Research Report

CBDC-based smart contract ecosystems

Imre Kocsis, László Gönczy, Attila Klenik, Pál Varga, Attila Frankó, Bence Oláh

July 2021

Contents

1 Introduction

Central Bank Digital Currency (CBDC) is "*a digital form of central bank money that is different from balances in traditional reserve or settlement accounts*"[8]. The evolving public stance of central banks in conjunction with CBDCs, the large number of experiments, proof of concept implementations and even a few production implementations[1] foreshadow that, in the coming years, more and more central banks will begin to create novel digital forms of central bank money.

Why the global interest in augmenting existing (digital) money and cash through digital central bank money is arising right now – and at the same time, practically everywhere – is not easy to pinpoint. The fundamental economic concept – or rather, concepts – coined as CBDC has, have been around for a long time; and although a consensus seems to be forming around the much sharper definition of a CBDC being a "*digital payment instrument, denominated in the national unit of account, that is a direct liability of the central bank*" [5], the various monetary, societal (and even fiscal) value drivers, which are cited globally, are very varied. For a stark contrast, it is worthwile to compare the opportunity analysis of the Bank of England [11] which largely focuses on maintaining a resilient payments landscape (even with the decline in the use of cash) with the eCNY pilot of the People's Republic of China, which, experts seem to agree, has strong finance control purposes.

That being said, in science fiction parlance, it seems to be steam engine time: all the precursor ideas and technological inventions[2] are in place now to answer to broadly similar requirements; and it seems like that, suddenly, everybody is building steam engines – that is, CBDCs.

With the ongoing developments as a backdrop, the work we undertook and report on in this paper tried to look at the next step ahead: when CBDCs will finally become a reality, how will they impact life *beyond* the original, payment and money transfer oriented CBDC use cases?

Our specific research goal was to demonstrate that it is technically viable to equip smart contract technology – already a game-changer in many sectors – with the ability to use retail[3] CBDCs and the *benefits* of doing so for *industrial* use cases.

However, in the current absence of open CBDC implementations, first we also had to design and implement a retail CBDC prototype. This report describes the following results:

1. Technical design considerations for a retail CBDC which uses distributed ledger technology to meet integrity, availability and resilience requirements.

2. The implementation of a CBDC prototype system.

[1] See, e.g., the site `https://cbdctracker.org`

[2] Including cryptocurrencies and blockchain technologies.

[3] Widely accessible; in contrast to wholesale CBDCs, which are restricted to financial institutions.

3. A representative smart contract based solution for an industrial asset management cooperation scenario.

4. Integration of the CBDC prototype with the consortial blockchain network hosting the industrial solution through *asset bridging*.

5. Demonstration of the integrated system.

This work is a report on *technological exploration*. Beyond the core "money" functionality (central bank issued electronic legal tender usable as a medium of exchange), CBDCs have a wide range of proposed use cases – from monetary through fiscal to public sector ones. Some of these don't seem to carry the potential to disrupt the established financial mechanisms of a nation in a major way; two examples are "colored" – spendable only for certain purposes – direct governmental transfers and facilitating the access of the unbanked to electronic money. However, very radical visions have been formulated, too, which envision, among others, influencing the movement of capital between sectors and gauging economic activity almost real-time through a CBDC [15]. This report focuses on the core functionality – and how it can be extended towards more sophisticated use cases through smart contracts. What focus there is on specific usage falls on industrial applications.

Technological exploration also means that we ignore most questions of *central bank policy* related to CBDCs. These include, but are not limited to, the question whether a CBDC should carry interest, and if yes, what should be the interest rate[4]. Last but not least, the report assumes that the reader is familiar with the basics concepts of Distributed Ledger Technology (DLT), the types of DLT networks (public-unpermissioned, public-permissioned, consortial-permissioned) and blockchains as their (today) predominant implementation approach[5].

[4]Central banks will likely want to avoid a "run to CBDC". [1] presents a compelling argument towards the ability of a CBDC to coexist with the banking sector without a detrimental effect on bank lending activity – if the interest on CBDC is set properly. At the same time, CBDCs can be also used to introduce negative interest on (electronic) "cash": something that is certainly impossible with physical cash, at least in direct, nominal terms. Setting a CBDC interest rate, however, is as much a question of applied economic theory as of central bank policy – and decidedly *not* one of CBDC technology.

[5][16] provides a very tractable and well structured overview.

2 CBDC ledger model and architecture

From an information system engineering point of view, the distinguishing, and, as we will see, to a great extent orthogonal, dimensions of a *widely accessible*, *electronic* and *central-bank issued* electronic money system are – at least – fourfold; all dimensions characterizing a key aspect of the *authoritative ledger* (*CBDC ledger*) that "accounts for" the to be accepted global state of the electronic asset. At this point, we treat the CBDC ledger – and its state – as a *logical* construct; allowing for scenarios where the CBDC ledger is to be understood as the composition of multiple different ledgers (not entirely unlike the way *federated databases* are created). That being said, the architecture we propose in this work will be based on a single, although decentralized, ledger.

- **Native ledger data model and transactions**: how the existence, ownership and other properties (e.g., authorization policies for various usages) of the electronic money are represented on the ledger as a "current state" and how past transactions are accounted for. (Technicalities, as whether those are stored and managed as a "chain of blocks", are not relevant at this level of abstraction.) Additionally, the types of transactions that can be initiated on the ledger state.

- **Extensibility through smart contracts**: to what extent can users of the ledger extend its data model and accepted transaction type set through the deployment of so-called smart contracts.

- **Ledger update management**: whether there are multiple parties involved in deciding on the order, acceptance and effect of incoming transactions targeting the current ledger state. If yes – i.e., the CBDC ledger is not, or parts of it are not *centralized* –, what is the consensus mechanism used. Ledger update management can also encompass an access management aspect – i.e., though the electronic money as a service is widely accessible, the general public may not directly access the system(s) actually maintaining the ledger.

- **Validation by the public**: *widely accessible electronic money* does not automatically translate to *widely validateable* electronic money – neither from the point of view of auditing whether a central bank is keeping itself to its stated monetary policies, nor from the point of view of checking the honest handling of one's electronic money.

In the following sections, we describe a number of key design options, identify the ones chosen for our purposes and provide our rationale. We build on the relatively recent design options overview of the Bank of International Settlements [3]. Our treatment also partially builds on [7], a research report which was made available during our project (and the philosophy of which reflects – in a very well written

way – some of the conclusions we were also converging towards at the time). That being said, our mini-taxonomy differs from both of these sources and emphasizes the system engineering aspects of creating a CBDC.

2.1 Ledger model

Ledger model: summary

Following the pattern set by major cryptocurrencies, pseudonymous CBDC ledgers, which maintain balances spendable by the owners of private cryptographic keys, promise to be a viable and robust technical approach, able to accomodate "account-based" as well as "token-based" CBDC approaches.

For pseudonymous ledgers, if required, proven and emerging privacy-preserving strategies are available from the cryptocurrency world; at the same time, KYC processes can create the knowledge necessary for linking pseudonyms and identities. This, in turn, facilitates the creation of such controlled processes as freezing funds, lawful seizure and the handling of lost keys.

Regardless of ledger programmability through smart contracts, a small core set of payment primitives ("transaction types") has been emerging for cryptocurrencies which can be treated as a template for the end-user usage of a (retail) CBDC. On the other hand, modelling efforts have synthesized the life cycle and behavioral facets of CBDC *assets* which most central banks seem to agree on.

Based on an overview of these results, we propose a simple, but already viable pseudonymous ledger data and transaction model for retail CBDCs. No assumption is made about ledger management and transaction processing, only that (regulated and explicitly authorized) financial institutions have the authority to ascertain the KYC status of our Ethereum-style pseudonyms on the ledger and that the central bank authorizes financial institutions to mint CBDC – up to central-bank managed allowances.

The available, relatively recent technological guidance from BIS on CBDCs emphasizes that the design decisions of any CBDC project shall be based on the envisioned role ("customer needs") of the the system and for retail CBDCs, defines three architectural patterns (see also Figure 1).

- **Indirect CBDCs** are (at least) two-tiered systems, where the "CBDC" is a claim on an intermediary; intermediaries handle retail payments and the central bank handles (or at least regulates and coordinates) wholesale payments.

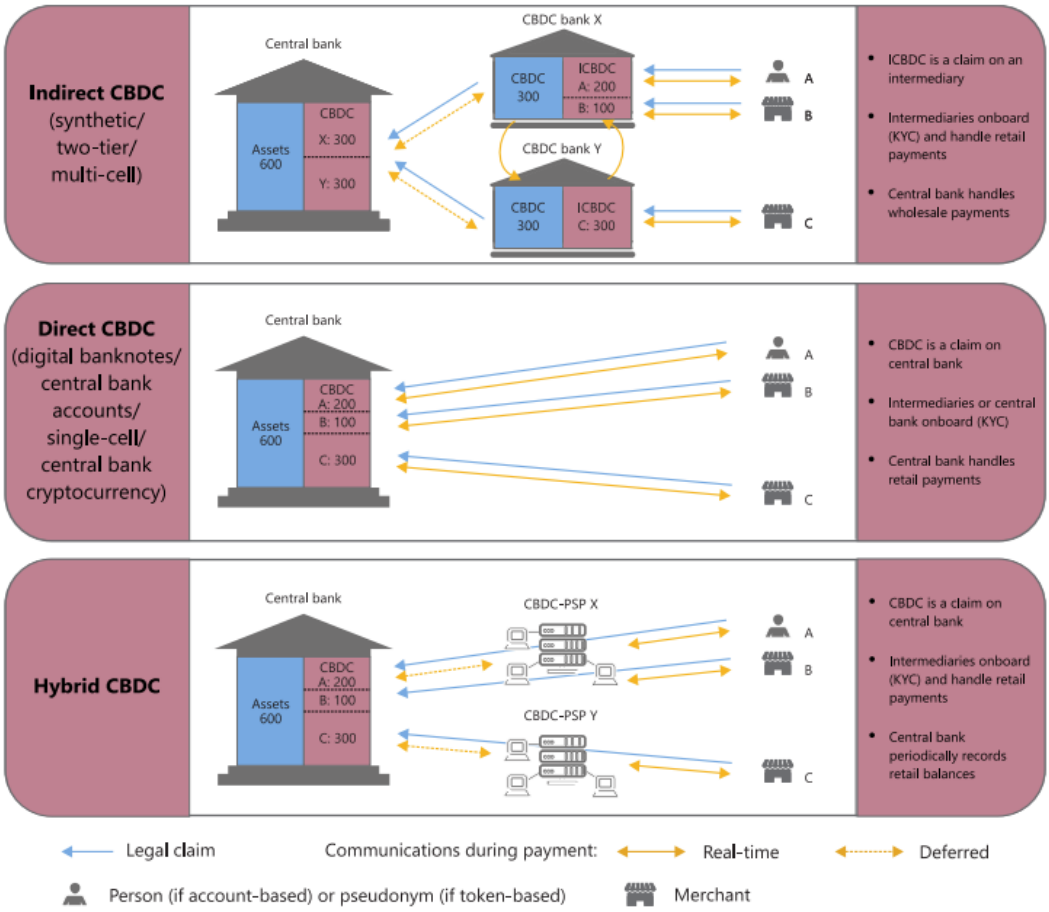

Figure 1: *"An overview of potential retail CBDC architectures"*, source: [3], p89

- In a **direct CBDC** architecture, the CBDC is a direct claim on the central bank and the central bank directly handles retail payments. The central bank may choose to offload KYC processes to commercial entities.
- In a **hybrid CBDC**, the asset is a direct claim on the central bank, but intermediaries handle KYC and have a role in the management of retail payments. The central bank may wish to only periodically record changes in retail balances.

It is important to note that these architectural patterns are high-level, *functional architectures*; they provide little to no guidance in terms of the structure of the managed ledger (or ledgers, in the case of indirect CBDCs). Later on, in the context of ledger updates and public verifiability, we will argue that as of now, hybrid architectures seem to be the technically viable choice which, at the same time, retain the true retail CBDC nature of the solution (it is debatable whether an indirect CBDC can be a "true" retail one, as it still relies on claims on intermediaries, even if the claims are matched 1:1 on the central bank ledger).

A further note – somewhat in advance – is in order with respect to the choice of wording of the BIS guidance. Expressions as the *"central bank handles retail payments"* are best understood as shorthands for the central bank *retaining appropriate control* over the given functionality.

In our view, a key component in the emergence of the CBDC idea is that the technological advances of the last 15 years made a whole range of responsiblity-sharing scenarios possible. At one end of the spectrum of possibilities, a central bank may opt to create a hierarchical infrastructure which is centralized at every level (a very conservative indirect CBDC); at the other end, it could (at least in theory) simply create a special stablecoin[6] on one of the large, public blockchain networks. In the first case, it does actually "handle" (wholesale) payments; in the second case, however, it essentially authorizes a network to do so, in terms of the money-handling rules captured in the smart contract underlying the stablecoin.

In case of a hypothetical stablecoin-CBDC, although the central bank retains at the very least its sovereign right to issue currency, the "handling" of transactions in the information technology sense is yielded to the network – operated by the public. Even if we put all questions of privacy, performance and actual security aside, it is doubtful that a central bank would – or *could* – relinquish its power to control its money through decentralization to that extent. At the same time, there are numerous intermediate options, as, e.g., decentralization of transaction processing among a number of trusted parties (governmental or non-governmental).

In the remainder of this subsection, we briefly review the applicable state of the art with respect to retail CBDC ledger modeling, draw conclusions and collect our recommendations.

2.1.1 The case for pseudonymization based accounting

Discussions on CBDCs tend to distinguish *token-based* and *account based* CBDCs. Tokens and accounts in this context tend to have their "normal" meanings – i.e., "bank account" like constructs versus fungible "e-cash" –, and not the ones that a reader well-versed in the cryptocurrency world may assume (i.e., Ethereum-like "accounts" versus smart contract based tokens or Bitcoin-style unspent transaction outputs).

The underlying core question here is, of course, that whether CBDC balances should be (relatively) easily attributable to persons and organizations, or the CBDC should function more like cash – anonymous, or potentially anonymous to a great extent.

[6] Informally, stablecoins are tokens on a blockchain, which represent the – transferrable – ownership of an asset with *stable* value. The schoolbook example is tokens representing units of some fiat currency, backed 1:1 through the currency kept in some form of reserve. For an in-depth introduction and classification, see, e.g., [14].

We believe that this dichotomy, while very valid from a classic point of view of how we account for money, has been superseded by technological development and arguably can be seen as a somewhat false one for CBDCs.

Practically all major, public-unpermissioned cryptocurrency networks – as Bitcoin and Ethereum – use *pseudonymization* today, and this is what we see as the most sensible choice for the upcoming first generation of CBDCs as the basic approach towards accounting.

A pseudonymization-based ledger, as we will see shortly, can support the whole gamut of options between fully identified accounts and anonymous units of electronic money – with pseudonymization, where a specific CBDC should lay on the spectrum is a question of central bank policy and regulatory environment. For instance, [19] proposes that citizens should enjoy robust privacy with protection even against government bodies being able to establish their CBDC holdings and transaction history, while firms should be subject to a high level of regulatory accountability.

2.1.2 Pseudonymization-based ledgers

In cryptocurrencies, the customary approach for handling identity and authorization is based on the end user creating her- or himself a *public-private crpytographic key pair*. From the public key, a so-called *address* is created via *hashing* — a seemingly unintelligible string of characters and numbers which serves as a pseudonym of the user. The user then can instruct other users to transfer funds to this address; when spending, she or he uses transactions requests signed by their private keys to prove that they own the funds and thus are authorized to request transfers to other addresses. A user can – and is generally advised to – use multiple pseudonyms.

Pseudonymization has multiple potential benefits in a CBDC context[7]. First, it decouples the core tasks of maintaining balances and identifying the owners of those balances. While an electronic money can be expected to be subject to various KYC, AML and other legal and regulatory requirements – i.e., it can't be expected to be fully cash-like for the ownership and transfer of nontrivial sums –, at the same time, a central bank may not wish to be able to directly "see" identity information on a CBDC ledger, not even if it is centralized and operated by it (and even much less so if the ledger is implemented in a distributed way). Pseudonymization can respond to such requirements – by storing and managing the relationship of addresses and actual identities off-ledger. Figure 2 demonstrates the concept of pseudonymous balance ledgers which use public-private key cryptography for transaction authorization.

[7]It is worth to note here that pseudonymization as a general mechanism is a fundamental concept in cybersecurity and, ideally, its goals, approaches, properties and risks would deserve a structured treatment, especially in light of its fundamental role in GDPR compliance. We refer the interested reader to [9].

