Patterns for Blockchain-Based Payment Applications

Qinghua Lu, Xiwei Xu, H.M.N. Dilum Bandara, Shiping Chen, Liming Zhu
Data61, CSIRO, Australia
University of New South Wales, Australia
{firstname.lastname}@data61.csiro.au

ABSTRACT

As the killer application of blockchain technology, blockchain-based payments have attracted extensive attention ranging from hobbyists to corporates to regulatory bodies. Blockchain facilitates fast, secure, and cross-border payments without the need for intermediaries such as banks. Because blockchain technology is still emerging, systematically organised knowledge providing a holistic and comprehensive view on designing payment applications that use blockchain is yet to be established. If such knowledge could be established in the form of a set of blockchain-specific patterns, architects could use those patterns in designing a payment application that leverages blockchain. Therefore, in this paper, we first identify a token's lifecycle and then present 12 patterns that cover critical aspects in enabling the state transitions of a token in blockchain-based payment applications. The lifecycle and the annotated patterns provide a payment-focused systematic view of system interactions and a guide to effective use of the patterns.

CCS CONCEPTS

• **Software and its engineering** → **Patterns**; Designing software.

KEYWORDS

Blockchain, Pattern, Architecture, Payment, Token, Cryptocurrency, Escrow, Channel

1 INTRODUCTION

Blockchain is an emerging distributed ledger technology that has attracted a broad range of interests from academia and industry to build the next generation of trustworthy applications. A large number of projects have been conducted to explore how to re-architect systems, and to build new applications and business models using
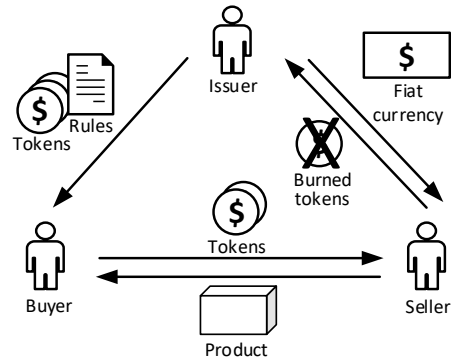
Figure 1: Overview of token-based payments.

blockchain as a general, decentralised computing and storage environment through the advent of smart contracts (i.e., programs on a blockchain) [5].

Payment is considered the killer application of blockchain technology, which executes monetary transactions for goods or services previously agreed upon by all parties involved. The "money" used on a blockchain includes both cryptocurrencies and tokens. *Cryptocurrencies* are baked into the blockchain platforms as the base currency (e.g., Bitcoin and Ether) for transactions and provide incentives for blockchain platform operations (e.g., mining). *Tokens* are application-tier coins that are created and exchanged using smart contracts on the blockchain. Tokens may represent cryptocurrencies (e.g., Binance Coin and Uniswap) or used as a proxy for fiat currency (e.g., Tether and US Dollar Coin).

Fig. 1 illustrates an overview of token-based payments. A token *issuer* issues a new token associated with a set of spending rules. One or more tokens are allocated to a *buyer*. For example, in food stamps, the issuer is the government agency that manages the program and buyers are the people classified as food insecure. The spending rules may specify eligible kinds of food, their quantities, and sellers. A buyer then purchases food products from a *seller* using the allocated tokens. A seller could be a supermarket that cashes out multiple tokens from the government agency. However, as different food stamps may carry different spending rules, it is in the seller's best interest to validate them at the time of payment. This ensures that the seller can convert the collected tokens to fiat currency for the products it sold. Alternatively, the buyer may also want to know the authenticity of the product and whether the seller has the authority to sell it. Once redeemed, the issuer needs to destroy the tokens (aka., burn token) to ensure the tokens are not reused (i.e., double spend).

As discussed in the above example, payments are usually conditional, where they are made subject to a set of conditions. Payment

conditions could be complicated in many environments, and could range from coupons with a designated product/service and an expiration date to insurance payouts and social welfare benefits. Further, the conditions need to be specified in advance (usually with the consent of parties involved in the transaction) such that they can be checked at the time of payment. Thus, in such use cases, the currency needs to be programmable to specify payment conditions, enable automatic validation and enforcement, and support immutable record-keeping of associated transactions. These properties are well aligned with the design of tokens realised by blockchains (i.e., provides immutable record-keeping and multi-party transactions) and smart contracts (i.e., provides programmability and automatic enforcement).

