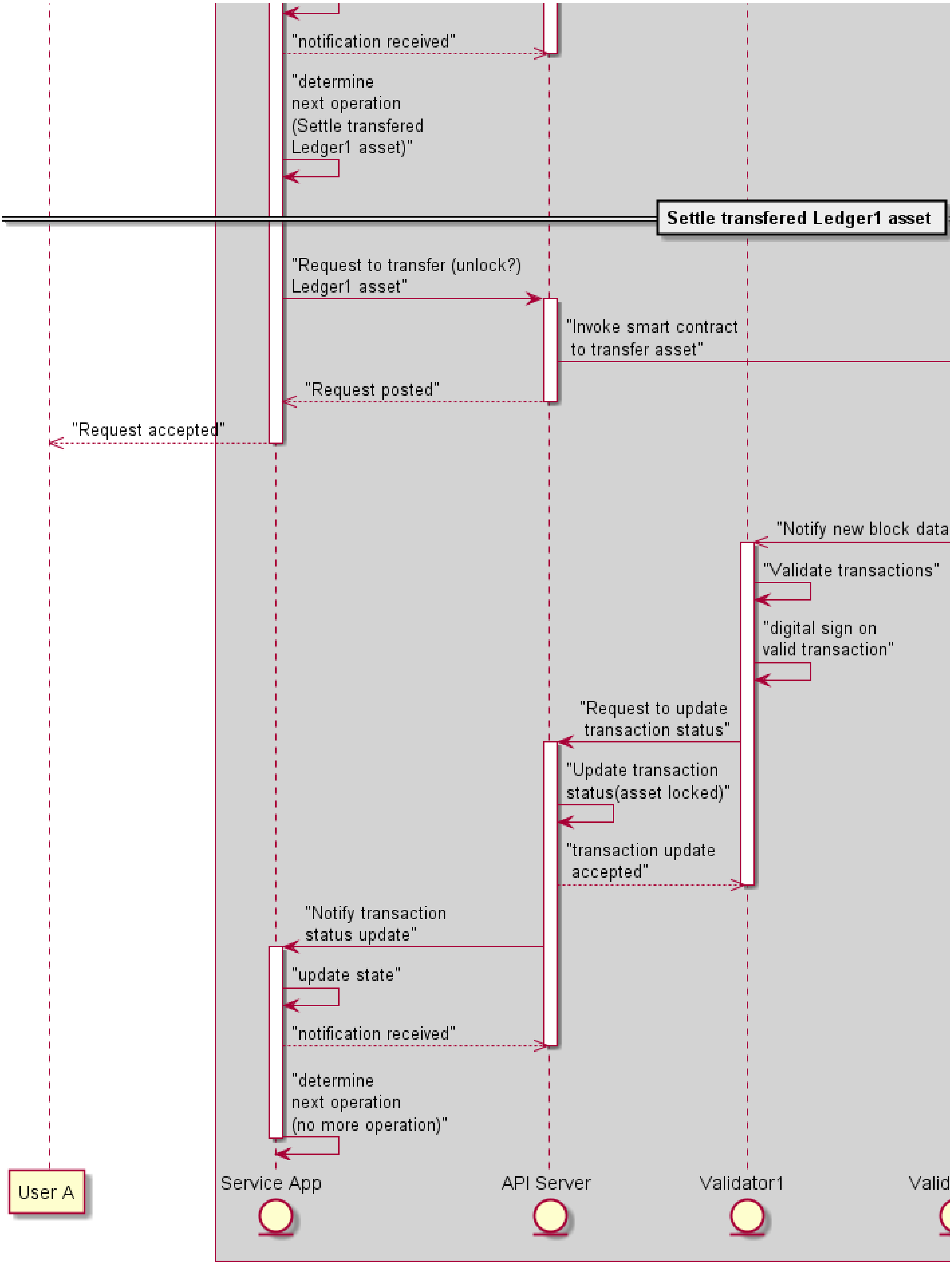
4.1.2 Value transfer

4.1.2.1 Desription of the pattern

4.1.2.2 Sequence diagram




"Notify new block data
"Validate transactions"
"digital sign on valid transaction"
"Request to update transaction status"
"Update transaction status(asset locked)"
"transaction update accepted"
"Notify transaction status update"
"update state"
"notification received"
"determine next operation (transfer Ledger2 asset)"
Transfer Ledger2 Asset
"Request to transfer Ledger2 asset"
"Invoke smart contract to transfer asset"
"Request posted"
"Request to update transaction status"
"Update transaction status"
"transaction update accepted"
"Notify transaction status update"
"update state"



4.1.3 Value-data transfer

4.1.3.1 Desription of the pattern

4.1.3.2 Sequence diagram

4.1.4 Data-value transfer

4.1.4.1 Desription of the pattern

4.1.4.2 Sequence diagram

4.1.5 Data transfer

4.1.5.1 Desription of the pattern

4.1.5.2 Sequence diagram

4.1.6 Data merge

4.1.6.1 Desription of the pattern

4.1.6.2 Sequence diagram

4.2 Technical Architecture

4.2.1 Monorepo Packages

The Blockchain Integration Framework is divided into a set of npm packages that can be compiled separately or all at once.

Naming conventions for packages: * cmd-* for packages that ship their own executable * sdk-* for packages designed to be used directly by application developers * All other packages should be named preferably as a single English word suggesting the most important feature/responsibility of the package itself.

4.2.1.1 core

Contains the kernel of the Blockchain Integration Framework. Code that is strongly opinionated lives here, the rest is pushed to other packages that implement plugins or define their interfaces.