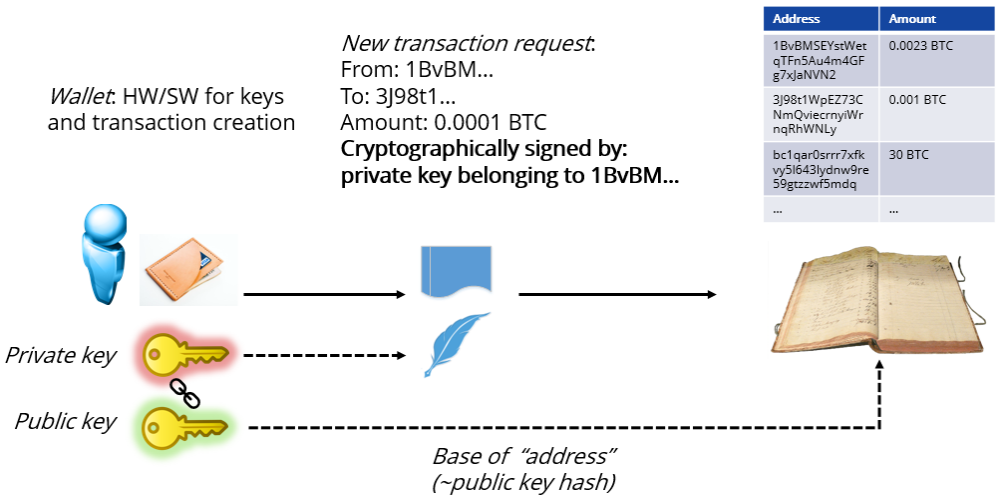

Figure 2: The basic model of pseudonymous ledgers relying on asymmetric (public key) cryptography.

2.1.3 Privacy and confidentiality of pseudonymized ledgers

It is important to note that a *pseudonymous* ledger of balances is not necessarily *anonymous*. Ledger history still reveals the time and monetary value of transactions between pseudonyms; thus, transaction graph pattern analysis may reveal the true identities behind the pseudonyms. To what extent this is an issue certainly depends on the extent the ledger content is distributed. For instance, for public-unpermissioned blockchains this can be a very valid threat and there are now commercial services specializing in transaction history analysis. If we assume a centrally managed CBDC ledger which is not fully revealed to the public, or one that is managed by a closed consortium of parties (using a blockchain-based distributed ledger technology), the issue is much more benign.

Still, it may continue to exist, as a central bank (or the operating parties) may want to avoid even the perception of being able link identities to transactions – that is, without first receiving authorization using the proper legal channels, which inevitably will have to exist for any truly impactful CBDC.

Luckily, as pseudonymization has been facing these privacy (and confidentiality) threats in the cryptocurrency world now for more than a decade, the privacy-preserving techniques invented for cryptocurrencies can be readily applied for CBDC ledgers using private-public key pair based pseudonyms, too.

At the most basic level, users can be encouraged to always use new pseudonyms (addresses) for each transaction (the requirement to perform KYC and AML for each new address can technically complicate matters somewhat, but is not real challenge). So-called transaction tumbling/mixing schemes can be also adopted which "mix together" transactions so that they effectively become untraceable by transaction

history analysis, although these techniques are largely associated with illicit activities in the cryptocurrency world [20].

Additionally, such privacy-preserving, pseudonymization based cryptocurrencies already exist (e.g., `Monero` and `ZCash`), which, to our current knowledge, effectively make the ledger content unintelligible for any unauthorized party by platform construction, without additional user measures, such as using a transaction mixer. (In the simplest case, for each transaction, the two sole "authorized parties" are the money sender and receiver).

The cryptography used by these platforms is fairly involved; e.g., ring signatures and various Zero-Knowledge Proof (ZKP) approaches. However, trust in the correctness of the mathematics and its application is robust and increasing.

It is also to note that the cryptography – especially ZKP-based approaches – is rapidly gaining support in smart contract technology (see, e.g., [10], or the Cairo language[8]). From the point of view of preserving privacy of asset handling in blockchains, this means that a smart contract deployed on a blockchain (on one supporting smart contracts, that is, but with the notable exception of Bitcoin this is largely a non-issue by today) can act as a "*privacy enclave*". In terms of pseudonymous addresses – and with some simplification – assets can be transferred to the address of the smart contract; clients make transfers between their addresses *within* the smart contract, using ZKPs to make the blockchain-recorded transactions unintelligible for third parties; and when they decide so, they can instruct the smart contract to transfer "out" their assets for normal handling.

In effect, the privacy and confidentiality supporting technologies created in the cryptocurrency world provide privacy design options and facilitate controlled privacy for CBDC ledgers, even if we have to assume as a worst case that the general public will be able to see the full ledger contents. This way, starting a CBDC project with a cryptocurrency-like, pseudonym-based balance handling approach also provides a viable *privacy evolution path*; even if a first design does not provide strong privacy and confidentiality, these can be realized later on, without a fundamental rethinking of the system design.

[8] `https://www.cairo-lang.org/`

2.1.4 The yet-unknown impact of Self Sovereign Identities

This way, in our view, a pseudonym (public-key derived "address") based balance[9] model is the most appropriate to propose for a general purpose, retail CBDC right now. As pseudonyms disconnect actual identity from technical identifiers by design, KYC and AML need off-ledger support; how we propose this to be solved will be described in the following sections.

That said, one area where further research and design thinking is critically necessary with respect to the core ledger model is *Self-Sovereign Identities* (SSI). In recent years, the state of the art in digital identity management has far surpassed the capabilities of classic Public Key Infrastructure (PKI). Blockchain-based solutions have been created to provide a registry of identity authorities, which can cryptographically issue such identities that can be used to prove identity properties without releasing further information.

The schoolbook example of such schemes is creating a proof that solely states that one's older than 18; this proof can be verified solely by access to the identity of the authority issuing the identity used for creating the proof. Such technologies are already implemented in mature code bases (see, e.g., `Hyperledger Indy`[10]), realized by blockchain networks (see, e.g., the Sovrin network[11]) and many key aspects are under standardization or have been already standardized.

How SSIs can – and whether shall – be integrated into CBDC solutions, and how they impact the core ledger model, is still a largely open question. On the other hand, SSIs and identity management approaches resembling SSIs can be expected to enter everyday life in Europe in the foreseeable future; importantly, this is one of the key use cases of the under-development *European Blockchain Services Infrastructure*[12].

2.1.5 Data and transaction models: native versus smart contract

At the time of this research, there is very little tangible – and openly accessible – prior art with respect to *specific* data models for a CBDC ledger. Matters are complicated by the fact that many proposals and ongoing activities rely on (blockchain-based) distributed ledger technology, which, as a rule, is programmable through *smart contracts*.

[9] As a more technical point of clarification, *balance* here is a *logical* notion. How the "amount of money spendable by the private key holder of a public address" is accounted for on the *technical* level has a number of possible approaches; famously, while the Ethereum technology uses actual "address balances", Bitcoin uses a so-called Unspent Transaction Output (UTXO) model, where essentially yet-unspent "packages" of currency are "labeled" with the address they belong to. However, at the very conceptual level, it is fair to say that in all cases, the ledger state, in effect, identifies the "balance" of an "address".

[10] `https://www.hyperledger.org/use/hyperledger-indy`

[11] `https://sovrin.org/`

[12] `https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/EBSI`

Ledgers which support smart contracts can certainly have a native ledger state, transaction and ledger transaction history model (for cryptocurrency networks, this is "the money ledger"). Usually, smart contracts are recorded on the ledger through *special* transactions – and after that, they can not only use the native transactions of the underlying ledger, but also utilize the ledger as a "custom database" to essentially create their own sub-ledger.

This mechanism is very prominent in the Ethereum technology (and its largest network deployment, the public Ethereum mainnet). Ethereum supports transferring its native unit of account – called "Ether" – between pseudonyms, but small, compiled programs – called "smart contracts" – can be also deployed to its addresses. These programs can be used to store and manage a custom `Ethereum address` $\leftrightarrow$ `currently owned amount` mapping for arbitrary assets over the Ethereum ledger, with the program defining the logic and authorization rules for creating, transferring, using and destroying units of the tracked asset. (Authentication is generally based on the same digital signature checking in the smart contracts as what is used for native transactions.)

This way, the (distributed) ledger technology can be used to account for arbitrary *tokens* – units of account having utility in some economic setting, ranging from securities through bus tickets to doctor's appointments. Additionally, not only can the tokens be bought and sold for the native unit of account (using well-established smart contract programming idioms), but themselves can be used as a money-like instrument in *other* smart contracts.

For instance, one can write a smart contract which remotely enables and disables physical locks (possibly, but not necessarily through a token-representing smart contract), while the payment for unlocking a room happens with a smart contract based token which represents fiat currency kept in reserve at a trustworthy and regularly audited financial institution.

Ledgers which are programmable through user-created and -deployed smart contracts originally appeared in the publicly accessible, unpermissioned consensus blockchain world[13]. However, the general concept is actually not tied to decentralized (including blockchain-based) systems[14].

Depending on the smart contract programming language, some smart contracts are more or less tied to a specific platform, as contracts written in the Solidity lan-

[13]While Ethereum was the milestone technology, even Bitcoin has some limited user-programmability, which can be used for creating a range of token and asset handling smart contracts.

[14]As it is widely known, what blockchains provide as "smart contract" programmability is actually a weak version of the original "smart contract" concept pioneered by the cryptographer Nick Szabo in the 90's. In the general case, they are neither contracts, nor smart and maybe have more in common with database stored procedures than actual contracts. Still, this is the terminology that has been widely adopted by industry as well as academia.

guage for the Ethereum platform[15]; other high-level languages, as notably DAML[16], have been designed to be able to run on multiple platforms, including classic, centralized databases. Blockchain smart contracts written in classic, general-purpose programming languages with some restrictions and using specific ledger interfaces constitute another category; these can be easily ported, e.g., to centralized databases (the key example here is the Hyperledger Fabric [2] blockchain framework, for which smart contracts can be created using the Java, JavaScript, Go and Python languages.) Last but not least, automatic creation of smart contract logic from models – as business process models and state charts – is also an emerging technique [13].

As a consequence, prior art with respect to specific CDBC ledger data models can be found in the following sources.

- The native "money" and "money handling" models of cryptocurrency platforms.
- Smart contracts created with the goal of modeling a CBDC. Note that these are important first and foremost from the conceptual point of view; what is available as a smart contract in a given domain, usually can be folded into another platform as a "native" capability, if need be.
- Legacy (i.e., existing) payment and settlement systems and their standards.

In this work, we omit the third category and review existing prior art for the first two categories.

2.1.6 Prior art: Bitcoin

Bitcoin was the first true – and in its amalgamation of some known design principles, quite revolutionary – "peer to peer electronic cash system". During the more than a decade since its inception, generally speaking, it also retained a narrow focus on electronic money handling. Consequently, although by the principle of issuance it maintains a ledger for a money-like asset which is exactly the *opposite* of a CDBC, it is worth to summarize its typical transaction types (through limited programmability, even Bitcoin provides space for user innovation) – it is a strong contemporary model of "handling money", unencumbered by any legacy from classic financial systems technology. Note that this is a bird's eye view – purposefully unencumbered by technical details as the specific transaction model.

In the Bitcoin network, the overwhelming majority of transactions are "money" transfers between public addresses acting as pseudonyms [6]. Some transactions

[15] Of course, with Ethereum there's always the choice to use the code on the world wide main net - or on a sector specific Ethereum network as the Energy Web Chain (`https://www.energyweb.org/technology/energy-web-chain/`), or on a bespoke consortial network which is closed to parties outside the consortium.

[16] `https://daml.com/`

simply record a small amount of data on the blockchain (without any monetary transfers); others use the terse, but powerful scriptability of Bitcoin to implement various "smart contracts" – mostly asset management schemes.

What is important to note is that the scriptability of transactions facilitates multiple-signature transactions as well as various conditional payment operations.

- Simple "pay to public key hash" transactions transfer money from a public address, by proving ownership through the ability of signing with the corresponding private key, to one or more different public addresses.
- Transfers from so-called multi-signature ("multisig") addresses require m signatures from a predetermined n. "Multisig" has many use cases; from "joint accounts" $(1 : n)$ through buyer-seller escrow with an arbitrator $(2 : 3)$ to majority spending decision use cases.

Conditional payments (i.e., conditions beyond the encumbrance to prove the ability to spend) introduce another, largely orthogonal, dimension. Arguably, the two most important such conditions are timelocks and hashlocks.

- "Timelocked" transfers: transfers, the funds of which become available for further spending only after some time has elapsed. The underlying funds may be locked until the lock time elapses, or spent beforehand, in which case the timelocked transaction becomes invalid.
- Hashlocked transfers: the transferred funds become available to the recipient for usage only after they prove that they know a secret input, the hash of which is included in the "hashlocked" transaction.

Both condition types have a number of straightforward and important use cases in using Bitcoin as electronic money. They are also instrumental in the construction of so-called Hashed Timelock Contracts[17] (HTLC) on the Bitcoin network. A HTLC specifies a payment, which the receiver has to cryptographically acknowledge, otherwise the transaction does not take place.

With HTLCs, schemes have emerged that enable the elimination of delivery without payment and payment without delivery risks in various asset transfer scenarios (at the expense of the HTLC-managing ledger locking funds for a predetermined time, even if the asset transfer does not take place in the end).

Notably, third party free *atomic swaps* between ledgers supporting HTLCs are possible and even commonplace today. (Suppose that Alice has some assets on ledger α; Bob has some assets on ledger β. An atomic swap either does not modify this state, or leads to a state where Bob becomes the owner of Alice's original assets on α and Alice becomes the owner of Bob's original assets on β.)

[17] `https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts`

HTLC-based atomic swaps, while pioneered by the cryptocurrency world, have wider ranging use cases. They provide a substrate for creating trustless and efficient, blockchain-backed peer to peer payment channels (as first demonstrated by the Bitcoin-backed Lightning network[18]). Secondly, HTLCs can provide robust means to implement cross border payments – potentially between two distinct CBDC DLTs. (This latter option was explored by the joint project "Stella" of the ECB and the Bank of Japan [12].

2.1.7 Prior art: CBDC token modeling

Smart-contract based tokens on the Ethereum network paved the way for financial innovation in "Decentralized Finance" (DeFi) [17]. Of the various token types, *stablecoins* – tokens representing fiat currency in reserve – can be actually approached as an *Ersatz* for central-bank issued CBDC on blockchains – they fill the gap of making "true" money available in a blockchain ecosystem. Their core problem is, of course, the same as with bank accounts (they can be actually interpreted in terms of narrow banking): they constitute a claim on a party other than a central bank. Tokens that "wrap" assets from another blockchain (e.g., Ethereum tokens representing Bitcoin, locked on the Bitcoin ledger specifically for "usage" on the Ethereum ledger) serve a similar purpose, although with cryptocurrency and not fiat backing: they create liquidity in the form of tokens.

The overabundance of token intents and capabilities led to various standardization efforts. The technical ones (as ERC-20[19]) are less important for our purposes; it is the Token Taxonomy Framework[20] (TTF) which bears here mentioning. Initially incubated by the Enterprise Ethereum Alliance (EEA), it is now under the auspices of the InterWork Alliance (IWA)[21]. The TTF models tokens with so-called *formulas*, which express behavioral facets (to be implemented by sets of interfaces on various platforms).

After a careful review of the then-available communications of major central banks and the literature, the eThaler project[22] in Hyperledger Labs created the following core TTF formula for CBDCs:

$$tF\{d, t, p, c, SC\} \tag{1}$$

The letters translate to the following behaviors:

- **tF**: a CBDC token should be fractional fungible

[18] `https://lightning.network/`
[19] `https://ethereum.org/en/developers/docs/standards/tokens/erc-20/`
[20] `https://github.com/InterWorkAlliance/TokenTaxonomyFramework`
[21] `https://interwork.org/`
[22] `https://github.com/hyperledger-labs/eThaler`

- **d**: it should be divisible
- **t**: it should be transferable
- **p**: it should be "pausable", meaning that it should be possible to "freeze" movements of the token on certain market conditions (i.e., primarily to stop a "run to CBDC")
- **SC**: supply control ensures that the token can be "minted" as well as "burned" only by a permitted authority (but at the same time, it also requires such an authority to exist).

Building on the results of the eThaler project, a proposal from otcDigital[23] suggests creating CBDCs as *dual, integrated ledgers*; one ledger handling the accounting of TTF-based CBDC tokens and one running their smart contract workflows. The authors suggest that the latter should be based on the standardized financial instrument workflow framework captured in the Common Domain Model (CDM)[24] of the International Swaps and Derivatives Association (ISDA). As the core life-cycle of a CBDC (especially a retail one) can be expected to be very simple (the key events are minting, transfers, freezes/unfreezes and burn), this seems to be a somewhat questionable approach for the *general* case; we expect CDM-based workflows to be a truly sensible choice for CBDCs which have to directly back the life cycle of sophisticated financial instruments as, e.g., swaps and options (for which the CDM also provides a standardized framework).

