BANCA D'ITALIA

EUROSISTEMA

Mercati, infrastrutture, sistemi di pagamento

(Markets, Infrastructures, Payment Systems)

Integrating DLTs with market infrastructures: analysis and proof-of-concept for secure DvP between TIPS and DLT platforms

by Rosario La Rocca, Riccardo Mancini, Marco Benedetti, Matteo Caruso, Stefano Cossu, Giuseppe Galano, Simone Mancini, Gabriele Marcelli, Piero Martella, Matteo Nardelli and Ciro Oliviero

July 2022

Number 26

BANCA D'ITALIA
EUROSISTEMA

Mercati, infrastrutture, sistemi di pagamento
(Markets, Infrastructures, Payment Systems)

Approfondimenti
(Research Papers)

Integrating DLTs with market infrastructures: analysis and proof-of-concept for secure DvP between TIPS and DLT platforms

by Rosario La Rocca, Riccardo Mancini, Marco Benedetti, Matteo Caruso, Stefano Cossu, Giuseppe Galano, Simone Mancini, Gabriele Marcelli, Piero Martella, Matteo Nardelli and Ciro Oliviero

Number 26 – July 2022

The papers published in the 'Markets, Infrastructures, Payment Systems' series provide information and analysis on aspects regarding the institutional duties of the Bank of Italy in relation to the monitoring of financial markets and payment systems and the development and management of the corresponding infrastructures in order to foster a better understanding of these issues and stimulate discussion among institutions, economic actors and citizens.

The views expressed in the papers are those of the authors and do not necessarily reflect those of the Bank of Italy.

The series is available online at www.bancaditalia.it.

Printed copies can be requested from the Paolo Baffi Library: richieste.pubblicazioni@bancaditalia.it.

ISSN 2724-6418 (online)
ISSN 2724-640X (print)

Banca d'Italia
Via Nazionale, 91 - 00184 Rome - Italy
+39 06 47921

Designed and printing by the Printing and Publishing Division of the Bank of Italy

Integrating DLTs with market infrastructures: analysis and proof-of-concept for secure DVP between TIPS and DLT platforms

by Rosario La Rocca*, Riccardo Mancini**, Marco Benedetti*, Matteo Caruso*, Stefano Cossu*, Giuseppe Galano*, Simone Mancini*, Gabriele Marcelli*, Piero Martella*, Matteo Nardelli* and Ciro Oliviero**

Abstract

The rise of a market for digital assets, both natively digital (e.g., non-fungible tokens, utility tokens) and those representing non-digital resources (e.g., security tokens, stablecoins, e-money tokens), is closely linked with the growing diffusion of distributed ledger technologies (DLT) platforms among investors. The increasing volume of transactions involving digital assets makes it worthwhile to investigate how central bank money can contribute to make the settlement safer and more efficient. While this is not strictly needed at the current juncture, central banks should arguably prepare to ensure that they can keep central bank money at the core of the settlement process of financially relevant assets. This requires the identification of possible solutions to increase interoperability between currently independent infrastructures: those regulating the exchange of the digital assets and those providing central bank money settlement services. A way to achieve this goal is to allow these transactions to flow through the market infrastructures, where central bank money is safely managed. This implies building a "bridge" between DLT platforms and market infrastructures.

This paper analyses several technical interoperability models and then describes the results of two experimental activities, which evaluate two different solutions for synchronizing the asset-leg and the cash-leg of a DvP (delivery versus payment) transaction. Both use the Target Instant Payment Settlement (TIPS) platform to provide the settlement services of the cash leg. The first, named "TIPS Hash-Link", is a lightweight, API based and DLT agnostic protocol which enables a loosely coupled integration of the market infrastructure with the majority of DLT platforms. Inspired by the Hash-Time Locked Contracts (HTLC) protocol, TIPS Hash-Link has been specifically tailored to overcome some failure scenarios commonly experienced with HTLC, leveraging TIPS as a trusted escrow for funds and a smart contract to coordinate the DvP operations on the DLT in a safe and consistent manner. The second one, named "TIPS-Algorand Just In Time Locking", takes advantage of the features offered by a specific DLT platform, Algorand. In particular, it leverages a native feature of the chosen DLT that simplifies DvP transactions guaranteeing atomicity; as such, this approach needs the two systems to be directly connected to interact with each other.

JEL Classification: E42.

Keywords: DvP, TIPS, DLT.

---

* Bank of Italy, Directorate General for Information Technology.

** Bank of Italy, Directorate General for Markets and Payment Systems.

Sintesi

La crescita di un mercato per asset digitali, sia nativi (ad es., non-fungible tokens, utility tokens) che relativi a risorse non digitali (ad es., security tokens, stablecoins, e-money tokens), è strettamente legata alla crescente diffusione, tra gli investitori, delle piattaforme basate su *Distributed Ledger Technologies* (DLT). L'aumento del volume delle transazioni che coinvolgono asset digitali rende utile indagare come la moneta di banca centrale possa contribuire a renderne il regolamento più sicuro ed efficiente. Sebbene ciò non sia strettamente necessario al momento attuale, le banche centrali dovrebbero prepararsi a garantire di poter mantenere la moneta di banca centrale al centro del processo di regolamento delle attività finanziariamente rilevanti. Questo richiede l'identificazione di possibili soluzioni per accrescere l'interoperabilità tra infrastrutture attualmente indipendenti: quelle che disciplinano lo scambio di asset digitali e quelle che forniscono servizi di regolamento in moneta di banca centrale. Un modo per raggiungere questo obiettivo è permettere a queste transazioni di fluire nelle infrastrutture dei mercati finanziari, dove la moneta di banca centrale è gestita in modo sicuro. Questo implica la costruzione di un "ponte" tra le piattaforme DLT e le infrastrutture dei mercati finanziari.

Questo articolo analizza diversi modelli di interoperabilità e descrive i risultati di due attività di sperimentazione, le quali valutano due diverse soluzioni per sincronizzare la "gamba" titoli (*asset-leg*) e la "gamba" contante (*cash-leg*) di una transazione di tipo DvP (*Delivery versus Payment*). Entrambe utilizzano la piattaforma TARGET Instant Payment Settlement (TIPS) per fornire i servizi di regolamento della "gamba" contante. La prima, chiamata "*TIPS Hash-Link*", è un protocollo leggero, basato su API e indipendente dalla DLT che consente un'integrazione a basso accoppiamento delle infrastrutture dei mercati finanziari con la maggior parte delle piattaforme DLT. Ispirato al protocollo *Hash-Time Locked Contracts* (HTLC), TIPS Hash-Link è stato specificamente progettato per superare alcuni scenari di errore comunemente sperimentati con HTLC, sfruttando TIPS come garante per i fondi e uno smart contract per coordinare le operazioni del DvP sulla DLT in modo sicuro e consistente. La seconda soluzione, chiamata "*TIPS-Algorand Just In Time Locking*", utilizza le funzionalità offerte dalla specifica piattaforma DLT utilizzata, Algorand. In particolare, sfrutta una funzionalità nativa della DLT scelta che semplifica le transazioni del DvP garantendone l'atomicità; in quanto tale, questo approccio richiede che i due sistemi siano direttamente collegati per interagire tra loro.

CONTENTS

1 Introduction[1]

1.1 Background and business context

The growing diffusion of distributed ledger technologies platforms among investors is one of the factors driving the rise of a market for digital assets. The transfer of ownership in securities transactions is a complex process; it is typically broken down into three main phases:[2] the *trading phase*, when a deal is established, with a seller agreeing to sell and a buyer agreeing to pay for a security; the *clearing phase*, when the trading parties arrange for the transfer of money and securities, typically by means of a central clearing counterparty (CCP), which aggregates trades and nets out transactions for the trading day; the *settlement phase*, when money and securities are actually exchanged in a coordinated manner between the parties involved on a settlement date. Typically, the settlement date is a couple of business days after the trade date. Exchange of money involves the use of a payment system (PS)[3] enabling transfers either in commercial bank or central bank money.[4] The usual recourse to clearing intermediaries is needed in order to minimize the counterparty credit risk between parties to a transaction and is obtained by various means (e.g., mutualization of risks among all CCP members, netting of transactions, deposit of collateral, creditworthiness monitoring of member parties, guarantee funds, and so on). The complexity of this process, and the number of intermediaries involved, make securities settlement often slow and costly.

In this context, the appealing promise of streamlining this process—lowering costs and execution times while maintaining comparable low levels of risk—is one of the key factors for the growing diffusion of distributed ledger technology (DLT) platforms (see the note on "DLTs, blockchains, and smart contracts").

1 We would like to thank Algorand Labs, and specifically Adriano Di Luzio, Cosimo Bassi, Stefano De Angelis and Federico Demicheli, for their valuable expertise and support in the experimental activities.

2 See ECB, 2022b.

3 A PS is used to settle financial transactions by transferring monetary value. PSs may be physical or electronic, each with its own procedures and protocols. Standardization enabled inter-operability, allowing some of these systems and networks to grow on a global scale. Nonetheless, there are still many country-specific or product-specific systems. A very high-level definition is provided in this document, because a PS is primarily responsible for transferring money and settling the so-called “cash-leg” of a Delivery-versus-Payment. The PS usually holds a ledger overseeing correct money transfer. This ledger can be organized using a token-based or an account-based abstraction. A token-based system holds a list of either spent or unspent tokens, representing money. An account-based system stores the balance for each customer correctly registered within the PS. For more details about token-based and account-based PS characterization, albeit focused on a central bank digital currency scenario, please refer to Urbinati *et al.*, 2021.

4 Central bank money: Liabilities of a central bank, in the form of either banknotes or bank deposits held at a central bank, which can be used for settlement purposes. Commercial bank money: Commercial bank liabilities that take the form of deposits held at a commercial bank which can be used for settlement purposes. See ECB, 2010.

DLTs, blockchains and smart contracts

The umbrella term *Distributed Ledger Technology* (DLT) encompasses a technological infrastructure together with its protocols allowing access, verification, and update of information stored on a common, shared ledger spread across multiple locations and/or entities. Since data is distributed, the so-called *consensus protocols* ensure the achievement of an agreement on data updates to be committed on the ledger.

A *blockchain* is a class of DLTs where the information is organized in blocks linked together in an orderly fashion. Each block is chained, in an append-only fashion, to its predecessor by means of a cryptographic hash, up to the first block in the chain — which is usually referred to as a *genesis block*.

While not strictly tied to, nor required by, the architectural style they represent, DLTs often support some form of programmability using scripts or smart contracts. *Smart contracts* are deterministic computer programs stored on the ledger that execute when predetermined conditions are met (e.g., see IBM, 2022). *Smart contracts* are usually adopted to automate the execution of agreements. Being available on a public ledger, they enable participants to audit implementation and verify outcomes without requiring any intermediary involvement. Smart contracts are also often used to create *tokens* representing assets like securities in digital form.

The adoption of a shared ledger among parties involved in financial trades is seen as a tremendous opportunity in terms of timely and secure information exchange, operating potentially 365/24/7. The usage of *smart contracts* enables the creation of complex financial instruments and enshrines the business logic in a guaranteed immutable and easily auditable form, potentially reducing recourse to intermediaries.

1.2 Scope of the document

This paper focuses on the aforementioned *settlement phase*, under the hypothesis that the financial instruments reside on some sort of DLT platform[5] as a tokenized asset[6], and the payment is made in central bank money. This relates to the *Delivery-versus-Payment* (DvP) in the case known in the literature as cross-ledger DvP, where cash and securities reside on separate ledgers, respectively the central bank payment infrastructures and DLT platforms (see the note on "Delivery-versus-Payment").

5 This excludes explicitly all the assets currently managed in the Eurosystem Platform Target2-Securities (T2S).

6 The programming capabilities of (some of the) blockchains enable the creation of special *tokens*, which can represent special *assets* - both fungible and non-fungible. Some examples of fungible tokens are stablecoins, security tokens, and utility tokens. Analyzing the legal aspects related to the issuance of tokenized assets is out of the scope of this paper. From the technical perspective, a prominent example of fungible tokens standard is represented by *ERC-20*, which forms the basis of other token-related standards on the Ethereum blockchain (see for more details: https://ethereum.org/it/developers/docs/standards/tokens/erc-20/); for security tokens, the former is often extended by *ERC-1400* (see: https://polymath.network/erc-1400, https://thesecuritytokenstandard.org/, and https://github.com/ethereum/EIPs/issues/1411). Non-fungible tokens (NFTs) are a non-interchangeable unit of data stored on a blockchain, which can be sold and traded. NFTs can represent a digital art-work or, in general, a digital file. Ownership of a token is associated with special rights to the related object. For example, owning a NFT is often associated with a license to use the underlying digital asset. The de facto standard in NFT issuance on the Ethereum blockchain is named *ERC-721*, and is representative of the vast majority of currently traded NFTs; see for more details: https://ethereum.org/en/developers/docs/standards/tokens/erc-721/.

Delivery-versus-Payment

Delivery-versus-Payment (DvP) is a form of settlement for securities, which links a securities transfer and a funds (cash) transfer in such a way that the delivery occurs *if and only if* the corresponding payment occurs (ECB, 2022a). A correct settlement requires that the systems perform operations with “all-or-nothing” semantic.
This concept is quite general as it essentially accounts for fulfilling an obligation to provide some kind of good or service if an agreed payment is finalised. However, the term is used specifically in the context of financial markets, where the good to be delivered is usually a *security* in exchange for a corresponding payment.
The presence of multiple ledgers leads to two main categories whereby a DvP can be conceptually and technically designed: in a *single-ledger DvP*, both cash and securities are on the same ledger; in a *cross-ledger DvP*, cash and securities are on two separate ledgers.

The objective of the study has been to investigate how central bank money can contribute to making the settlement of DvP trades in digital assets safer and more efficient, increasing interoperability between the DLT platforms regulating the issuance and exchange of those assets and the systems providing central bank money settlement services.

From a theoretical perspective, multiple possible interoperability and integration models for DvP have been explored and a comparison framework has been defined, with the identification of relevant dimensions of analysis - from both a business and a technological point of view. Each solution has been analysed and classified; moreover, a comprehensive failure analysis—with an approach somewhat remindful of *Failure mode and effects analysis* (FMEA)—has been put in place, pinpointing for each solution potential failure scenarios and effects.

In this context, a novel DvP interoperability model has been proposed based on this study, called *"API-based DvP with Hash-Link Contract"*, which draws upon the well-known Hash Time Locked Contracts (HTLC) approach and overcomes some limitations of the latter in terms of safety of DvP transactions, which is paramount in defining an interoperability model with central bank money settlement systems, while at the same time imposing minimum technical requirements on the asset-managing DLT.[7] To our knowledge, this model has not been introduced before in the literature, either in the central banking community or in the wider academic community.

From the experimental perspective, two interoperability models have been selected and evaluated. The first one, *API-based DvP with Hash-Link Contract* model, has been selected to represent loosely-coupled solutions which do not require the operation of a node on the asset-managing DLT. The second one, *Orchestrated DvP with Just In time Locking of the Cash-leg*, has been chosen to represent solutions which require tight-coupled integration, operating a node of the asset-managing DLT. In both cases, the Target Instant Payment Settlement (TIPS) platform was used to provide payment services for the cash leg; TIPS was selected from the Eurosystem infrastructures, for, among other reasons, its 365/24/7 availability. Algorand was the DLT platform chosen among a number of candidates to manage the asset leg due to its permissionless forkless nature, with a deterministic consensus protocol, and low block finalization time.

The main goal of the experiment was to verify the feasibility of the proposed models, effectively validating interoperability between a DLT and the Eurosystem Market Infrastructures in the DvP use-case. Several test-cases, both functional and non-functional, have been executed to verify the system response both in positive cases ("happy path") and in various failure scenarios. The architecture of both solutions is presented in depth from a technological point of view, along with comprehensive testing results.

7 The requirements on the DLT side are essentially limited to the support of some cryptographic *hash functions*, which are widely implemented by the vast majority of DLTs and blockchains, being an essential component of their inner working.

The conclusion section summarizes the results and findings, highlighting the strengths and weaknesses of the experimented solutions. In particular, it shows how the proposed implementation of a novel interoperability model—named *"TIPS Hash-Link"*—represents a viable solution in terms of performance and exhibits desirable characteristics in terms of low-coupling with respect to the specific asset-managing DLT.

1.3 Related work

Several central banks investigated the use of DLTs in securities settlement, in order to gain a better understanding of the opportunities offered by this new technology. This section lists some advanced research works that have involved the settlement of payments, securities, and the definition of DvP models.

In March 2018, the European Central Bank and the Bank of Japan, in the context of the second phase of the collaborative Project Stella, focused on securities DvP in a DLT environment (Project Stella, 2018). Besides providing a clear definition of different DvP scenarios, the Stella report details and analyzes the process flows of novel techniques for single-ledger DvP and cross-ledger DvP, based on Hash Time Locked Contracts. The main findings include that DLTs offer a new approach for achieving DvP between ledgers, which could give rise to additional challenges in terms of safety and efficiency that would need to be addressed.

In October 2018, the Bank of Canada, in collaboration with Canadian financial institutions and technological companies, published the results of the third phase of Project Jasper.[8] The project focused on a PoC for a DLT-based integrated securities infrastructure providing DvP settlement to help re-imagine the payment exchange process of CDSX - Canada's clearing and settlement system for securities. Jasper Phase III successfully demonstrated that a DLT platform can be used for a payment and securities settlement system.

In November 2018, the Monetary Authority of Singapore published the results of the third phase of Project Ubin.[9] Ubin was launched in 2016 to explore the use of Blockchain and DLT in the financial system, and the third phase analysed the DvP capabilities for the settlement of tokenized assets across different blockchain platforms. The project demonstrated that DvP settlement finality, inter-ledger interoperability, and investor protection can be achieved through specific solutions designed and built on blockchain technology. Albeit the project only provides a high-level description of the different DvP models, noting the importance of an arbitrator for dispute resolution. Section 2 also discusses its importance in guaranteeing the safety of different DvP models.

In December 2020, the BIS Innovation Hub, the Swiss National Bank and the financial infrastructure operator SIX announced the conclusion of the first phase of Project Helvetia. The experiments demonstrated the functional feasibility and legal robustness of settling tokenized assets in two different ways, by means of two different experiments. The first, with a wholesale Central Bank Digital Currency (CBDC) and the second by linking a DLT platform to the existing central bank payment system.[10] A second phase, completed in 2022, focused specifically on wholesale CBDC.

In March 2021, Deutsche Börse, Deutsche Bundesbank and Germany's Finance Agency published the results of an experiment in which they developed and successfully tested a settlement interface for electronic securities, working with a range of other market participants. Securities settlement using DLT is performed with the aid of a "trigger" solution and a transaction coordinator in TARGET2, the Eurosystem's large-value payment system. In doing so, the participants demonstrated that it is possible to establish a technological bridge between blockchain technology and conventional payment systems to settle securities in central bank money with no need to create central bank

8 See Bank of Canada *et al.*, 2018.
9 See MAS *et al.*, 2018.
10 See BIS Innovation Hub *et al.*, 2018.

digital currency.[11]

In December 2021, Banque de France, the BIS Innovation Hub Swiss Centre, the Swiss National Bank and a private sector consortium announced Project Jura[12], which explores the direct transfer of Euro and Swiss Franc wholesale CBDCs on a single DLT platform operated by a third party. Tokenized asset and digital currencies are successfully settled on a DLT using single-ledger DvP mechanisms.

Our document specifically focuses on cross-ledger DvPs as identified in the Stella Phase 2 report. Section 2 identifies four main models to achieve cross-ledger DvP, and analyses them in detail across a number of different dimensions. The DvP models are also divided into two categories, named orchestrated DvP models and API-based DvP models. Among API-based DvP models, this document presents a novel solution that, besides being DLT agnostic, introduces the role of the DvP oracle, a trusted entity which guarantees the DvP safety in case of disputes between Buyer and Seller. For two of these models (the novel API-based and an orchestrated one), the analysis is also complemented with practical experiment that involves the TIPS payment system and the Algorand DLT (see Section 3). Similarly to other works, this paper also concludes that it is possible to establish a technological bridge between a DLT and a conventional payment system for the settlement of securities in central bank money. Furthermore, this paper shows that API-based DvP models can be made DLT-agnostic with advantages in terms of the effort required to develop and integrate them with multiple and heterogeneous DLTs. In contrast, the orchestrated DvP models can rely on specific DLT capabilities to provide additional functionality (see Section 5).

2 DvP Integration Models

This section presents various models for cross-ledger DvP. Each model is assessed against a set of relevant dimensions of analysis described in Section 2.1. Being cross-ledger DvP the scope of this work, this document considers a scenario where the asset-leg of each DvP transaction is settled on a Distributed Ledger Technology (DLT) on which the asset is issued, whereas its cash-leg is settled on a Payment System (PS). For each DvP transaction, there are two involved counterparties, named Buyer and Seller. Buyer and Seller have agreed to exchange a given amount of securities against payment in cash. The exchange needs to take place within a given time, which has also been agreed upon by the two counterparties and depends, among other factors, also on the latency of the PS and DLT instances adopted. The details of how this trading agreement between the parties is carried out is out of the scope of this work. This section instead focuses on how to achieve atomicity of the cash payment on the PS and delivery of securities on the DLT. It is assumed that both Buyer and Seller can access the two systems, either directly or via a trusted intermediary (e.g., a bank). For each DvP model, an *Interoperability Solution* is described: this is a system component that needs to be developed in order to facilitate the DvP in that specific model. It is assumed that the Interoperability Solution can access the PS and (if needed) the DLT. Moreover, the Interoperability Solution can have accounts on the PS, and (if needed) a node and a wallet on the DLT. It is also assumed that the Interoperability Solution is operated by the central bank that also operates the PS. In the reference scenario, the PS, the DLT, and the Interoperability Solution are trusted, i.e., they are not subject to persistent crash or Byzantine failures and do not deviate from their expected behaviour. In particular, all DvP models assume to operate in absence of forks on the DLT, because either the DLT is *forkless* (e.g., when deterministic consensus protocols are adopted—as usually the case of permissioned DLTs) or it is highly improbable to revert confirmed transactions (e.g., due to the difficulty of performing a 51% attack). It is also assumed that communication channels among the different actors are weakly synchronous, i.e., may be subject to unpredictable delays

11 See Deutsche Börse *et al.*, 2021.
12 See Banque de France *et al.*, 2021.

that do not grow indefinitely. This is likely to be true in a production system with transient network faults. Moreover, communication channels are secure with exchanged messages assumed to be confidential and authenticated, i.e., no impersonation attack can take place. Four main DvP models are identified and distributed into two classes, namely "Orchestrated" and "API-based". In the first two DvP models (Sections 2.2 and 2.3), the Interoperability Solution is also called *Orchestrator*, because it coordinates the protocol steps on both the payment system and the DLT. In the other two DvP models (Sections 2.4 and 2.5), the Interoperability Solution is also called *API Gateway*, because it can be seen as an extension of the PS functionalities, by means of an Application Programming Interface (API), which is independent from the specific DLT technologies adopted for the asset-leg.