As blockchain technology is still emerging, systematically organised knowledge providing a holistic and comprehensive view on designing blockchain-based payment applications is yet to be established. Therefore, to form such knowledge, in this paper, based on a literature review and our project experience, we collect a set of patterns that cover the critical aspects in the design of blockchain-based payment applications. We identify the lifecycle of a payable token, in which the identified patterns are associated with the state transitions along the lifecycle. The lifecycle and the annotated patterns provide a payment-focused systematic view of system interactions and a guide to the effective usage of the patterns to both architects and developers. In this paper, we focus on blockchain-based payable tokens such as those based on ERC-1363[1] standard that can be spent to purchase products or services. We also consider native cryptocurrencies, fungible tokens (e.g., ERC-20[2]), and non-fungible tokens (e.g., ERC-721[3]) that are used within smart contracts to attach to a specific buyer (e.g., a voucher given to a patient), specify payment conditions, and execute payments.

The remainder of this paper is organised as follows: Section 2 discusses related work. The overview of payment patterns is introduced in Section 3. Section 4 presents each pattern in details. Section 5 gives concluding remarks.

2 RELATED WORK

Patterns have been collected to document the reusable solutions for designing blockchain-based applications [3, 7, 9, 10, 15, 19, 23, 24]. Some of the blockchain patterns focus on the design for generic applications [7, 15, 23], while others are collected only for specific applications [3, 9, 24]. Eberhardt and Tai [7] identified five data and computation design patterns for blockchain-based applications and discussed what data and computation to be kept on-chain and what to be placed off-chain.

Many patterns are summarised for smart contracts [4, 10, 19, 20]. For example, Liu et al. [10] presented eight design patterns for writing smart contracts. Wöhrer and Zdun [19] summarised six design patterns to improve the security of smart contracts. Further, empirical studies on smart contracts have been performed to identify the common smart contract design patterns [4, 20]. There are also examples of applications recognising the adoption of specific pattern collections to showcase their robust designs. For example,

OriginChain [12, 22] is a supply chain application that adopts data management and smart contract design patterns to improve system adaptability. uBaaS is a service platform that encapsulates patterns as services to generate code skeletons to facilitate the development of blockchain-based applications [13]. In [21], we proposed a decision model that assists developers and architects in selecting a set of patterns for blockchain-based applications.

There have been significant efforts in both industry and academia to address payment challenges using the blockchain technology. In 2018, CSIRO's Data61 and Commonwealth Bank of Australia developed a proof of concept app to create and manage programmable "smart money" using blockchain technology [17] through the case study of the National Disability Insurance Scheme (NDIS). In 2019, TietoEVRY and Kela (The Social Insurance Institution of Finland) had cooperated to develop a proof of concept prototype for smart money [18]. Goldfeder et al. [8] studied the escrow problem for physical products and design a set of escrow schemes for Bitcoin.

Compared to the existing literature and application of known patterns, our study focuses on the patterns in the context of blockchain-based payments, which have not been covered before. Although patterns such as *burned token*, *escrow*, and *oracles* are already discussed in previous work [9, 23], in this paper, we adapt them in the area of payments and aim to provide a holistic and comprehensive guide on the design of blockchain-based payment applications.

3 OVERVIEW OF BLOCKCHAIN-BASED PAYMENT PATTERNS

A variety of constraints need to be satisfied while building payment applications on a blockchain. While each application is unique, we can always learn from the commonly recurring problems and the corresponding reusable design solutions. Those reusable design solutions can be established as a set of design patterns.

The payment patterns are derived from our blockchain project experience and literature review. We started with identifying our research question, which is "What are the effective patterns for designing a blockchain-based payment application?". We used Google Search and Google Scholar as the source databases. The search keywords include "blockchain", "payment", "smart contract", "token", "asset", and their combinations. We also used snowballing to expand the pool of source literature. Next, we filtered the sources according to the predefined selection criteria, e.g., "payment must include ownership transfer of money". Finally, we used open and axial coding to analyse the selected sources and derive related patterns iteratively.