ConsenSys, a leading Ethereum solutions provider, is also entering the CBDC technology market[25]. What is publicly available from their approach does not delve into data modeling details (can be expected to be along the lines of the results of the eThaler project) and is more important from the architectural point of view – thus, we will shortly comment on it later.

2.1.8 Prior art: modeling CBDCs in DAML

Digital Asset **ref** has also created a Proof-of-Concept CBDC implementation in their now-open DAML smart contract language and a white paper[26] on "Principles for Technical Implementation". DAML is a high-level, domain specific smart contract programming language which, through so-called *drivers*, can be "hosted" on a range of different ledger technologies – from centralized databases through blockchain-based distributed ledgers to non-blockchain based ones. DAML is a *high level*, *domain specific* language in the following senses.

[23] `https://interwork.org/cbdc-central-bank-digital-currency-and-the-iwa-open-standards-token-taxonomy-framework/`

[24] `https://www.isda.org/category/infrastructure/common-domain-model/`

[25] See `https://pages.consensys.net/central-banks-and-the-future-of-digital-money`

[26] `https://blog.digitalasset.com/news/cbdc-principles-for-technical-implementation`

- **A high-level metaphor**: it has a nontrivial, domain specific core ledger metaphor – *contract instances* created from *contract templates*. Various parties noted on a contract instance can forward the predetermined life-cycle of a contract; to the point where a contract is "closed", with the potential side effect of one or more new contracts being created. The language also has a privacy model specifying and controlling the visibility properties of contract instances at the *abstract level* of DAML itself, based on the roles defined in contract templates.

- **Execution through adaptation layers**: for "executing" DAML on a ledger, a highly nontrivial "driver" technological layer is necessary. The price to pay for the ability to run DAML on various "backends" is certainly threefold:

  - not all backends are necessarily able to support all intended features of the language from a systemic point from view (visibility, for instance, being being critical),
  - any serious effort to validate and verify a DAML-based solution also has to involve the drivers themselves, increasing complexity very considerably, and
  - a DAML-based application may not be able to utilize core and distinguishing features of certain hosting ledger platforms (either due to conceptual mismatches or driver deficiencies).

- **Domain specific**: while many problems can be expressed more or less naturally through evolving graphs of contracts, DAML was originally created with sophisticated financial instruments in mind (the Common Domain Model of ISDA has actually received a DAML implementation[27]). Initial experiences of the authors of this report suggest that certain problem types (e.g., state machine like control problems, securing data stream applications and high-performance distributed ledger applications) are ill suited to a DAML-based implementation.

DAML is also accompanied by Canton[28], a cross-ledger interoperability protocol which is broadly similar to the widely known Corda platform[29] in the way that it facilitates information flows between ledgers on a need-to-know basis and uses a centralized "sequencer" service to establish a global ordering of transactions.

The CBDC prototype of Digital Asset presents an implementation for two key use cases: atomic, "cross border" currency exchange (payment versus payment setup and

[27] `https://medium.com/daml-driven/the-isda-cdm-much-more-than-just-a-standard-for-the-derivatives-lifecycle-71c367373743`

[28] `https://www.canton.io/`

[29] **??**

settlement) and the use of "earmarked" CBDC assets (paying rent with government stimulus). Relying on Canton, interoperability between different CBDC platforms is also showcased.

After a review of the publicly available DAML CBDC demo smart contracts, the impression of the authors is that DAML may be a good fit for formulating sophisticated smart contracts which *rely on the existence of a CBDC*, but not necessarily for the CBDC itself (note: as we will see later, and as we implicitly already assumed with the IWA proposal, the asset and the smart contracts utilizing it do not necessarily have to use the same authoritative ledger). As a matter of fact, in the demo contracts a CBDC is merely an "asset", which a Central Bank can mint. There is considerable uncertainty at this point that whether DAML is able to express the privacy controls actually needed by a CBDC asset (while meeting security and dependability requirements) in the practice, and its scaling to CBDC workloads.

2.1.9 Outstanding requirements

While retail CBDCs can be expected to have an "electronic cash"-like facet (up to a holding or transaction value threshold), from a legal point of view they can be expected to fall into a more regulated category than cash (e.g., in Hungarian law, *"elektronikus pénz"*, lit. *electronic money*). Consequently, practical retail CBDC implementations will have to not only support KYC and AML, but also facilitate the state exercising such controls as the freezing and court-ordered seizure of funds.

2.1.10 A simple CBDC ledger model

For our research prototype implementation, we created a simple CBDC ledger and transaction data model which does not rely on post-initialization programmability – through smart contracts or otherwise[30]. Figure 3 presents the data model.

Ethereum style accounts are at the core of the ledger model. These are "Ethereum style" in the sense that they have an associated Ethereum address and transaction authorization checks are assumed to be performed based on checking digital signatures of transaction requests (that is, whether the request was signed with the corresponding private key). Accounts have a CBDC balance and can be multisig, meaning that all outgoing transactions are required to have a necessary minimum number of signatures. Accounts can be also frozen – meaning that no financial transactions are permitted; and an amount can be also forcefully locked. Initiating the necessary state changes is tied to special roles in the system (in the implementation we will present, the central bank and/or authorized financial intermediaries).

[30]That being said, the transactions in our implementation are certainly implemented by smart contracts.

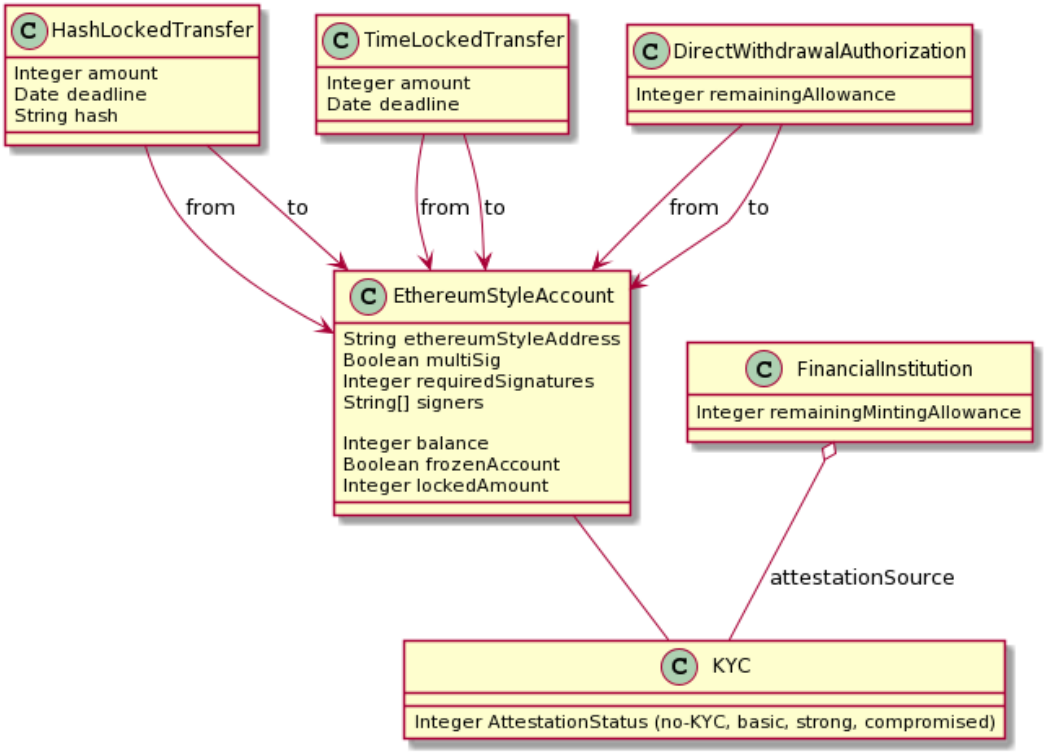

Figure 3: A simple ledger model for a pseudonymous CBDC

In our model, financial institutions appear as first class concepts. As a controlled form of CBDC distribution, they can be authorized by the central bank to "mint" CBDC to addresses, up to a minting allowance managed by the central bank. Also, they create KYC information to accompany the addresses; in our simplified model, KYC levels include the following.

1. No KYC performed for the account yet
2. Basic KYC performed (e.g., tied only to simple authentication schemes and mobile phone identifiers or emails)
3. Strong KYC performed (compliant with current banking requirements)
4. compromised: it has been revealed that the KYC earlier set for the account was erroneous

We will delve into the ledger-modifying and query transactions over this simple data model later, as it requires the discussion of our proposed architecture, too. Two further important points here are that those certainly include the basic transfer operation and that the implementations of all operations will include "policy hooks", enabling a pluggable approach to create central-bank defined *authorization* policies. Authorization policies include the maximum amounts transferrable at various KYC

levels and the question that who and under what conditions is authorized to lock funds on an account. Notice that the sensitive part of KYC data is *not* part of the ledger model.

The ledger model also contains support for hash locked and time locked transfers; upon these operations, the associated funds are effectively under an escrow by the ledger, thus, they don't form a part of any account balance temporarily. Hash-locked transfers have an associated deadline, after which the sender can revoke the transfer. In their current form, our hash-locked and time-locked transfers are not able to support the same kind of HTLCs as Bitcoin; as later discussed, we are focusing on different integration solutions. That being said, Bitcoin-style HTLC support for third-party free atomic swaps can be introduced in our model and implementation.

Last but not least, we support *direct withdrawals*: that is, accounts can be authorized to withdraw funds from other accounts up to an allowance controlled by the owner of the source address. The associated allowance management data is also part of the ledger.

2.2 Ledger update management

Architecture: hybrid CBDC with DLT core

We argue that currently, the most practical and risk-minimizing technical approach for a retail CBDC is a high-performance consortial DLT core with indirect (financial intermediary based) end user access. At the same time, the CBDC should remain a direct claim on the central bank; that is, a hybrid CBDC. We also describe our prototype implementation of this pattern.

By update management of a CBDC ledger, we mean the mechanisms through which the integrity of the ledger is maintained; that is, the processes for

1. initializing the ledger,
2. accepting of refusing incoming transactions for processing,
3. ordering incoming transactions (even if only partially),
4. determining whether the initiator of the transaction is authorized to request the transaction and other logical prerequisites and business constraints are met,
5. determining the ledger updates the acceptance of the transaction lead to and
6. modifying the accepted-as-current status of the ledger accordingly.

Clearly, these activities can be fulfilled even by a classic, centralized system, operated by a central bank. Mathematical techniques (primarily in the *authenticated data structures* and *zero-knowledge proof* domains) are also becoming available, which, by releasing a series of cryptographic "commitments" to the current state of the ledger, ensure that the maintainer of the database (in our case, a central bank) can perform neither retroactive nor "unathorized" (e.g., modifying the owners of funds without following the established and published procedures) modifications without these becoming evident.

Still, numerous – if not most – central banks are looking into Distributed Ledger Technology (DLT) based CBDC implementations[31]; with many DLTs being based on blockchain technology. In the following, when necessary, we will refer to CBDCs relying on a DLT technological basis as dCBDC.

2.2.1 Rationales for DLT platforms

Why would a central bank strive for creating a dCBDC? It is worth to look at the potential benefits, as they are rather specific to the CBDC application.

To prepare our argument, we have to point out that "distributed trust" with respect to ledger state maintenance – beside algorithmic currency issuance, the original core tenet of cryptocurrencies – is arguably *not* the cornerstone DLT capability for dCBDCs, at least not how we understand it in the context of public, unpermissioned blockchain networks.

It is easy to argue that a dCBDC does not need the financially incentivized "distributed trust" and "algorithmic money" issuance aspects of cryptocurrency blockchain networks. The second statement is obvious; with respect to the first statement, it is hard to imagine a scenario where a central bank truly wants to mirror cash with a CBDC to the extent that it does not retain at least the *option* of strong control over transactions of the asset. (Apart from legal requirements, an electronic asset with the untraceability and uncontrollability of cash and *without* its natural, physical burdens of movement would pose unacceptable illicit activity related as well as systemic risks.)

Theoretically, it does not deterministically follow from a requirement for strong control over the asset that the central bank should have a definitive say in the set and approach of parties jointly maintaining the ledger through some form of majority consensus (if there are actually multiple parties); but from a practical point of view, this *will* be a requirement[32]. On the other hand, if we opt for a so-called *permissioned*

[31]The site `https://cbdctracker.org/` provides a good global overview, with the ability for comparisons along multiple dimensions – including the "DLT nature" of the CBDC projects.

[32]Until recently, the majority "mining power" of the Bitcoin network – with simplification: the weight to influence the currently accepted distributed ledger state – was located in the People's Republic of China, meaning that no other sovereign state could – or should – even theoretically contemplate using bitcoin as "money".

DLT, where a controlled set of parties participate in ledger maintenance, then trust will be much less "distributed" – so won't the public be far less inclined to truly "trust" the CBDC solution?

This is actually a false argument. We should realize that it can't be the goal of a CBDC to eliminate the need to trust central banks – this follows, by definition, from the expression "Central Bank Digital Currency"[33]. We should, then, look for the DLT benefits in other areas.

Following a similar train of thought, various industries have already created *quality models* to find the various business value creating applications of DLTs. Maybe the most complete is the *Blockchain Value Framework*[34] of the World Economic Forum, which looks at "value drivers" – that is, established DLT use case categories as payments, process automation, "track and trace", records reconciliation, marketplace creation, ... – and identifies how these can lead to

- improving profitability and quality, or
- increasing transparency among parties, or
- reinventing products and processes.

In the WEF framework, DLT *capabilities* constitute the "glue" between use cases and these *key dimensions*. The WEF capabilities include the following[35]:

1. **Automation** – smart contract based automatic business rule execution.
2. **Full traceability** – as a rule, blockchain-based DLTs maintain a complete transaction journal.
3. **Speed and efficiency** – multiparty cooperations can become more efficient with a single, common authoritative ledger of the "common business" of the parties, especially when earlier necessary intermediaries are also eliminated during the process.
4. **Evidence (of) tampering**: attempts to modify historical transaction data in the journal can be made straightforward to detect.
5. **Distributed**: ledger data is stored and maintained in a highly replicated manner, with a majority consensus process applied to incoming transactions.
6. **Holistic view**: a DLT can act as an authoritative "single source of truth" to all stakeholders. This capability is especially powerful when multiple business

[33] As a brief aside, one can also argue that using traditional cash implies a level of trust in the central bank, too.

[34] `http://www3.weforum.org/docs/WEF_Building_Value_with_Blockchain.pdf`

[35] This list is slightly abridged as well as slightly modified – reflecting the view of the authors on the topic.

cooperation types are connected by a DLT (possibly in an incremental manner); e.g., financial institutions entering a supply chain management DLT with various trade finance offerings.

7. **DAx (Decentralized Autonomous x)**: by encoding sophisticated business rule sets in smart contracts in a transparent manner, autonomous organizations, products, services, ... can be created.

8. **Enhanced identity**: see earlier with respect to SSIs.

9. **Tokenization and digital assets**: digital representation, ownership management and transfer of physical assets. (For financial instruments, the capability is very widely known.)

Purpose-built quality models are appearing for other domains, too – as, for instance, the Internet of Things (IoT), robotics and security services.

A full, structured analysis of the potential DLT benefits for CBDCs is beyond the scope of this paper. It is evident, however, that point 5 in itself offers a very strong driver, even if it is *solely the central bank itself* who operates the nodes of the network. The dependability and security benefits stemming from the distributed nature of DLTs can be a very strong proposition even in a "single organization" setting, providing defenses against malicious (internal) modification attempts at a single node as well as a wide range of software, hardware and communication fault classes. Point 8 may also prove to be a strong secondary driver in the mid-term.

Additionally, for CBDC-based *applications*, essentially all remaining points can carry significant benefits. However, for applications a more serious analysis is necessary, as it can be quite context-dependent and situational that whether a DLT-based approach is the most beneficial (among others, the exact nature of the assumed underlying dCBDC has to be considered).

2.2.2 dCBDCs and DLT types

To create a wide-access (retail) CBDC implementation on a DLT basis, first and foremost a *DLT network type*[36] has to be selected.

- **Public-unpermissioned** distributed ledgers, such as the major cryptocurrency blockchains as Bitcoin and Ethereum, offer a mirage of true "distributed trust". However, "outsourcing" ledger maintenance almost fully to the maintainers of anonymous peer to peer systems has such risks which are not bearable for a central bank; there has to be a true, non-probabilistic guarantee in place that malicious actors (state or private) cannot gain the power to make majority

[36] One can argue that wholesale CBDCs are almost "by defintition" consortial – closed, permissioned – networks.

decisions about ledger updates and authoritative ledger content. Additionally, these networks currently lack the required performance and their cost to use is largely unpredictable. (See also our comments on stablecoin-CBDCs in the previous subsection.)