2.1 A list of relevant dimensions of analysis

This section presents a non-exhaustive list of relevant dimensions for analysing the DvP integration models. The solutions described in Sections 2.2–2.5 will be assessed against these dimensions. Some dimensions should be considered crucial for the success of the solution, such as the safety of the DvP. Others deal with technological coupling between the Interoperability Solution and the DLT platform, and may affect the development and maintenance cost of the analysed solution; these dimensions include, for example, the zero-code support for new DLTs or the solution agnosticism towards the DLT. Finally, the relevance of some other dimensions depends on the scenario in which the solution is deployed.

D1 *DLT agnosticism.* The capability to apply the solution to different DLTs/blockchain, under minimal assumptions and considering that nothing is known or can be known about the correct functioning and operation of the DLT platform. A DLT agnostic solution allows to interoperate with a new DLT, not originally supported, without requiring new developments specifically tied to that platform.

D2 *No DLT node to operate.* The capability of the solution to interoperate with a DLT without requiring to manage a node for each integrated platform.

D3 *No DLT wallet to own.* The capability of the solution to interoperate with a DLT without requiring to own a wallet for the native crypto-asset or any other asset that is issued on the DLT.

D4 *Presence of a trusted DvP oracle.* Whether the solution guarantees DvP safety by relying on a trusted oracle that, acting impartially, provides the outcome of the DvP in case of a dispute.

D5 *Safety, atomicity guarantees, and potential DvP failures.* The guarantees provided by the solution in terms of safety, regarding transaction atomicity (i.e., “all-or-nothing” transfer of cash and security) and failure avoidance in case of unforeseen operational behaviour from one or both parties involved, or the network connection between the parties and the platform.

D6 *Liquidity efficiency.* The capability of the solution to avoid locking cash liquidity (i.e. in escrow-like scenarios) until the whole transaction is completed.

D7 *Absence of the free option.* Whether the solution does not offer, to one of the involved parties, the right to choose to finalise the transaction or not while keeping the other party engaged, without corresponding a fee for this right.[13]

D8 *Specific features requirements of the DLT.* Whether the solution relies on features that are specific of a particular DLT. While it is reasonable to assume that a DLT should support widespread cryptographic primitives (e.g., SHA-256), constraints upon specific programmability features or signature algorithms of the platform are considered a specific feature requirement.

13 Equivalent of a “call” or “put” option for free.

2.2 Orchestrated DvP with Asset-leg Locking

The most simple DvP model in the DLT ecosystem involves an Orchestrator that acts as an escrow for the asset-leg and an arbitrator of the DvP transaction. Two variants of this protocol exist. In the simpler one, the Orchestrator acts as an actual temporary custodian of the asset, i.e., it receives the asset from the Seller, and delivers it to the Buyer upon evidence of payment. During the DvP, the Orchestrator has exclusive control over the asset that has to be delivered.

In a more advanced one, the Orchestrator shares the control of the asset with Buyer and Seller, usually in an escrow arrangement that uses a 2-of-3 threshold signature[14], where the 3 possible signers are the Buyer, the Seller, and the Orchestrator, and two signatures are required to transfer the asset. If Buyer and Seller agree among them on the result of the DvP transaction, then the Orchestrator is not involved, since no arbitration is needed. This is what happens during normal operation. On the contrary, if a dispute between the counterparties arises, either the Buyer or the Seller can involve the arbitrator in the process (i.e., the Orchestrator), in order to solve the dispute and release the asset from the escrow.

The more advanced protocol is considered, where the Orchestrator shares control of the assets during the DvP. The steps of a successful scenario are illustrated in Figure 1: in **Step1**, the Seller (and Buyer) initialize the DvP on the Orchestrator; in **Step2**, the Seller transfers the securities to the escrow; in **Step3**, the Buyer makes the payment within the agreed timeout; in **Step4**, the Seller authorizes the transfer of the securities from the escrow to the Buyer. In this scenario the DvP is considered cooperatively executed, because no arbitrator was involved.

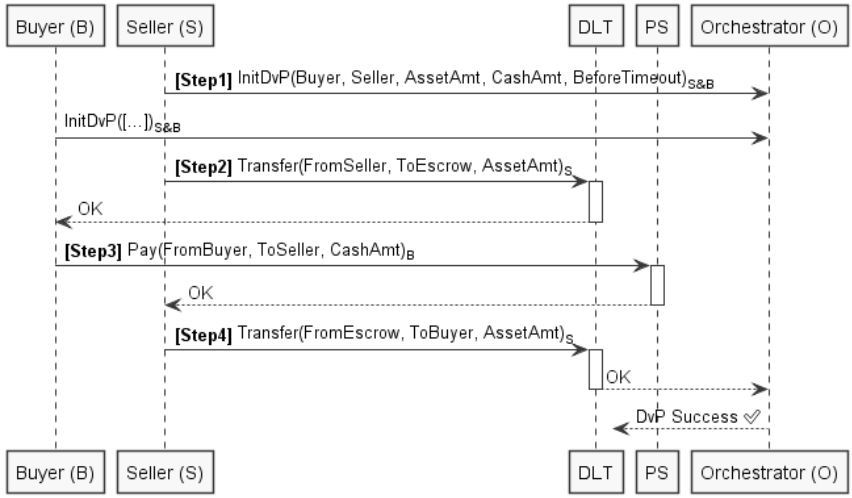

Figure 1. Orchestrated DvP with Asset-leg locking, success scenario.

Potential settlement failures may happen in the scenarios listed below.

- In Step3, if Buyer does not carry out the payment within the agreed timeout. In this case, the Buyer can authorize the transfer of the securities from the escrow to the Seller (cooperative cancellation). If this does not happen, the Seller has to involve the Orchestrator, acting as arbitrator, to unlock the assets from the escrow. In this latest scenario, the DvP is considered forced cancelled, because the arbitrator had to be involved.
- In Step4, if the Seller does not authorize the securities transfer from the escrow. In this case, the Buyer has to involve the arbitrator, that acknowledges the payment and transfers the asset to the Buyer (forced execution).

14 Multi-signature: https://en.bitcoin.it/wiki/Multi-signature#Multi-signature_application_examples.

DvP failures cannot happen in the hypothesis that Buyer and Seller act in their own interest, and the Orchestrator eventually fulfils its own responsibilities.

This DvP model relies on the Orchestrator as a trusted DvP oracle (D4), which acknowledges payments and returns/transfers assets on the DLT. For this reason, the solution is not DLT agnostic (D1) and the arbitrator has full visibility of the DvP transaction on the DLT. The solution requires new development for each new DLT that needs to be integrated, since the arbitrator implementation depends on the characteristics of the DLT that hosts the assets. The arbitrator operates a node on the DLT (D2) and owns a wallet to sign transactions (D3). Since DvP failures are excluded, safety is always guaranteed (D5). The solution locks the asset-leg only, while the cash is not locked (D6). The Buyer owns an option to buy the asset, and may refuse to carry out the payment in the second step (D7). The solution does not require special features to be supported by the DLT (D8), since it only requires standard transactions (in the first, simpler, variant) and a threshold signature escrow (in the second variant).

2.3 Orchestrated DvP with Cash-leg Locking

In this section, we illustrate two models that rely on the locking of the cash-leg of the DvP transaction, in order to ensure the safety of the DvP. The first model relies on an Orchestrator keeping on-hold a pre-authorized asset transfer transaction on the DLT until the payment occurs; by doing so, it can reduce the amount of time in which the cash-leg is locked. For this reason, we call this solution "Just In Time Locking" of the cash-leg, abbreviated as JITL. The second model relies on an Orchestrator acting as bridge component that can transfer value from the PS to the DLT, and vice-versa, in the form of a Non-Fungible Token (NFT)[15] representing the cash on the DLT, within the context of a specific DvP transaction. We call this solution "Tokenized Cash-leg Locking".

2.3.1 Just In Time Locking of the cash-leg

The JITL model uses a pre-signed transaction (or a package of transactions) that transfers the securities from Seller to Buyer on the DLT. The transaction is not immediately broadcasted to the DLT network, but it is given to the Orchestrator, that will release it only when the Buyer successfully reserves, on the PS, an amount of funds needed to carry out the payment.

15 A token is non-fungible when it is unique (i.e., non-interchangeable) and cannot be divided into tokens with lower amounts. For these reasons, the NFT can be used within the context of a specific DvP transaction only, and not as a general purpose cash token on the DLT.

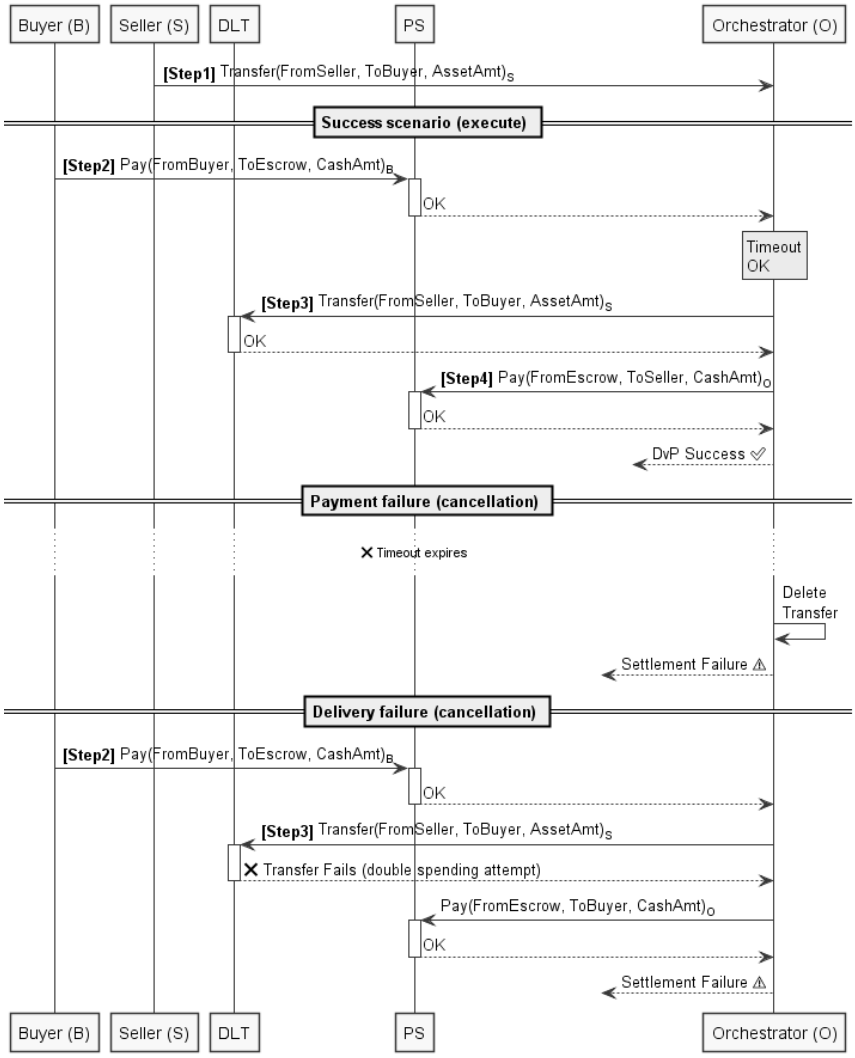

Figure 2. Orchestrated DvP with Just In Time Locking of the cash-leg.

A JITL DvP is illustrated in Figure 2; the steps of a successful scenario are the following. In **Step1**, Buyer and Seller initialize the DvP with the Seller providing the Orchestrator with a valid and pre-signed DLT transaction that eventually transfers the securities from Seller to Buyer on the DLT.[16] In **Step2**, Buyer transfers, within a timeout, cash needed for the payment to an escrow technical account owned by the Orchestrator on the PS. In **Step3**, the Orchestrator broadcasts Seller's transaction to the DLT network, and the asset transfer to the Buyer completes on the DLT. In **Step4**, the Orchestrator transfers cash from the escrow to the Seller account on the PS.

Potential settlement failures may happen in the scenarios listed below, while DvP failures cannot happen in the hypothesis that Buyer and Seller act in their own interest, and the Orchestrator eventually fulfils its own responsibilities.

- In Step2, Buyer does not perform the payment, or performs the payment with an incorrect amount. In this case, the timeout expires and either Seller or Buyer can request the Orchestrator to cancel

16 The pre-signed transaction can either be a simple transfer of the asset from Seller to Buyer, signed by the Seller only, or a more complex atomic package of DLT transactions signed by Seller, Buyer and also the Orchestrator, including the issuance and redemption of a cash-leg representation on the DLT. For sake of clarity, this section presents the simplest model. In Section 3.3, it is possible to find technical details on a more complex variant of the same kind, that provides additional safety guarantees (e.g., in case the pre-signed transaction leaks from either the Seller or the Orchestrator, or their communication channel) but has a different trade-off (e.g., it requires the Orchestrator to own a wallet on the DLT).

the DvP, delete the pre-signed transaction and, if necessary, refund partial cash payments.

- In Step3, Seller's transaction does not get confirmed, e.g., because Seller spent the securities on the DLT in the meantime. In this case, the Orchestrator cancels the DvP and returns cash to Buyer.

The DvP protocol relies on the Orchestrator as a trusted DvP oracle (D4), which handles cash transfers on the PS. For this reason, the model is not DLT agnostic (D1) and the Orchestrator has full visibility of the DvP transaction on the DLT. The model requires new development for each new DLT that needs to be integrated, since the Orchestrator implementation needs to validate and broadcast DLT transactions. The Orchestrator operates a node on the DLT (D2). A wallet to sign transactions is not needed because the DLT transaction is a simple transfer from Seller to Buyer (D3). Since DvP failures are excluded by the presence of the oracle, safety is always guaranteed (D5). This DvP model locks the cash-leg for the time that is needed for the securities transaction to confirm on the DLT (D6). Neither Buyer nor Seller own an option to buy/sell the asset, and both can cause the third step to fail (D7). The model does not requires specific features on the DLT side: only the ability to transfer an asset is needed. Since the Seller gives a valid and pre-signed transaction to a third party, it would be desirable the ability to define self-expiring transactions, so that the DLT can autonomously prevent asset transfers after expiration (D8).

2.3.2 Tokenized Cash-leg Locking

Most DLTs allow participants to issue assets on the ledger in the form of tokens, which are different from the native asset of the ledger.[17] For these ledgers, it becomes possible to achieve an orchestrated DvP with cash-leg locking.

At the beginning of the process, the Buyer reserves an amount of cash that is needed for the payment of the asset, e.g., by transferring it to an account controlled by the Orchestrator. Then, the Orchestrator issues, on the DLT, a special token that represents Buyer's funds, and transfers this token to the Buyer. The token is non-fungible (i.e., an NFT), and can be used only in the context of the specific DvP transaction for which it was issued: Buyer can either transfer the token to Seller, which in turn will return it to the Orchestrator, or the Buyer can directly return the token to the Orchestrator. The transfer of the token from Buyer to Seller happens contextually with the transfer of the asset from Seller to Buyer, directly on the DLT. For this to happen safely, the DLT must provide support for Single Ledger DvP[18]. Ultimately, Seller can claim the funds locked by the Orchestrator, by transferring the received token back to it: this will release the funds from the escrow and credit Seller's account.

The steps of a successful scenario of the cash-leg locking with tokenization are the following. In **Step1**, Buyer transfers cash needed for payment to a technical account owned by the Orchestrator on the PS. In **Step2**, Orchestrator issues the NFT on the DLT, and transfers it to Buyer. In **Step3**, Buyer and Seller engage in a single ledger DvP between asset and cash NFT. In **Step4**, Seller transfers the NFT to the Orchestrator. In **Step5**, Orchestrator transfers cash from the technical account to Seller, and PS credits Seller's account of the payment amount.

Potential settlement failures may happen in the scenarios listed below, while DvP failures cannot happen in the hypothesis that Buyer and Seller act in their own interest, and the Orchestrator eventually fulfils its own responsibilities.

- At Step3, if Buyer and Seller do not engage in the single ledger DvP; Buyer can return the NFT to the Orchestrator, to get its account credited.

17 The native asset of a DLT is here defined as the one used to pay the transaction fees to the DLT confirmation network; examples are bitcoin for the Bitcoin blockchain, ether for the Ethereum blockchain, and algo for the Algorand blockchain.

18 Existing DLTs offer three main technical solutions to implement a single ledger DvP. Some DLTs (e.g., Algorand Chen *et al.*, 2019) offer dedicated primitives to swap of two assets defined on the ledger. From the user-developer perspective, this is the simpler solution to adopt. Other DLTs (e.g., Bitcoin Nakamoto, 2008) allow to create and sign a single transaction with different asset types in the inputs and outputs. DLTs with expressive script languages (e.g., Ethereum Buterin, 2014) offer a more general approach with the creation of dedicated smart contract to encode the swap logic.

This DvP model does not rely on a third party acting as trusted DvP oracle (D4), because it models the DvP as single-ledger DvP that takes place on the DLT. The DvP is performed as an atomic transaction, where the atomicity is guaranteed by the DLT functioning. For this reason, the DLT recalls the functionalities of a DvP oracle; nevertheless, it is not considered as such, because the DvP oracle definition includes only third parties (i.e., it excludes the PS and the DLT). For this reason, the model is not DLT agnostic (D1) and the Orchestrator has full visibility on the DvP transaction on the DLT. The model requires new development for each new DLT to be integrated, since the Orchestrator implementation depends on the characteristics of the DLT hosting the assets. The Orchestrator operates a node on the DLT (D2) and owns a wallet to sign transactions (D3). Since DvP failures are excluded, safety is always guaranteed (D5). This DvP model locks the cash-leg, potentially with indefinite time, so it is not efficient (D6). Neither Buyer nor Seller own an option to buy/sell the asset, and both may decide to not execute the third step, i.e., the single ledger DvP (D7). The model requires the DLT to support issuance of tokenized and non-fungible assets, as well as functionalities for single-ledger DvP (D8).

2.4 API-based DvP with Hash-Time Locked Contract

This DvP model relies on API Gateway, which is sitting in front of the PS and exposes additional functionalities by mean of APIs. The API gateway operates a technical account on the PS, which can receive and hold funds to be transferred from Buyer to Seller. It also enables the secure exchange of messages from Seller to Buyer. We call this DvP model "API-based DvP with HTLC", because it leverages Hash Time Locked Contracts (HTLC[19], Poon *et al.*, 2016) on the PS and DLT to regulate both the cash and securities transfer. An HTLC is a special contract that is widely available on DLTs; it enables time-bounded conditional transfers. Besides acting as an escrow, it combines two types of locks: hash-lock and time-lock. The hash-lock allows unlocking the funds (or assets) in escrow by providing a correct secret phrase (referred to as *preimage*) within an agreed timeout. The time-lock adds an expiration time to the escrow. The recipient of the funds (or assets) locked within the escrow should claim them before expiration.[20] Otherwise, the original sender can claim back the funds (or assets).

An API-based DvP with HTLC is illustrated in Figure 3; the steps of a successful scenario are the following. In **Step1**, the Seller randomly generates a preimage and keeps it secret; then, they calculate a commitment H of the preimage (e.g., its cryptographic hash). In **Step2**, the Seller transfers securities from his account to an HTLC on the DLT. The HTLC locks securities on the commitment H until a timeout $Timeout_1$ expires; In **Step3**, using the API gateway, the Buyer moves funds from his account to a technical HTLC account on the PS. The HTLC locks funds on the same commitment used for the securities, until a timeout $Timeout_2$ (with $Timeout_2 < Timeout_1$) expires; In **Step4**, the Seller claims the Buyer's funds providing the preimage to the API gateway: the funds move from the HTLC account to the Seller account on the PS. The API gateway timely notifies the Buyer and provides him with the preimage. In **Step5**, leveraging the preimage, the Buyer can unlock the Seller's securities from the HTLC and moves them to his account on the DLT.

Potential failures may happen in the scenarios listed below.

- In Step3, the Buyer does not move funds to the technical account. This results in a settlement failure. However, the Seller can get back his securities as soon as $Timeout_1$ expires.
- In Step4, the Seller does not claim the Buyer's funds within $Timeout_2$. A settlement failure occurs. The Buyer can get back his funds as soon as $Timeout_2$ expires, and, as soon as $Timeout_1$ expires, the Seller can also get back the securities.

19 Hash Time Locked Contracts: https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts.

20 In a blockchain, the time-lock can either be expressed in terms of minimum number of blocks to be confirmed from the lock transaction (relative time-lock) or with a blockchain height (absolute time-lock).