Table 1 presents an overview of the collected patterns. We classify the patterns into two categories: token design and payment management patterns. We provide a summary for each pattern. Although some patterns (e.g., *burned token*, *stealth address*, and *multisignature*) are provided as embedded features by some blockchain platforms, they can still be considered at the application-level as architects and developers can implement those features in smart contracts and applications. For example, the burning of native tokens like BNB is baked into blockchain platform logic, while ERC-20 like tokens could be burned by the smart contracts that created them. Hence, such tokens are at the control of the application developer. Similarly, while token swap could be supported at the blockchain platform

---

[1]https://eips.ethereum.org/EIPS/eip-1363
[2]https://eips.ethereum.org/EIPS/eip-20
[3]https://eips.ethereum.org/EIPS/eip-721

Table 1: Overview of blockchain payment patterns.

| Category	Name	Summary
Token Design Patterns	Token Template	Tokens are designed in a customised way using a *token template* that can be extended or instantiated.
	Token Registry	A token registry maintains the ownership information of tokens through a mapping of token ID and wallet address.
	Policy Contract	A *policy contract* specifies the policies that a token must satisfy, e.g., whether a token must be spent on the product or services supplied by eligible sellers or how many tokens can be spent in a transaction.
	Burned Token	When redeemed, a spent token is marked as unusable/unspendable (aka., *burned*).
Payment Management Patterns	Escrow	Before making a transaction, tokens are transferred to a third-party smart contract called the escrow. The escrow holds the deposited tokens until the payment conditions are satisfied.
	Payment Channel	A two-way pathway (aka., *payment channel*) is established between transacting parties to perform transactions off-chain while recording opening and final settlement transactions on the blockchain.
	Seller Credential	The buyer can validate the seller's business qualification through seller credentials.
	Stealth Address	A *stealth address* protects the privacy of the parties involved in the payment by creating a one-time address for the transaction.
	Oracle	To examine the fulfilment of payment conditions, an *oracle* is used to introduce the state of external systems into the closed blockchain execution environment.
	Multisignature	Given a pool of n addresses that could authorise a payment, a subset m of them can authorise a payment by signing the respective transaction ($m \leq n$).
	Token Swap	Token swap allows users to trade directly between two types of tokens as an atomic transaction.
	Authorised Spender	An authorised third-party spender is allowed to spend a certain amount of tokens on behalf of the approver.
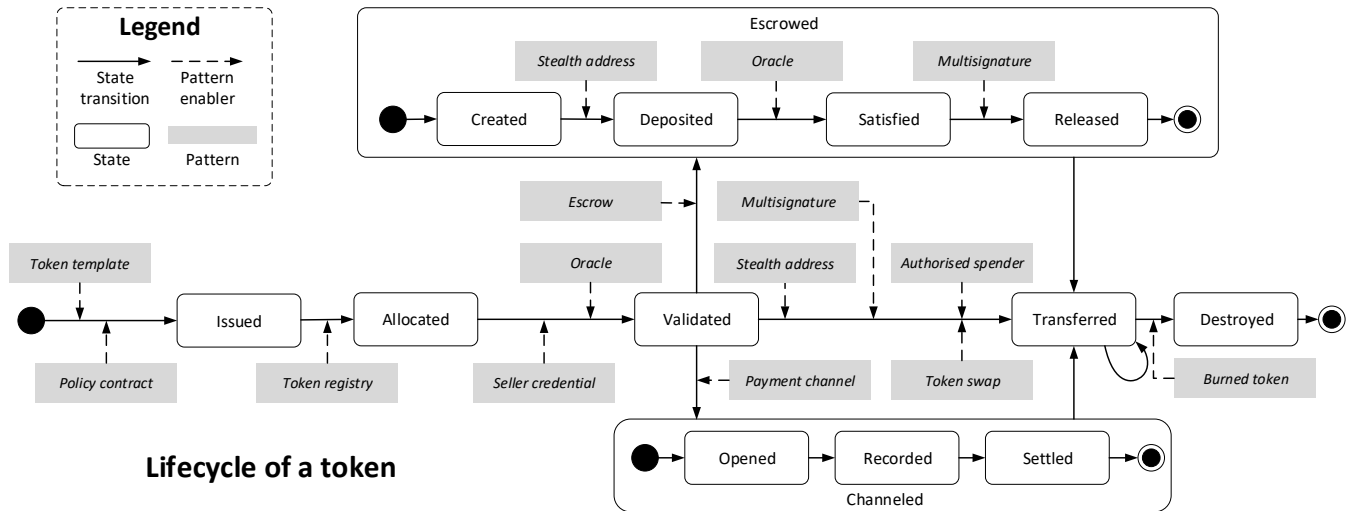
Figure 2: Lifecycle of a token with the annotated patterns.

level, it becomes more complicated when token are swapped across blockchains. In such cases, an application-level solution, such as Hash-Lock Time-Lock Contracts (HTLCs), is needed to link two blockchains. Therefore, while we focus on patterns that assist the blockchain-based payment application design in this paper, some of those patterns may also be applied to a different level.