- **Public-permissioned** DLTs distribute the required trust in honest ledger maintenance among a set of known parties across whom a sufficiently large malicious clique is unlikely to form. At the same time, the network remains openly accessible for the general public. These types of networks are steadily emerging and usually address a specific sector and its challenges (e.g., the Sovrin network in identity management, the XRP Ledger[37] for cross-border payments, the Energy Web Chain in the energy industry). In theory, these networks can be scaled to the performance required by a CBDC; the core challenges of creating a public-permissioned dCBDC are
  - finding a sufficiently trustworthy set of peers for ledger maintenance;
  - *securing* the network sufficiently against all attacks (with an emphasis on smart contract vulnerabilities and common mode software faults across the machines in the network),
  - creating the appropriate level of privacy on the publicly accessible ledger – the applicable technologies are still rather new; and
  - if user-deployed smart contracts are supported, ensuring that they don't compromise the performance of the network and can't perform financial operations which are illegal or don't comply with financial regulations (as, e.g., unlicensed money lending).
- **Private-permissioned**, or *consortial* DLTs are maintained by a set of organizations (usually with a majority-vote like mechanism on transactions) and are accessible only to a controlled set of actors. In addition to the same (maintainer) *consortium forming problem* which arises in public-permissioned networks[38], a consortial CBDC in itself, by definition, is not widely accessible.

It's easy to see that all three DLT network types pose significant challenges for CBDC application. Our stance on choosing a network type can be summarized as follows.

- Direct CBDC issuance on a public-permissioned network or incentivizing a public-permissioned one with a CBDC is right now not a practically viable option.

[37] `https://xrpl.org/`

[38] Participants of public-unpermissioned networks are incentivized with the native cryptocurrency of the network; whether this would be a viable approach with a retail CBDC seems to be an open question.

- Public-permissioned networks are an option for retail CBDCs, possibly implementing a direct CBDC. (Notice that while the BIS architectures do "rhyme" with the DLT types, there's not a one to one correspondence by any means.) However, from a technological risk management point of view (chiefly: security and performance), a direct-access CBDC promises to be a rather risky proposition, even if the network is created by directly reusing the "battle hardened" software of large, established cryptocurrency networks.

- A core consortial network with a perimeter of authorized gateways is a viable and appropriate choice for a CBDC, with practically the only serious drawback – if we treat it as such – that the public, lacking direct access, can not directly verify the contents of the ledger.

2.2.3 A hybrid CBDC with a consortial DLT core

The architecture we propose and implemented in a research prototype implements a *CBDC ledger architecture* and *DLT system architecture* pattern which are structurally very similar to each other:

- (a form of) hybrid CBDCs, and

- a *consortial DLT network with gateway-based access for users.*

The architecture is demonstrated by Figure 4. This architecture bears strong similarities with [7], which was published mid-way of our prototype project and acted as a confirmation of our approach.

The core **CBDC DLT** is a Hyperledger Fabric network. Hyperledger Fabric (HLF) is a blockchain framework in the Hyperledger umbrella project (under the stewardship of the Linux Foundation), one of the leading platforms for creating consortial blockchain networks in the support of cross-organizational cooperation. Authentication and authorization in HLF networks is based on standard, enterprise PKI; however, as we will see, in our case this will involve only the central bank and financial institutions and not the retail CBDC end users.

The data model and transactions are implemented as HLF smart contracts – in HLF parlance, chaincode. One of the key features of HLF is that it supports a range of classic programming languages and not (just) blockchain specific ones, as the Solidity language, which targets primarily the Ethereum platform – in our current project, we used JavaScript (for which HFL supports the `Node.js` runtime).

Also, the performance capacity of a HLF network can be the subject of proper design for performance and very high throughputs and low latencies are achievable, without resorting to additional overlay techniques.

In this paper, we do not describe the specific ledger key-value model and the chaincode implementation in great technical detail; the API documentation of the

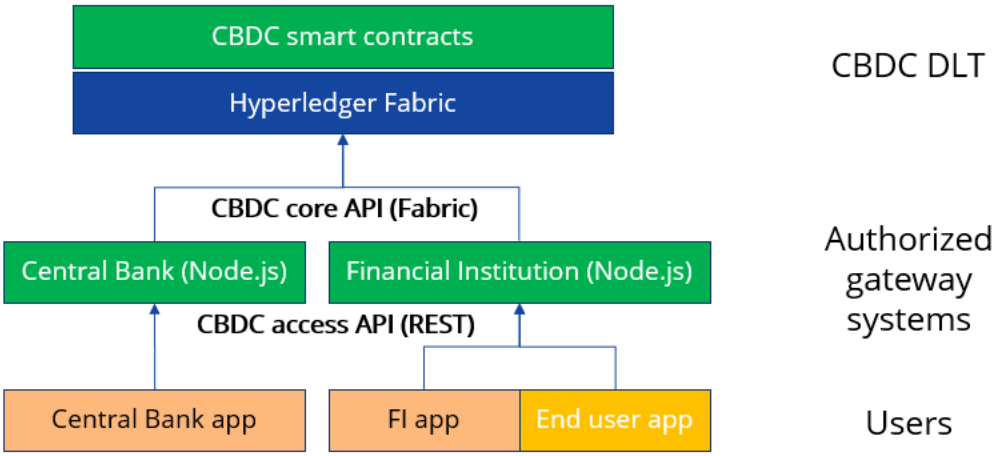

Figure 4: High-level architecture of the CBDC research demonstrator

ledger chaincode is available from the authors on request. The data model is an implementation of the logical CBDC ledger model on Figure 3.

To note is that although the CDBC core API was not designed to accept Ethereum transaction requests *per se*, transactions originating in the end user applications (e.g., wallets; see below) very closely resemble those by structure and content. Handling addresses, transaction content signing and signature checks are actually performed across the board by using the official Ethereum `web3.js` JavaScript API.

To prevent replay attacks, we also maintain a *nonce* for each pseudonymous account in a way very similar the Ethereum; each transaction originating "from" an account has to have a larger nonce field than what is recorded for the account. (And after a successful transaction, the nonce is increased by one for the originating account.) A difference from the Ethereum network is that the system expects transaction nonces to be exactly one higher than the recorded "number of transactions already performed from the originating account"; if the transaction nonce is more than one bigger, then the transaction is refused and not "put on hold" until the presumably missing transactions arrive.

Authorized financial institutions (FIs) provide a gateway/access layer to the core system. Their role is twofold: monetary as well as technical.

From the ***monetary point of view***, in our current model, financial institutions mint CBDC to end-users upon request, to the extent allowed to them by the central bank.

To this end, end users first register one or more address *through* an FI on the CBDC ledger; crucially, the end user keeps the private keys private and such a request is only complemented by the registering FI with KYC information. This gives room to a great deal of variability and different scenarios; incumbent banks will find it easy to provide KYC for their already existing customers, while other, potentially novel

intermediaries – "financial institutions" only in a very broad sense – can provide easy account registration with low KYC levels (e.g., based on mobile phone identification, which, among others, mobile operator companies can easily perform).

The end user having one or more registered accounts can initiate the minting of CBDC to those accounts. Our current model does not tie CBDC minting to the FI through which an address was registered; we envision a "multi-entrance" model, where, as a rule, indirect access to the CBDC system is provided by any FI for a small nominal fee (subject to free market mechanisms in pricing) – or for free, e.g., if the end user is otherwise a customer of the FI.

That being said, minting in the current model is envisioned predominantly as a *conversion* operation – directly from retail accounts or cash (e.g., for the clients of banks) or by "buying CBDC" through payment providers (which can be a compelling option for mobile operators if they wish to enter this service space). For such conversions, the party authorized and requested for minting can certainly require an account to be registered *through* them. The non-CBDC leg of the mechanisms associated with minting CBDCs – and its reverse, "burning", that is, conversion back to some other form of money – will be varied; the research prototype only goes so far as to enforce the central bank set allowances of FIs.

We would like to note here that direct governmental transfers (and earmarked or "colored", as well as "expiring" CBDC) are not ruled out in our current model by any means; a prototype implementation would be rather straightforward.

Also, from the point of view of societal inclusion in using digital money and digital money transfers, we would like to note here that schemes are possible where not even a mobile phone is necessary to participate in CBDC usage. At least in theory, the digital signing capabilities of modern electronic identity cards can be tied into a CBDC system and provide a form of government-supplied, "citizen default" access.

Non-government issued physical smart cards can also provide access through their cryptographic capabilities and pre-registered keys – albeit most probably with low KYC levels and thus serious restrictions on transfers. (Whether these cards can be used in an offline manner is another, orthogonal concern, outside the scope of our current research.)

From the ***technical point of view***, FIs act as access gateways; they "forward" the digitally signed requests of the end users to the core CBDC ledger (for actual CBDC transaction requests as transfers, hashlocks and timelocks; administrative matters, as for instance, registering addresses, are certainly handled differently). As the gateway role is provided by licensed and regulated organizations, their participation can be tied to enforcing basic security controls as a first line of defense, including rate-limiting the incoming transactions and combating denial of service attack attempts.

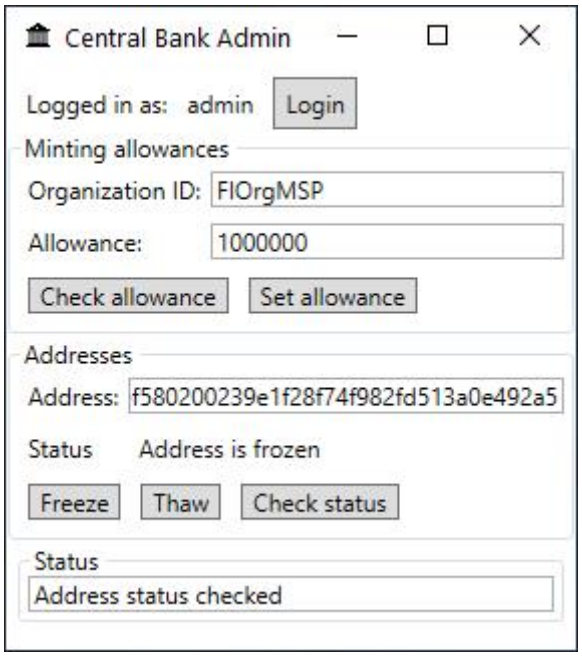

Figure 5: Central bank admin client prototype

We have implemented a prototype for the FI gateway functionality which supports end user access as well as FI administrative access. In a similar manner, we have prototyped a server for central bank operations. Both the central bank and FI gateways provide REST APIs downstream; the Swagger documentation is available from the authors upon request.

The **user layer** constitutes end user applications which communicate with the middle layer through REST API calls. Our current implementation relies on password-based authentication and constitutes desktop clients; however, due to the REST API based approach, both the security model and delivery model are easily adaptable. (E.g., certificate-based authorization and mobile clients.)

Figures 5 through 8 demonstrate user access. Note that the GUI capabilities closely match the upstream REST API interfaces, thus they also give a good, nontechnical overview of the operation which can be requested at that level.

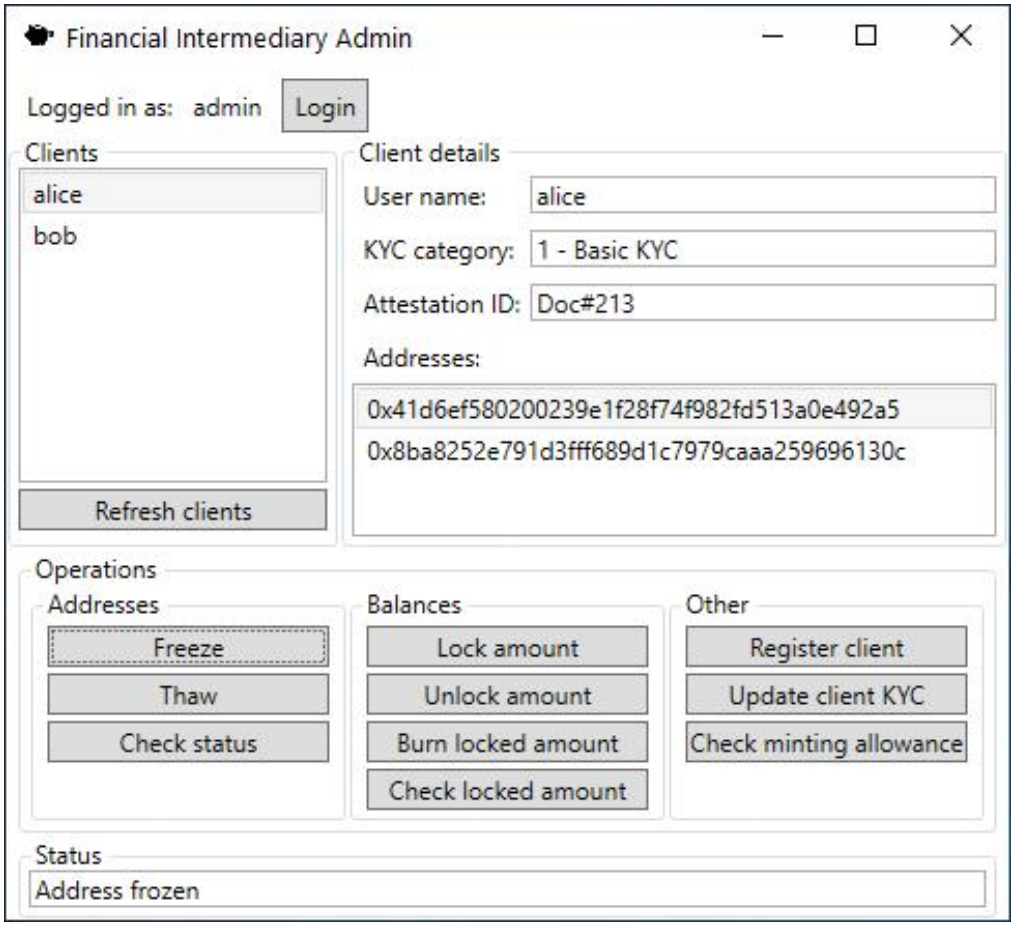

Figure 6: Financial institution client prototype

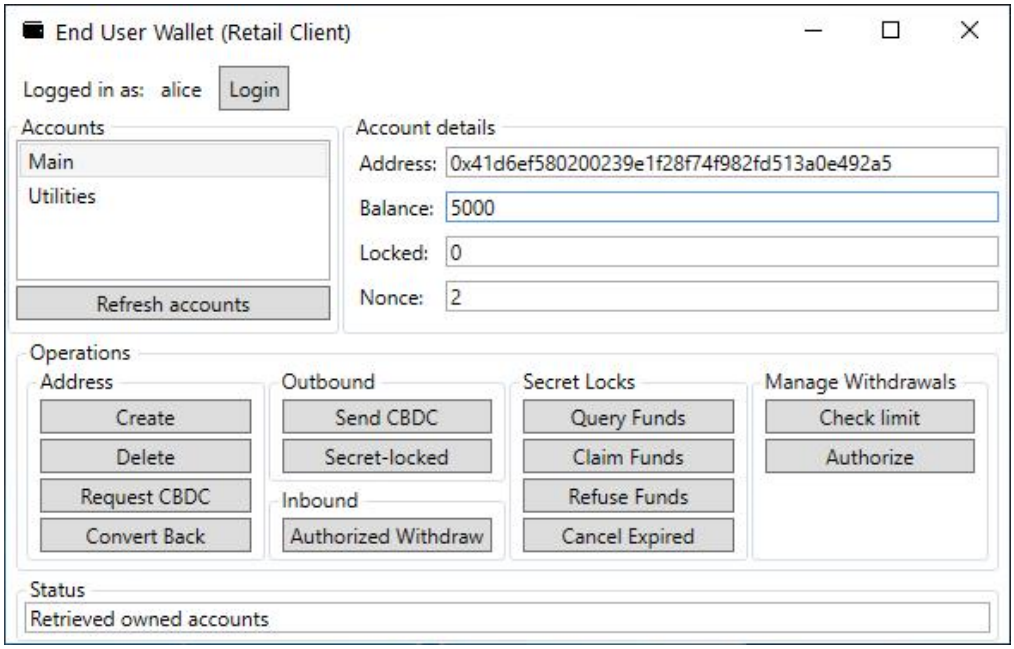

Figure 7: Retail end user wallet prototype

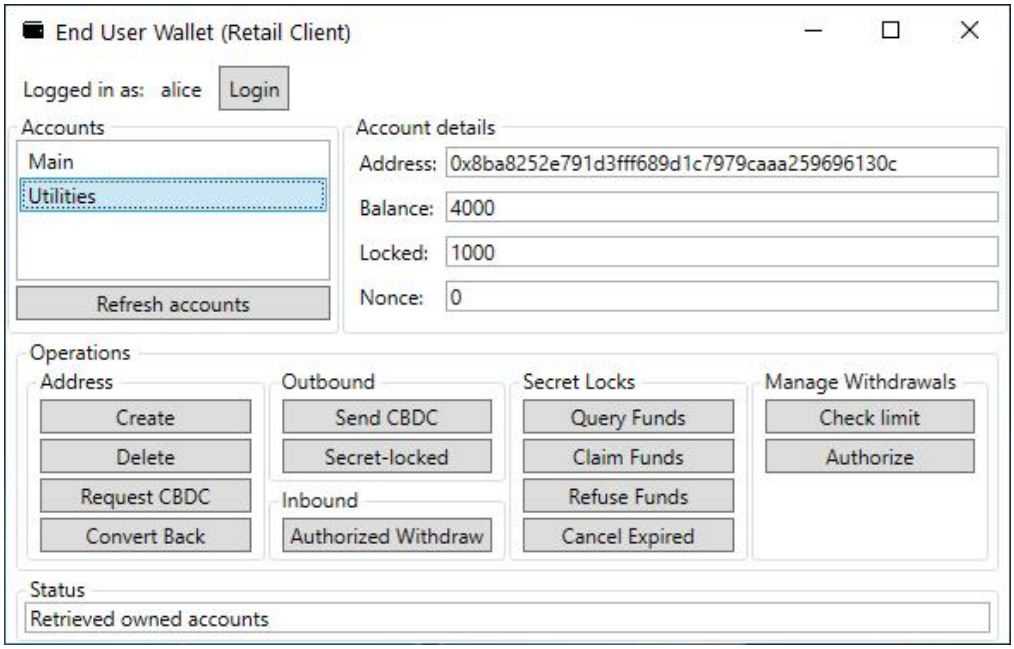

Figure 8: Retail end user wallet prototype – multiple account capability

2.3 Smart contract support

Bridging and other integrations instead of smart contracts

Wide access to smart contract deployment capability is inadvisable for a dCBDC due to dependability and security risks. Instead, CBDC liquidity for smart contracts can be realized with integration schemes, such as bridging. Our demonstrator includes a Hyperledger Cactus based centralized bridging solution (see in later sections). For regulatory control of financial activities on the foreign side, we created an initial customized consensus implementation for Hyperledger Burrow.