The main responsibilities of the core package are:

4.2.1.1.1 Runtime Configuration Parsing and Validation

The core package is responsible for parsing runtime configuration from the usual sources (shown in order of precedence): * Explicit instructions via code (config.setHttpPort(3000);) * Command line arguments (--http-port=3000) * Operating system environment variables (HTTP_PORT=3000) * Static configuration files (config.json: { "httpPort": 3000 })

The Apache 2.0 licensed node-convict library to be leveraged for the mechanical parts of the configuration parsing and validation: https://github.com/mozilla/node-convict

4.2.1.1.2 Configuration Schema - Validator

Parameter	Type	Config Key: CLI	Config Key: Env	Config Key: JSON	Description
Etcd Hosts	Array<string>	--etcd-hosts	ETCD_HOSTS	etcdHosts	The hosts of Etcd nodes the validator node should connect to for the purpose of leadership election.
Private Key	string	--private-key	PRIVATE_KEY	privateKey	The private key of the validator node to be used when signing validated messages.
Public Key	string	--public-key	PUBLIC_KEY	publicKey	The public key of the validator node that pairs with the Private Key of the same node.

4.2.1.1.3 Configuration Schema - API Server

Parameter	Type	Key: CLI, Env, JSON	Description
Validator Hosts	Array<string>	--validator-hosts VALIDATOR_HOSTS validatorHosts	List of hosts to connect to when requesting validation related tasks from the validator nodes.

Parameter	Type	Key: CLI, Env, JSON	Description
HTTPS_PORT	number	--https-port HTTPS_PORT httpsPort	The TCP port to listen on for HTTPS connections.
CORS Domains	Array<string>	--cors-domains CORS_DOMAINS corsDomains	Optional. Zero or more domain patterns (wildcards are allowed).
Virtual Hosts	Array<string>	--virtual-hosts VIRTUAL_HOSTS virtualHosts	Optional. When specified, constrains the acceptable incoming requests to ones that specify their host HTTP header in a way that matches at least one of the patterns specified in this configuration parameter.
Authentication Strategies	Array<string>	--authentication-strategies AUTHENTICATION_STRATEGIES authenticationStrategies	Optional. Specifies the fully qualified name, version and exported module of one or more npm packages that are to be loaded and used as the providers for the authentication strategies. For example to use PassportJS's OpenID Connect strategy one with specify the value ["passport-oidc-strategy@0.1.1###Strategy"] which will get parsed as a JSON string containing an array of strings.
Authentication Options	Array<string>	--authentication-options AUTHENTICATION_OPTIONS authenticationOptions	Used to provide arguments to the constructors (or factory functions) exported by the modules specified by AUTHENTICATION_STRATEGIES. For example, in this configuration parameter you can specify the callback URL for an Open ID Connect provider of your choice, the client ID, client secret, etc. Important: The order in which the items appear have to match the order of items in AUTHENTICATION_STRATEGIES.
Package Registries	Array<string>	--package-registries PACKAGE_REGISTRIES packageRegistries	Optional. Defaults to the public npm registry at https://registry.npmjs.org/. Can be used to specify private registries in the event of closed source plugins. If multiple registry URLs are provided, they all will be tried in-order at bootstrap time.

4.2.1.1.4 Plugin Loading/Validation

Plugin loading happens through NodeJS's built-in module loader and the validation is performed by the Node Package Manager tool (npm) which verifies the byte level integrity of all installed modules.

4.2.1.2 cmd-api-server

A command line application for running the API server that provides a unified REST based HTTP API for calling code.

By design this is stateless and horizontally scalable.

Comes with Swagger API definitions.

4.2.1.3 cmd-validator

Command line application to run a single BIF validator node.

4.2.1.4 sdk-javascript

Javascript SDK (bindings) for the RESTful HTTP API provided by cmd-api-server. Compatible with both NodeJS and Web Browser (HTML 5 DOM + ES6) environments.

4.2.1.5 keychain

Responsible for persistently storing highly sensitive data (e.g. private keys) in an encrypted format.

For further details on the API surface, see the relevant section under Plugin Architecture.

4.2.1.7 tracing

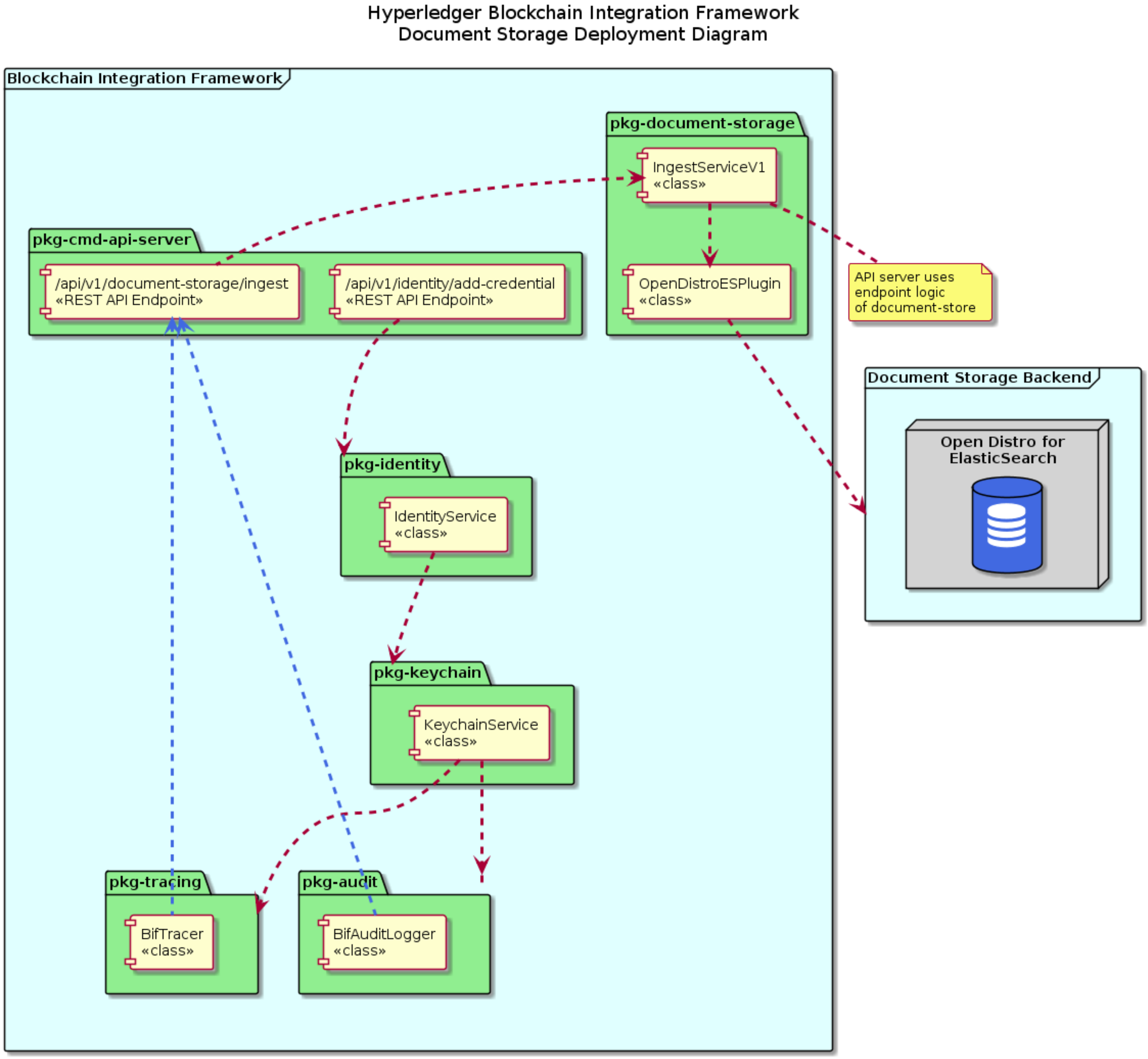
Contains components for tracing, logging and application performance management (APM) of code written for the rest of the Blockchain Integration Framework packages.