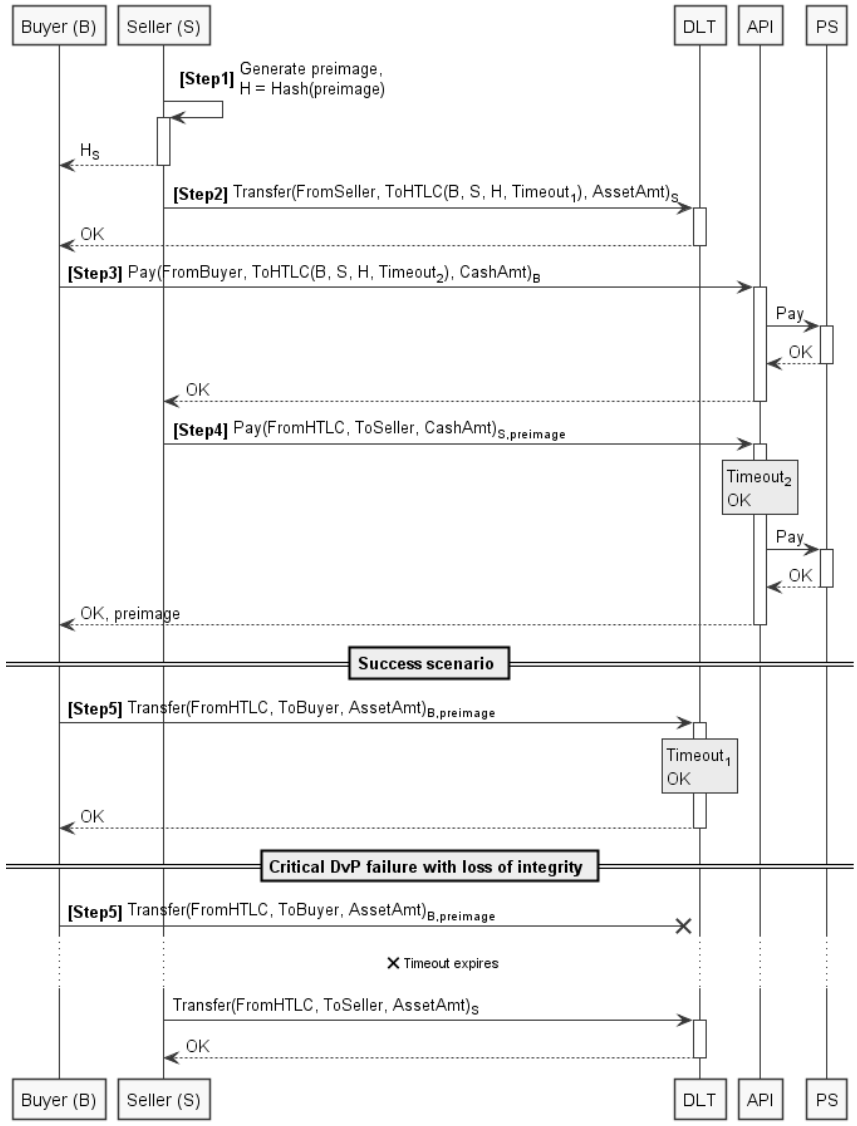

Figure 3. API-based DvP with HTLC, success scenario and critical DvP failure scenario.

- In Step5, the Buyer cannot claim the securities before $Timeout_1$ expires. A DvP failure occurs, resulting in a critical loss of integrity. The DvP safety is compromised, because the Seller can collect both the funds and the securities. It is important to highlight that this failure may materialize for external causes which do not depend on the Buyer. In particular, a temporary network unavailability on the communication channel between the API and the Buyer, may prevent the Buyer to timely receive the cash transfer notification in Step4. Moreover, a delay on the communication channel between the Buyer and the DLT, may prevent the Buyer to timely send the asset transfer request in Step5. A mitigation technique for this risk is to set $Timeout_1$ reasonably high, as a function of the principal amount: all other factors being equal, a higher amount implies a higher risk and requires a longer timeout. Note that the timeout value determines a trade-off between safety and efficiency in the DvP process: a longer timeout increases safety but decreases efficiency (during the DvP the asset is locked in the HTLC and cannot be transferred by the Seller), while a shorter timeout increases efficiency but can be detrimental for safety.

This model facilitates the DvP settlement by introducing specific APIs in front of the PS. The model is DLT agnostic, and does not require new development for supporting new DLTs (D1). There-

fore, it does not require to operate a node on the DLT (D2), or to own a wallet to sign transactions (D3). This model does not rely on a trusted DvP oracle (D4); as a direct consequence, it introduces critical failures that compromise the DvP (step 5): the safety of the DvP transaction is not always guaranteed (D5). The weakness of this model revolves around the idea of solving the DvP using timeouts instead of arbitrators: for this reason, a failure to deliver a given instruction within a given timeout may cause a DvP failure (see Project Stella, 2018, section on DvP failure). The API-based DvP with HTLC solution locks the cash-leg for the time that is needed for confirming the securities transaction (D6). The Seller owns a free option, as they can unlock securities by not claiming the Buyer's funds in the third step (D7). The model requires the DLT to support HTLC functionalities (D8).

An advantage of this model is that it allows participants to reuse an existing hash value to create an HTLC linked with other DvP transactions using the same hash value. In such a way, related DvP transactions across multiple ledgers can be created. This characteristic may enable additional applications, such as cross-border payments, that are however beyond the scope of this work.

The recent definition of the adaptor signature scheme[21] has introduced an alternative way of defining this DvP model. In particular, an adaptor signature can be used in place of the hash preimage, to link the cash and asset transfer. Adaptor signatures rely on elliptic curve cryptography. The smart contracts that use adaptors instead of hash values are called Point-Time Locked Contracts (PTLC), where the point on a curve is a public key against which the signature is verified. The adoption of PTLCs in place of HTLCs improves the confidentiality of the DvP on public ledgers, because the logic underlying the smart contract can be hidden in the success scenario, so that the asset transaction appears indistinguishable from a regular transfer on the DLT. However, the adoption of PTLCs does not change the safety considerations regarding this DvP model: Not introducing a trusted DvP oracle, it admits critical DvP failures that could compromise the safety of DvP transactions.

2.5 API-based DvP with Hash-Link Contract

This section presents a novel DvP model that relies on an API gateway to simplify the DvP definition and settlement. The "API-based DvP with Hash-Link Contract" builds upon the primitives of an HTLC, and introduces a trusted oracle, as in the orchestrated models, that solves possible disputes between Buyer and Seller. The involvement of a trusted oracle allows to guarantee the safety of DvP transactions, also in case of temporary unavailability of one of the involved actors (and, in particular, of the API gateway itself). The API gateway plays the role of trusted DvP oracle. This model defines a *Hash-Link Contract* (HLC) on the DLT that locks securities until some specific conditions are met. From a business perspective, these conditions are the following: (i) the Buyer agrees on the return of securities to the Seller, or (ii) the Seller agrees on the transfer of securities to the Buyer, or (iii) there is a dispute between Buyer and Seller and the oracle is involved. The knowledge of preimages is used to signal the decision of the oracle, that has exclusive knowledge of two preimages, namely cancellation preimage and execution preimage. If the DvP oracle decides to solve the dispute in favour of the Seller, e.g., because the payment did not occur within an agreed timeout, then it reveals the *cancellation preimage* to the Seller, which in turn uses it to get back the securities from the HLC. On the contrary, if the DvP oracle solves the dispute in favour of the Buyer, e.g., because the corresponding payment did already occur, then it reveals the *execution preimage* to the Buyer, which uses it to claim the securities from the HLC.

21 Adaptor signature scheme: https://bitcoinops.org/en/topics/adaptor-signatures/.

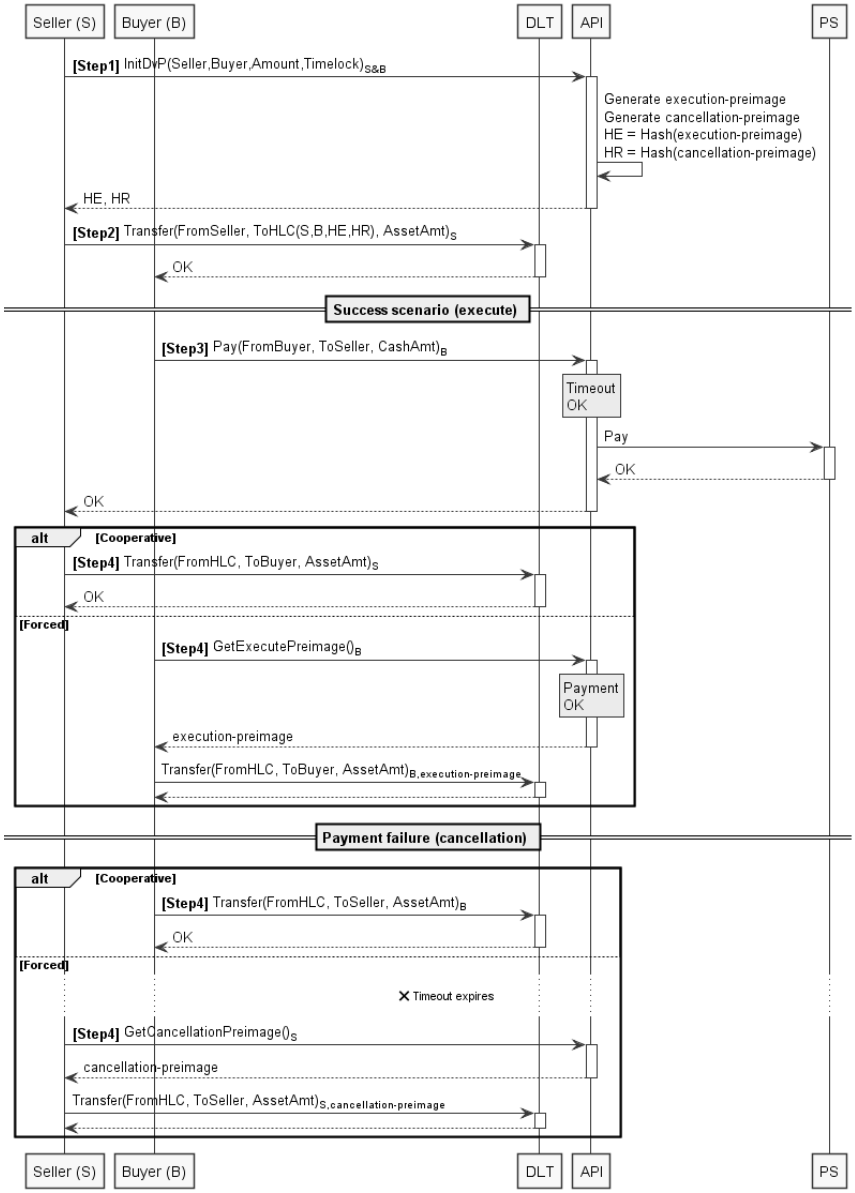

Figure 4. API-based DvP with Hash-Link Contract.

An API-based DvP with HLC is illustrated in Figure 4; the steps of a successful scenario are the following. In **Step1**, the Seller initializes a DvP transaction by interacting with the API gateway. The latter generates the two preimages and communicates their commitment (e.g., a cryptographic hash) to both Buyer and Seller. To resolve possible future disputes, the API gateway, acting as oracle, starts a timer for the current DvP, which expires according to the information received. In **Step2**, the Seller transfers securities from his account to an HLC on the DLT. The HLC locks securities until it is provided with a valid signature (from either the Seller or the Buyer) or a preimage (either execution or cancellation). Unlocking with the Seller's signature or the execution preimage moves the securities to the Buyer's account on the DLT. Unlocking with the Buyer's signature or the cancellation preimage takes back the securities to the Seller's account on the DLT. In **Step3**, using the API gateway, the Buyer moves funds from his account to the Seller's account on the PS. The API gateway readily notifies both Buyer and Seller of the correct completion of the funds reserve. In **Step4**, the Seller unlocks the HLC to move securities to the Buyer's account on the DLT. This case is called *cooperative execution*, because Buyer and Seller collaborate to successfully complete the

DvP transaction.

Potential settlement failures may happen in the scenarios listed below. DvP failures cannot happen in the hypothesis that Buyer and Seller act in their own interest, and the API gateway eventually fulfils its own responsibilities. The API gateway, acting as DvP oracle, guarantees the protocol safety, also in case of temporary unavailability of involved actors and arbitrary delays on the communication channels.

- In Step3, the Buyer does not move funds to the Seller's account on the PS. This results in a settlement failure. The Seller can get back his securities by retrieving the cancellation preimage from the API gateway as soon as the timeout expires. This case is called *forced cancellation*.
- In Step4, the Seller does not collaborate with the Buyer to open the HLC. The Buyer can obtain the execution preimage from the API gateway and collect the securities from the HLC on the DLT. This case is called *forced execution*.

The API-based DvP with Hash-Link Contract model introduces the presence of a trusted DvP oracle for solving disputes between Buyer and Seller (D4). Using hash preimages instead of digital signatures allows to maintain a very low level of coupling between API gateway and DLT. This model is DLT agnostic, and does not require new development for supporting new DLTs (D1). Therefore, this model does not require to operate a node on the DLT (D2), or to own a wallet to sign transactions (D3). The API gateway only needs to support the same hash function of the DLT platform. Differently from the API-based DvP with HTLC, this DvP model guarantees the safety of the DvP transaction (D5). This DvP model locks only the asset-leg for the time needed for confirming the funds transfer transaction (D6). The Buyer has the free option, as they can avoid transferring funds in the third step described above, while the Seller has already engaged the securities in the HLC (D7). The model requires the DLT to support HLC functionalities (D8).

2.6 Summary of DvP models analysis

In this section, the different DvP models are compared against the dimensions of analysis introduced in Section 2.1. Table 1 summarizes the results. Notice that the different dimensions are not equally important. First, if a DvP model does not guarantee settlement safety (D5), it will be hardly considered for an actual DvP implementation. This is, for example, the case of the API-based with HTLC model. Second, the dimensions assessing the neutrality of the model and the coupling with the DLT platform (D1, D2, D3, D8) are prioritized. For example, the DLT agnosticism can be useful when it enables the integration of new DLTs with a very limited effort, thus avoiding the need of developing new DLT-specific adapters. Dedicated development for new DLT could slow down the integration process, thus indicating a less extensible solution. The need for the Interoperability Solution to own a wallet for the native DLT crypto-asset is also a relevant downside, because it introduces strong coupling with the DLT. Also, the need for specific features could affect the neutrality of the Interoperability Solution Operator. However, specific DLT features might be necessary to implement a DvP model, such as the ability to deploy contracts, non-fungible assets, or perform atomic transfers. Third, other dimensions that affect the DvP cost for participants are considered, such as the liquidity efficiency (D6) and the absence of the free option (D7). Finally, other dimensions such as presence of a trusted oracle (D4), are less important in the specific scenario considered in this document. Overall, there is not a single DvP model that dominates all the others. The DvP models using an Orchestrator enable safety and can be adopted when a tighter control of what happens on the DLT is desirable. Nevertheless, these models have higher coupling with the DLT platform and require higher effort on the development side. To improve the Interoperability Solution operator neutrality with respect to DLTs, API-based DvP with HLC appears as the most suited DvP model.

Table 1 below summarises the analysis of the DvP models presented in this section.

Table 1. Summary of the DvP models analysis.

Dimensions	Asset-leg locking	Just in time cash-leg locking[a]	Tokenized cash-leg locking	Hash-Time Locked Contract	Hash-Link Contracts
D1: DLT agnosticism	No	No	No	Yes	Yes
D2: No DLT node to operate	No	No	No	Yes	Yes
D3: No DLT wallet to own	No	Yes	No	Yes	Yes
D4: Use of a DvP oracle	Yes	Yes	No	No	Yes
D5: Safety of the DvP	Yes	Yes	Yes	No	Yes
D6: Liquidity efficiency	Yes	No[b]	No	No	Yes
D7: No free option	No	Yes	Yes	No	No
D8: Specific DLT features	Yes[c]	No	Yes[d]	Yes[e]	Yes[f]

[a] This column shows the analysis for the simplest JITL model only. Other models may differ in: (i) the need to hold a wallet on the DLT, (ii) the need for specific DLT features, such as token issuance and single ledger DvP, (iii) additional safety guarantees in scenarios with a more powerful adversarial model; [b] Cash-leg is locked only for the time required for the asset transfer; [c] Threshold signatures wallet; [d] Token issuance and single-ledger DvP; [e] Hash functions and Time-locked transactions; [f] Hash functions only.

3 Experiment

The experiment aims to develop and evaluate Proof-of-Concept (PoC) solutions for secure execution of a DvP between a distributed ledger and a payment system, hosting respectively assets and cash. Section 3.1 introduces the actors and systems involved in the PoC: the Target Instant Payment Settlement (TIPS) system presented in Section 3.1.1, which provides the settlement services of the DvP cash-leg; the Algorand blockchain, presented in Section 3.1.2, where the asset-leg of the DvP resides. Among the different DvP integration models, two of them were selected for a deeper investigation: Section 3.2 presents "TIPS Hash-Link", an instance of the API-based DvP with Hash-Link Contract model; Section 3.3 presents "TIPS-Algorand Just In Time Locking", an instance of the Orchestrated DvP with Just In Time Locking model. The former represents a loosely-coupled solution that does not require any connection with the DLT. The latter represents a solution tailored for a specific DLT and, as such, requires a tight-coupled integration and also to operate a node on the DLT.

3.1 Actors of a cross-ledger DvP

The experiment considers a typical cross-ledger DvP scenario in which a Seller and a Buyer, interact to exchange assets on a DLT for cash on a payment system. Figure 5 represents the two involved actors, together with the asset and cash ledgers: the Algorand[22] DLT implements the platform holding the asset-leg, while TIPS implements the platform holding the cash-leg of the DvP. Buyer and Seller own the assets on the DLT thanks to private keys stored in their respective wallets, represented in blue.

22 The name "Algorand" is used to represent the network of nodes collectively operating the DLT.

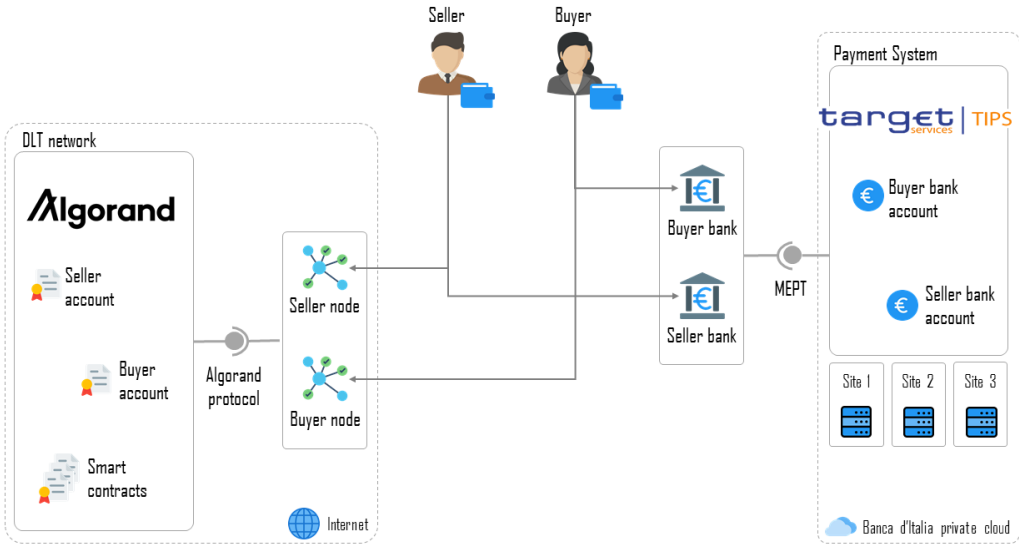

Figure 5. Actors of a cross-ledger DvP: Seller owns assets on the Algorand DLT and wants to sell them to Buyer for an agreed amount in cash; the cash-leg resides on a separate ledger, i.e., on the TIPS system.

3.1.1 The Target Instant Payment Settlement System (TIPS)

In this experiment, the payment system is instantiated as the Target Instant Payment Settlement (TIPS[23]) platform, which provides the settlement services of the cash-leg. TIPS was selected, among the other market infrastructures owned by the Eurosystem such as Target2, for its 365/24/7 operational model, which most closely resembles the service offered by a DLT platform.

TIPS manages commercial banks' accounts and regulates central bank money flows among these accounts. Customers (e.g., Buyer and Seller) have no visibility on TIPS and cannot hold accounts in it: they can only access the TIPS services through a commercial bank acting as a trusted intermediary. The interaction with TIPS takes place via the Message Exchange Processing for TIPS (MEPT, 4CB, 2019) protocol. Without loss of generality of the DvP models, and only for the scope of this experiment, a simplifying assumption is considered: the PoC assumes that Buyer and Seller have direct access to the settlement services provided by TIPS, they own a TIPS account and have (direct or indirect) access to it. Buyer and Seller accounts are used to move the liquidity for the cash-leg of the DvP transactions.

The realization of the PoC required the set up of a dedicated TIPS processing environment. The new environment was implemented within the Bank of Italy's Private Cloud infrastructure (Figure 5) and inherited all the characteristics of the production one. From a physical point of view, the TIPS environment of the PoC was distributed over three data centers, thus creating an *active-active* architecture that replicates the components of the TIPS settlement engine as in the production environment. Virtual machines located in each data center host the application components and the middleware necessary for the construction of the TIPS settlement engine. All virtual machines have been created using the API exposed by the Private Cloud platform, which in turn is based on OpenStack.

23 See Renzetti *et al.*, 2021 and Arcese *et al.*, 2021.

3.1.2 The Algorand DLT

Algorand[24] is a permissionless and public blockchain founded in 2017 by the ACM Turing Award winner and MIT professor Silvio Micali and developed by the Algorand Foundation[25] and by Algorand Inc., a company based in Boston. This PoC benefited from a collaboration with experts from Algorand, who provided advice on the best use of constructs and specific features of the DLT.

Algorand consensus. Algorand uses a *Pure Proof of Stake* (PPoS) consensus protocol to update the distributed ledger. The PPoS algorithm allows Algorand to confirm transactions and update the blockchain with latency in the order of seconds while scaling to many users. PPoS leverages a novel *Byzantine Agreement* protocol to reach consensus, which is aimed to ensure a high level of *security* without lacking in *scalability* and *decentralization*.[26] Algorand ensures that users never have divergent views of confirmed transactions (i.e. it ensures that forks do not occur), even if some of the users are malicious and the network is temporarily partitioned.[27]

Algorand infrastructure. To interact with the blockchain, a user (e.g., Buyer, Seller) must operate a DLT node, which acts as a peer of the Algorand blockchain network. To simplify resource demand, lightweight nodes exist: they act only as a client of the network and do not directly participate in the consensus protocol that grows the chain. This experiment adopts an Algorand node hosted by Purestake[28]. Furthermore, to sign transactions and operate upon their assets, a user holds a pair of public/private cryptographic keys; when a transaction is signed with the user's private key, the authorship is clear and cannot be repudiated. To improve privacy, a user can hold multiple public/private keys. A wallet helps to store such keys and simplify the process of transaction signing.