Up until this point, we did not discuss the extensibility of the system by user-provided smart contracts: whether holders of CBDC (or financial institutions) should be able to deploy smart contracts to the system.

In our view, the answer to this question is a very clear no; the functional scope of the dCBDC should be limited, and focus on the direct electronic money aspects of the CDBC it provides the bookkeeping for. Our argumentation starts from the premise that the *user-perceived* availability, reliability and security of a CBDC system is critical, if it is to supplement other forms of electronic money as the "most trusted" option.

As a consequence, neither the performance (and possibly availability), nor the (percieved) security implications of "user programmability" are acceptable.

On the performance side, the resource usage of smart contracts has to be controlled – after all, a DLT is only a network of computers with finite capacity. The Ethereum mainnet solves the finite capacity problem with a payment-for-execution-steps scheme, where the unit price changes with overall demand; however, this only *seems* to be a solution, as high utilization (with high prices) renders the network effectively unavailable for large numbers of users by at least temporarily "pricing them out". This is not a viable approach for a CBDC.

Consortial-permissioned networks, such as Hyperledger Fabric ones, do not have to use such payment schemes; for these, simple timeout mechanisms (e.g., a smart contract can run at most for 5 seconds) suffice. However, overall overload of the system still has to be avoided, meaning that at the very least, the set of smart contracts available for execution has to be under control. As a consequence, schemes as hosting well-screened and vetted smart contracts (e.g., to support government services) are possible, but general "user programmability" is not adivsable.

Security is another critical aspect, where the actual problem is not necessarily straightforward. Blockchain platforms functionally *isolate* smart contracts to a great

degree; software vulnerabilities in a smart contract should not be able to have any impact on other smart contracts (and the system as a whole).

However, these *technical* defenses are not enough from a *public perception* point of view. The infamous DAO attack on the Ethereum network was due to a smart contract vulnerability; the platform itself functioned as specified. Still, 60 million USD worth of Ether was stolen, prompting the majority of the community of Ethereum node operators to collectively roll back the state of the blockchain, reversing this fiasco which was deemed unbearable from a public perception point of view. Should we allow then the possibility of DAO-like attack on a dCBDC open?

The well-recognized and numerous benefits of CBDCs can be realized without broad smart contract support, too. At the same time, in a broader societal and economic context, providing limited functionality and "robbing" the end users of the option of customizability does put constraints on innovation and experimentation with novel financial, business and collaboration models.

The solution to this conundrum lies in *blockchain integration and interoperability* solutions. Many existing interoperability schemes of the cryptocurrency world have potential dCBDC applications which can extend the potential use cases of otherwise restricted-functionality CBDCs – albeit not without some additional risks.

As an example: the same way Bitcoin can be "locked down" on the public BTC ledger, subsequently issued and used on the Ethereum mainnet as a token, and finally released on the BTC ledger (while the token is also burned), a dCBDC may support creating CBDC-based liquidity on so-called "sidechain" distributed ledgers. Or as batches of Ethereum transactions can now be "rolled up" (ordered and executed) by a third party who may register only partial data about the transactions on the mainnet, a dCBDC may support rollup schemes to offload transaction execution as well as to limit the on-chain tracked data strictly to the digital currency aspects (emphatically not including data that the parties participating in the rollup scheme deem private or confidential).

However, what the terms "integration" and "interoperability" mean in general is becoming increasingly hard to describe with any level of sufficient technical specificity; new challenges of blockchain interoperability and approaches to address them have been appearing at a significant pace for years now. A reasonably general survey – and a taxonomy – is provided by [4]. A large (and quite representative) class of interoperability patterns is employed for scaling of Ethereum – and documented very well on `ethereum.org`[39]. For cross-organizational settings, the whitepaper of the Hyperledger Cactus project provides a good overview[40].

For our demonstrator, we implemented *centralized bridging*; authorized financial institutions can receive requests to "move" amounts of CBDC to another ledger.

[39] `https://ethereum.org/en/developers/docs/layer-2-scaling/`

[40] `https://github.com/hyperledger/cactus/blob/main/whitepaper/whitepaper.md`

Technically, this means that the funds of the requester become locked-down with the financial insitution, who instructs the "other (foreign) side" (in our case: a consortial Ethereum network) to create the equivalent funds for the requester. The foreign ledger can then use these funds in smart contracts; the bridge is responsible for "bringing back" the funds to the CBDC ledger upon request.

Broadly speaking, the other integration schemes – such as rollups – are variants of this theme; the CBDC ledger is used to "back" more sophisticated operations on one or more different ledgers (which are not necessarily distributed and not necessarily blockchain-based).

The conceptual problem with these integration approaches borrowed from the cryptocurrency world is that a CBDC will be a *controlled* asset; temporary release to a different ledger does not mean that a central bank is willing to lose all control (temporarily). Importantly, KYC and AML regulations are still to be adhered to.

The technical and conceptual solutions to this problem are still in early stages; in our work, we created a modified consensus algorithm for Hyperledger Burrow – a blockchain framework which is able to run Ethereum smart contracts – which allows central banks to set and monitor whitelisting/blacklisting rules without the modification of smart contracts. This development was not integrated into the final demonstrator.

3 Associating the physical word with smart contract elements

3.1 Smart contracts in industrial use cases

The benefits of using blockchain (BC) technologies for recording transactions in industrial use-cases outweighs the drawbacks of it. The speed, reliability, security, immutability and traceability, to name a few advantages provided, can be of great value for most companies. However, there could be obstacles in some special cases.

One of the basic principles of many blockchain scenarios is that the ledger is distributed, so there are no authorities, and every account is equal. These are great things, but in a business-to-business case or within some industrial setting, it might be needed to have some sort of authority over a group of accounts. Companies are is usually organized hierarchically, and consist of many divisions or even have subsidiaries. Since a division is a part of a company, there are certain dependencies – and a division is often not an autonomous entity, so it can't do whatever it wants. Logically, it doesn't make sense to give away accounts that are fully independent to every device the company wants to connect to the blockchain (or use its ledger).

Companies' acceptance of using blockchain technologies in their operations can only be improved if the problem of authority over their own accounts is solved, or if there is a viable alternative that overcomes the related obstacles.

3.2 Individual and shared accounts for company-bound devices

Suppose there are multiple companies that are working together, or are part of a supply chain, for example, the manufacturer, suppliers, shipping companies etc. They are considering using blockchain to track where each part of the product or the final product itself is in the supply chain.

Let's look at an archetypical shipping company. They might have devices such as cranes, chain-hoists, robotic arms or conveyor belts in their parcel center, where they want to keep track of the locations of the assets (in this case: packages). Tracking can simple mean that we know for each given point of time: which robotic arm has the asset or which conveyor belt is carrying it.

The abstract model for tracking "ownership" of assets in this setting is that when the device takes or handles a given asset, it takes the "ownership" for it as well. The device does not only create a record in an event log about this, but at the same time "pays" for the ownership, as well.

As an example: if a robotic arm picks up a package, it sends a transaction to the blockchain that it has the package, and "pays" a token at the same time (it might be the case that a robotic arm can only handle a predetermined number of packages a

day). Since these devices are in the same company (or the same division), they have to be grouped together on blockchain, and establish some form of hierarchy.

3.3 Issues with using individual accounts

A possible solution could be that every device has an own account on the given blockchain, with an own balance. This way it is obvious where an asset is and which device handles it, since the device becomes the 'owner' of the asset. The problem with this approach is that it is not possible to deauthorize a device to handle assets, because every account (i.e., *externally owned account* on the Ethereum blockchain) has the same rights, and is independent of the other accounts. Another problem is that anybody on the blockchain can see the exact location of every asset (which device handles it). For example, the manufacturer doesn't need to know the exact location of an asset in the supplier's factory (even if it is just an account address), it only needs to know which member of the supply chain has the particular asset.

3.4 Issues with using shared accounts

The other approach could be that every device use the same account. It solves the problem of individual locations, but introduces many more. Obviously, an outside party could not tell the exact location, but nobody else either. The company would not be able to tell where an asset is without individual identifiers, such as wallet addresses. The other problem is even worse: since every device uses the same address, every one of them knows the private keys of the account, so they could send any transaction they wanted to, which is unacceptable and could cause serious losses.

3.5 Companies represented through smart contracts

On many BC platforms (i.e., including the Ethereum blockchain), it is possible to deploy smart contracts, which is a huge advantage compared to other blockchains without this feature. Using smart contracts, it is possible to solve the aforementioned problems.

One possible solution is the following. Every company is represented as a smart contract on the blockchain, where it has an address, just like the externally owned accounts, making the use of smart contracts invisible to the outside world. Inside the smart contract, there are two key-value lists: the first one (i.e., *authorized*) tells if an address is part of the company and therefore allowed to initiate transaction on behalf of the company. The other one (i.e., *allowance*) stores the maximum amount each address is allowed to spend from the company's balance. The combination of the two lists makes it possible to easily manage the individual limits and rights of every device of the company.

Using this approach, every device has an own, unique address, so it is known exactly where an asset is, and it is possible to keep a history of previous transactions of each device.

The company's smart contract has to be deployed by an administrator of the company, who will authorize and manage the allowances of the accounts that are in the same logical group that the smart contract represents (e.g., a floor, office, division, the whole company).

In this initial example, the group is a company, but it can be any subset or superset as well. Initially, every individual address is excluded from the list of authorized addresses, so none of the accounts can initiate transactions on behalf of the company. The authorized addresses are added by the administrator, and by default, their allowance is 0. Allowances also have to be defined explicitly for each address to ensure that every account has the sufficient allowance and to prevent overspending. The administrator can also deauthorize addresses. In this case, the address will no longer be able to initiate transactions in the name of the company, and at the same time, the allowance of the address will be set to 0 to prevent inconsistencies in the state of the contract.

The transaction steps on how an asset's ownership gets exchanged in our model is depicted by Figure 9. The blockchain stores various *Asset Contracts* and *Token Contracts*. The Asset Contract stores the ownership information regarding the given asset – in this case, bound to the Company address. The Token Contract stores information of the token balances of the individual wallets as well as the balances of company contracts, and transfers the tokens between them. Besides, as the Figure shows, the company takes care of authorising and keeping track of the allowances of their "devices" (that are represented as wallets).

Those devices that are part of a company, instead of calling the main contract (that handles transactions, assets etc), have to call the contract of the company. The individual devices do not have separate balances, instead they can spend from the company's balance, as long as their allowance is enough to pay for the asset. If the caller address is authorized in the company, and the total cost of the transaction is less than the allowance of the caller, the company contract will handle the transaction and buy the asset. In this case, the bought asset will be owned by the company contract, so the outside world will only know which company has the asset, but not the exact device inside the company. The company contract will record the transaction and the address of the device that initiated the transaction. After the transaction, the buyer device's allowance will be lowered by the amount of the transaction cost.

Representing companies as smart contracts has multiple benefits, such as:

- Authorization management

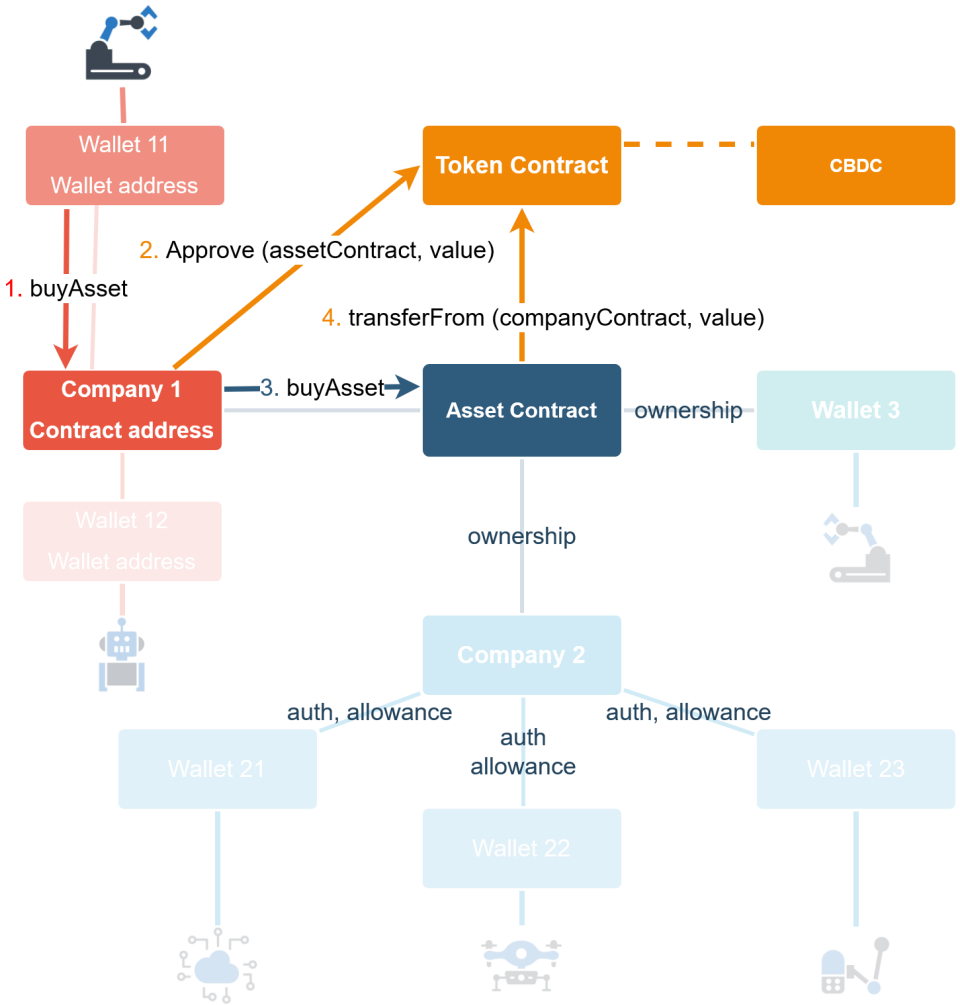

Figure 9: Steps of a transaction initiated by a member of a company

- Set spending limits to prevent overspending
- Realistic ownership (an asset is owned by a company, not by a device)
- Hides the exact address from the outside world
- Assets can still be tracked inside the company

This approach also allows that a device can be a part of multiple companies at same time, and can initiate transactions on their own, not in the company's name while being registered as a member of one or more companies. This is safe and does not present any threat to the company's balance, because the individual accounts don't have tokens (that belong to the company) tied to their addresses, so it is not possible that an address uses the company's funds and becomes the owner of the asset instead of the company. Besides this, accounts can have own funds that they

can spend however they want. This is completely independent of their allowances at specific companies.