Fig. 2 illustrates the lifecycle of a token and highlights the patterns associated with state transitions within the lifecycle. The lifecycle starts with the issuance of a token. The token can be designed in a customised way using a *token template* pattern which specifies necessary details (e.g., token name, symbol, number of decimal places, and the total number of tokens supplied) and advanced

features (e.g., the minting process, initial distribution, and token burning process). A *policy contract* pattern stipulates the rules for using a group of tokens such as how many vouchers can be spent in a transaction and whether the token must be spent on the specified products or services supplied by eligible sellers.

Once issued, the predefined number of tokens are allocated to the buyer's wallet addresses. The *token registry* pattern could be used to maintain the token ownership information as a mapping between a wallet address and the number of tokens or token identifiers associate with it. The buyer's wallet receives the allocated tokens and sends a notification to the consumer that the tokens are received.

Before purchasing a product or service, a consumer's wallet could use the *seller credential* pattern to validate the seller's business qualifications. Further, the *oracle* pattern can introduce the external state/data into the blockchain execution environment, which is self-contained. For example, a payment may be subject to an external state confirmed by a human like a truck driver confirming the collection of an item or an IoT device automatically reporting quality attribute of the item or its ambient conditions through the oracle.

Once the required transaction information is validated, the token(s) are transferred to the seller. The *escrow* pattern can be used for conditional payments, which is suitable for high-value, infrequent payments that require strong security guarantees. The escrow holds the deposited tokens until the payment conditions are satisfied. *Multisignature* pattern can be applied to sign transactions requiring the approval of multiple parties, e.g., to control the release of tokens held by an escrow. Please note that *stealth address*, *oracle*, and *multisignature* patterns can be applied to a transaction with or without an escrow.

To deal with frequent, low latency, and low-value payments, a *payment channel* can be designed to perform transactions off-chain with opening and final settlement transactions recorded on-chain. The *authorised spender* pattern is used to allow a third party (e.g., a smart contract) to spend a certain amount of tokens on behalf of the approver. *Token swap* pattern enables users to trade directly between two types of tokens as an atomic swap. The *stealth address* pattern enables a given payment to be made to a one-time account/address, such that the anonymity set is only provided to the parties involved in the transaction. Once the transaction is settled, the seller is notified that the payment has been executed successfully.

Depending on the use case, tokens could be transferred multiple times. Tokens might need to be redeemed in some use cases. Thus, the spent/redeemed tokens might become *burned tokens*, which means the tokens are no longer usable.

4 BLOCKCHAIN-BASED PAYMENT PATTERNS

In this section, we present patterns that shape the architectural elements and their interactions in blockchain-based payment applications. Each pattern is described following the extended pattern form in [14], which includes the pattern name, summary, problem statement, forces that make the problem difficult to solve, solution, consequences of applying the solution, and real-world known uses. The proposed patterns' intended audience is software architects and developers who have to deal with blockchain-based payment application design and development.

4.1 Token Template

Summary: Tokens can be designed in a customised manner using a template smart contract that can be extended or instantiated.

Context: Blockchain provides a trustworthy platform to realise tokenisation. In a blockchain application, a token represents a programmable digital unit of value (i.e., assets) recorded on the blockchain for a specific purpose, like buying a product or consuming a service. The same application may use different tokens or asset classes, e.g., a food stamp issued for a child is usually different from an adult.

Problem: A token is usually defined as a data structure embedded in a smart contract to represent an asset. How can developers design and develop different types of tokens with various features?

Forces:

- *Flexibility.* A blockchain application might support different types of tokens with different properties and rules.
- *Interoperability and liquidity.* Tokens supported by different applications might need to be exchanged frequently.
- *Vulnerabilities.* Handling tokens with a high value is risky, as lost tokens cannot be replaced/recovered on the blockchain. Further, vulnerabilities or bugs might be introduced when developing a blockchain-based application. As blockchain is an immutable data store, updating already deployed smart contracts is impossible. This makes it difficult to fix bugs by releasing new versions of a smart contract.
- *Productivity.* Blockchain is an emerging technology with limited tooling and documentation. Thus, developers can have a steep learning curve, which affects the developer productivity.
- *Cost.* On a public blockchain, transaction fees need to be paid to deploy and execute smart contracts. The execution fee is proportional to the computational and storage complexity of the smart contract.