Algorand ledger. Algorand's native asset, i.e., the asset that is used to pay transaction fees in the Algorand ledger, is called the *Algo*. By holding Algos, users can also register to participate in consensus, which means that they will participate in the process of proposing and voting on new blocks to grow the chain. The Algorand functionalities include the creation of non-native assets on the blockchain, called Algorand Standard Assets (ASA). An ASA can be configured to represent either a fungible or a non-fungible token (i.e., NFT) and, in addition, allow transfer restrictions to be placed on it. Before a new ASA can be transferred to a specific user, the receiver must *opt-in* to receive it.[29] The Algorand ledger organizes the state using accounts (account-based ledger), which live on the blockchain and are associated with specific on-chain data, e.g. a balance; an Algorand address is used to identify an Algorand account. The account-based organization of the DLT state in Algorand smoothly fits with the ability to run smart contracts to update the DLT state.

Algorand programmability. Two main forms of programmability exist in Algorand: smart contracts and smart signatures. *Smart contracts* are pieces of code deployed on the DLT, which have public addresses, and can hold an internal state. Their execution is triggered by transactions directed to their address. Their business logic often includes pre-conditions and actions, whose output leads to internal state update or to funds/assets transfer. The smart contract logic is executed by the DLT nodes, and its state is also kept "on-chain". Running the smart contract logic requires the payment of fees, whose value is proportional to the amount of operations performed.[30] Instead, storing the state requires the escrow of a minimum balance that will be returned after clearing the state. A *Smart signature*[31] is a programmable logic executed while spending a transaction. It can

24 See Chen *et al.*, 2019.
25 https://algorand.foundation/about-us
26 https://algorand.foundation/algorand-protocol/about-algorand-protocol/pure-proof-of-stake
27 https://www.algorand.com/technology/white-papers
28 https://www.purestake.com/
29 https://developer.algorand.org/docs/get-details/asa/
30 https://developer.algorand.org/tutorials/understanding-teal-opcode-budget/
31 https://developer.algorand.org/docs/get-details/dapps/smart-contracts/#smart-signatures

be thought as an account whose private key includes the source code: knowledge of the source code is a necessary condition to issue a transaction on behalf of the smart signature account. The transaction will be successfully completed only if the smart signature allows it: e.g., by hard-coding the transaction beneficiaries, a smart signature can enforce payment towards their accounts only. Smart signatures do not store additional state on the DLT but the transaction itself. They can be used to define escrows as well as delegate signature authority. For this reason, they are usually created with a disposable semantic. An instance of a smart signature is created from a template with a limited scope, e.g., in the context investigated in this document, to serve a single DvP. Serving a limited number of actors, smart signatures are also less risky than smart contracts: an exploit in a smart contract would affect all users interacting with it. Smart contracts and signatures can be written in a programming language called TEAL, or in higher level language such as Python, that is then compiled to TEAL. The compiled TEAL program produces an Algorand address and a private key, called *Logic Signature*. The latter can be used to evaluate the business logic while spending a transaction: the transaction is allowed when it evaluates to True against the TEAL logic.

3.2 TIPS Hash-Link Contract DvP

This section describes the PoC implementing the API-based DvP with Hash-Link Contract model described in Section 2.5, sitting between the TIPS and Algorand platforms introduced in Section 3.1. The core component of the PoC is the TIPS Gateway, represented in Figure 6, which implements the Hash-Link specific functionalities.

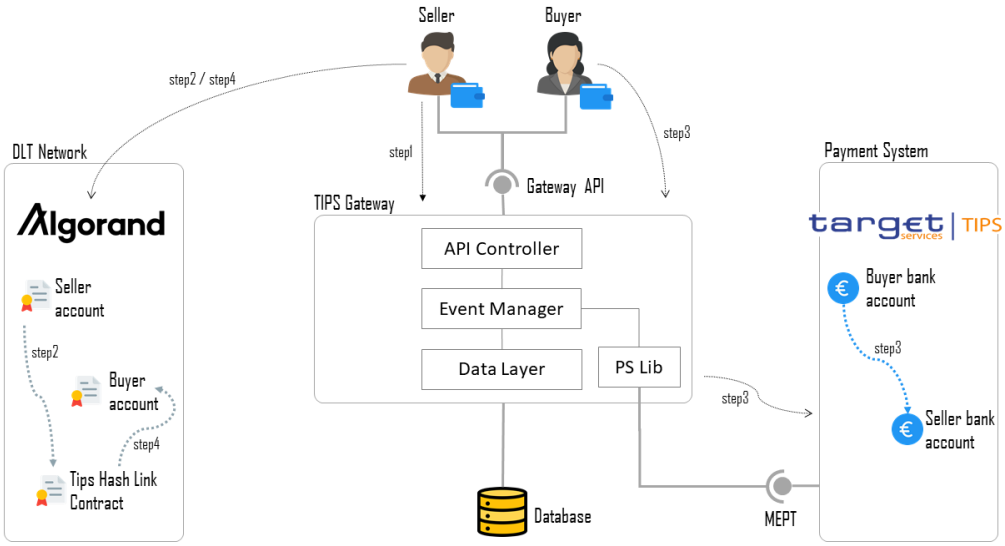

Figure 6. The TIPS Hash Link proof of concept.

3.2.1 TIPS Gateway

Software Architecture. The TIPS Gateway is a modular application written in Java, and composed of the following modules:

- The *API Controller* exposes the functionalities to initialize the DvP (as in Step1), to forward payments messages to the payment system (as in Step3), and to *force* either the cancellation or the execution of the DvP. It also executes preliminary checks on the validity of the received messages, and translates them into the corresponding events that will be internally forwarded to the Event

Manager. The API is designed to be *asynchronous*: when the controller receives a request, it dispatches the corresponding event to the Event Manager, and sends a response back to the client, indicating that the request has been successfully received and taken in charge. The result of the request, once ready, will be sent back via a callback URL provided by the client.

- The *Event Manager* oversees the overall DvP flows, coordinating the interactions with the other systems as well as Buyer's and Seller's client. It is an asynchronous component that activates upon reception of internal events. This allows to easily scale the TIPS Gateway when the system is overloaded.[32] The Event Manager manages the DvP phases as follows: in the *initialization phase* (Step1), it creates a new DvP descriptor—filled with a unique DvP identifier, execution and cancellation preimages, Seller and Buyer identifiers— and stores it using the Data Layer. To compute the public preimage commitments, TIPS Gateway uses the SHA2 hash function.[33] Finally, to solve the DvP in case of dispute, the event manager starts a timer that locks the DvP for a predefined amount of time (i.e., timelock). In the *payment phase* (Step3), the Buyer pays the Seller through gateway, which forwards the request to the TIPS system. Then, it updates the DvP instance according to the TIPS response, and notifies Buyer and Seller of the payment outcome. In the *forced execute phase*, it queries TIPS to get information about the existence of a payment with a specific identifier and, if it exists, it provides the execution preimage to the Buyer. In the *Forced cancellation phase*: it checks that the timeout provided for the DvP is expired. If it is, it queries TIPS to get information about the payment with the specific identifier. If it does not exist, it prevents any future payment, for the specific DvP id, from being entered and, finally, it provides the cancellation preimage to the Seller.
- The *Data Layer* provides primitives to store and retrieve a DvP descriptor, which holds information regarding an instance of DvP transaction between Seller and Buyer; the DvP descriptor stores data and metadata such as the DvP and assets identifiers, preimages and timelocks. The Data Layer allows decoupling the persistence layer from the other components of TIPS Gateway.
- The *PS Library* (PS Lib) uses message queues to communicate with TIPS, either by sending instructions or by receiving messages signaling the occurrence of events that need to be forwarded to the Event Manager. It has been designed to introduce an abstraction layer over different payment systems, and to define a simple interface that masks the logic needed to operate on a specific one.

Gateway Application Programming Interface (API). The *API Controller* exposes simple APIs enabling users to access the TIPS Gateway functionalities, while controlling the exposed objects and actions, and hiding the underlying implementation details. To design the TIPS Gateway with ease of interoperability with existing standards, the API Controller has been designed considering two principles: REST-like API definition and compliance with standards for data transmission. The REST-like API definition allows to simplify the Buyer/Seller interaction with the TIPS Gateway. Due to the asynchronous nature of the communication, the designed APIs are asynchronous as well: they return HTTP response status codes, informing whether the request has been taken in charge or not. To receive TIPS Gateway answers, the customers are requested to expose specific endpoints. In a real world scenario, these endpoints can be hosted by a trusted party (e.g., a commercial bank as currently occurs for payments managed by TIPS); however, in this experiment, no intermediaries are considered between Buyer/Seller and TIPS. The designed APIs accept a payload defined according to the design principles of the pan-European interoperability standard for payment systems defined by The Berlin Group[34]. Moreover, the APIs take into account the work conducted so far within the

32 In an asynchronous application, commands are encapsulated in events that are stored in internal queues, waiting for workers able of executing them.

33 The experiment uses SHA2 (Dang, 2015), although other hash functions can be used as long as the DLT supports them (e.g., Dworkin, 2015, which is already supported by several DLTs, including Algorand).

34 The Berlin Group (https://www.berlin-group.org) is a pan-European payments interoperability standards initiative and

Financial Stability Board (FSB) Stage3 Roadmap[35]. Specifically, business information is carried-out by ISO20022[36] (`pain` and `pacs`) messages. This promises to easily support future services and needs. Also, the usage of ISO20022 messages would allow a simple connectivity through existing Network Service Providers for TIPS, which offer connectivity services via XML message. More details about the APIs can be found in Appendix B. When the API Controller receives a request, it performs a preliminary validation and translates it into an internal event that will be forwarded to the Event Manager. While doing so, the API Controller sends an HTTP response back to the request client, indicating that the request has been successfully taken in charge (i.e., `202 ACCEPTED` status code). Otherwise, an error code is returned.

Infrastructure. From an infrastructure point of view, the TIPS Gateway is deployed in Bank of Italy private cloud.

3.2.2 Algorand Hash-Link Contract (HLC)

The implementation of TIPS Hash-Link is quite straightforward because it has a low coupling with the DLT: The TIPS Gateway manages the creation of execution and cancellation preimages, and accepts payments from Buyer; The Seller is in charge of creating the HLC on the Algorand DLT; The Buyer has to perform the payment, and then complete the DvP in a cooperative or forced manner.

Particularly interesting for the Algorand side of this DvP model is the Step2, in which the Seller transfers the assets from his account to an HLC, acting as an escrow and managing the assets-leg settlement. It is important to note that the HLC definition lies outside the responsibility perimeter of the TIPS Gateway, and its development can be left entirely to the market. Nevertheless an example of such smart contract template for the Algorand blockchain is provided to the reader in this section as a reference.

On Algorand, the HLC can be implemented using either a smart contract or a smart signature, and this PoC implements HLC using smart signatures. Specifically, an HLC is created using a TEAL template with the following parameters: `S`, address of Seller account on Algorand; `B`, address of Buyer account; `HE`, hash of execution preimage; `HR`, hash of cancellation preimage; `TIPS_GW_TX_ID`, DvP identifier generated by TIPS Gateway; `ASSET_ID`, assets to exchange (i.e., ASA). The compiled parameterized template produces the smart signature that populates the transaction Logic Signature; note that, by changing the template parameters (e.g., another DvP involving the same Seller and Buyer), also the generated smart signature is different.

To publish the HLC on the Algorand blockchain, the Seller creates an *atomic transfer*[37] with the following three transactions. With the first transaction, the Seller deposits some Algos in the HLC (i.e., 0.201 A): This amount represents the minimum amount of Algos necessary for the HLC to execute the steps required by the protocol. Further details in Appendix A. The second transaction prepares the HLC account to receive the ASA representing the assets, i.e., opt-in; this is a mandatory operation for an account to receive an ASA in Algorand. With the third transaction, the Seller sends the ASA representing the assets to the HLC. As a whole, this atomic transfer creates and funds an HLC acting as an escrow.

API framework with the aim of defining open and common schemes that every payment system should adopt to reach a concrete harmonization.

35 Financial Stability Board (FSB) Stage3 Roadmap and in particular the Building Block 15 “Harmonising API protocols for data exchange” is a standardization approach for the definition of a global standard for the API applied to the cross-border usage. See: https://www.fsb.org/2020/10/enhancing-cross-border-payments-stage-3-roadmap.

36 https://www.iso20022.org

37 https://developer.algorand.org/docs/get-details/atomic_transfers

$$AtomicTransfer \begin{bmatrix} PayTx & \left\{ \begin{array}{ll} amount: & 0.201A \\ receiver: & HLC \end{array} \right\}_{Seller} \\ OptInTx & \left\{ asset: \quad \text{asset} \right\}_{LogicSig(HLC)} \\ AssetTransferTx & \left\{ \begin{array}{ll} asset: & \text{asset} \\ receiver: & HLC \end{array} \right\}_{Seller} \end{bmatrix}$$

The DvP completes on the DLT with one of the following scenarios: cooperative execution, cooperative cancellation, forced execution, or, alternatively, forced cancellation.

In *Cooperative Execution*, Seller is notified about the payment and publishes a transaction on Algorand to move the assets on the Buyer's account. In Algorand, this requires to create an atomic transfer with three transactions. The first is a technical transaction (with amount 0) needed to encode through the digital signer, the Seller's willingness to perform the operation. This transaction piggybacks (using the note[38] field) the DvP identifier. The second transaction moves assets to the Buyer. The third transaction gives back the deposit to the Seller. This atomic transfer also closes the HLC account: the DvP is indeed completed.

$$AtomicTransfer \begin{bmatrix} PayTx & \left\{ \begin{array}{ll} amount: & 0 \\ receiver: & HLC \end{array} \right\}_{Seller} \\ AssetTransferTx & \left\{ \begin{array}{ll} asset: & \text{asset} \\ receiver: & Buyer \end{array} \right\}_{LogicSig(HLC)} \\ PayTx & \left\{ \begin{array}{ll} amount: & 0 \\ receiver: & Seller \end{array} \right\}_{LogicSig(HLC)} \end{bmatrix}$$

In *Cooperative Cancellation*, Buyer and Seller jointly agree to abort the DvP. The Buyer submits, to the Algorand DLT, an atomic transfer to return the assets to the Seller. This atomic transfer looks like the one used in cooperative execution, with the transaction signer and assets receiver accordingly set: Buyer signs the transaction; Seller receives the assets.

In *Forced Execution*, the Buyer queries the TIPS Gateway to obtain the execution preimage. TIPS Gateway checks with TIPS whether the payment has been performed by the Buyer for the DvP. If so, it returns the preimage. The Buyer can now submit an atomic transfer on the DLT to unlock the assets from the escrow by providing his signature and the execution preimage to the HLC. Also in this case, the atomic transfer includes three transactions: The first runs the HLC with the Buyer signature and the cancellation preimage; the second transfers the assets; the last returns the HLC deposit to the Seller.

38 https://developer.algorand.org/docs/get-details/transactions/transactions/#common-fields-header-and-type

$$AtomicTransfer \begin{bmatrix} PayTx & \{amount: 0, receiver: HLC\}_{Buyer} \\ AssetTransferTx & \{asset: asset, receiver: Buyer\}_{LogicSig(HLC)} \\ PayTx & \{amount: 0, receiver: Seller\}_{LogicSig(HLC)} \end{bmatrix}$$

In *Forced Cancellation*, the Seller queries the TIPS Gateway to obtain the cancellation preimage. TIPS Gateway checks whether the DvP timeout is expired. If it is expired, the gateway interact with TIPS to check whether Buyer performed the payment. If the payment does not exist, TIPS prevents any future payments for the DvP. Hence, TIPS Gateway returns the cancellation preimage to the Seller, who can submit an atomic transfer to unlock the assets. The atomic transfer includes three transactions: The first runs the HLC with the Seller signature and the cancellation preimage; the second transfers the assets to the Seller's account; the last returns the HLC deposit to the Seller.

Further details on the HLC design can be found in Appendix A.

3.3 TIPS-Algorand Just In Time Locking DvP

This section describes the PoC implementing the Just In Time Locking DvP model described in Section 2.3.1, where a DvP Orchestrator coordinates operations between the TIPS and Algorand platforms introduced in Section 3.1. The core component of the PoC is the *TIPS-Algorand Orchestrator*, represented in Figure 7, which implements the business logic to carry out the DvP.

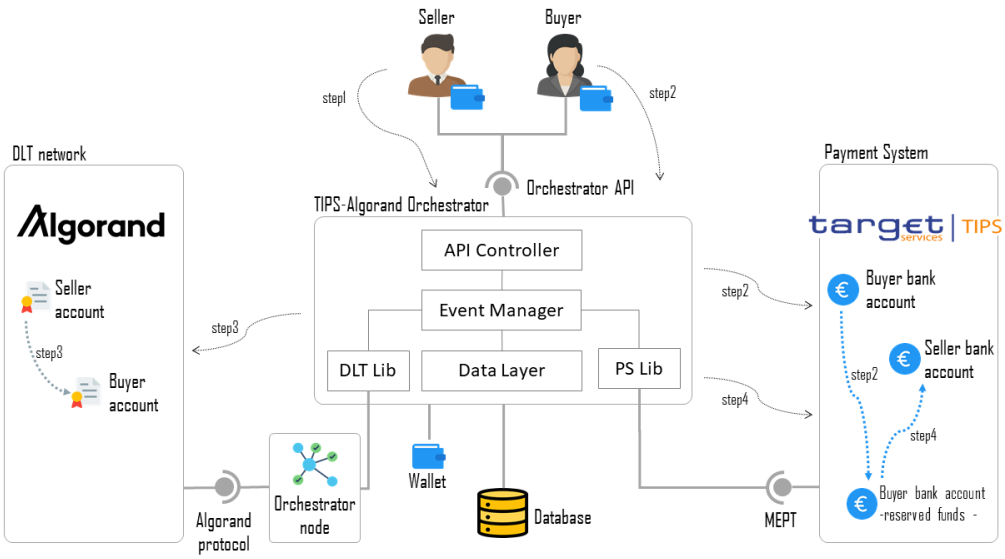

Figure 7. Tips-Algorand JITL proof of concept.

3.3.1 TIPS-Algorand Orchestrator

Software Architecture. From a software developer point of view, the orchestrator application is a modular software written in Java, and composed of the following modules:

- The *API Controller* exposes the functionalities to initialize the DvP (as in Step1) and to forward payments messages to the PS (as in Step2). It also executes preliminary checks on the validity of the received messages, and translates them into the corresponding events that will be internally forwarded to the Event Manager.
- The *Event Manager* handle the events coming from other software modules (e.g. the API controller), and manages the DvP flows, orchestrating the interactions with the other software components. In the *initialization phase* (Step1) it creates a new DvP descriptor, with a new unique identifier, and stores it. Then, it prepares an atomic transfer skeleton whose transactions must each be signed by the corresponding DvP actors (see details later in the section). In the *execution phase* (Step2 and Step3), it tries to reserve the buyer funds on TIPS. If the reserve completes successfully, it signs the remaining transactions in the package, and submits it to the DLT. Once the transactions have been finalized on DLT, the Event Manager proceeds with the confirmation of the transaction on TIPS. If any error occurs between the reserve on TIPS and the transaction finalization on the DLT, a reject and cancellation request will be send to TIPS. Finally, in the *cancellation phase*, if the DvP timeout expires before the payment occurs, then the Event Manager destroys the transactions signed during the initialization phase to prevent further DvP executions. As detailed in Section 2.3.1, by overseeing the DvP settlement, the TIPS-Algorand Orchestration has to be a trusted component of the system.
- The *Data Layer*, is used to store and retrieve DvP records in the database, in particular the pre-signed DvP transactions that are delivered to the orchestrator by Buyer and Seller in Step1 of the process.
- The *PS Library* (PS Lib), uses message queues to communicate with TIPS, either by sending instructions or by receiving messages signaling the occurrence of events that need to be forwarded to the Event Manager.
- The *DLT Library* (DLT Lib), is a new software module developed for the experiment phase, which provides an high-level abstraction layer over different DLTs. The library defines a simple interface that masks the logic to operate on a specific DLT. Interacting with a DLT usually introduces both some complexity, and often, a formalism with technical details that are specific to the chosen DLT. For example, each DLT interaction usually implies the execution of some transactions that need to be formed with a specific DLT formalism in mind. In order to keep the software modular and maintainable, the implementation of the DLT integration logic has been grouped within a library and abstracted to the above layers. This allows exposing an high level API. The current DLT lib implementation supports Algorand, and uses the Algorand Java SDK[39].

Infrastructure. The TIPS-Algorand Orchestrator is deployed in the Bank of Italy private cloud, just like the PS. It has to directly reach the Algorand network: to this end it communicates with an Algorand node hosted by Purestake through the public Internet. Finally, it also manages a wallet for the Algorand blockchain, that is used to sign transactions, as we will detail later.

3.3.2 Algorand Atomic Transfer

This PoC implements an improved version of the protocol, which modifies the pre-signed transaction used in the first step of the JITL DvP model. The pre-signed transaction is actually implemented as a package of transactions, which are executed atomically on the Algorand DLT using its *atomic transfer* functionality.

The DvP package is composed by a set of seven transactions, whose purpose is the transfer of the assets and a Locked Liquidity ASA (LLA), representing reserved funds on the Algorand DLT:

- The first and the second ensure that Buyer and Seller are entitled to the ownership of the LLA

39 Algorand Java SDK: https://developer.algorand.org/docs/sdks/java/.

(opt-in);

- The third transfers the LLA, initially owned by the TIPS-Algorand Orchestrator in its own wallet, to the Buyer;
- The fourth and the fifth transactions represent the actual DvP: they respectively transfers the LLA from Buyer to Seller, and the assets from Seller to Buyer;
- The sixth transfers the LLA from Seller to the TIPS-Algorand Orchestrator;
- Finally, the seventh transaction destroys the LLA.