4.2.1.8 audit

Components useful for writing and reading audit records that must be archived longer term and immutable. The latter properties are what differentiates audit logs from tracing/logging messages which are designed to be ephemeral and to support technical issues not regulatory/compliance/governance related issues.

4.2.1.9 document-storage

Provides structured or unstructured document storage and analytics capabilities for other packages such as audit and tracing. Comes with its own API surface that serves as an adapter for different storage backends via plugins. By default, Open Distro for ElasticSearch is used as the storage backend: https://aws.amazon.com/blogs/aws/new-open-distro-for-elasticsearch/



> The API surface provided by this package is kept intentionally simple and feature-poor so that different underlying storage backends remain an option long term through the plugin architecture of BIF.

4.2.1.10 relational-storage

Contains components responsible for providing access to standard SQL compliant persistent storage.

> The API surface provided by this package is kept intentionally simple and feature-poor so that different underlying storage backends remain an option long term through the plugin architecture of BIF.

4.2.1.11 immutable-storage

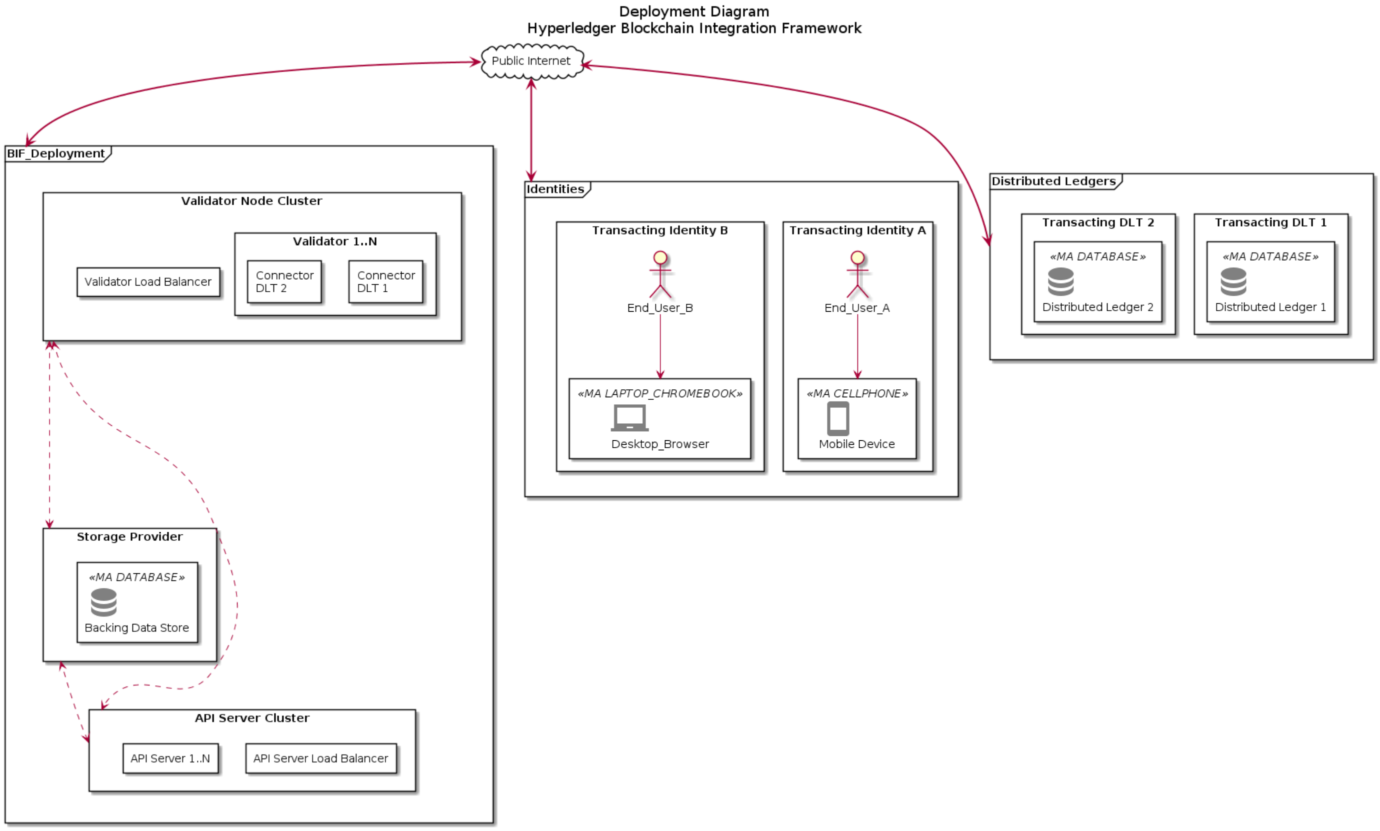
Contains components responsible for providing access to immutable storage such as a distributed ledger with append-only semantics such as a blockchain network (e.g. Hyperledger Fabric).