Solution: Blockchain-based payment applications might need to issue different types of tokens with customisation. However, as most tokens have a common set of properties and functions, a token can be defined as a template smart contract. The template can then be either extended or instantiated by another smart contract to issue a new token at run time. Multiple tokens may also be issued based on the same template by setting different properties. There can be two types of tokens: cash tokens and voucher tokens. *Cash tokens* are fungible tokens that are fully exchangeable with each other and can be divided into smaller denominations for multiple transactions. In contrast, *voucher tokens* are non-fungible tokens that are unique and must be spent in a single transaction regardless of the price of a product or service. Both cash tokens and voucher tokens can be developed as token templates.

Fig. 3 shows the sequence of activities required to create and utilise a token template. In the token template smart contract, the user can define the token's attributes and methods to manage the token. The common attributes usually consist of essential information such as token name, symbol, number of decimal places, and type. Other attributes such as the minting process, initial distribution of tokens, and burning process could be specified depending on the token's intended use. The specified methods enforce rules that govern how the tokens are minted, owned, and transferred. Optional attributes and methods in the template allow customisation. If that is insufficient, the template smart contract can be extended by another smart contract to support customised attributes and methods. Once developed, the template would change rarely. Thus, it can be thoroughly tested and certified to be free of defects before the deployment. Once developed, deploy the template to the blockchain as a factory smart contract. The template contract can be made a factory contract by adding a public method to spawn
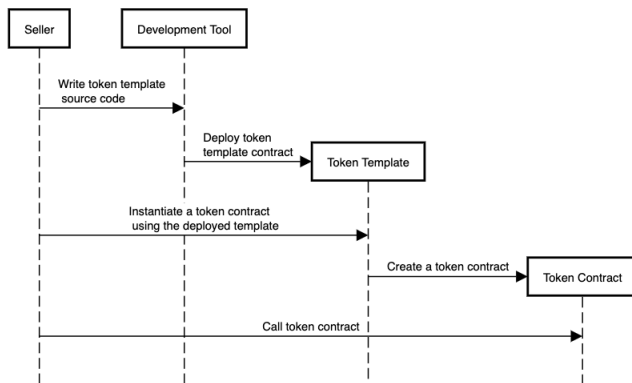
Figure 3: Sequence diagram of token template.

new child contracts with given properties. Whenever a new token type is to be created, issue a transaction with relevant properties to the spawn function in the factory contract. In turn, the factory contract instantiates the template with the given properties to spawn the token contract. Subsequently, transactions could be issued to the token contract to transact using the newly created tokens. Alternatively, if the new token contract is developed by extending the template, the template can be kept off-chain. The new token contract can then be directly deployed on to the blockchain, which internally carries template contract code (i.e., parent contract).

Consequences:

Benefits:

- *Productivity and security.* A token template can support developing a particular type of token by creating an instance of the token template. A well-defined and tested template simplifies blockchain application development, reduces time, and enhances security and reliability.
- *Flexibility.* A blockchain application might support different types of tokens with different rules. The use of token templates in the design can improve modularity.
- *Interoperability and liquidity.* As the token template can be designed based on a token standard, it offers high interoperability and liquidity through token transfer and swap.

Drawbacks:

- *Cost.* If a public blockchain is used, an extra cost is required to deploy the token template contract on the blockchain and call its token contract instance creation function. Further, the generalisation of the template increases the cost of computational and storage complexity of the smart contract, increasing the transaction fees needed to deploy and manipulate it.
- *Productivity.* It takes time to design, develop, and test a token template carefully. However, this is a one time process.

Related patterns: The *token registry* pattern can be used to maintain the ownership information of tokens generated using the token contract. Terms and conditions that a generated token needs to satisfy can be specified by linking the token to policies specified using the *policy contract* pattern. Once redeemed, a spent token can be destroyed or marked as unusable using the *burned token* pattern.

Known uses:

- *ERC-20*[4] and *ERC-721*[5] have emerged as the de-facto standards for implementing fungible and non-fungible tokens in Ethereum and sister blockchains, respectively, by instantiating reference implementations like OpenZeppeline[6].
- *Lorikeet* [11] is a model-driven engineering tool that provides off-chain templates for the developers to customise fungible and non-fungible asset data contracts based on ERC-20 and ERC-721 interface definitions.
- *Token Mint*[7] is a tool to create fixed supply and mintable ERC-20 and ERC-777 tokens and crowd sale contracts without coding.
- *Token Launcher*[8] creates mintable ERC-20 tokens by setting the desired specification.