The package of the atomic transfer transactions is generated by the TIPS-Algorand Orchestrator and initially signed only by Buyer and Seller. The partially signed package is transferred to the Orchestrator (Step1), which will in turn sign the remaining unsigned transactions, just before Buyer's funds are reserved (Step2). Then, the Orchestrator broadcasts the final version of the atomic transfer to the DLT nodes for confirmation (Step3). The actual transfer of the assets takes place thanks to the fifth transaction, while all the other transactions are necessary to ensure that the LLA circulates from TIPS-Algorand Orchestrator to Buyer, from Buyer to Seller, and from Seller back to the Orchestrator, which finally destroys it. In this way, it is not possible to use the LLA outside of a specific DvP. When the atomic transfer is processed by the DLT, the TIPS-Algorand Orchestrator proceeds with the confirmation of the payment depending on the atomic transfer result: the payment is performed only if the DLT successfully confirms all the transactions in the atomic transfer.

The following diagram shows the details of the package of transactions composing the atomic transfer just described, used in the PoC. The subscript at the end of each transaction indicates the transaction sender, which coincides with its signer. The use of a LLA, and the package of transactions in place of a single transaction that simply transfers the asset from Seller to Buyer, further improves the security of the arrangement, because it always requires an additional signature from the Orchestrator. A drawback of this approach is an increased complexity of the transaction, that requires the use of an ASA.

$$
AtomicTransfer \left[\begin{array}{lll}
OptInTx & \{asset: & LLA\}_{Buyer} \\
OptInTx & \{asset: & LLA\}_{Seller} \\
AssetTransferTx & \{asset: \ LLA, \ receiver: \ Buyer\}_{Orchestrator} & \\
AssetTransferTx & \{asset: \ LLA, \ receiver: \ Seller\}_{Buyer} & \\
AssetTransferTx & \{asset: \ asset, \ receiver: \ Buyer\}_{Seller} & \\
AssetTransferTx & \{asset: \ LLA, \ receiver: \ Orchest.\}_{Seller} & \\
AssetDestroyTx & \{asset: & LLA\}_{Orchestrator}
\end{array}\right.
$$

4 Tests and Results

In order to show the feasibility of the designed solutions and analyze some specific flows, the two implemented DvP models were tested. Section 4.1 presents a suite of test cases evaluated for both the DvP solutions: test cases investigate the normal operation (i.e., cooperative execution) as well as exceptional flows (i.e., forced execution, forced cancellation, failures, duplicate requests).

To get a rough idea of the DvP solutions' performances, Section 4.2 presents a simple derivation

of throughput and complexity of TIPS Hash-Link (T-HL) and TIPS-Algorand Just-in-time Locking (TA-JITL). We combined this approach with systematic latency measurements, aiming at reaching robust conclusions regarding performances.

4.1 Test scenarios

For TA-JITL the tests mainly covered the TIPS Gateway component because it manages the whole logic. In T-HL, instead, also the smart contract was tested, but in a separated way, because it is thought to be independent and with a different ownership from the TIPS Gateway. Also complete end-to-end tests were executed in both protocols. For T-HL they only covered the initialization phase of all use cases and the final phases of *forced* use cases only: *cooperative* ones are based upon an agreement between buyer and seller, external to TIPS, and then do not need the TIPS Gateway intervention for the final phases.

Regarding the TIPS Gateway, the designed test scenarios depend on the available operations offered by the specific integration model. Since the two models are different, also the test cases need to be different. In particular, for the Hash-Link protocol, the test scenarios depend on the following operations: initialization, payment, forced execution and forced cancellation. For the TA-JITL protocol, instead, the test scenarios are restricted to three operations: initialization, execution and cancellation. The only common element is the way the tests were executed. In order to facilitate this step an ad-hoc software module was developed. It is written in Java and provides a controller that exposes a set of APIs that allow to manage the several flows to be tested. It orchestrates the invocations to the Buyer and Seller, triggering the needed actions time after time.

Tables 2 and 3 list the test cases executed for Hash-Link and Just-In-Time Locking respectively. The first column contains a brief description of the scenario, the “Initial state” column is the set of the conditions that has to be met before the event specified in the column “Action” is triggered, and in the “Expected result” column is described what has to happen in order for the test to complete without errors.

Table 2. Hash-Link test cases.

Scenario	Initial state	Action	Expected result
Bad init	-	init, wrong data	Error: operation not permitted
Good init	-	init	New DvP generated, hashed preimages returned
Bad cancellation after init	init	forced cancellation before timelock	Error, no preimage returned
Execution after init	init + no pay	forced execution	Error, no preimage returned
Good cancellation after init	init + no pay	forced cancellation after timelock	Cancellation preimage returned
Pay	init	pay (before timelock)	Payment forwarded to TIPS, result returned
Pay after timelock	init	pay (after timelock)	Error, payment rejected
Double payment	init + pay	pay	Error, payment rejected
Cancellation after pay	init + pay	forced cancellation	Error, no preimage returned
Execution after pay	init + pay	forced execution	Execution preimage returned
Cancellation after execution	init + pay + execution	forced cancellation	Error, no preimage returned
Double execution	init + pay + execution	forced execution	Execution preimage returned
Pay after cancellation	init + cancellation	pay	Error, payment rejected
Execution after cancellation	init + cancellation	forced execution	Error, no preimage returned
Double cancellation	init + cancellation	forced cancellation	Cancellation preimage returned
Execution after wrong pay in TIPS	init + pay wrong data[g] in TIPS	forced execution	Error, no preimage returned

[g] This case evaluates the TIPS-Gateway ability to check whether the Buyer’s payment matches what was agreed in the DvP initialization. Payment mismatches may occur if the Buyer maliciously performs the payment in TIPS directly, without interacting with the TIPS-Gateway, which holds DvP-related information.

Table 3. Just-In-Time Locking test cases.

Scenario	Initial state	Action	Expected result
Bad init	-	init, wrong data	Error: operation not permitted
Good init	-	init	New DvP generated, DLT transactions prepared
Bad cancellation after init	init	cancellation before timelock	Error: operation not permitted
Good cancellation after init	init	cancellation after timelock	Transactions deleted
Execution with cash-leg error	init	cash-leg error	Error, data cleaned on TIPS and DLT
Execution with asset-leg error	init	asset-leg error	Error, data cleaned on TIPS and DLT
Wrong execution after timelock	init	execution after timelock	Error, reservation rejected
Execution	init	execution before timelock	DvP executed on TIPS and DLT
Cancellation after execution	init + execution	cancellation	Error: DvP in executed state
Double execution	init + execution	execution	Error: DvP already executed
Execution after cancellation	init + cancellation	execution	Error: DvP in cancelled state
Double cancellation	init + cancellation	cancellation	Error: DvP already cancelled

Regarding the smart contract, two wallets, representing the Buyer and the seller, were created in the Algorand TestNet[40] and were used to exchange the assets on the DLT. The DLT operations involving the two accounts and, for TIPS Hash-Link, the flows allowed by the smart contract were validated using AlgoExplorer[41] as counterproof. Moreover, the smart contract was also tested from a security point of view, trying to manipulate it with a set of tampering attempts: all the executed attempts failed, without compromising the smart contract and then the correctness of the nested flows. More details about the smart contract can be found in Appendix A.

4.2 Results

The two solutions can be compared from a performance perspective. Every DLT has two key performance indicators: block time and block size. The block time is the time between two consecutive blocks confirmations, whereas the block size is the maximum number of transactions that can be added in a single block. Currently the Algorand DLT has a block time (t) of 4.5 seconds and a block size (k) of 5000 transactions.[42] Neglecting the execution time and delays introduced by the Buyer, Seller, and Interoperability Solution, the finalization of a DvP requires two blocks, both for the T-HL solution and for the TA-JITL solution. As detailed in Section 3, the T-HL protocol requires the execution of two Atomic Transfer operations: one for the HLC creation and another for the DvP settlement; likewise, TA-JITL needs a block for the creation of the LLA; once the LLA is created, its identifier can be included into the atomic transfer with asset exchange in a second block. Since the first transaction (i.e., HLC or LLA creation) can be broadcasted immediately after (hence, with a waiting time of t) or immediately before a block is published (with no waiting time): its average approval time is therefore $\frac{1}{2}t$, with t the block time. To publish the second atomic transfer, it is always necessary to wait at least for another block time t. Therefore, the average minimum completion time $\bar{C}(t)$ of the asset-leg settlement on the DLT is on average $\frac{3}{2}t$ for both solutions:

$$\bar{C}(t) = \frac{1}{2}t + t \tag{1}$$

Under the assumption that the DLT is not overloaded, and Buyer, Seller, and the Interoperability Solutions are the only active actors participating in the DvP, the maximum DLT throughput can be computed. The maximum throughput (T) of both the solutions depends on the number of transactions (n) required for each DvP, the number of transactions the DLT can include in each

40 Algorand offers three public networks where the blockchain is deployed with different purposes: MainNet, TestNet, and BetaNet (https://developer.algorand.org/docs/get-details/algorand-networks/). This PoC uses the TestNet, which allows testing application with the current version of the protocol and realistic network conditions.

41 https://algoexplorer.io/

42 https://developer.algorand.org/docs/get-started/basics/why_algorand/#performance

block k, and on the block time t, as follows:

$$T(k,t,n) = \frac{k}{nt} \tag{2}$$

Given that T-HL requires 6 transactions to complete a DvP, while TA-JITL requires 8 transaction, from (3) follows that the former has a theoretic throughput that is 33% higher than the latter:

$$\frac{T_{T-HL}(k,t,n) - T_{T_A-JITL}(k,t,n)}{T_{T_A-JITL}(k,t,n)} = \frac{\frac{k}{6t} - \frac{k}{8t}}{\frac{k}{8t}} = \frac{1}{3} \sim 33\% \tag{3}$$

Table 4 summarizes theoretic performance of the two DvP solutions: T-HL and TA-JITL. All the calculations are theoretical, and strictly dependent on the Algorand block time t and block size n. They however provide a meaningful estimation of the performances of the two solutions, since the interactions with the DLT are the only parts of the schemes neither tunable nor scalable in the scope of the current paper.

Table 4. Theoretic Performance of TIPS Hash-Link (T-HL) and TIPS-Algorand Just-in-time Locking (TA-JITL).

Performance Metric	$T-HL$	$TA-JITL$[h]
Average completion time (s)	6.75	6.75
Throughput (DvP/s)	185	139

[h] The implemented variant of the protocol was presented for its simplicity. Other definitions that reduce the number of required transactions improving both performance metrics can be defined.

The improvement just of the blockchain parameters (k and t) would result in better performance for the two DvP solutions, both in terms of minimum completion time and maximum throughput.

4.3 Discussion

While showing similar performance in terms of average completion time, T-HL results in higher throughput; this is motivated by the usage of fewer transactions to complete the DvP with respect to TA-JITL. The experiment outcome provides further confirmation of the opportunities offered by the adoption of an API-based DvP with Hash-Link Contract model to solve cross-ledger DvPs while keeping a low coupling with the DLT, as discussed in Section 2.

The experiment allowed also to compare the two solutions against the effort needed for their realization. In addition to handling the life cycle of the DvP, the TA-JITL protocol requires the development of a software component that has also to manage the wallet on the DLT and has to operate a DLT node to execute operations on it. All these activities would be the responsibility of developers and maintainers of the Interoperability Solution. The off-chain communication among Buyer, Seller, and the trusted oracle could be offered as an additional feature of the Interoperability Solution (e.g., offered by the TIPS Gateway as done in the experiment) or as a value-added third party service. Also in the latter case, the Interoperability Solution needs to interact with the third party, thus increasing its functioning responsibilities and workload. The TA-JITL solution is closely intertwined with the Algorand DLT so as to make full use of the Algorand technical features and constructs. Although interesting, this solution has some peculiarities that could be limiting in some use cases (e.g., when the Interoperability Solution must not own a wallet).

Being a more general solution, T-HL promises an easy integration with other DLT technologies, as it requires neither direct DLT interaction nor wallet management. This is a relevant key point for a central bank that aims to preserve neutrality by not clinging to specific technologies or suppliers. Nonetheless, T-HL assumes a solid definition of the HLC template, which is crucial to avoid DvP threats or failures. To this end, HLC templates should be carefully developed, extensively tested,

and then conveniently secured. These activities could be performed by a third party service. The PoC does not focus on the definition of a solid HLC template, being outside the responsibility of the Interoperability Solution developers. Conversely, TA-JITL uses only simple DLT transactions, which require the DvP actors to pay special attention only to the transactions to be signed. In a sense, T-HL shifts the complexity of the asset-leg settlement on the smart contract definition instead of the transaction orchestrations performed by TA-JITL.

5 Conclusions

This paper investigates different interoperability models for a cross-ledger DvP, between assets residing on a DLT, and the cash residing on the ledger of a different payment system (e.g., operated by a central bank, for settlement in central bank money). In particular, four DvP models are identified, and classified according to whether the Interoperability Solution acts as a gateway for the payment system, or as a DvP orchestrator between the payment system and the distributed ledger. The different models are analyzed across different dimensions, including—among the others—the DLT agnosticism, presence of a trusted DvP oracle, safety, liquidity efficiency, absence of a free option. The analysis favored cash-leg locking efficiency - as money is exchanged more frequently than assets - which should be achieved without any particular deterioration as the number of connected DLTs increases.

Afterwards, the paper describes a detailed experiment, which instantiates the payment system as TIPS (for providing the settlement services of the DvP cash-leg), and the DLT as the Algorand blockchain (for regulating the asset-leg of the DvP). The experiment implements two PoC for two different DvP models: TIPS Hash-Link is an instance of the API-based DvP with Hash-Link Contract model; and TIPS-Algorand Just In Time Locking is an instance of the Orchestrated DvP with Just In Time Locking model.

The experiment confirmed the feasibility of the chosen DvP models as viable solutions for enabling interoperability between a DLT and the Eurosystem Market Infrastructures.[43] For both solutions, the ability to address a potentially extended use of digital asset management services with central bank money settlement services has emerged. The flexibility offered by these models allows them to be applied to retail business cases, taking into consideration the new ongoing experiment being conducted with external market players, such as ABI[44].

In the current context of growing interest in transactions involving the exchange of digital assets—such as stablecoins, security tokens, non-fungible tokens and so on—this experiment demonstrated a possible usage of central bank money for cash settlement. In particular, the proposed implementations can effectively contribute to safer and more efficient settlement of DvP trades, enabling interoperability with a wide array of DLT platforms—and different degrees of integration effort compared to other already established approaches.

The implemented solutions preserve the atomic nature of a DvP transaction, building a “bridge” between the DLT asset management platform and the central bank payment system. In accomplishing this task, this architecture offers the potential to standardize communications with DLTs and, crucially, avoid the need to issue central bank money as cash tokens on DLTs—a key point of other approaches—thus preserving the central bank’s control over the cash-leg of the DvP.

Adopting such approaches could offer the opportunity to build services capable to settle transactions involving tokenized assets in central bank money, without influencing the TARGET infrastructure, monetary policy, oversight, and financial stability, and, notably, without requiring any change on the DLT side[45]. This approach would enable interoperability with DLTs, responding to

43 This in order to create a payment solution that complements “the existing” one, for assets and business cases currently not managed by other TARGET Services (e.g., T2S) since issued or represented on a DLT.

44 ABI, Associazione Bancaria Italiana https://www.abi.it/Pagine/default.aspx

45 Changes to DLTs are usually difficult to apply especially considering their nature, their adoption, and consequently the

current market trends, in a comparatively shorter time frame than approaches requiring the definition of a native digital central bank money (i.e., a cash tokenization approach) on self or third-party operated DLTs, which could better fit a medium or long term solution from the perspective of the Eurosystem's market infrastructures.

importance of getting the changes approved by the underlying communities.

References

4CB. (2019). *Message Exchange Processing for TIPS (MEPT)* (tech. rep.). International Organization for Standardization. https://www.ecb.europa.eu/paym/target/tips/profuse/shared/pdf/TIPS_MEPT_-_Message_Exchange_Processing_for_TIPS_-_v1.2_final_rev.pdf

Arcese, M., Di Giulio, D. & Lasorella, V. (2021). Real-Time Gross Settlement systems: breaking the wall of scalability and high availability. (2021-02). https://www.bancaditalia.it/pubblicazioni/mercati-infrastrutture-e-sistemi-di-pagamento/approfondimenti/2021-002/N.2-MISP.pdf

Bank of Canada, TMX Group, Payments Canada, Accenture & R3. (2018). *Jasper phase III: Securities Settlement using Distributed Ledger Technology* (tech. rep.). Payments Canada. https://www.payments.ca/sites/default/files/jasper_phase_iii_whitepaper_final_0.pdf

Banque de France, BIS Innovation Hub, Swiss National Bank & a private sector consortium. (2021). Project Jura: cross-border settlement using wholesale CBDC. https://www.bis.org/about/bisih/topics/cbdc/jura.htm

BIS Innovation Hub, Swiss National Bank & SIX. (2018). *Project Helvetia Phase I: Settling Tokenised Assets in Central Bank Money* (tech. rep.). BIS Innovation Hub. https://www.bis.org/publ/othp35.pdf

Buterin, V. (2014). *A next-generation smart contract and decentralized application platform* (tech. rep.). Ethereum Foundation. https://ethereum.org/en/whitepaper/

Chen, J. & Micali, S. (2019). Algorand: A Secure and Efficient Distributed Ledger. *Theoretical Computer Science*, *777*(100), 155–183. https://doi.org/10.1016/j.tcs.2019.02.001

Dang, Q. (2015). Secure Hash Standard. https://doi.org/10.6028/NIST.FIPS.180-4

Deutsche Börse, Deutsche Bundesbank & Germany's Finance Agency. (2021). Dlt-based securities settlement in central bank money successfully tested. https://www.bundesbank.de/en/press/press-releases/dlt-based-securities-settlement-in-central-bank-money-successfully-tested-861444

Dworkin, M. (2015). Sha-3 standard: Permutation-based hash and extendable-output functions. https://doi.org/10.6028/NIST.FIPS.202

ECB. (2010). The payment system. https://www.ecb.europa.eu/pub/pdf/other/paymentsystem201009en.pdf

ECB. (2022a). Glossary: Delivery versus payment. https://www.ecb.europa.eu/services/glossary/html/glossd.en.html

ECB. (2022b). Securities trading, clearing and settlement. https://www.ecb.europa.eu/stats/payment_statistics/securities/html/index.en.html

IBM. (2022). What are smart contracts on blockchain? https://www.ibm.com/topics/smart-contracts

MAS, SGX, Anquan Capital, Deloitte & Nasdaq. (2018). *Delivery versus Payment on Distributed Ledger Technologies: Project Ubin* (tech. rep.). Monetary Authority of Singapore. https://www.mas.gov.sg/-/media/MAS/ProjectUbin/Project-Ubin-DvP-on-Distributed-Ledger-Technologies.pdf

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 9. https://bitcoin.org/bitcoin.pdf

Poon, J. & Dryja, T. (2016). *The bitcoin lightning network: Scalable off-chain instant payments* (tech. rep.). Lightning Network. https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf

Project Stella. (2018). *Securities Settlement Systems: Delivery-Versus-Payment in a Distributed Ledger Environment* (tech. rep.). European Central Bank and Bank of Japan. https://www.ecb.europa.eu/pub/pdf/other/stella_project_report_march_2018.pdf

Renzetti, M., Bernardini, S., Marino, G., Mibelli, L., Ricciardi, L. & Sabelli, G. M. (2021). TIPS - TARGET Instant Payment Settlement Il sistema europeo per il regolamento dei pagamenti istantanei. *Markets, Infrastructures, Payment Systems*, (2021-01). https://www.bancaditalia.it/pubblicazioni/mercati-infrastrutture-e-sistemi-di-pagamento/questioni-istituzionali/2021-001/MIS-20210129.pdf

Urbinati, E., Belsito, A., Cani, D., Caporrini, A., Capotosto, M., Folino, S., Galano, G., Goretti, G., Marcelli, G., Tiberi, P. & Vita, A. (2021). A digital euro: a contribution to the discussion on technical design choices. *Markets, Infrastructures, Payment Systems*, (2021-10). https://www.bancaditalia.it/pubblicazioni/mercati-infrastrutture-e-sistemi-di-pagamento/questioni-istituzionali/2021-010/N.10-MISP.pdf

Appendices

A Hash-Link Contract: Implementation Details

This appendix aims to show the structure of the HLC smart contract and describes how the execution result of each phase is displayed on AlgoExplorer, a tool for browsing blocks of the Algorand blockchain. Below is shown the main section of the smart contract template.

```
return \
    Seq([
        Cond([
            Global.group_size() == Int(3),
            Assert(Or(prepare_checks,
                    cooperative_exec_checks,
                    cooperative_refnd_checks,
                    forced_exec_checks,
                    forced_refnd_checks,))
            ],),
        assert_security_check(),
        Approve(),
    ])
```

Listing 1. HLC: main condition

The source code in Listing 1 shows that the HLC smart contract returns an ‘‘approve’’ only if it is involved in one of the five expected scenarios: *Prepare*, *CooperativeExecution*, *CooperativeCancellation*, *ForcedExecution*, or *ForcedCancellation*.