> The API surface provided by this package is kept intentionally simple and feature-poor so that different underlying storage backends remain an option long term through the

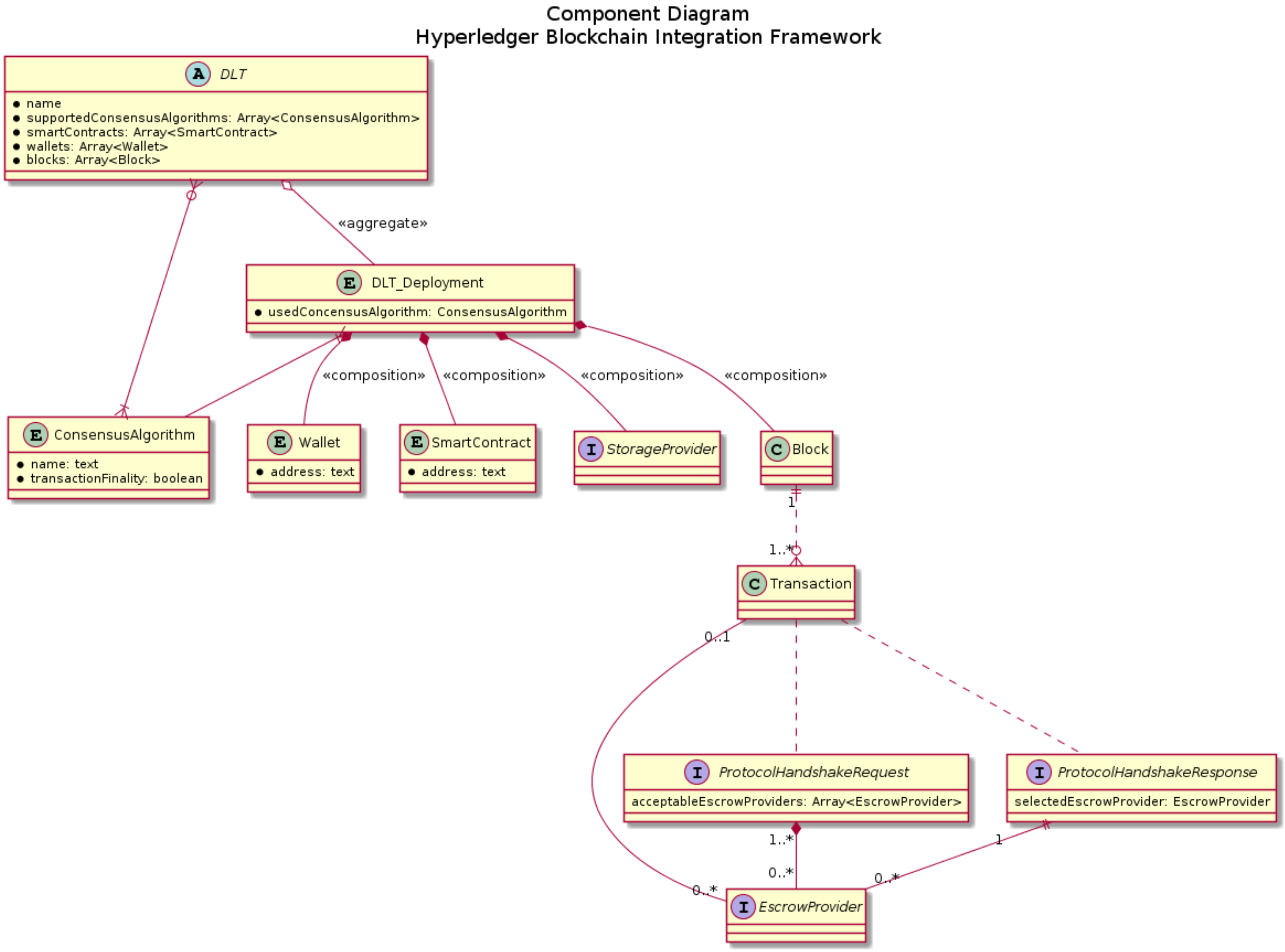
plugin architecture of BIF.

4.2.2 Deployment Diagram

Source file: ./docs/architecture/deployment-diagram.puml



4.2.3 Component Diagram


Component Diagram
Hyperledger Blockchain Integration Framework
A DLT
name
supportedConsensusAlgorithms: Array<ConsensusAlgorithm>
smartContracts: Array<SmartContract>
wallets: Array<Wallet>
blocks: Array<Block>
«aggregate»
E DLT_Deployment
usedConcensusAlgorithm: ConsensusAlgorithm
«composition»
«composition»
«composition»
«composition»
E ConsensusAlgorithm
name: text
transactionFinality: boolean
E Wallet
address: text
E SmartContract
address: text
I StorageProvider
C Block
1
1..*
C Transaction
0..1
I ProtocolHandshakeRequest
acceptableEscrowProviders: Array<EscrowProvider>
I ProtocolHandshakeResponse
selectedEscrowProvider: EscrowProvider
1..*
0..*
1
0..*
0..*
I EscrowProvider

4.2.4 Class Diagram

4.2.5 Sequence Diagram - Transactions

TBD

4.3 Transaction Protocol Specification

4.3.1 Handshake Mechanism

TBD

4.3.2 Transaction Protocol Negotiation

Participants in the transaction must have a handshake mechanism where they agree on one of the supported protocols to use to execute the transaction. The algorithm looks an intersection in the list of supported algorithms by the participants.