4 Demonstrator environment

4.1 General overview of the multi-level CBDC demonstrator

The abstract representation of the multi-level architecture for the demonstrator is depicted by Figure 10. Let us briefly describe the demonstrator through this illustration, from bottom to top.

On the physical level there are devices that handle – create, move, transfer, receive – assets. The devices in our demonstrator are various robotic arms and conveyor belts, whereas the asset is a little rubber duck.

This physical level has one speciality that is not necessary but advisable: it is created as a service oriented architecture. In here, systems are providing and/or consuming "services", where a service can be information exchange – or even exchange of physical assets. Systems are loosely coupled, the connection is created through late binding, and the service consumers find producers through discovery or lookup. This service oriented architecture in our demonstration is provided through the Arrowhead Framework. Figure 10 shows the loosely coupled systems with green colour.

The next level is the Smart contracts associated with the interaction between companies and their devices. The ownership and allowances are exchanged here; the underlying ideas are described in Chapter 3. This is implemented on a private Ethereum blockchain network, marked as blue in Figure 10.

The CBDC smart contract, where the monetary transactions happen with the involvement of digital currency, resides at the highest level of the demonstration architecture. This is implemented on Hyperlegder Fabric, as the hosting blockchain network – marked as blue on the top of Figure 10.

The two smart contract infrastructures can be completely independent from each other. This gives great flexibility for the (industrial) companies and their ecosystems (on how they recording their transactions) as well as to financial institutions using traditional or digital currencies. As part of the demonstrator we have implemented a bridge between two widely used platforms, Ethereum and the Hyperledger Fabric. This is implemented through Hyperledger Cactus, and marked as an orange bridge in Figure 10.

4.2 Getting physical: devices and their transactions

During the demonstration there are several companies that provide services to each other. This is demonstrated by "creating" a rubber duck (a.k.a the asset) by one company's robot, "transferring" it to another location by a conveyor belt of a different company, then "receiving" by a robotic arm of a third company.

At each step the devices get ownership of the asset in the smart contract, but they transfer the fee in return to the previous owner or service provider.

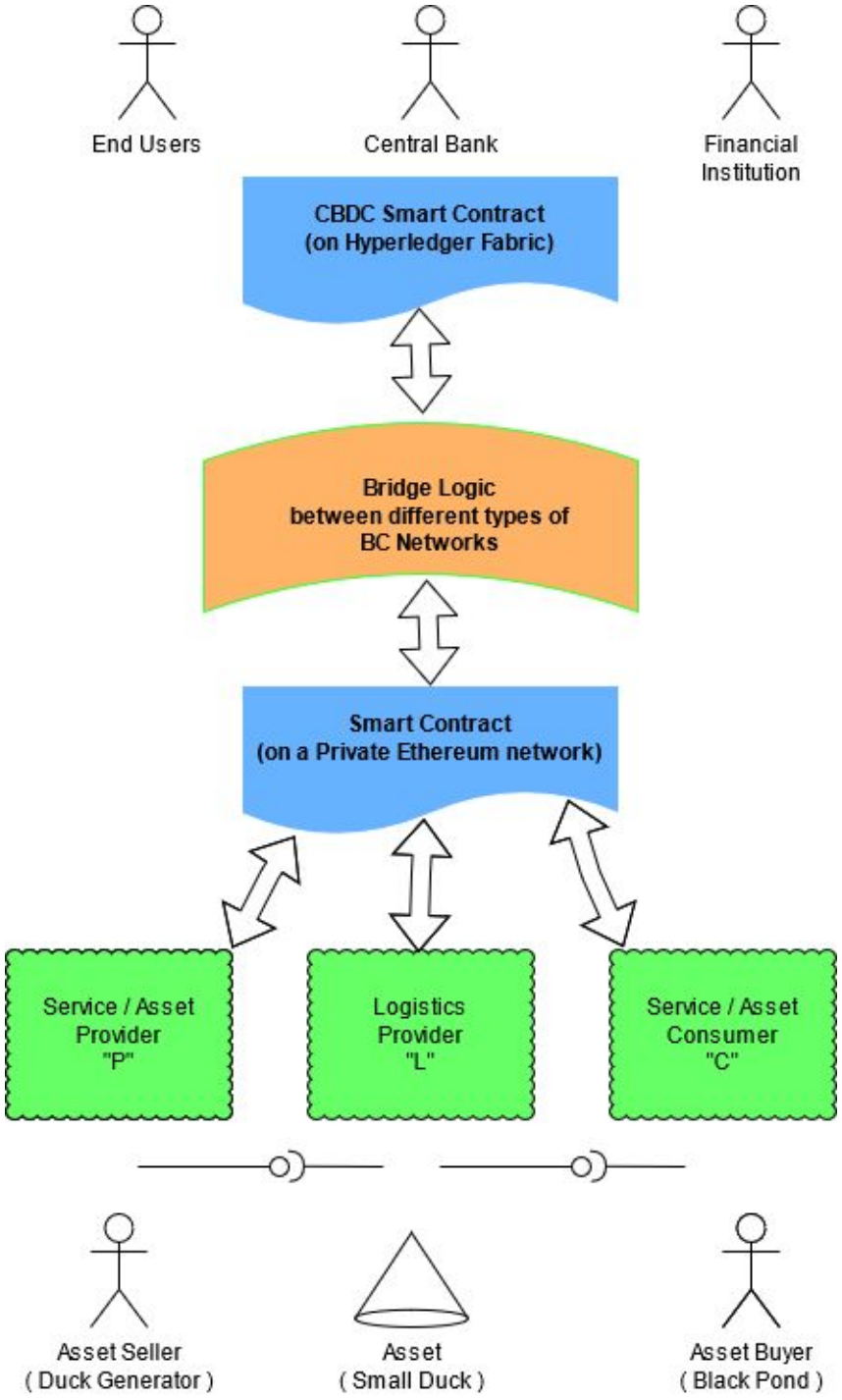

Figure 10: The simplified view of the demonstrator, where the CBDC and the smart contract are implemented on two independent BC technology sets. Notice the flexibility of the architecture: the physical (industrial) entities are independent from the smart-contract's BC, which is independent from the CDBC's BC. All are communicating through APIs that are standard but also secure.

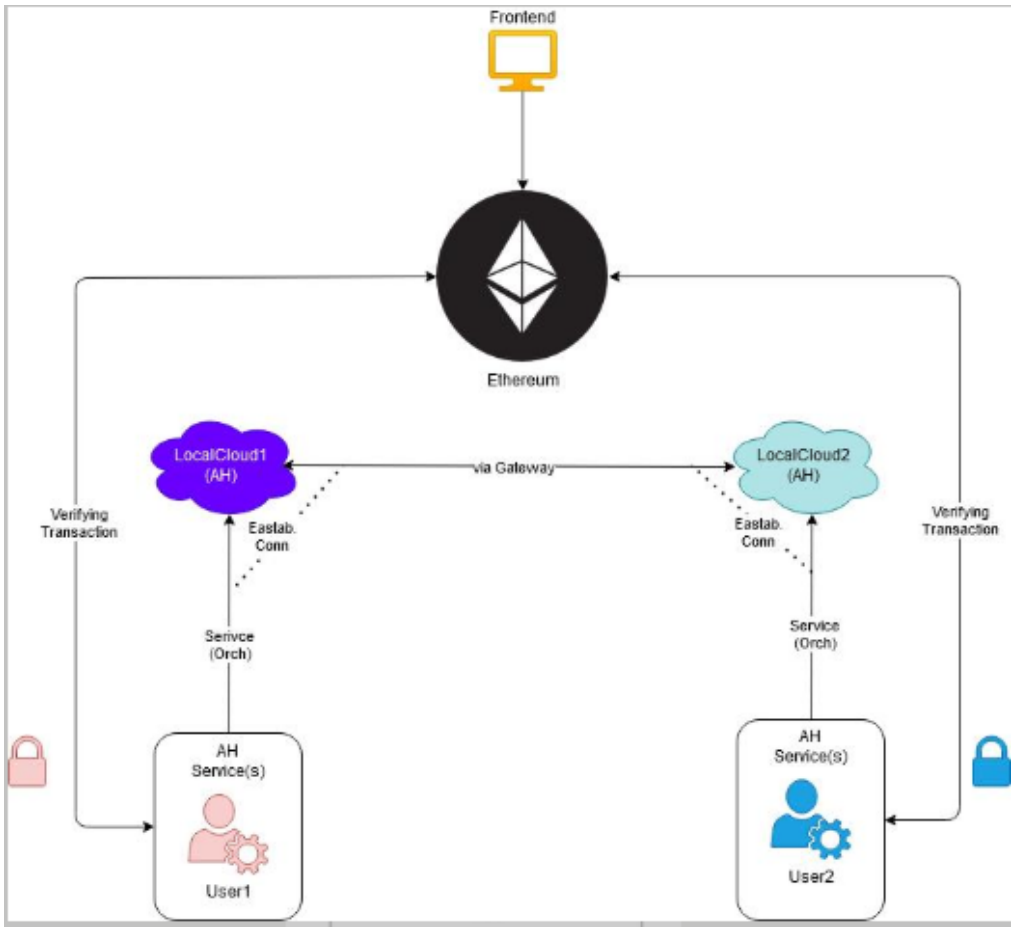

Figure 11: Physical entities at different companies access the same Smart contract. Using Arrowhead Local Clouds provide dynamic management of physical resources (sensors, actuators) and data security at the partner companies.

During the demonstration we see (i.e. through the monitoring GUI shown in the Appendix, Figure 21) as the ownership of the asset gets transferred to company A, B and C, respectively, as the demo scenario goes along.

Within each company there could be several systems or devices that are able to provide the given service (creating the duck, transferring it, receiving it). It is the task of the underlying Arrowhead Framework local cloud at each company to connect the service provider systems with the service consumers. The service can be provided in-between these Arrowhead local clouds (which can be associated with the different companies), where one company does not have to know or care about which given device provided the service at the other company. Once the transaction is successful, it does not matter, between which loosely coupled devices made it happen, the execution of the asset and token smart contracts (Ethereum, in this case) follows up at a company level.

This architecture is represented by Figure 11.

4.3 Smart contracts for the physical transactions: Asset and Token

These contracts are implemented in the Ethereum level that are related to the companies, their devices, and corresponding transactions.

As described through Figure 9, the asset contract and the token contract are two important terms in the above described, company-bound smart contracting model.

For managing and keeping track of availability, amount, and ownership of different types of assets, there is an **asset contract**. This contract stores every asset type and their parameters, such as the address of the owner, the available quantity, and the unit price. Anybody can add a new type of asset, view and buy existing assets, and manage their own assets. An owner can increase or decrease the quantity available for purchase, and update the unit price of a given asset that they have.

Anything that has to do with the assets happens on the asset contract, but managing payments is the responsibility of the **token contract**. The token contract keeps track of the balances of the individual accounts, verifies transactions, and transfers funds between accounts.

The asset contract is linked to the token contract, so every time someone buys an asset, the token contract needs to be called, and is also called by the asset contract. It is similar to when someone wants to sell their house, they ask a real estate agent to sell the house on their behalf. The agent is not the owner of the house, but they are authorized by the seller to sell it.

The reference implementation of the smart contracts are briefly described in the Appendix I as Chapter B. The Asset monitoring GUI is briefly described in Appendix II as Chapter C.

4.4 Supplying CBDC to the smart contract platform

In the demonstrator, the platform running the smart contracts is supplied with CBDC through a mechanism which the cryptocurrency world calls *bridging*.

Informally, "bridging out" an asset from a ("home") ledger to another ("foreign") one involves the following key steps:

- A user initiates a special transaction on the home ledger, locking the funds to be bridged.
- The party responsible for the bridging operation (or multiple parties, as decentralized solutions exist, too) is notified about the bridging intent; as a consequence, it takes ownerhip of the to-be bridged assets and instructs the foreign ledger to create tokens representing the assets locked on the home side. These new tokens are typically put under the ownership of the initiating party.
- The tokens representing the bridged funds on the home side can be used in transactions in the foreign network.

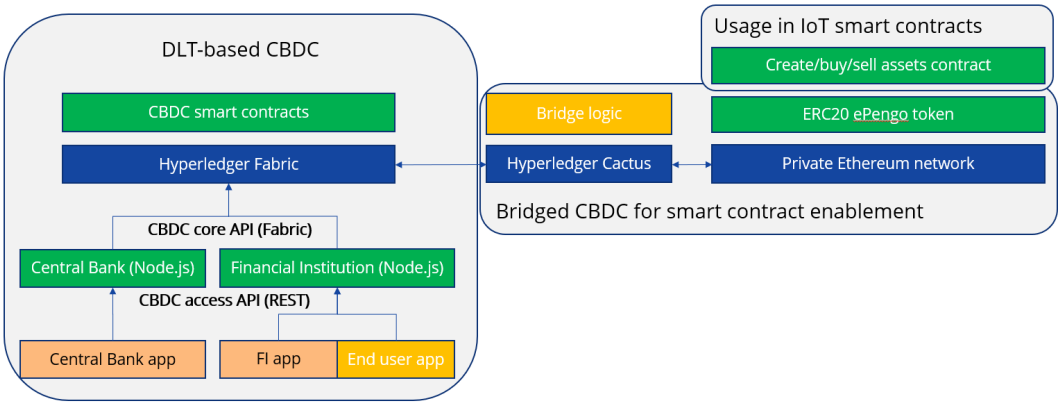

Figure 12: Bridging CBDC to the industrial smart contract platform.

- When an owner of bridged-out tokens wants to receive the equivalent CBDC on the home ledger instead, he or she "burns" the tokens via the smart contract handling them.
- The bridging party/parties get notified and release CBDC on the home side to the appropriate account.

To implement bridging, we used the Hyperledger Cactus project in the demonstrator. Figure 12 depicts the architectural setup.

5 Demonstrator scenario descriptions

Figure 13 shows the simplified dialogue sequence for the demonstrator from the physical (Arrowhead-bound) devices through the Ethereum smart contract to the Hyperledger-based CBDC.

Besides, the following subsections briefly describe the scenario steps that applies for this diagram, moreover, the scenarios described in Figure 9.

5.1 Adding new assets

A user can add a new asset that they want to sell by calling the *createAsset* function. This function takes two parameters: the first one is the unit price, that specifies how much one unit of the new asset costs, and the second one is the quantity of the new asset that is up for sale. The new asset will have a unique ID, that identifies the asset, and the asset will be tied to the caller of the function (the seller of the asset).

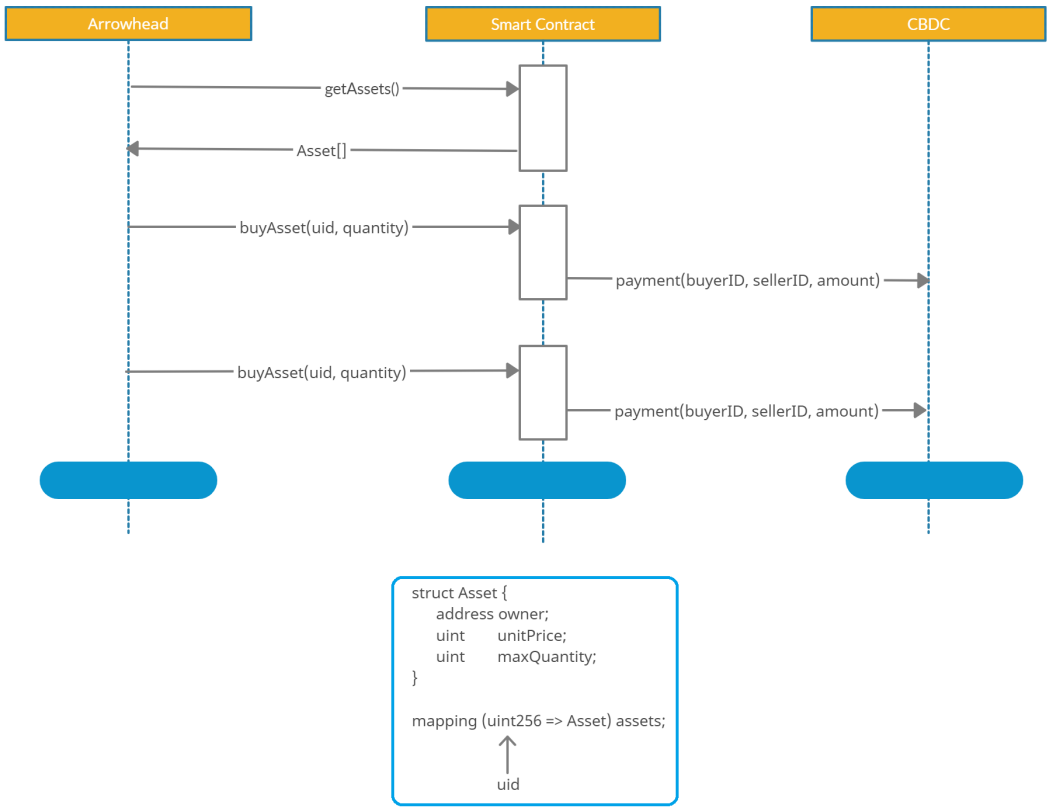

Figure 13: The simplified dialogue sequence chart of the demonstrator.