A.1 Prepare phase

The following code section represents which checks are performed by the HLC in order to detect and approve a valid prepare phase.

```
prepare_at_checks = And(
    Global.group_size() == Int(3),
    Gtxn[0].type_enum() == TxnType.Payment,
    Gtxn[1].type_enum() == TxnType.AssetTransfer,
    Gtxn[2].type_enum() == TxnType.AssetTransfer,
)
prepare_participants_checks = And(
    Gtxn[0].sender() == seller,
    Gtxn[2].sender() == seller,
    Gtxn[0].receiver() == Gtxn[2].asset_receiver(),
    Gtxn[0].receiver() == Gtxn[1].asset_receiver(),
    Gtxn[1].sender() == Gtxn[1].asset_receiver(),  # opt-in tx
)
prepare_assets_checks = And(
    Gtxn[0].amount() >= Int(min_funding_amount),
    Gtxn[1].xfer_asset() == asset,
    Gtxn[1].asset_amount() == Int(0),  # opt-in tx
    Gtxn[2].xfer_asset() == asset,
    Gtxn[2].asset_amount() == Int(1),
)
prepare_checks = And(
    prepare_at_checks,
    prepare_participants_checks,
    prepare_assets_checks,
)
```

Listing 2. HLC: prepare checks

In particular, three kind of checks are performed:

1. The atomic transfer-related checks (`prepare_at_checks`): it expects an atomic transfer made of three transactions; the first is a Payment transaction, the second and the third are AssetTransfer transactions.
2. The participant-related checks (`prepare_participants_checks`): it expects that the sender addresses of the first and the third transactions are equal to the Seller address, and that the remaining addresses are equal to the HLC one. The logic executes, substantially, the following chain of equality:

$$Tx[0].receiver == Tx[2].receiver == Tx[1].receiver == Tx[1].sender$$

   Note that, even if the HLC address is not explicitly checked, we can infer that it is present into $Tx[1].sender$, since the HLC logic has been triggered. All the transaction parties are then either explicitly or implicitly validated.
3. The assets-related checks (`prepare_assets_checks`): here the focus is on the ASA identifier (asset id), to be used in the second and third transaction, respectively with an amount of 0 (opt-in semantics) and 1. The first check is related to the minimum amount of Algos the HLC needs in order to complete the DvP through one of the four foreseen finalization phases.

Funding amount

Currently, the funding amount check is hardcoded to 0.201 Algos (row 15 of Listing 2). This value represents the minimum amount of Algos necessary to the HLC to be able to execute the steps foreseen by the protocol.
The smart contract must be able to cover the fees associated with the operations it will have to carry out, and must own the minimum balance required by the Algorand DLT to operate. Every account, in order to be able to issue a transaction, must have a balance of 0.1 Algos, plus 0.1 Algos for each asset holding. Since the HLC, after the execution of the prepare phase, owns one ASA, its balance must never go under 0.2 Algos.
The HLC performs exactly three operations: an opt-in, during the prepare phase, an asset transfer and a pay, during the finalization phase. The execution of each of these operations costs 0.001 Algos. Then, after the prepare phase the HLC balance goes from 0.201 to 0.200 Algos, while after the finalization phase the balance is 0.198 Algos. This seems to violate the check on the minimum balance just described, but this is not the case: the check is not performed when an account is going to be closed. In particular, the finalization phase consists of an atomic transfer where the pay transaction (third transaction) has the "$close\_remainder\_to$" field set, which requires to close the HLC account, transferring all the remaining funds to the Seller. This is the reason why the amount of this pay transaction is 0 Algos. The close operation would be permitted only if the account does not own any ASA. To delete the ASA, the AssetTransfer Transaction (second transaction of the atomic transfer) has the "$asset\_close\_to$" field set: it tells Algorand that the account does not want to hold and receive assets of the type specified inside the transaction. This action is also known as `"opt-out"`, and leads to a reduction of the account minimum balance (the opposite of what happens with the `"opt-in"` operation).

Figure 8 shows the HLC account after a prepare phase is concluded. It shows the HLC balance of 0.2 Algos and 1 BOND and the 3 transactions which constitute the Atomic Transfer. Inspecting one of the three transactions it is visible also the "$Group\ ID$" of the Atomic Transfer (Figure 9).

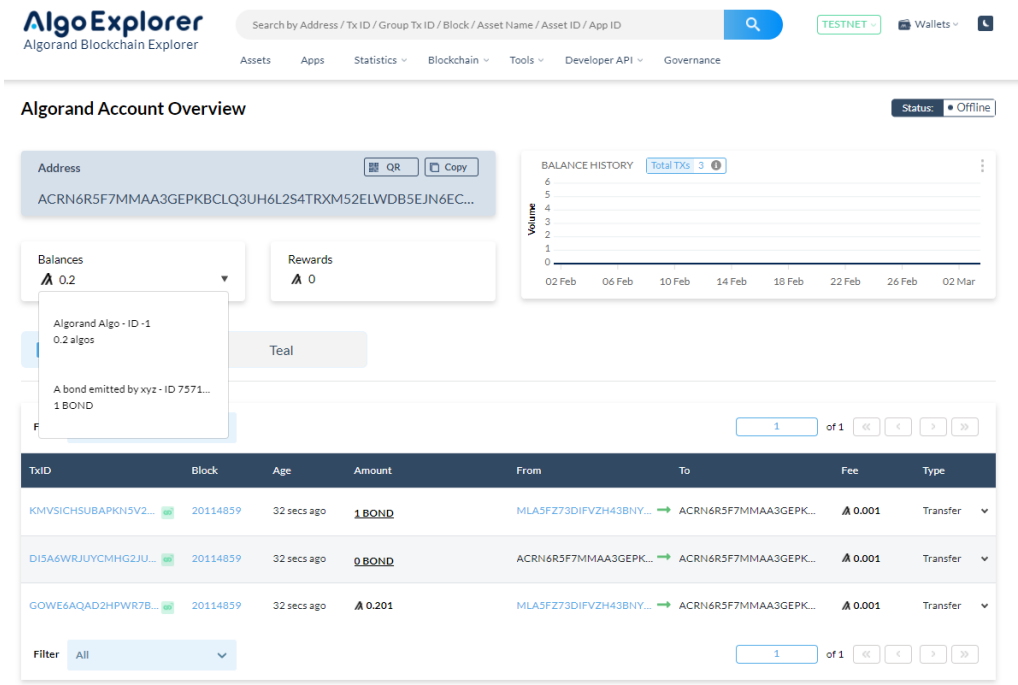

Figure 8. The prepare phase on AlgoExplorer.

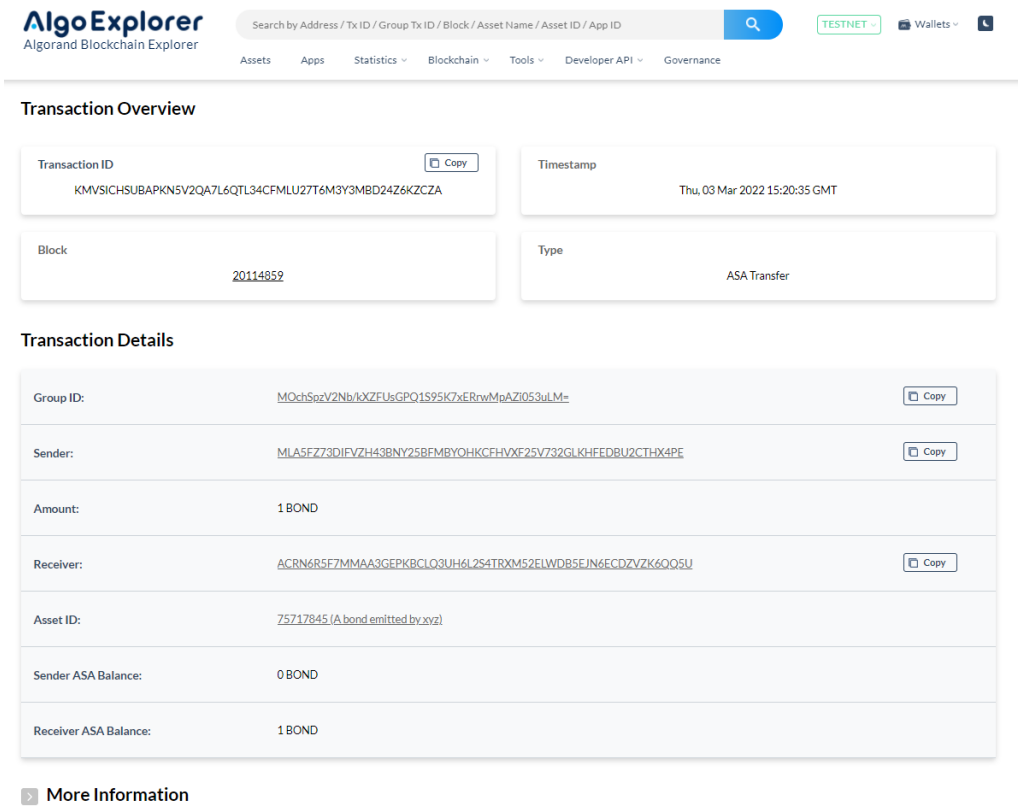

Figure 9. The prepare phase on AlgoExplorer: Detail of a transaction including the Group ID of the atomic transfer.

A.2 Finalization phases

This section describes the checks that the HLC performs in order to detect and approve a valid request of DvP finalization phase: Listing 3 for the *cooperativeExecution*, Listing 4 for the *cooperativeCancellation*, Listing 5 for the *forcedExecution* and Listing 6 for the *forcedCancellation*.

The approach is very similar to the one seen in the prepare phase (Section A.1). The checks are performed in lines 9 and 20–22 of Listing 3 (and lines 3 and 15–17 of Listings 4, 5, and 6, as well). The first validates the note field of the first transaction: it must contain the expected DvP identifier (i.e., `TIPS_GW_TX_ID`) and the expected command to execute (i.e., bilateral execution `"Bx"`, bilateral cancellation `"Br"`, forced execution `"Ux"`, or forced cancellation `"Ur"`) depending on the finalization phase scenario). The second checks relate to the `closeRemainderTo` and `assetCloseTo` transaction fields: these validations checks aim to avoid unwanted appropriation of assets or funds. For the *ForcedExecution* (*ForcedCancellation*), line 4 of Listing 5 (Listing 6) defines the validation checks for the provided exeuction preimage (cancellation preimage).

Figures 10, 12, 14 and 17 shows the HLC account after the conclusion of the respective finalization phase. It is visible the balance of the HLC, empty for both funds and assets.

The three transactions on the top of the table are related to the finalization phase, whereas the three at the bottom to the prepare phase. Following the order, row 3 includes the pay transaction with the command (details respectively in Figures 11, 13, 15 and 18); row 2 includes the asset transfer to the correct party and the opt-out of the HLC from the asset; at row 1 includes the HLC account cancellation. In Figures 16 and 19 it is visible respectively the execution and the cancellation

preimages passed into the note field of the second transaction.

```
bx_at_checks = And(
    Global.group_size() == Int(3),
    Gtxn[0].type_enum() == TxnType.Payment,
    Gtxn[1].type_enum() == TxnType.AssetTransfer,
    Gtxn[2].type_enum() == TxnType.Payment,
)
bx_asset_checks = And(
    Gtxn[0].amount() == Int(0),
    BytesEq(Gtxn[0].note(), Concat(tips_tx_id, Bytes("|Bx"))),
    Gtxn[1].xfer_asset() == asset,
    Gtxn[1].asset_amount() == Int(1),
    Gtxn[2].amount() == Int(0),
)
bx_participants_checks = And(
    Gtxn[0].sender() == seller,
    Gtxn[1].asset_receiver() == buyer,
    Gtxn[0].receiver() == Gtxn[1].sender(),
    Gtxn[0].receiver() == Gtxn[2].sender(),
    Gtxn[2].receiver() == seller,
    Gtxn[0].close_remainder_to() == Global.zero_address(),
    Gtxn[1].asset_close_to() == buyer,
    Gtxn[2].close_remainder_to() == seller,
)
cooperative_exec_checks = And(
    bx_at_checks,
    bx_asset_checks,
    bx_participants_checks,
)
```

Listing 3. HLC: cooperative execution checks

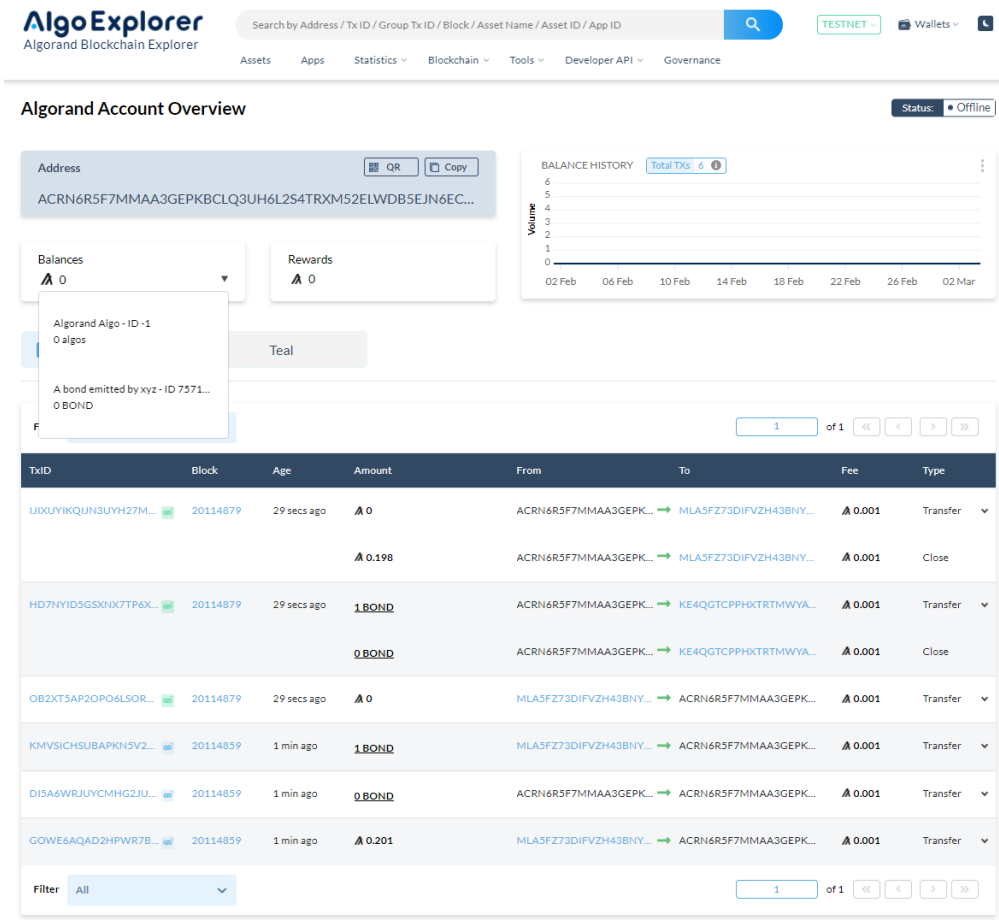

Figure 10. The cooperative execution on AlgoExplorer.

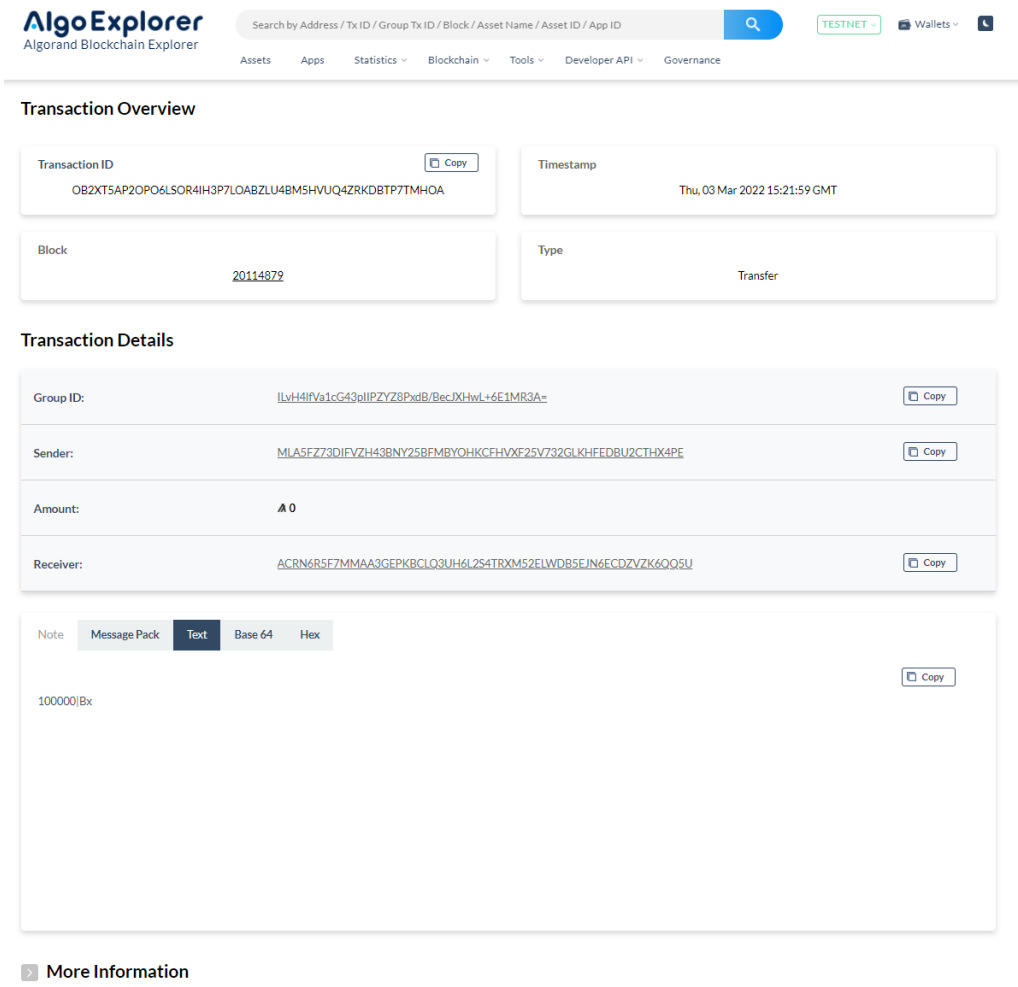

Figure 11. The cooperative execution on AlgoExplorer: Details of a transaction including the atomic transfer Group Id and notes.

```
br_asset_checks = And(
    Gtxn[0].amount() == Int(0),
    BytesEq(Gtxn[0].note(), Concat(tips_tx_id, Bytes("|Br"))),
    Gtxn[1].xfer_asset() == asset,
    Gtxn[1].asset_amount() == Int(1),
    Gtxn[2].amount() == Int(0),
)

br_participants_checks = And(
    Gtxn[0].sender() == buyer,
    Gtxn[1].asset_receiver() == seller,
    Gtxn[0].receiver() == Gtxn[1].sender(),
    Gtxn[0].receiver() == Gtxn[2].sender(),
    Gtxn[2].receiver() == seller,
    Gtxn[0].close_remainder_to() == Global.zero_address(),
    Gtxn[1].asset_close_to() == seller,
    Gtxn[2].close_remainder_to() == seller,
)

cooperative_refnd_checks = And(
```

```
    bx_at_checks,
    br_asset_checks,
    br_participants_checks,
)
```

Listing 4. HLC: cooperative cancellation checks

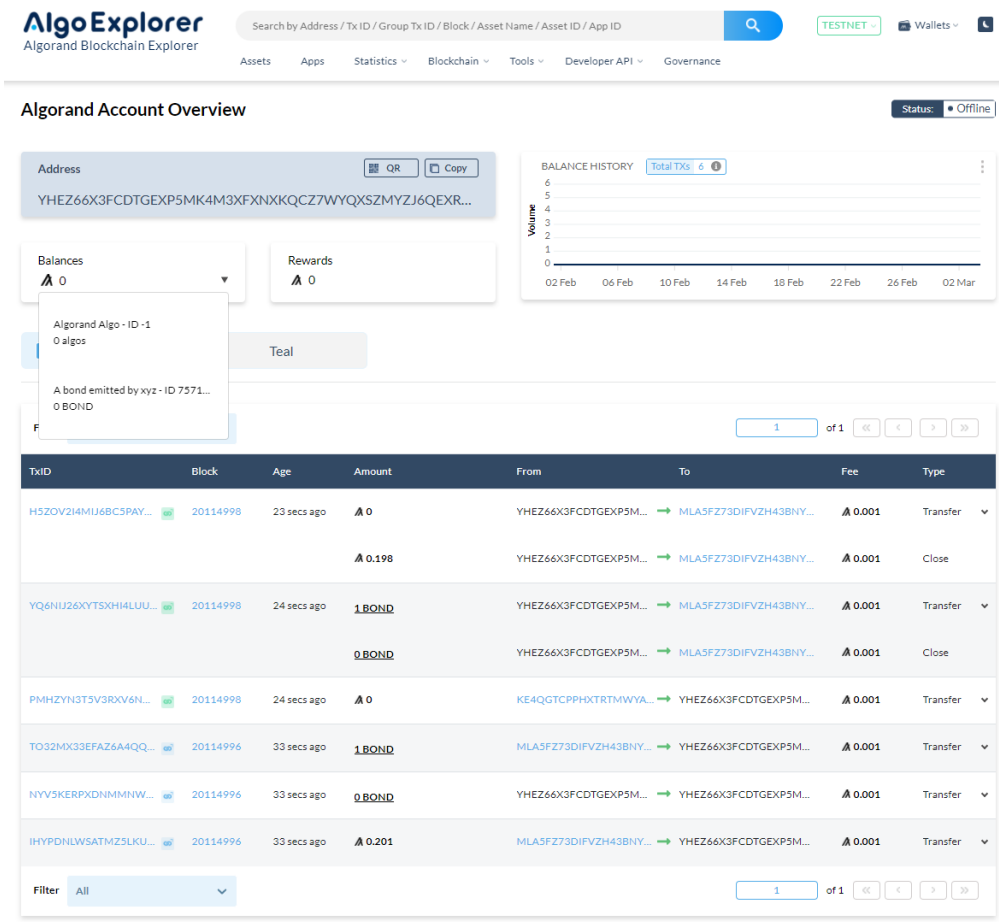

Figure 12. The cooperative cancellation on AlgoExplorer.