Participants can insist on a specific protocol by pretending that they only support said protocol only. Protocols can be versioned as the specifications mature. Adding new protocols must be possible as part of the plugin architecture allowing the community to propose, develop, test and release their own implementations at will. The two initially supported protocols shall be the ones that can satisfy the requirements for Fujitsu's and Accenture's implementations respectively. Means for establishing bi-directional communication channels through proxies/firewalls/NAT wherever possible

4.4 Plugin Architecture

Since our goal is integration, it is critical that BIF has the flexibility of supporting most ledgers, even those that don't exist today.

> A plugin is a self contained piece of code that implements a predefined interface pertaining to a specific functionality of BIF such as transaction execution.

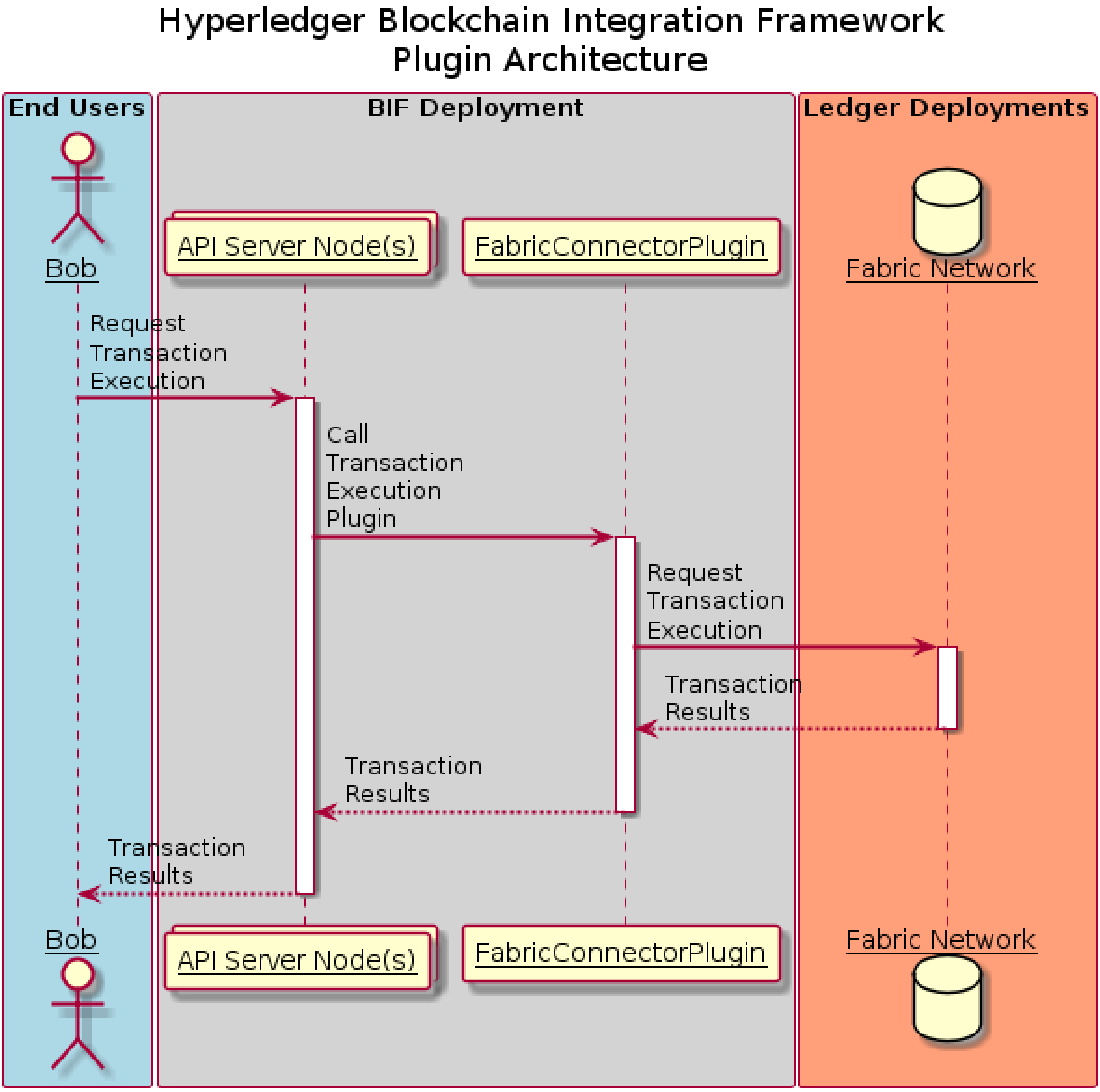
Plugins are an abstraction layer on top of the core components that allows operators of BIF to swap out implementations at will.

> Backward compatibility is important, but versioning of the plugins still follows the semantic versioning convention meaning that major upgrades can have breaking changes.

Plugins are implemented as ES6 modules (source code) that can be loaded at runtime from the persistent data store. The core package is responsible for validating code signatures to guarantee source code integrity.

An overarching theme for all aspects that are covered by the plugin architecture is that there should be a dummy implementation for each aspect to allow the simplest possible deployments to happen on a single, consumer grade machine rather than requiring costly hardware and specialized knowledge.

> Ideally, a fully testable/operational (but not production ready) BIF deployment could be spun up on a developer laptop with a single command (an npm script for example).