5.2 Buying an asset

5.2.1 Before calling the *buyAsset* function

By default, calling only the *buyAsset* function of the asset contract will result in reverting the transaction because the asset contract is not allowed to transfer funds on behalf of the buyer. To prevent this, before buying a specific asset by calling the *buyAsset* function, the buyer must call the *approve* function of the token contract first that authorizes the asset contract to initiate the transfer of the funds from the account of the buyer to the account of the seller. The allowance is passed as a parameter of the *approve* function as well as the address of the asset contract. After calling this function, the asset contract will be authorized to spend money from the buyer's account (but not more than the allowance), so it is recommended to call the *approve* function every time before buying an asset with the exact amount given as the parameter.

5.2.2 Buying the asset

To buy an asset, a user has to call the *buyAsset* function with two parameters: 1) the ID of the asset, 2) the number of units he/she wants to buy. If the requested amount exceeds the available amount of the given asset, the transaction will be reverted. This function will call the *transferFrom* function of the token contract, which transfers the total amount of money that needs to be paid to the seller for the assets. If the buyer has authorized the asset contract to transfer the necessary amount from their account before trying to buy the asset, and they have sufficient funds, the transaction will be successful, and the buyer becomes the new owner of the asset(s).

5.3 Increasing or decreasing quantity

It is necessary for an owner to be able to update the available quantity of an asset. It might be needed, for example, if there is a restock or some units become damaged and are no longer in a condition to be up for sale. In this case, the owner of an asset can call the *addAsset* or the *removeAsset* function to increase or decrease the quantity, respectively. Both functions take to parameters: the ID of the asset, and the difference between the old and the new quantities. In case of the *removeAsset* function, if the quantity to be subtracted is larger than the current quantity, the transaction will be reverted and the existing quantity will not be changed. The quantity of a given asset can only be change by its owner.

5.4 Updating the unit price of an asset

Another useful feature is the ability to change the unit price of an asset. One thing to keep in mind is that it is not possible to differentiate between units of the same

asset. Changing the unit price results in every unit costing the new amount of money thereafter. If it is needed to have a separate batch of an asset that costs more or less than the others, a new asset has to be created with the new price and the quantity of the batch.

6 Summary

In this work, we have reported on the design and prototyping of

- a Hyperledger Fabric based retail CBDC system,
- an industrial smart contract application,
- the Hyperledger Fabric based issuance of CBDC to the blockchain supporting the industrial cooperation
- and the usage of the so created funds in the industrial cooperation smart contracts.

To the best of the knowledge of the authors, these initial results are at the forefront of CBDC and CBDC application R&D; hoever, at the same time, they are not final and definitive – rather, should be treated enablers of further research.

After this initial demonstration, the most important avenues of research also seem to be clear: on the one hand, the impact of CBDC availability on smart contract use cases should be assessed in a structured manner and on the other hand, the pros and cons of the various blockchain integration approaches – of which bridging is just one – should be identified.

A Key demoed workflows of the CBDC component

The following figures document the internal interaction flows of the key demo scenarios of the CBDC-demonstrator.

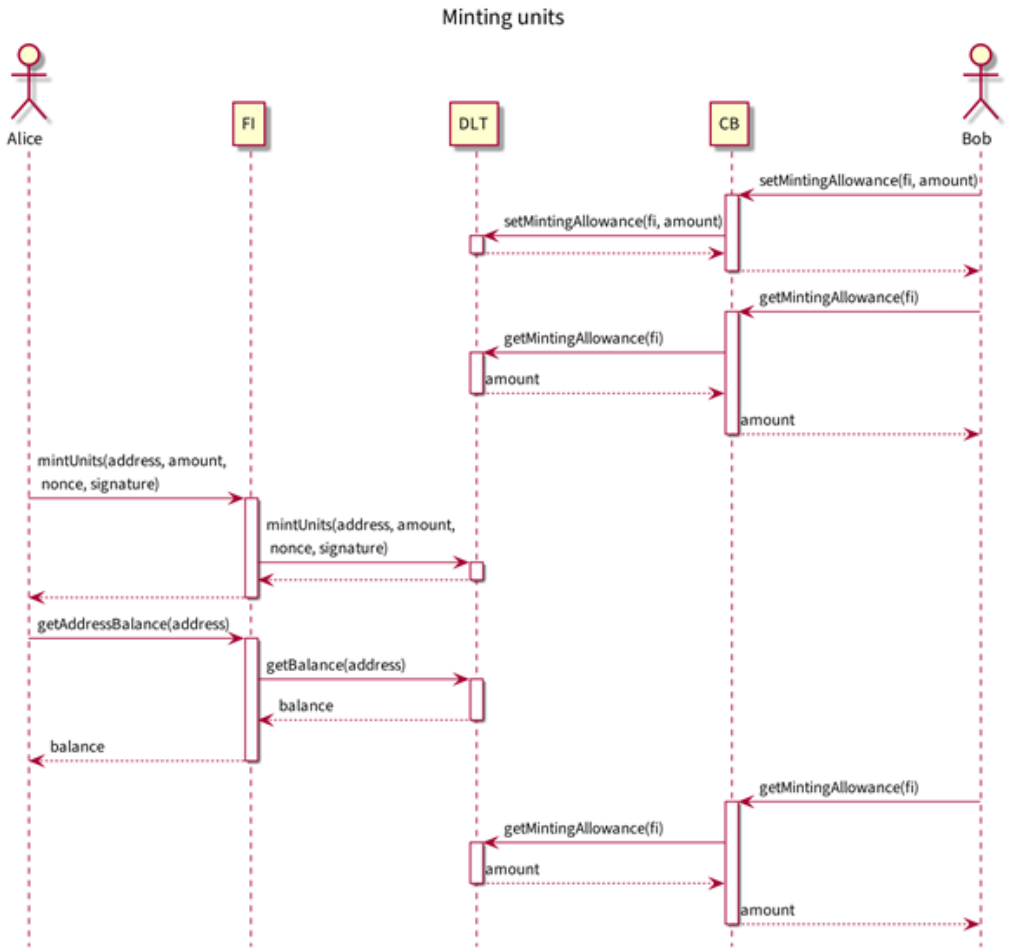

Figure 14: Alice, a customer of FI, requests the minting of CBDC to one of her addresses from an FI. Previously, an appropriate CBDC minting allowance has been set for the FI by a central bank employee.

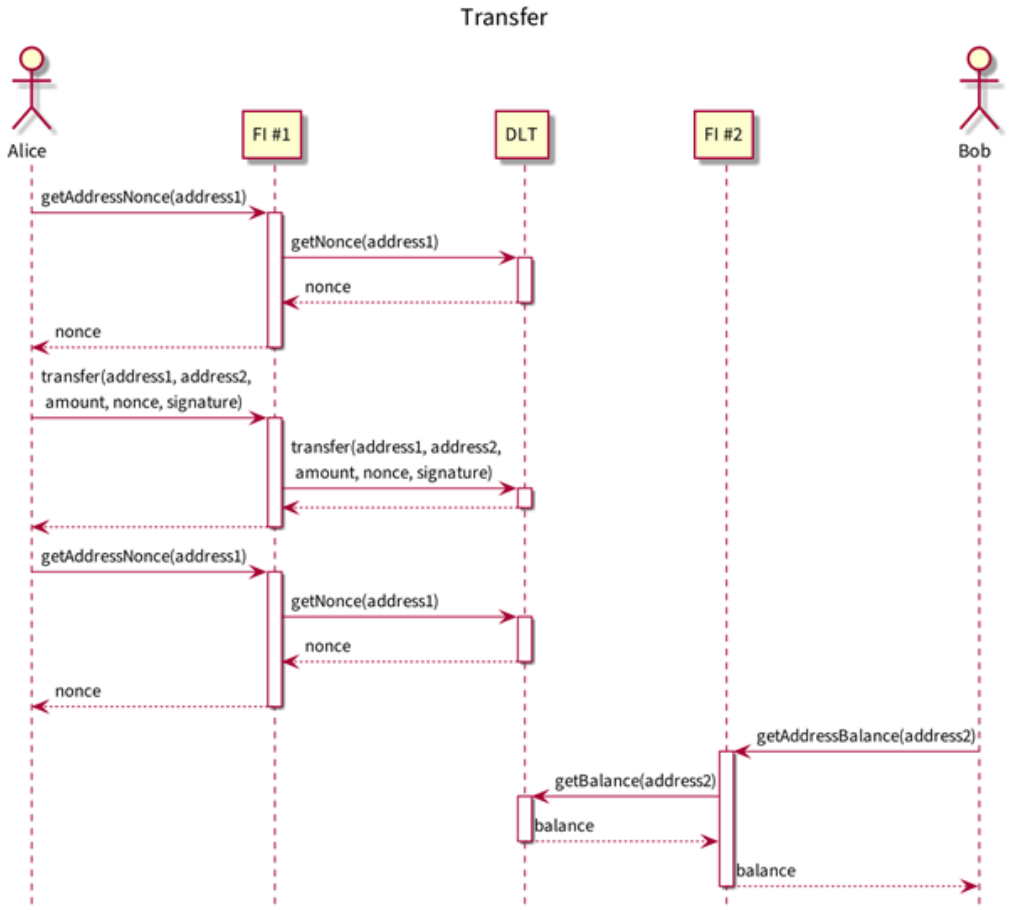

Figure 15: Alice queries the current nonce from the CBDC DLT through an FI for one of her addresses. Using that nonce, she assembles and signs a transfer transaction request to one of Bob's addresses. She gives the signed transaction request to an FI, which forwards it to the CBDC DLT.

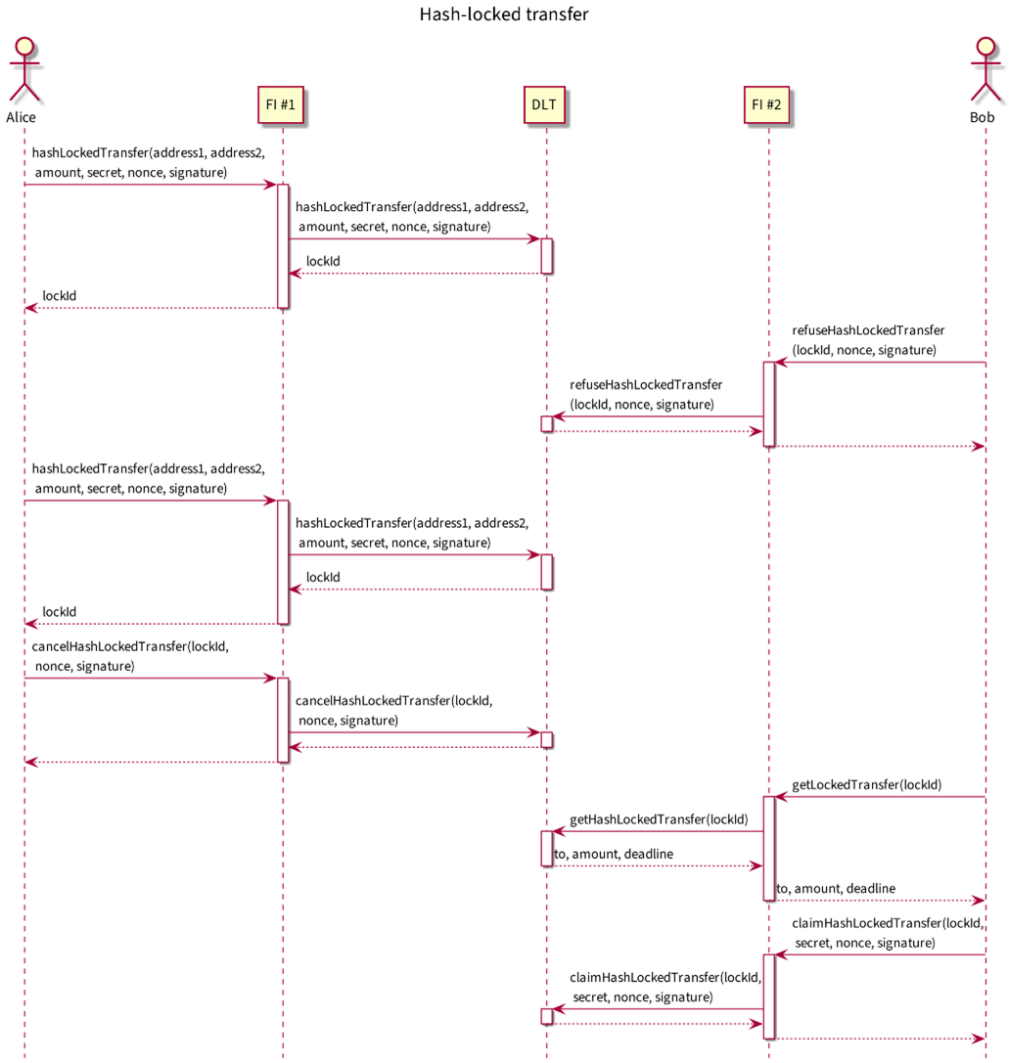

Figure 16: Hash-locked transfer.

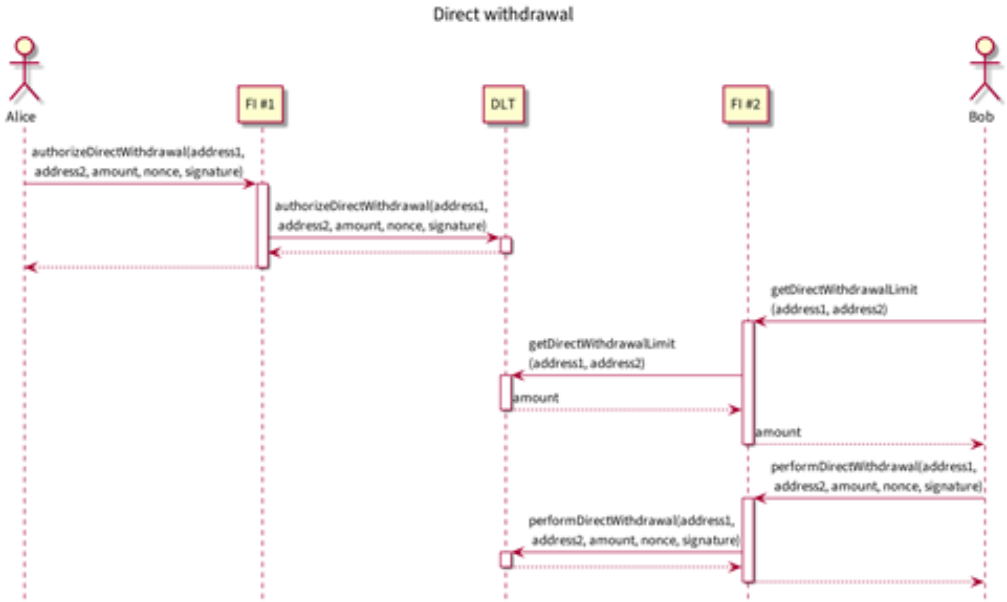

Figure 17: Direct withdrawal.

B A reference implementation for the related smart contracts

We have created a reference implementation of the contract to demonstrate that the idea of companies as smart contracts is in fact a possible solution to the initial problem. The code was written in Solidity [18] and the 0.8.4 compiler version was used to compile the contract.