4.2 Token Registry

Summary: A token registry maintains the ownership information of tokens by mapping the token identifier and wallet address.

Context: A buyer (i.e., token owner) needs to pay a certain number of tokens to purchase a product or service. When a payment is made, the respective tokens' ownership needs to be transferred from the buyer to the seller. Further, tracking historical ownership of tokens is required to ensure traceability. The payment application needs to manage a large number of transacting parties, tokens, and transactions.

Problem: How can a large number of tokens, their owners, and transactions be effectively tracked?

Forces:

- *Dynamism.* The ownership of tokens could change frequently.
- *Scalability.* A large number of token and their owners need to be mapped.
- *Transparency.* Transfer of token ownership needs to be tracked for compliance auditing.

Solution: A table in the form of a token registry contract can be used to record the ownership information of tokens by mapping token identifier (ID) to the owner's wallet address. As seen in Fig. 4, the issuer first registers both the token ID and wallet address of the buyer as variables in the token registry smart contract. The registry may also keep track of the token balance. This registry provides the ground truth on token ownership, as it could be manipulated only through the smart contract by authorised parties. For example, while both the buyer and seller can verify their token status, only the buyer can initiate a request to transfer a token. When a product to be purchased, the buyer requests the registry to transfer the desired number of tokens. As it is in the seller's best interest to ensure it receives only the accepted types of tokens, the seller could then confirm the token transfer to indicate its willingness to accept the tokens. Once confirmed, the registry smart contract updates the buyer's and seller's token balances. Delivery of product is handled off-chain.

---

[4]See footnote 2
[5]See footnote 3
[6]https://docs.openzeppelin.com/contracts
[7]https://tokenmint.io/
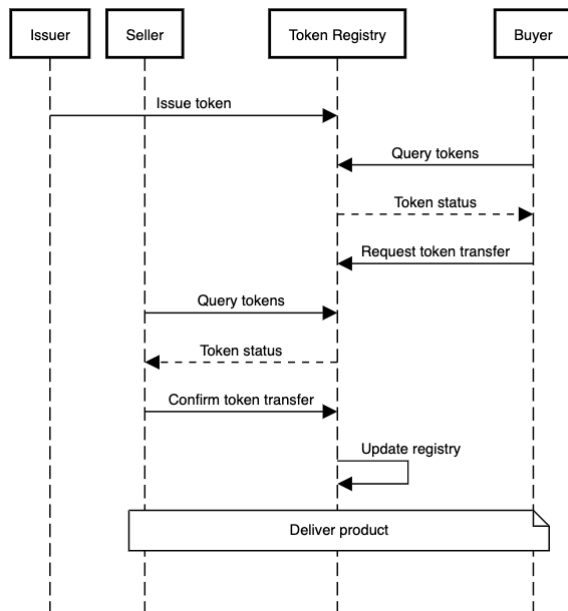[8]https://thetokenlauncher.com/

Figure 4: Sequence diagram of token registry.

Writing permission to the registry can be added through a permission control module. For example, only the issuer should register tokens and perform the forced transfer for legal action or fund recovery. Further, it can be specified that ownership transfer should proceed only if the token is in the spendable state or once the seller approves. Such conditions can be checked through a method in the registry. If the conditions are not met, the registry can throw an error rather than receiving the token. This pattern can be generalised to keep track of eligible sellers and a matrix that maps products and sellers. For example, a registry contract can maintain the seller attributes such as names, wallet addresses, certifications, and accolades.

Benefits:

- *Traceability.* Traceability is enhanced, e.g., how many tokens are owned by whom at what time is reflected in the registry.
- *Upgradability.* Because the token ownership and balance data are managed separately by the token registry, the rest of the smart contract code can be upgraded without changing the token registry contract.
- *Cost.* By separating token ownership data from the application code, there is no additional cost for data migration when the application logic is upgraded.

Drawbacks:

- *Cost.* Recording ownership data in smart contract incur an additional cost if a public blockchain is adopted. Further, there could be additional cost for updating the registry while executing a transaction.

Related patterns: The *token template* pattern can use the token registry to maintain the ownership information of tokens generated

using the token contract. It can also be used to build the credentials registry of sellers used in the *seller credential* pattern. Once redeemed, a spent token can be destroyed or marked as unusable by the registry using the *burned token* pattern.