4.4.1 Ledger Connector Plugins

Success is defined as: 1. Adding support in BIF for a ledger invented in the future requires no core code changes, but instead can be implemented by simply adding a corresponding connector plugin to deal with said newly invented ledger. 2. Client applications using the REST API and leveraging the feature checks can remain 100% functional regardless of the number and nature of deployed connector plugins in BIF. For example: a generic money sending application does not have to hardcode the supported ledgers it supports because the unified REST API interface (fed by the ledger connector plugins) guarantees that supported features will be operational.

Because the features of different ledgers can be very diverse, the plugin interface has feature checks built into allowing callers/client applications to **determine programmatically, at runtime** if a certain feature is supported or not on a given ledger.

```
export interface LedgerConnector {
  // method to verify a signature coming from a given ledger that this connector is responsible for connecting to.
  verifySignature(message, signature): Promise<boolean>;

  // used to call methods on smart contracts or to move assets between wallets
  transact(transactions: Transaction[]);

  getPermissionScheme(): Promise<PermissionScheme>;

  getTransactionFinality(): Promise<TransactionFinality>;

  addForeignValidator(): Promise<void>;
}

export enum TransactionFinality {
  GUARANTEED = "GUARANTEED",
  NOT_GUARANTEED = "NOT_GUARANTEED"
}

export enum PermissionScheme {
  PERMISSIONED = "PERMISSIONED",
  PERMISSIONLESS = "PERMISSIONLESS"
}
```

4.4.2 Identity Federation Plugins

Identity federation plugins operate inside the API Server and need to implement the interface of a common PassportJS Strategy: https://github.com/jaredhanson/passport-strategy#implement-authentication

```
abstract class IdentityFederationPlugin {
  constructor(options: any): IdentityFederationPlugin;
  abstract authenticate(req: ExpressRequest, options: any);
  abstract success(user, info);
  abstract fail(challenge, status);
  abstract redirect(url, status);
  abstract pass();
  abstract error(err);
}
```

4.4.1.1 X.509 Certificate Plugin

The X.509 Certificate plugin facilitates clients authentication by allowing them to present a certificate instead of operating with authentication tokens. This technically allows calling clients to assume the identities of the validator nodes through the REST API without having to have access to the signing private key of said validator node.

PassportJS already has plugins written for client certificate validation, but we go one step further with this plugin by providing the option to obtain CA certificates from the validator nodes themselves at runtime.

4.4.3 Key/Value Storage Plugins

Key/Value Storage plugins allow the higher-level packages to store and retrieve configuration metadata for a BIF cluster such as: * Who are the active validators and what are the hosts where said validators are accessible over a network? * What public keys belong to which validator nodes? * What transactions have been scheduled, started, completed?

```
interface KeyValueStoragePlugin {
  async get<T>(key: string): Promise<T>;
  async set<T>(key: string, value: T): Promise<void>;
  async delete<T>(key: string): Promise<void>;
}
```

4.4.4 Serverside Keychain Plugins

The API surface of keychain plugins is roughly the equivalent of the key/value *Storage* plugins, but under the hood these are of course guaranteed to encrypt the stored data at rest by way of leveraging storage backends purpose built for storing and managing secrets.

Possible storage backends include self hosted software [1] and cloud native services [2][3][4] as well. The goal of the keychain plugins (and the plugin architecture at large) is to make BIF deployable in different environments with different backing services such as an on-premise data center or a cloud provider who sells their own secret management services/APIs. There should be a dummy implementation as well that stores secrets in-memory and unencrypted (strictly for development purposes of course). The latter will decrease the barrier to entry for new users and would be contributors alike.

Direct support for HSM (Hardware Security Modules) is also something the keychain plugins could enable, but this is lower priority since any serious storage backend with secret management in mind will have built-in support for dealing with HSMs transparently.

By design, the keychain plugin can only be used by authenticated users with an active BIF session. Users secrets are isolated from each other on the keychain via namespacing that is internal to the keychain plugin implementations (e.g. users cannot query other users namespaces whatsoever).

```
interface KeychainPlugin extends KeyValueStoragePlugin {
}
```

[1] https://www.vaultproject.io/ [2] https://aws.amazon.com/secrets-manager/ [3] https://aws.amazon.com/kms/ [4] https://azure.microsoft.com/en-us/services/key-vault/

5. Identities, Authentication, Authorization

BIF aims to provide a unified API surface for managing identities of an identity owner. Developers using the BIF REST API for their applications can support one or both of the below requirements: 1. Applications with a focus on access control and business process efficiency (usually in the enterprise) 2. Applications with a focus on individual privacy (usually consumer-based applications)

The following sections outline the high-level features of BIF that make the above vision reality.

An end user (through a user interface) can issue API requests to * register a username+password account (with optional MFA) **within** BIF. * associate their wallets to their BIF account and execute transactions involving those registered wallet (transaction signatures performed either locally or remotely as explained above).

5.1 Transaction Signing Modes, Key Ownership

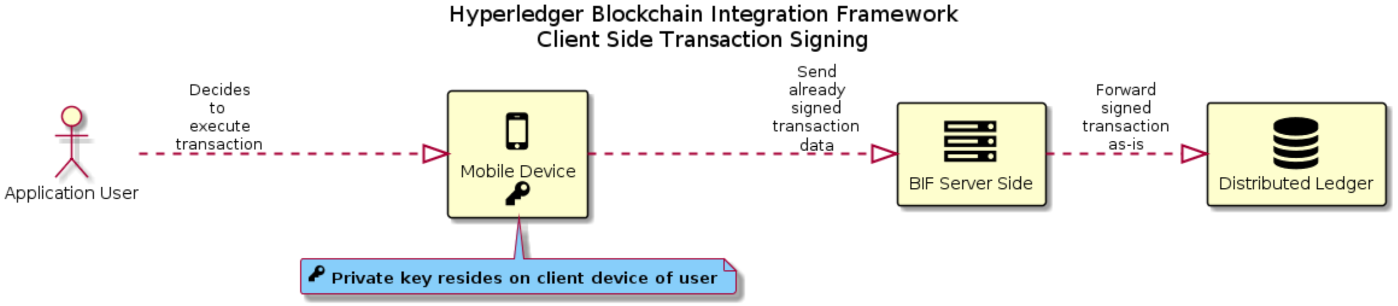
An application developer using BIF can choose to enable users to sign their transactions locally on their user agent device without disclosing their private keys to BIF or remotely where BIF stores private keys server-side, encrypted at rest, made decryptable through authenticating with their BIF account. Each mode comes with its own pros and cons that need to be carefully considered at design time.