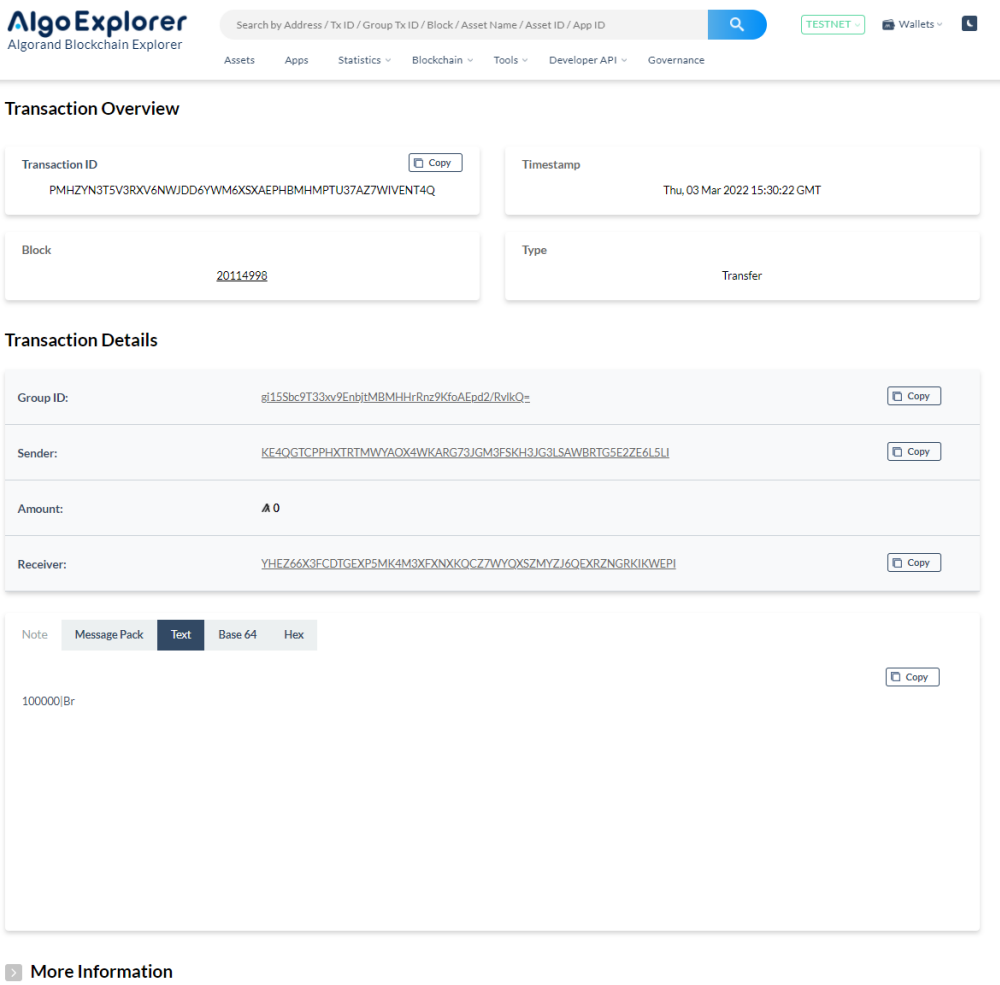

Figure 13. The cooperative cancellation on AlgoExplorer: Details of a transaction including the atomic transfer Group ID and notes.

```
ux_asset_checks = And(
    Gtxn[0].amount() == Int(0),
    BytesEq(Gtxn[0].note(), Concat(tips_tx_id, Bytes("|Ux"))),
    Sha256(Gtxn[1].note()) == exec_hash,
    Gtxn[1].xfer_asset() == asset,
    Gtxn[1].asset_amount() == Int(1),
    Gtxn[2].amount() == Int(0),
)

ux_participants_checks = And(
    Gtxn[0].sender() == buyer,
    Gtxn[1].asset_receiver() == buyer,
    Gtxn[0].receiver() == Gtxn[1].sender(),
    Gtxn[0].receiver() == Gtxn[2].sender(),
    Gtxn[2].receiver() == seller,
    Gtxn[0].close_remainder_to() == Global.zero_address(),
    Gtxn[1].asset_close_to() == buyer,
    Gtxn[2].close_remainder_to() == seller,
)

```

```
forced_exec_checks = And(
    bx_at_checks,
    ux_asset_checks,
    ux_participants_checks,
)
```

Listing 5. HLC: forced execution checks

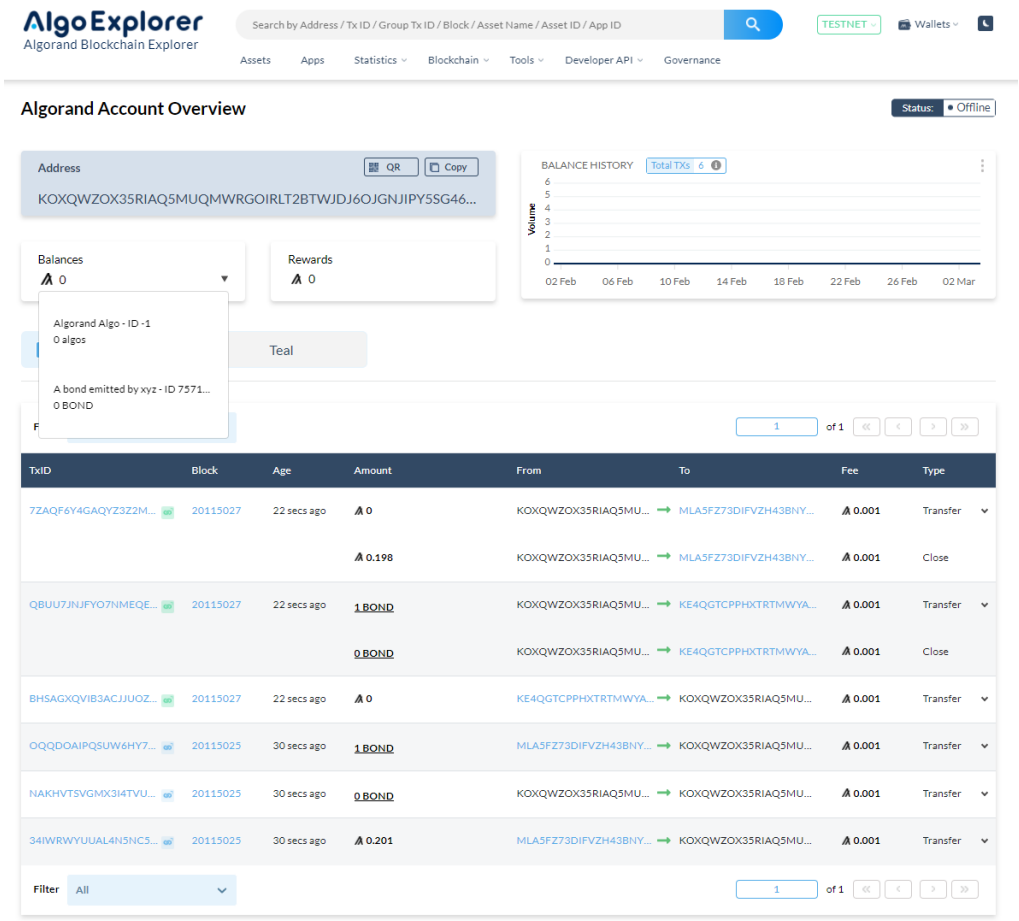

Figure 14. The forced execution on AlgoExplorer.

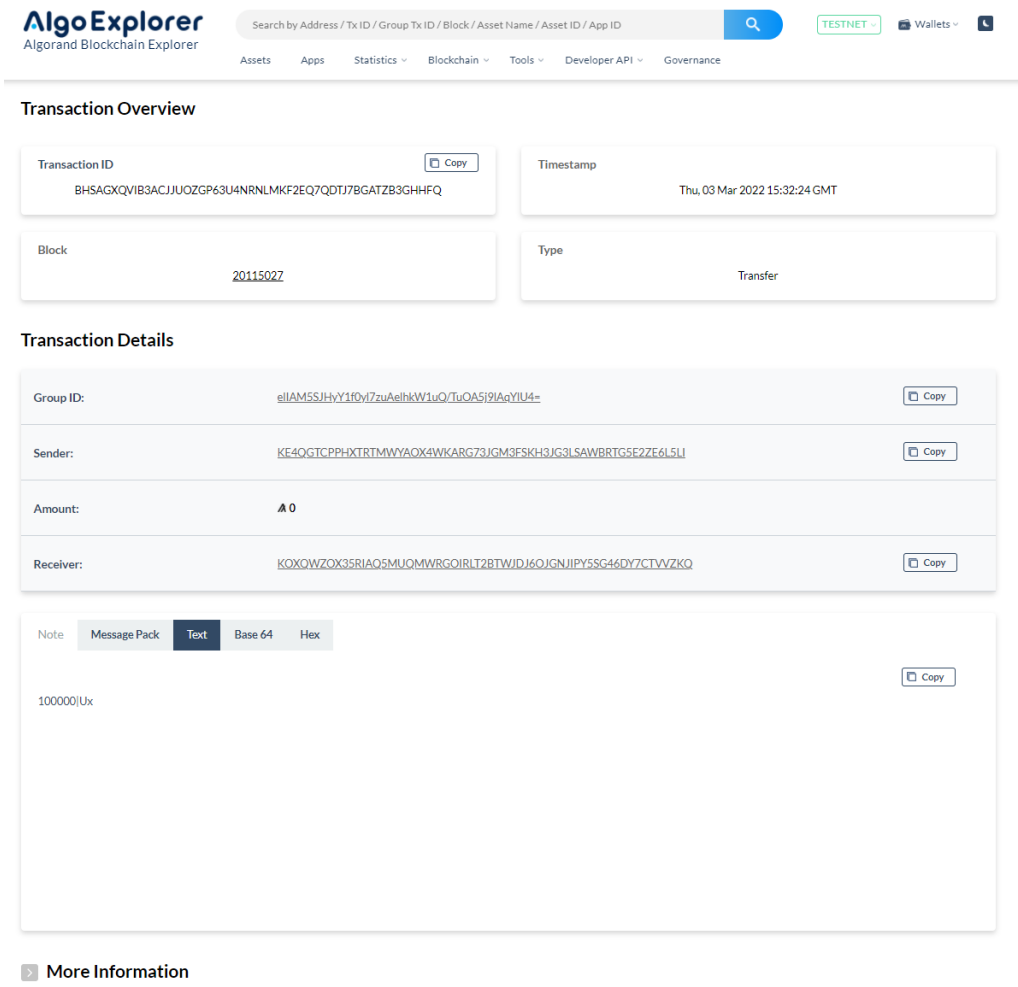

Figure 15. The forced execution on AlgoExplorer: Details of the first transaction including the atomic transfer Group ID and notes.

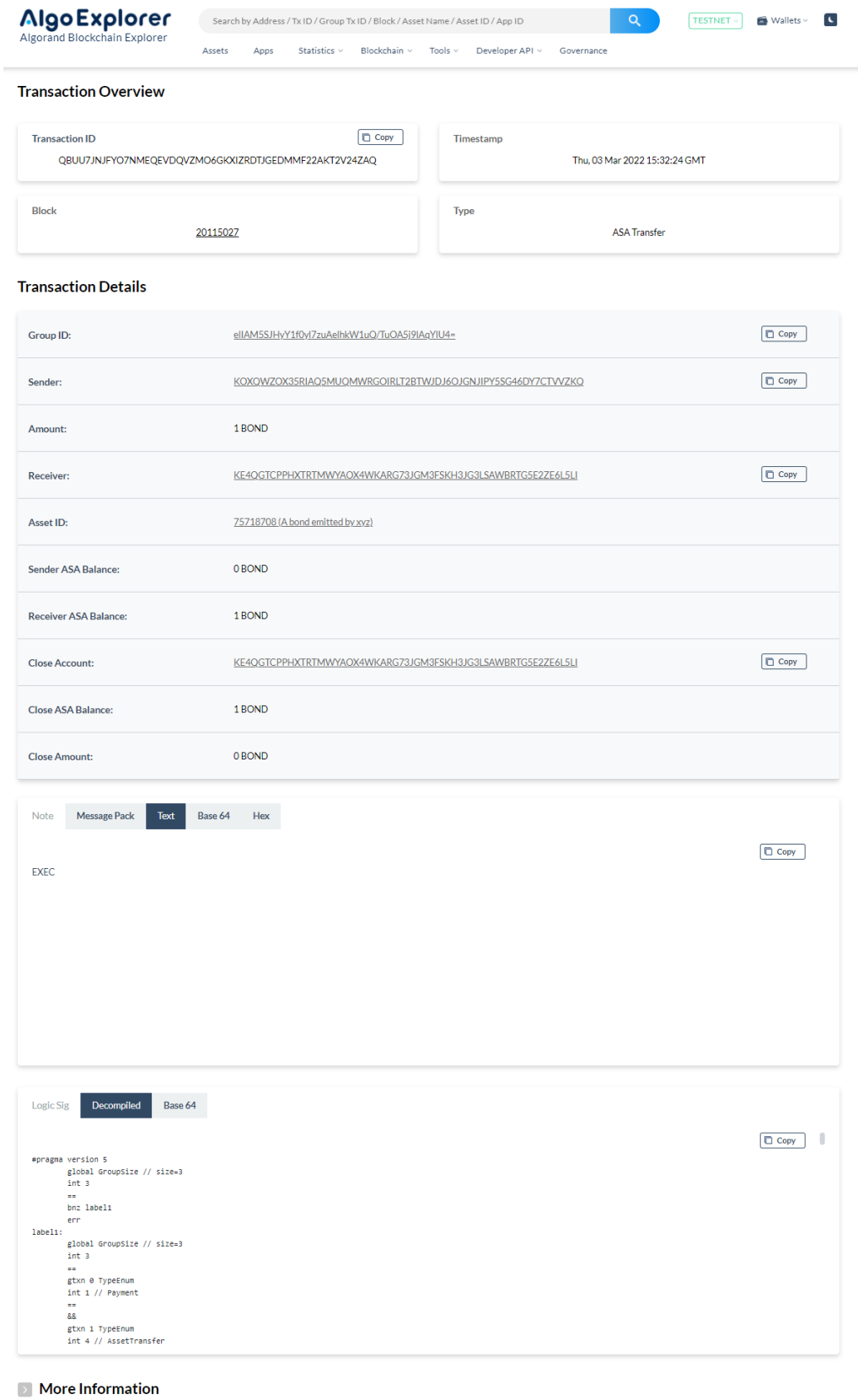

Figure 16. The forced execution on AlgoExplorer: Details of a second transaction with the execution preimage.

```
ur_asset_checks = And(
    Gtxn[0].amount() == Int(0),
    BytesEq(Gtxn[0].note(), Concat(tips_tx_id, Bytes("|Ur"))),
    Sha256(Gtxn[1].note()) == rfnd_hash,
    Gtxn[1].xfer_asset() == asset,
    Gtxn[1].asset_amount() == Int(1),
    Gtxn[2].amount() == Int(0),
)

ur_participants_checks = And(
    Gtxn[0].sender() == seller,
    Gtxn[1].asset_receiver() == seller,
    Gtxn[0].receiver() == Gtxn[1].sender(),
    Gtxn[0].receiver() == Gtxn[2].sender(),
    Gtxn[2].receiver() == seller,
    Gtxn[0].close_remainder_to() == Global.zero_address(),
    Gtxn[1].asset_close_to() == seller,
    Gtxn[2].close_remainder_to() == seller,
)

forced_refnd_checks = And(
    bx_at_checks,
    ur_asset_checks,
    ur_participants_checks,
)
```

Listing 6. HLC: forced cancellation checks

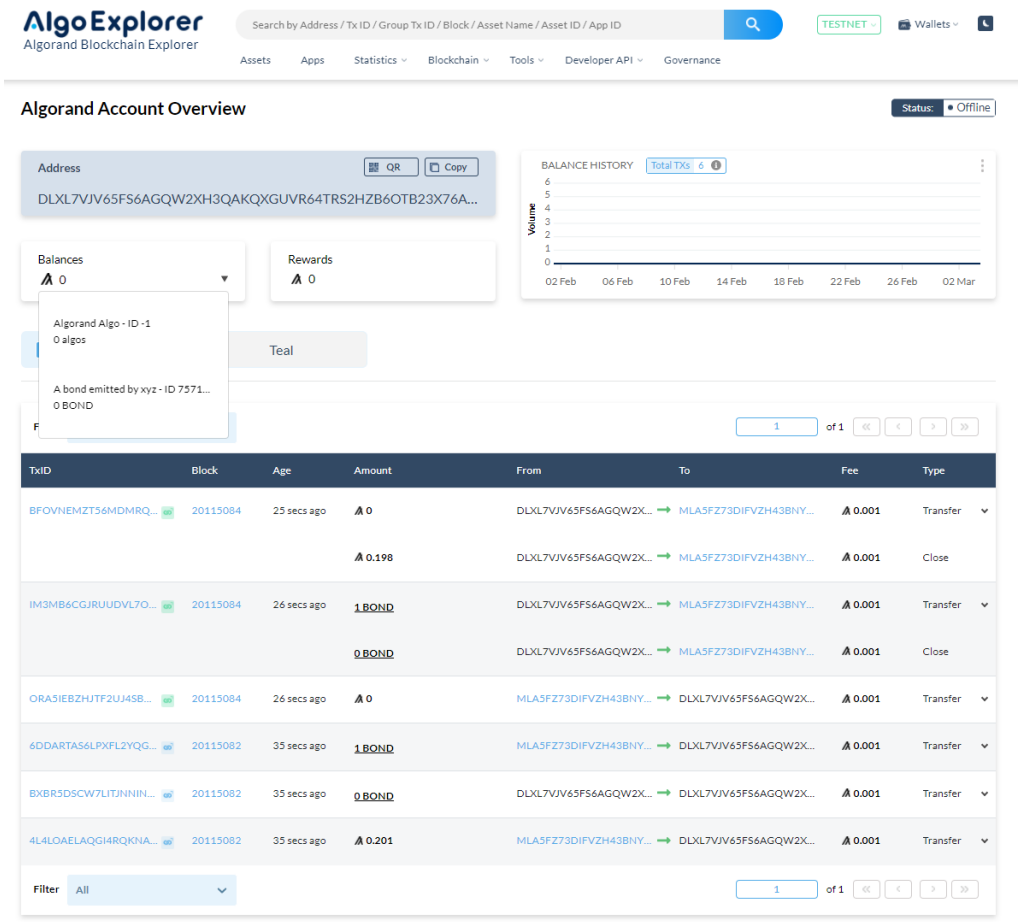

Figure 17. The forced cancellation on AlgoExplorer.

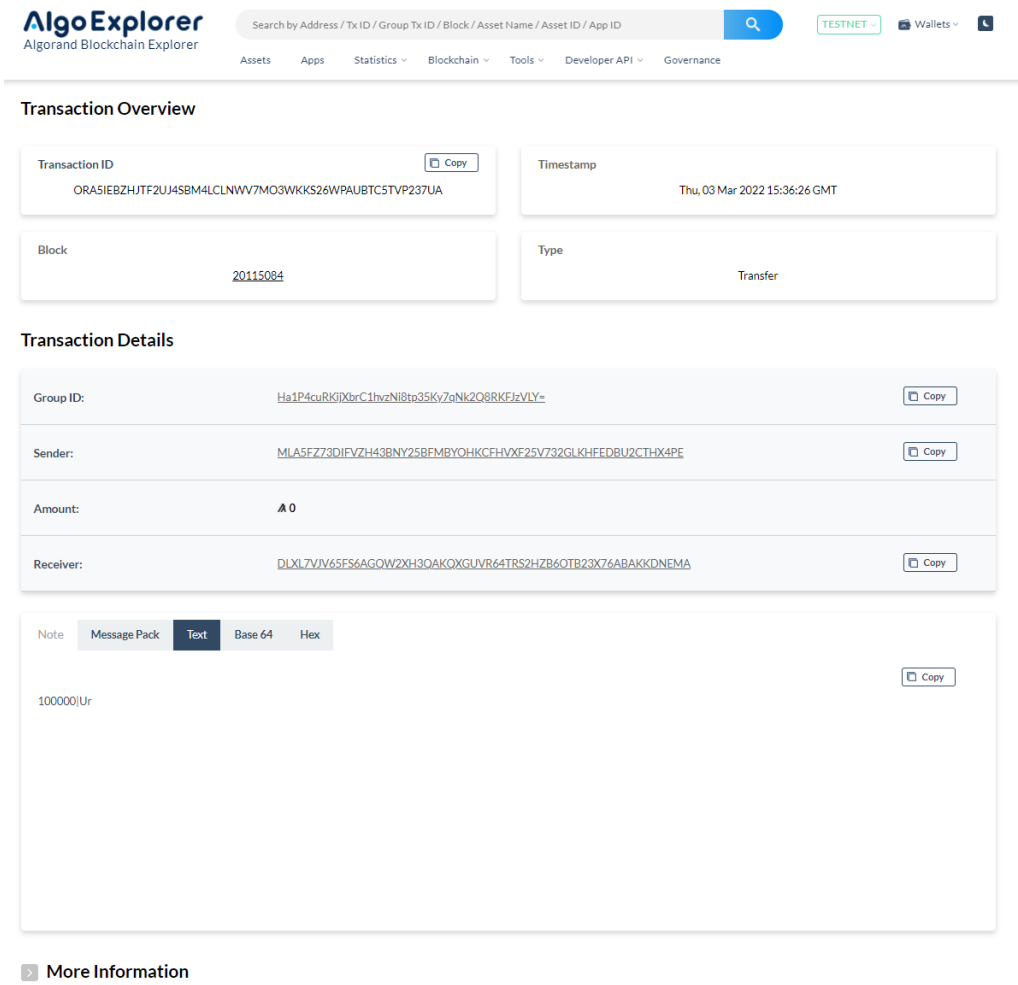

Figure 18. The forced cancellation on AlgoExplorer: Detail of first transaction including the atomic transfer Group ID and notes.