Known uses:

- *Perth Mint Gold* token [1] uses a registry contract to map physical gold ownership (specified via GoldPass certificates) to the owner's Ethereum address.
- *Lorikeet* [11] is a model-driven engineering tool that generates centralised and distributed asset registry smart contracts compliant with ERC-20 and ERC-721 standards. It could further specify relationships and many-to-many mappings between attributes (within the same or a different registry), enforce access control, and track the history of changes made to assets tracked by the registry.
- *Parity Token Registry*[9] enables all the registered token to be visible to Parity wallet users.
- *Codefi*[10] and *Cardano*[11] provide an ownership registry for Universal and Ada tokens, respectively.
- *Making Money Smart*[12], *Stadjerspas Smart Vouchers*[13], and *Pension Infrastructure*[14] use registries to specify eligible sellers and fund providers.

4.3 Policy Contract

Summary: A *policy contract* specifies the policies that a token must satisfy, e.g., whether a token must be spent on the product or services supplied by eligible sellers or how many tokens can be spent in a transaction.

Context: A payment should be settled only when specific business rules or policies are satisfied. For example, government support programs or charitable donations usually indicate limits and constraints regarding products/services that could be purchased, their quantities, and eligible sellers.

Problem: How to specify policies governing the use of one or more tokens and validate them by the payment applications before allowing a payment?

Forces:

- *Diversity.* A variety of policies often govern the use of a token. They are challenging to find and interpret in relation to one another.
- *Dynamism.* The policies or business rules are dynamically attached/removed to/from tokens.
- *Complexity.* The policies or business rules are complex and difficult to codify.

Solution: A policy smart contract can be designed to specify policies that a token must satisfy before use. Fig. 5 shows the sequence of activities to specify and use a policy contract. First, the issuer creates a policy smart contract and deploys it to the blockchain.

---

[9]https://github.com/openethereum/token-registry
[10]https://github.com/ConsenSys/UniversalToken
[11]https://github.com/cardano-foundation/cardano-token-registry
[12]https://commbank.com.au/business/business-insights/making-money-smart.html
[13]https://stadjerspas.nl/
[14]https://apg.nl/en/publication/apg-and-pggm-develop-a-blockchain-application-for-pension-administration/
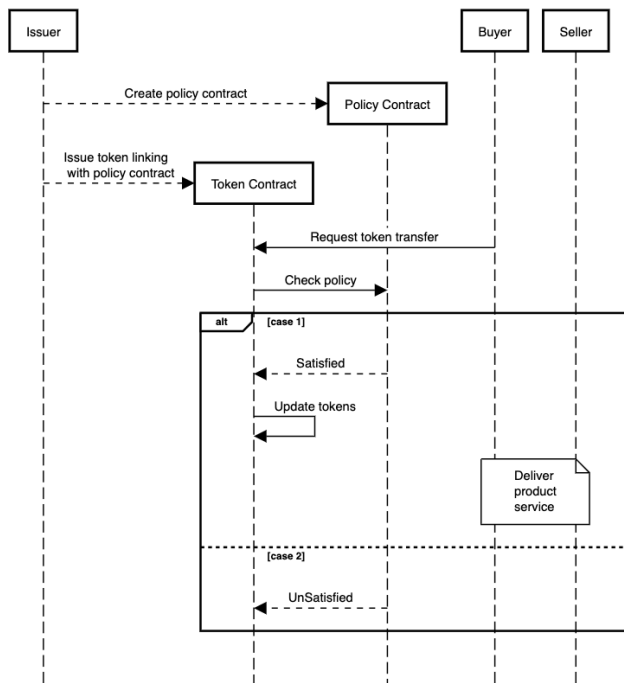
Figure 5: Sequence diagram of policy contract.

For example, the contract could specify that an attached token is spendable only on medicine, and the corresponding policy check logic could validate whether a given pharmacy is registered in the approved seller registry (i.e., a registry smart contract that maintains the information of eligible sellers from whom the token can be spent to purchase that medicine). Then the token contract can link the deployed policy to any newly issued token. When the buyer requests the token transfer during payment, the token contract (or token registry, not shown in the figure) can validate the policy by calling the policy contract. When all the policy conditions are satisfied, the token balance is updated on the token contract (or token registry). If the payment conditions are unsatisfied, the transaction is rejected.