5.1.1 Client-side Transaction Signing

Usually a better fit for consumer-based applications where end users have higher expectation of individual privacy.

Pros * Keys are not compromised when a BIF deployment is compromised * Operator of BIF deployment is not liable for breach of keys (same as above) * Reduced server-side complexity (no need to manage keys centrally)

Cons * User experience is sub-optimal compared to sever side transaction signing * Users can lose access permanently if they lose the key (not acceptable in most enterprise/professional use cases)

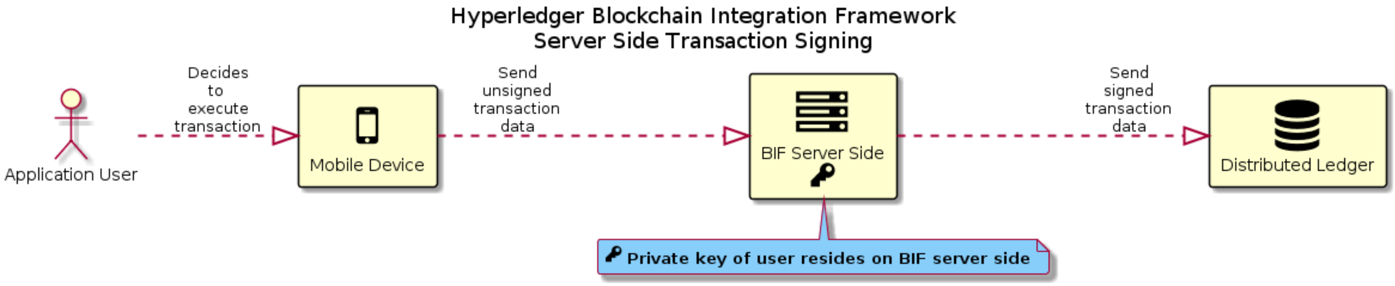

Hyperledger Blockchain Integration Framework
Client Side Transaction Signing
Application User
Decides to execute transaction
Mobile Device
Send already signed transaction data
BIF Server Side
Forward signed transaction as-is
Distributed Ledger
Private key resides on client device of user

5.1.2 Server-side Transaction Signing

Usually a better fit for enterprise applications where end users have most likely lowered their expectations of individual privacy due to the hard requirements of compliance, governance, internal or external policy enforcement.

Pros * Frees end users from the burden of managing keys themselves (better user experience) * Improved compliance, governance

Cons * Server-side breach can expose encrypted keys stored in the keychain


Hyperledger Blockchain Integration Framework
Server Side Transaction Signing
Application User
Decides to execute transaction
Mobile Device
Send unsigned transaction data
BIF Server Side
Send signed transaction data
Distributed Ledger
Private key of user resides on BIF server side

5.2 Open ID Connect Provider, Identity Provider

BIF can authenticate users against *third party Identity Providers* or serve as an *Identity Provider* itself. Everything follows the well-established industry standards of Open ID Connect to maximize information security and reduce the probability of data breaches.