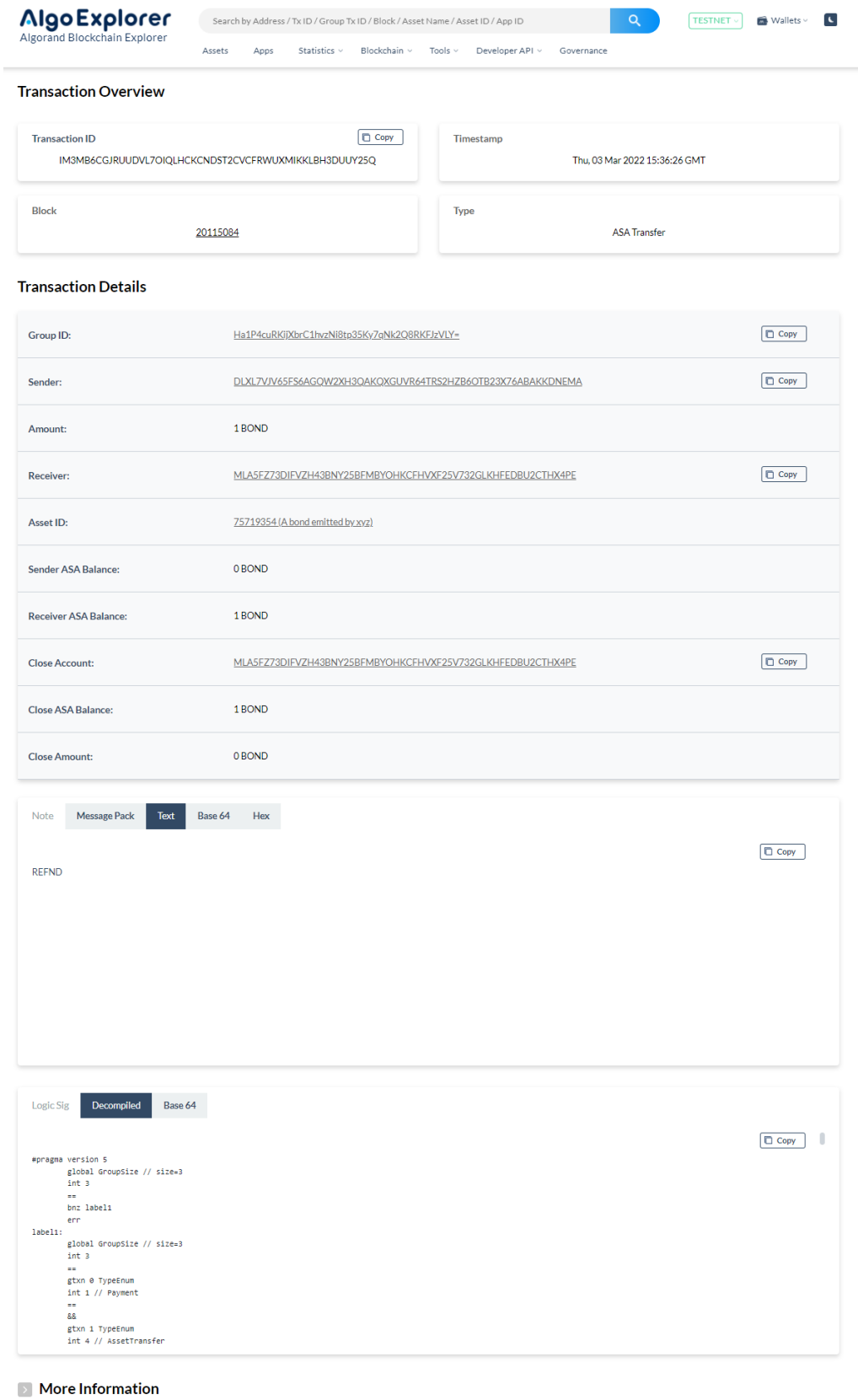

Figure 19. The forced cancellation on AlgoExplorer: Detail of second transaction including the cancellation preimage.

A.3 Global security checks

In addition to all checks previously described, there are further ones that must always be carried out on every transaction interacting with the HLC, regardless of the use case. In the context of the current work, two important ones have been identified.

```
...
max_fee = 1000
...

@Subroutine(TealType.none)
def assert_security_check():
    i = ScratchVar(TealType.uint64)
    return Seq([
        For(i.store(Int(0)), i.load() < Global.group_size(), i.store(i.load()
    + Int(1))).Do(
            Assert(And(
                Gtxn[i.load()].fee() <= Int(max_fee),
                Gtxn[i.load()].rekey_to() == Global.zero_address(),
            ))
        )
    ])
```

Listing 7. HLC: global security checks

The first enforces, on each transaction, a fixed `fee`[46], set to the minimum fee of the Algorand DLT. This control ensures that the HLC can never go out of funds due to outbound transactions with unexpected high fees set. This avoids Denial of Service scenarios where the DvP cannot complete due to HLC lack of funds. Otherwise, this kind of attack would lead to the situation where the Seller obtains the payment, but the Buyer does not obtain the asset, which remains locked inside the HLC.

It is also checked that `rekey`[47] field is always unset. The rekey address has a permanent delegation semantics: if an account A1 issues a transaction with the rekey address set to the address A2, the Algorand DLT will start accepting transactions spending assets or funds owned by the account A1, signed with the private key associated to A2. This feature is particularly dangerous if used by a HLC address acting as escrow. In fact, rekeying the HLC address would lead to the theft of the assets and funds deposited within it.

B TIPS Gateway API detail

B.1 Documentation

The API documentation describes all the services offered by the interface with a detailed description of how to use them, aiming to cover everything a customer would need to know for practical purposes.

Documentation is crucial for the development and maintenance of applications using the APIs. For this reason, the designed API has been documented using Swagger[48], which presents, for each endpoint, the following information:

- the method;
- the XML business payload through the XSD definition, an explanation of the semantic applied and a description on how to populate the needed fields;
- an example of the business payload, applied to the specific business context.

46 https://developer.algorand.org/docs/get-details/transactions/transactions/#common-fields-header-and-type
47 https://developer.algorand.org/docs/get-details/accounts/rekey/
48 https://swagger.io/

This approach has been adopted for all the TIPS Gateway functionalities exposes through APIs.

B.1.1 TIPS Gateway Endpoints

DvP initiation	
Path	/v0/init
Method	POST
Caller	Seller
Details	Asynchronously triggers the generation of a TIPS Transaction ID and of two preimages H(R) and H(R')
Request body	pain.009.001.06 - MandateInitiationRequest
Response code	202

Payment	
Path	/v0/instruct-payment
Method	POST
Caller	Buyer
Details	Called by the buyer to instruct the payment of the TIPS transaction with the given Transaction ID
Request body	pacs.008.001.02 - FIToFICustomerCreditTransfer
Response code	202

Forced cancellation	
Path	/v0/refund
Method	POST
Caller	Seller
Details	Enables the seller to ask TIPS Gateway to disclose the secret R' used in the forced cancellation scenario
Request body	pain.013.001.08 - CreditorPaymentActivationRequest
Response code	202

Forced execution	
Path	/v0/execute
Method	POST
Caller	Buyer
Details	Enables the buyer to ask TIPS Gateway to disclose the secret R used in the forced execution scenario
Request body	pain.013.001.08 - CreditorPaymentActivationRequest
Response code	202

B.1.2 Buyer and Seller Endpoints

DvP initiation answer	
Path	/v0/init-answer
Method	POST
Caller	TIPS Gateway
Details	Receives the answer of the DVP initialization
Request body	pain.012.001.06 - MandateAcceptanceReport
Response code	204

Payment answer	
Path	/v0/instruct-payment-answer
Method	POST
Caller	TIPS Gateway
Details	Receives the answer of a payment instruction
Request body	pacs.002.001.03 - FIToFIPaymentStatusReport
Response code	204

Forced cancellation answer	
Path	/v0/refund-answer
Method	POST
Caller	TIPS Gateway
Details	Receives the answer of the forced cancellation
Request body	pain.014.001.08 - CreditorPaymentActivationRequestStatusReport
Response code	204

Forced execution answer	
Path	/v0/execute-answer
Method	POST
Caller	TIPS Gateway
Details	Receives the answer of the forced execution
Request body	pain.014.001.08 - CreditorPaymentActivationRequestStatusReport
Response code	204

B.2 Example

To better understand how the API documentation has been structured with Swagger, the following example describes in detail the *Forced execution* case of TIPS Hash-Link.

B.2.1 Data constellation

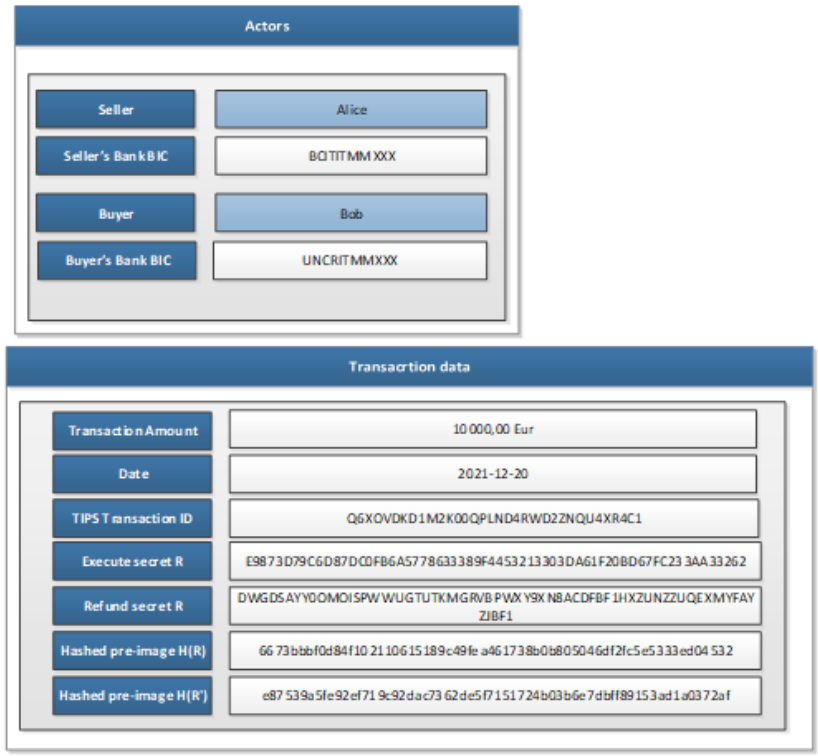

Figure 20. Data constellation

B.2.2 Actors

- The seller (Alice) owns the asset *A* in the DLT network. She wants to sell the asset reaching the final settlement of the cash leg in TIPS. Her bank is *BCITITMMXXX*;
- The buyer (Bob) wants to buy the asset *A*. At the end of the DvP, every party of the DLT network recognize it as the new owner of the asset *A*. His bank is *UNCRITMMXXX*;

B.2.3 Scenario

With the *Forced execution* functionality, Bob asks TIPS Gateway to disclose the execution preimage *R* (e.g., its value is `E9873D79C6D87DC0FB6A5778633389F4453213303DA61F20BD67FC233AA33262`). The TIPS Gateway releases *R* if and only if the cash-leg of the DVP has already been settled in TIPS. Providing the execution preimage *R* to the HLC, the asset *A* ownership is transferred to Bob, thus successfully terminating the DvP.

B.2.4 TIPS Gateway Endpoint

POST /v0/execute
Required Request Body
Based on ISO20022 *pain.013.001.08 - CreditorPaymentActivationRequest* message, with the following usages for fields with main business importance:

Field Name	Description	XML path	Mand.	Usage
Number of transactions	Number of transactions carried out by the message	/Document/CdtrPmtActvtnReq/GrpHdr/NbOfTxs	Yes	Fixed value 1
Buyer BIC	BIC of the party who sent the message	/Document/CdtrPmtActvtnReq/GrpHdr/InitgPty /Id/OrgId/AnyBIC	Yes	Debtor/Buyer, in case of forced execution
Requested action	Type of the action requested	/Document/CdtrPmtActvtnReq/PmtInf/PmtTpInf /CtgyPurp/Prtry	Yes	“EXE” for forced execution
Transaction ID	TIPS Transaction Identification	/Document/CdtrPmtActvtnReq/PmtInf/CdtTrfTx /PmtId/InstrId	Yes	
End to End Identification	End to End Identification	/Document/CdtrPmtActvtnReq/PmtInf/CdtTrfTx /PmtId/EndToEndId	Yes	Not used in the forced execution processing

XML body example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Document xmlns="urn:iso:std:iso:20022:tech:xsd:pain.013.001.08" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance">
  <CdtrPmtActvtnReq>
    <GrpHdr>
      <MsgId>001</MsgId>
      <CreDtTm>2020-12-20T22:30:00.00Z</CreDtTm>
      <NbOfTxs>1</NbOfTxs>
      <InitgPty>
        <Id>
          <OrgId>
            <AnyBIC>UNCRITMMXXX</AnyBIC>
          </OrgId>
        </Id>
      </InitgPty>
    </GrpHdr>
    <PmtInf>
      <PmtMtd>TRF</PmtMtd>
      <PmtTpInf>
        <CtgyPurp>
          <Prtry>EXE</Prtry>
        </CtgyPurp>
      </PmtTpInf>
      <ReqdExctnDt>
        <Dt>2020-12-20</Dt>
      </ReqdExctnDt>
      <Dbtr>
        <Id>
          <OrgId>
            <AnyBIC>UNCRITMMXXX</AnyBIC>
          </OrgId>
        </Id>
      </Dbtr>
      <DbtrAgt>
        <FinInstnId>
          <BICFI>UNCRITMMXXX</BICFI>
        </FinInstnId>
      </DbtrAgt>
      <CdtTrfTx>
        <PmtId>
          <InstrId>Q6XOVDKD1M2K00QPLND4RWD2ZNQU4XR4C1</InstrId>
          <EndToEndId>NONREF</EndToEndId>
        </PmtId>
        <Amt>
          <InstdAmt Ccy="EUR">10000</InstdAmt>
```

```
        </Amt>
        <ChrgBr>SLEV</ChrgBr>
        <CdtrAgt>
          <FinInstnId/>
        </CdtrAgt>
        <Cdtr/>
      </CdtTrfTx>
    </PmtInf>
  </CdtrPmtActvtnReq>
</Document>
```

Response: 202 - Asynchronous request accepted. The result, when ready, will be sent to customer endpoint

B.2.5 Buyer endpoint

POST /v0/execute-answer
Required Request Body
Based on ISO20022 *pain.014.001.08 - CreditorPaymentActivationRequestStatusReport* message, with the following usages for fields with main business importance:

Field Name	Description	XML path	Mand.	Usage
Request result	Result of the requested action	/Document/CdtrPmtActvtnReqStsRpt /OrgnlPmtInfAndSts/TxInfAndSts/StsRsnInf/Rsn/Prtr	Yes	“ACCP” for accepted, “RJCT” for rejected
Secret	Returned secret	Document/CdtrPmtActvtnReqStsRpt /OrgnlPmtInfAndSts/TxInfAndSts/StsRsnInf/AddtlInf	No	Secret R, in case of accepted forced execution answer

XML body example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Document xmlns="urn:iso:std:iso:20022:tech:xsd:pain.014.001.08" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance">
  <CdtrPmtActvtnReqStsRpt>
    <GrpHdr>
      <MsgId>005</MsgId>
      <CreDtTm>2020-12-20T22:30:00.00Z</CreDtTm>
      <InitgPty/>
    </GrpHdr>
    <OrgnlGrpInfAndSts>
      <OrgnlMsgId>002</OrgnlMsgId>
      <OrgnlMsgNmId>pain.013.001.08</OrgnlMsgNmId>
    </OrgnlGrpInfAndSts>
    <OrgnlPmtInfAndSts>
      <OrgnlPmtInfId>NONREF</OrgnlPmtInfId>
      <TxInfAndSts>
        <StsRsnInf>
          <Rsn>
            <Prtry>ACCP</Prtry>
          </Rsn>
          <AddtlInf>
    DWGDSAYYOOMOISPWWUGTUTKMGRVBPWXY9XN8ACDFBF1HXZUNZZUQEXMYFAYZJBF1</AddtlInf
    >
        </StsRsnInf>
      </TxInfAndSts>
    </OrgnlPmtInfAndSts>
  </CdtrPmtActvtnReqStsRpt>
</Document>
```

Response: 204 - Message received.

Papers published in the 'Markets, Infrastructures, Payment Systems' series

n. 1 TIPS - TARGET Instant Payment Settlement – The Pan-European Infrastructure for the Settlement of Instant Paymentsi, *by Massimiliano Renzetti, Serena Bernardini, Giuseppe Marino, Luca Mibelli, Laura Ricciardi and Giovanni M. Sabelli* (Institutional Issues)

n. 2 Real-Time Gross Settlement systems: breaking the wall of scalability and high availability, *by Mauro Arcese, Domenico Di Giulio and Vitangelo Lasorella* (Research Papers)

n. 3 Green Bonds: the Sovereign Issuers' Perspective, *by Raffaele Doronzo, Vittorio Siracusa and Stefano Antonelli* (Research Papers)

n. 4 T2S - TARGET2-Securities – The pan-European platform for the settlement of securities in central bank money, *by Cristina Mastropasqua, Alessandro Intonti, Michael Jennings, Clara Mandolini, Massimo Maniero, Stefano Vespucci and Diego Toma* (Institutional Issues)

n. 5 The carbon footprint of the Target Instant Payment Settlement (TIPS) system: a comparative analysis with Bitcoin and other infrastructures, *by Pietro Tiberi* (Research Papers)

n. 6 Proposal for a common categorisation of IT incidents, *by Autorité de Contrôle Prudentiel et de Résolution, Banca d'Italia, Commissione Nazionale per le Società e la Borsa, Deutsche Bundesbank, European Central Bank, Federal Reserve Board, Financial Conduct Authority, Ministero dell'Economia e delle Finanze, Prudential Regulation Authority, U.S. Treasury* (Institutional Issues)

n. 7 Inside the black box: tools for understanding cash circulation, *by Luca Baldo, Elisa Bonifacio, Marco Brandi, Michelina Lo Russo, Gianluca Maddaloni, Andrea Nobili, Giorgia Rocco, Gabriele Sene and Massimo Valentini* (Research Papers)

n. 8 The impact of the pandemic on the use of payment instruments in Italy, *by Guerino Ardizzi, Alessandro Gambini, Andrea Nobili, Emanuele Pimpini and Giorgia Rocco* (Research Papers) (in Italian)

n. 9 TARGET2 – The European system for large-value payments settlement, *by Paolo Bramini, Matteo Coletti, Francesco Di Stasio, Pierfrancesco Molina, Vittorio Schina and Massimo Valentini* (Institutional Issues) (in Italian)

n. 10 A digital euro: a contribution to the discussion on technical design choices, *by Emanuele Urbinati, Alessia Belsito, Daniele Cani, Angela Caporrini, Marco Capotosto, Simone Folino, Giuseppe Galano, Giancarlo Goretti, Gabriele Marcelli, Pietro Tiberi and Alessia Vita* (Institutional Issues)

n. 11 From SMP to PEPP: a further look at the risk endogeneity of the Central Bank, *by Marco Fruzzetti, Giulio Gariano, Gerardo Palazzo and Antonio Scalia* (Research Papers)

n. 12 TLTROs and collateral availability in Italy, *by Annino Agnes, Paola Antilici and Gianluca Mosconi* (Research Papers) (in Italian)

n. 13 Overview of central banks' in-house credit assessment systems in the euro area, *by* L*aura Auria, Markus Bingmer, Carlos Mateo Caicedo Graciano, Clémence Charavel, Sergio Gavilá, Alessandra Iannamorelli, Aviram Levy, Alfredo Maldonado, Florian Resch, Anna Maria Rossi and Stephan Sauer* (Institutional Issues)

n. 14 The strategic allocation and sustainability of central banks' investment, *by Davide Di Zio, Marco Fanari, Simone Letta, Tommaso Perez and Giovanni Secondin* (Research Papers) (in Italian)

n. 15 Climate and environmental risks: measuring the exposure of investments, *by Ivan Faiella, Enrico Bernardini, Johnny Di Giampaolo, Marco Fruzzetti, Simone Letta, Raffaele Loffredo and Davide Nasti* (Research Papers)

n. 16 Cross-Currency Settlement of Instant Payments in a Multi-Currency Clearing and Settlement Mechanism, *by Massimiliano Renzetti, Fabrizio Dinacci and Ann Börestam* (Research Papers)

n. 17 What's ahead for euro money market benchmarks?, *by Daniela Della Gatta* (Institutional Issues) (in Italian)

n. 18 Cyber resilience per la continuità di servizio del sistema finanziario, *by Boris Giannetto and Antonino Fazio* (Institutional Issues) (in Italian)

n. 19 Cross-Currency Settlement of Instant Payments in a Cross-Platform Context: a Proof of Concept, *by Massimiliano Renzetti, Andrea Dimartina, Riccardo Mancini, Giovanni Sabelli, Francesco Di Stasio, Carlo Palmers, Faisal Alhijawi, Erol Kaya, Christophe Piccarelle, Stuart Butler, Jwallant Vasani, Giancarlo Esposito, Alberto Tiberino and Manfredi Caracausi* (Research Papers)

n. 20 Flash crashes on sovereign bond markets – EU evidence, *by Antoine Bouveret, Martin Haferkorn, Gaetano Marseglia and Onofrio Panzarino* (Research Papers)

n. 21 Report on the payment attitudes of consumers in Italy: results from ECB surveys, *by Gabriele Coletti, Alberto Di Iorio, Emanuele Pimpini and Giorgia Rocco* (Institutional Issues)

n. 22 When financial innovation and sustainable finance meet: Sustainability-Linked Bonds, *by Paola Antilici, Gianluca Mosconi and Luigi Russo* (Institutional Issues) (in Italian)

n. 23 Business models and pricing strategies in the market for ATM withdrawals, *by Guerino Ardizzi and Massimiliano Cologgi* (Research Papers)

n. 24 Press news and social media in credit risk assessment: the experience of Banca d'Italia's In-house Credit Assessment System, *by Giulio Gariano and Gianluca Viggiano* (Research Papers)

n. 25 The bonfire of banknotes, *by Michele Manna* (Research Papers)