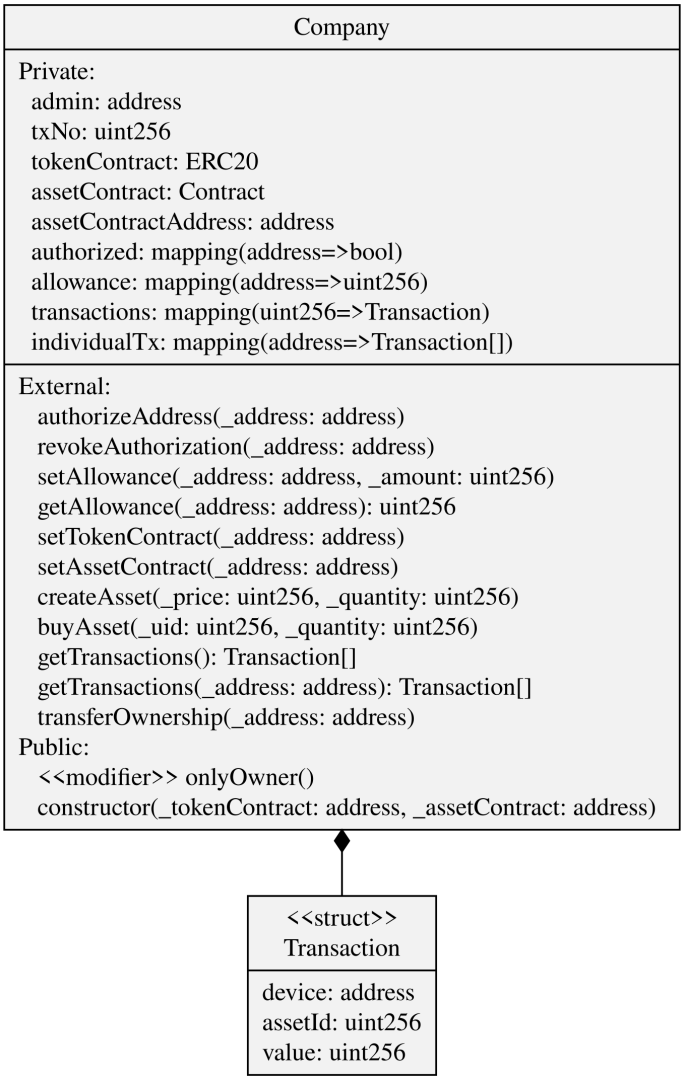

Figure 18: Class diagram of the Company contract

Figure 18 shows the class diagram of the Company contract. It can be seen that the constructor needs the address of a token contract (i.e., the address of a deployed, ERC20 compliant token contract) and the address of another contract (that (in the example) manages the ownership of assets), so they have to exist when the company

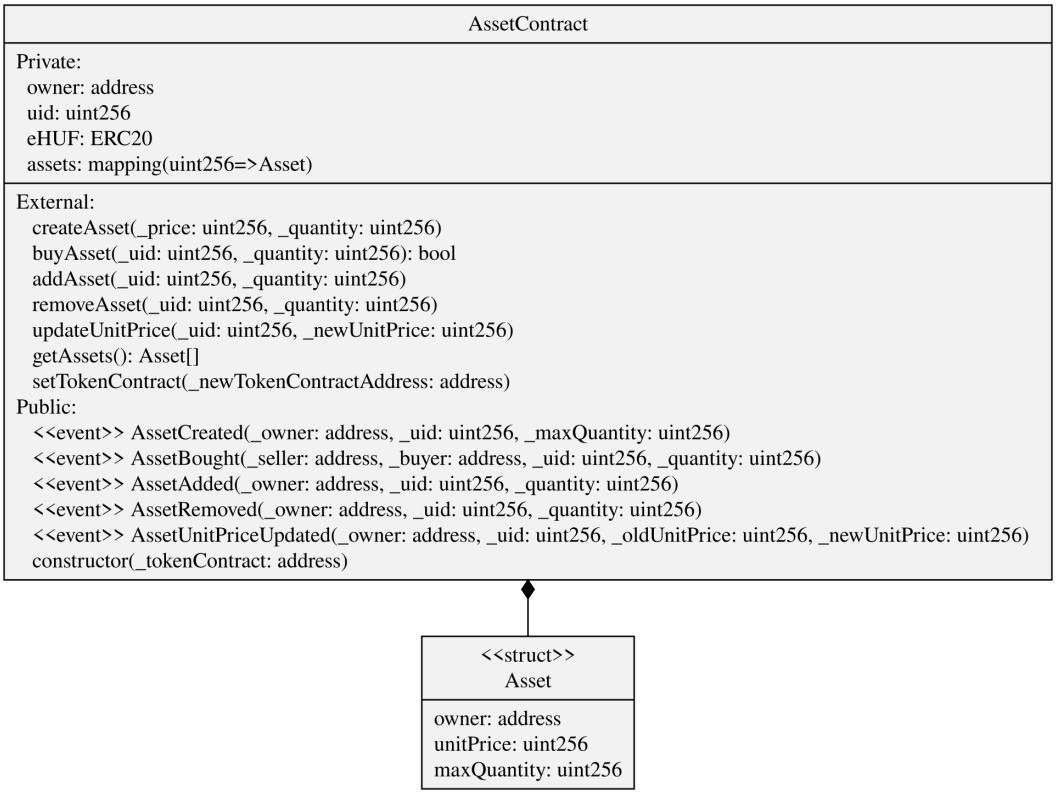

Figure 19: Class diagram of the Asset contract

contract is deployed to the blockchain. However, this implementation can be changed whenever the model changes.

Figure 19 shows the class diagram of the Asset contract, which the Company contract communicates with. It also needs the address of the same token contract that the Company contract uses, to ensure correct token transfers. Like in the case of the Company contract, the address of the token contract can be changed later. In case of the Asset contract changing its token contract address, it is the responsibility of the administrators of the companies to change their addresses accordingly and keep them up to date at all times.

Figure 20 shows the class diagram of the ERC20 token contract. Any contract can be used as a token contract that is ERC20 compliant. The Asset contract calls the transferFrom method inside of its buyAsset method to initiate the payment for the assets. For the payment to be successful, any buyer, whether it is an externally owned account or a contract, has to approve the asset contract to transfer funds on their behalf.

The contracts were deployed to a private Ethereum blockchain, then a series of test were carried out, including buying from an unauthorized address, buying from an address that is authorized but has an insufficient allowance to buy an asset, buying from an address that is authorized and has sufficient allowance to buy an

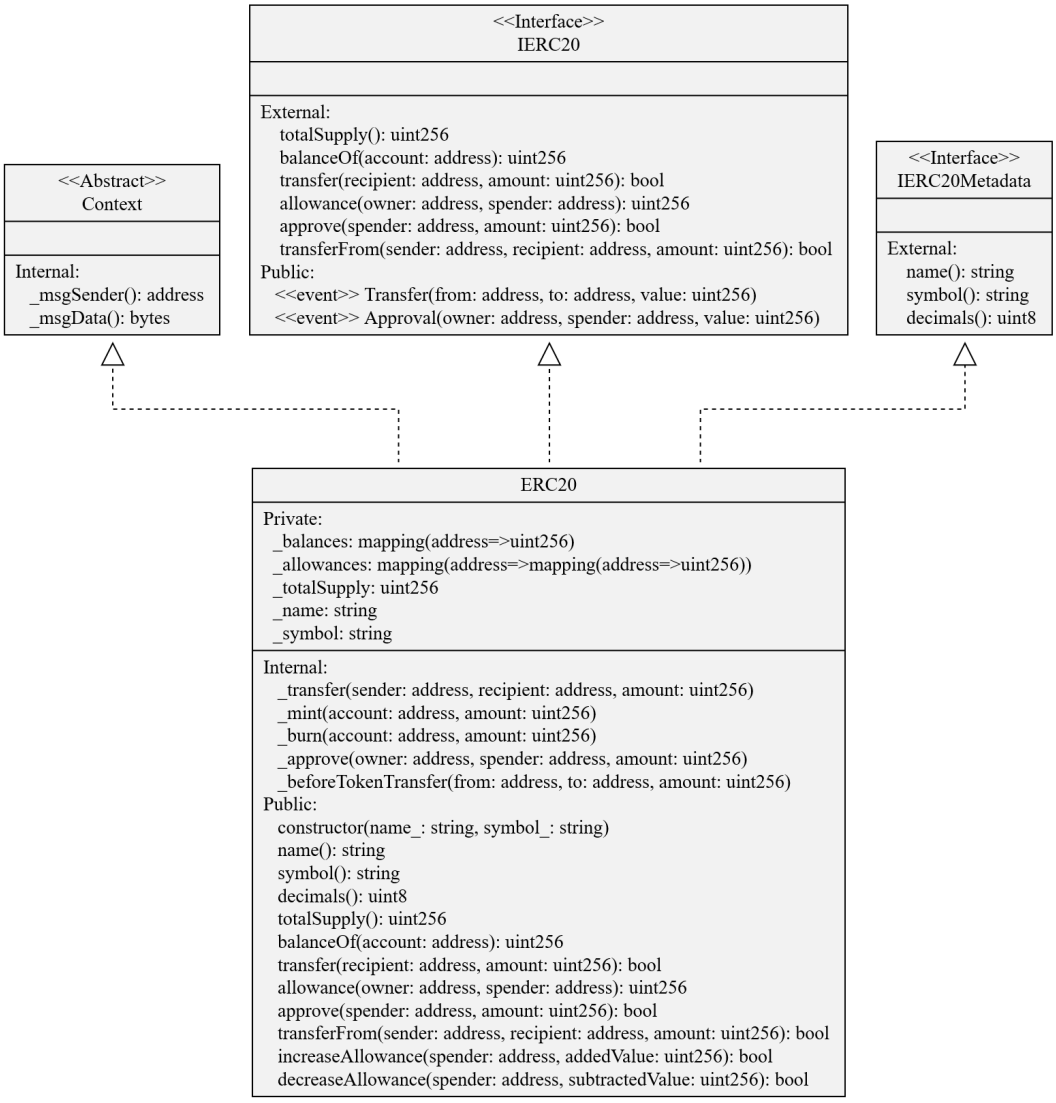

Figure 20: Class diagram of the ERC20 token contract

asset, calling methods from a non-admin address that can be called only from the admin address, etc.

The contract behaved as expected and every test case was successful during the test.

C Asset monitoring interface

On the **BME - MNB Asset Monitoring Interface** the changes in the ownership of the assets can be observed as they are recorded on the blockchain. The page shows the assets that are added in the current session and the location (owner) of every one of them. The locations and the corresponding balances are also listed, so it can be verified that the transactions were successful and the funds were transferred between the parties involved in the particular transaction.

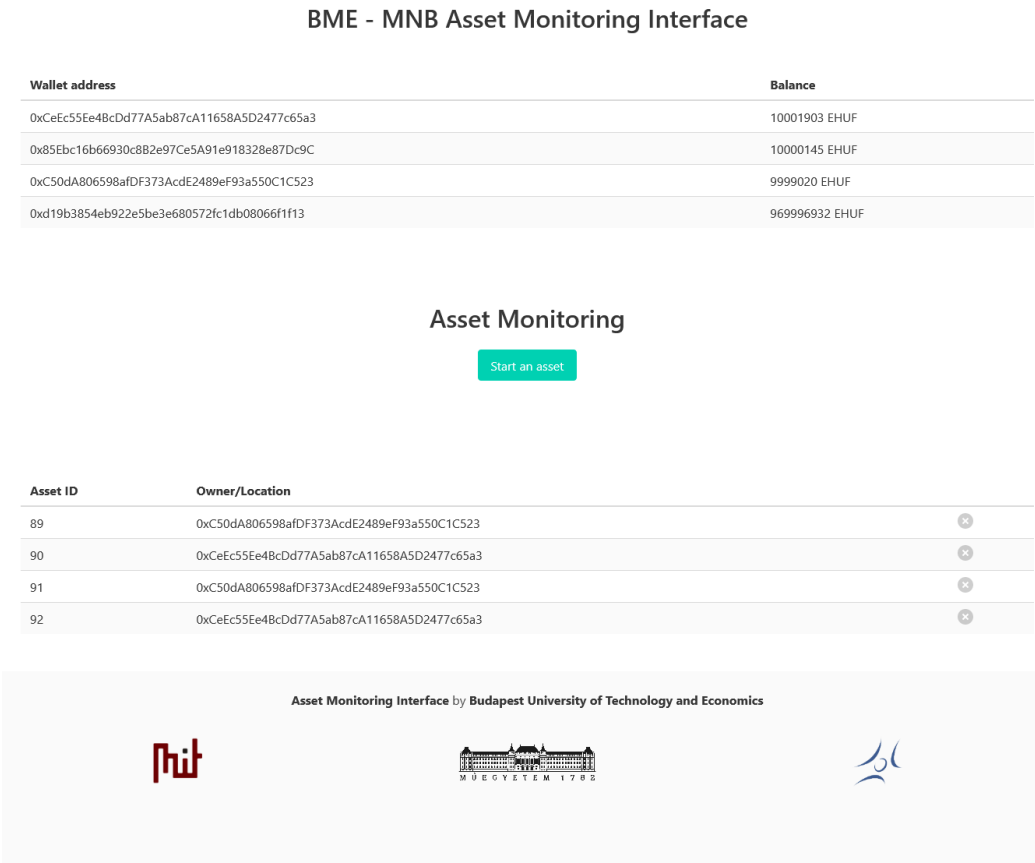

Figure 21: The Asset Monitoring Interface

Every account is tied to an actuator (robotic arm, conveyor belt, etc.) that handles assets. When an actuator handles an asset, it is considered to be the owner of that asset. In case of actuators exchanging assets (for example, a robotic arm places an asset on the conveyor belt), the receiving actuator buys the asset by calling the

approve and *buyAsset* functions, in this order, as described in the previous section. The next actuator can only obtain the asset if the payment was successful.

The Asset Monitoring Interface shows how a series of transactions happen: when the initial owner of an asset registers the new asset, it shows up at the end of the list. A second actuator then buys that asset, therefore becoming the new owner, and at the same time, the funds get transferred to the account of the initial actuator. Finally, a third actuator buys the asset from the second, becoming the final and current owner of the asset. The list of assets and balances get updated according to the transactions, so the current state can be observed anytime.

References

[1] David Andolfatto. *Assessing the Impact of Central Bank Digital Currency on Private Banks*. 2018. URL: https://doi.org/10.20955/wp.2018.026.

[2] Elli Androulaki et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the thirteenth EuroSys conference*. 2018, pp. 1–15.

[3] Raphael Auer and Rainer Boehme. "The technology of retail central bank digital currency". In: *BIS Quartlerly Review* (Mar. 2020). URL: https://www.bis.org/publ/qtrpdf/r_qt2003j.htm.

[4] Rafael Belchior et al. "A Survey on Blockchain Interoperability: Past, Present, and Future Trends". In: *CoRR* abs/2005.14282 (2020). arXiv: 2005.14282. URL: https://arxiv.org/abs/2005.14282.

[5] BIS Innovation Hub. *Central bank digital currencies: foundational principles and core features*. Oct. 2020. URL: https://www.bis.org/publ/othp33.pdf.

[6] Stefano Bistarelli, Ivan Mercanti, and Francesco Santini. "An analysis of non-standard bitcoin transactions". In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE. 2018, pp. 93–96.

[7] Kyoung Jin Choi et al. *A Proposal for a Canadian CBDC (Model X Final Report)*. Feb. 2021.

[8] Committee on Payments and Market Infrastructures. "Central bank digital currencies". In: *Markets Committee Papers* (Mar. 2018). URL: https://www.bis.org/cpmi/publ/d174.htm.

[9] European Union Agency for Cybersecurity. *Pseudonymisation techniques and best practices*. 2019. URL: https://www.enisa.europa.eu/publications/pseudonymisation-techniques-and-best-practices/at_download/fullReport.

[10] Jacob Eberhardt and Stefan Tai. "Zokrates-scalable privacy-preserving off-chain computations". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2018, pp. 1084–1091.

[11] Bank of England. *Discussion Paper: Central Bank Digital Currency Opportunities, challenges and design*. Mar. 2020. URL: https://www.bankofengland.co.uk/-/media/boe/files/paper/2020/central-bank-digital-currency-opportunities-challenges-and-design.pdf.

[12] European Central Bank and Bank of Japan. *Project Stella –Synchronised cross-border payments*. 2019. URL: https://www.ecb.europa.eu/paym/intro/publications/pdf/ecb.miptopical170101.en.pdf.

[13] Péter Garamvölgyi et al. "Towards Model-Driven Engineering of Smart Contracts for Cyber-Physical Systems". In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 2018, pp. 134–139. DOI: 10.1109/DSN-W.2018.00052.

[14] Amani Moin, Kevin Sekniqi, and Emin Gun Sirer. "SoK: A classification framework for stablecoin designs". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2020, pp. 174–197.

[15] Yao Qian. "Central Bank Digital Currency: optimization of the currency system and its issuance design". In: *China economic journal* 12.1 (2019), pp. 1–15.

[16] Michel Rauchs et al. "Distributed ledger technology systems: A conceptual framework". In: *Available at SSRN 3230013* (2018).

[17] Fabian Schär. "Decentralized finance: On blockchain-and smart contract-based financial markets". In: *FRB of St. Louis Review* (2021).

[18] *Solidity Programming Language*. URL: https://soliditylang.org/. (accessed: 05.07.2021).

[19] Katrin Tinn and Christophe Dubach. *Central bank digital currency with asymmetric privacy*. 2021.

[20] Rolf Van Wegberg, Jan-Jaap Oerlemans, and Oskar van Deventer. "Bitcoin money laundering: mixed results? An explorative study on money laundering of cybercrime proceeds using bitcoin". In: *Journal of Financial Crime* (2018).