5.3 Server-side Keychain for Web Applications

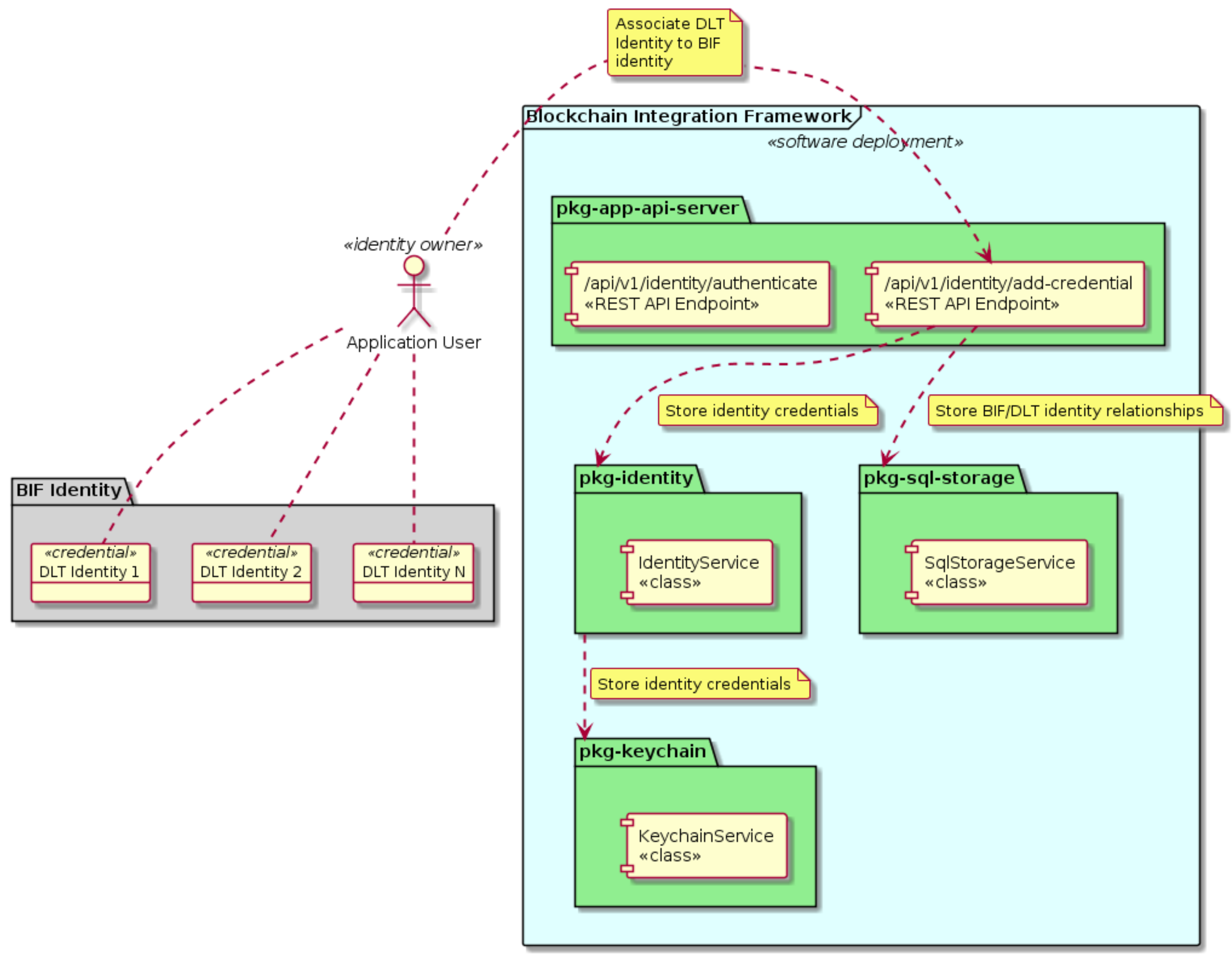
There is a gap between traditional web/mobile applications and blockchain applications (web 2.0 and 3.0 if you will) authentication protocols in the sense that blockchain networks rely on private keys belonging to a Public Key Infrastructure (PKI) to authenticate users while traditional web/mobile applications mostly rely on a centralized authority storing hashed passwords and the issuance of ephemeral tokens upon successful authentication (e.g. successful login with a password). Traditional (Web 2.0) applications (that adhering security best practices) use server-side sessions (web) or secure keychains provided by the operating system (iOS, Android, etc.) The current industry standard and state of the art authentication protocol in the enterprise application development industry is Open ID Connect (OIDC).

To successfully close the gap between the two worlds, BIF comes equipped with an OIDC identity provider and a server-side key chain that can be leveraged by end user applications to authenticate once against BIF and manage identities on other blockchains through that single BIF identity. This feature is important for web applications which do not have secure offline storage APIs (HTML localStorage is not secure).

Example: A user can register for a BIF account, import their private keys from their Fabric/Ethereum wallets and then have access to all of those identities by authenticating once only against BIF which will result in a server-side session (HTTP cookie) containing a JSON Web Token (JWT).

> Native mobile applications may not need to use the server-side keychain since they usually come equipped with an OS provided one (Android, iOS does).

Hyperledger Blockchain Integration Framework
Unified Identity Management


Associate DLT Identity to BIF identity
Blockchain Integration Framework
«software deployment»
pkg-app-api-server
/api/v1/identity/authenticate «REST API Endpoint»
/api/v1/identity/add-credential «REST API Endpoint»
Store identity credentials
Store BIF/DLT identity relationships
pkg-identity
IdentityService «class»
pkg-sql-storage
SqlStorageService «class»
Store identity credentials
pkg-keychain
KeychainService «class»
«identity owner»
Application User
BIF Identity
«credential» DLT Identity 1
«credential» DLT Identity 2
«credential» DLT Identity N

In web 2.0 applications the prevalent authentication/authorization solution is Open ID Connect which bases authentication on passwords and tokens which are derived from the passwords. Web 3.0 applications (decentralized apps or *DApps*) which interact with blockchain networks rely on private keys instead of passwords.

6. Terminology

API Server: A module of BIF which provides a unified interface to control/monitor Blockchain ledger behind it.

Validator: A module of BIF which verifies validity of transaction to be sent out to the blockchain application.

Lock asset: An operation to the asset managed on blockchain ledger, which disable further operation to targeted asset. The target can be whole or partial depends on type of asset.

Abort: A state of BIF which is determined integrated ledger operation is failed, and BIF will execute recovery operations.

Integrated ledger operation: A series of blockchain ledger operations which will be triggered by BIF. BIF is responsible to execute 'recovery operations' when 'Abort' is occurred.

Restore operation(s): Single or multiple ledger operations which is executed by BIF to restore the state of integrated service before start of integrated operation.

End User: A person (private citizen or a corporate employee) who interacts with BIF and other ledger-related systems to achieve a specific goal or complete a task such as to send/receive/exchange money or data.

Business Organization: A for-profit or non-profit entity formed by one or more people to achieve financial gain or achieve a specific (non-financial) goal. For brevity, *business organization* may be shortened to *organization* throughout the document.

Identity Owner: A person or organization who is in control of one or more identities. For example, owning two separate email accounts by one person means that said person is the identity owner of two separate identities (the email accounts). Owning cryptocurrency wallets (their private keys) also makes one an identity owner.

Identity Secret: A private key or a password that - by design - is only ever known by the identity owner (unless stolen).

Credentials: Could mean user authentication credentials/identity proofs in an IT application or any other credentials in the traditional sense of the word such as a proof that a person obtained a masters or PhD.

Ledger/Network/Chain: Synonymous words meaning referring largely to the same thing in this paper.

OIDC: Open ID Connect authentication protocol

PKI: Public Key Infrastructure

MFA: Multi Factor Authentication

7. References

1: Heterogeneous System Architecture - Wikipedia, Retrieved at: 11th of December 2019