The policy check logic could be implemented as smart contract methods that are invoked before the payment function. It is also possible to specify a generic method to validate a policy document given in a pre-specified format. For example, JSON schema could be used to specify policies that can be codified as key-value pairs. The mapping of policy contracts can happen at the level of either a single or group of tokens. In some instances, policies are expected to remain the same until the token is redeemed (or at least within a given time window). In other cases, the policies governing the tokens may change with time (e.g., changes in service agreements). Thus, the issuers may want to enforce the latest policy at the time of spending. Typically, a policy contract needs to be designed with a generic interface such that it can be dynamically attached/detached to/from the token. The attachment can be specified as a tuple consisting of the address of policy smart contract and list of policy functions.

Benefits:

- *Dynamism.* Policy contracts can be flexibly attached, updated, or removed from a token.
- *Diversity.* Policy contracts enable issuers to specify a variety of policies in policy contracts for the usage of different tokens.
- *User Productivity.* Multiple policies and tokens could be automatically checked using smart contracts without needing the buyer and seller to interpret them manually.

Drawbacks:

- *Cost.* If a public blockchain is used, an extra cost is required to deploy a policy contract on blockchain and call its functions at the time of transacting.
- *Developer Productivity.* Policies are challenging to codify; hence, it takes time to carefully design, develop, and test a policy contract. However, this is a one time process.

Related patterns: *Token template* and *token registry* patterns can use the policy contract to link and validate the policies that a token needs to satisfy. A policy contract can also be referred when a third party conducts a transaction using the *authorised spender* pattern.

Known uses:

- *Making Money Smart*[15][17] is a joint pilot project by the Commonwealth Bank of Australia and CSIRO's Data61 that attaches policy contracts to each pouch of tokens issued to redeem National Disability Insurance Scheme (NDIS) benefits in Australia. The policy defines what healthcare services could be used, their quantities, and eligible sellers.
- *Smart Money*[16][18] is a blockchain-based payment platform jointly developed by Kela (The Social Insurance Institution of Finland) and TietoEVRY in cooperation with the Finnish Tax Administration, Financial Supervisory Authority of Finland, and Borenius Attorneys. Spending rules are defined as constraints for smart money tokens and are verified at the time of a transaction.
- *ERC-1400*[17] is a library of standards for security tokens on Ethereum that specify transfer restrictions, rights, and obligations. Polymesh[18] specify how those restrictions can be specified using JSON schema in a KYC (Know Your Customer) setting.
- *CAIPY* [2] is a blockchain-based car insurance policy framework. In the design of *CAIPY*, policy contracts model process fragments defined by the physical insurance policies between customers and insurers. The policy contracts in *CAIPY* only govern the conditions for claims, not the use of tokens as insurance payments.

4.4 Burned Token

Summary: Once the redeem process is finalised, the spent tokens become unspendable.

Context: Once the sellers receive tokens from the buyers, they need to redeem the tokens for fiat currency (e.g., US Dollars). Once the tokens are redeemed in return for a fiat currency payment, they should not be further usable (i.e., prevent double-spending).

---

[15]See footnote 12
[16]https://tietoevry.com/en/campaigns/2020/smartmoney--a-conditional-digital-payment-guarantee/
[17]https://github.com/ethereum/EIPs/issues/1411
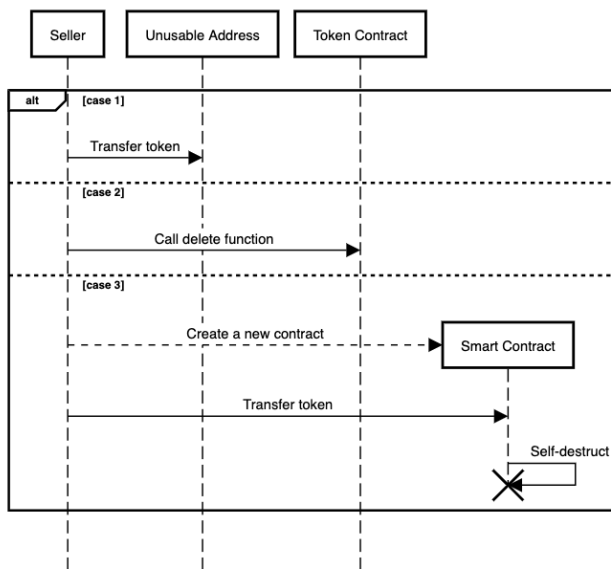[18]https://developers.polymesh.live/originate/sdk

Figure 6: Burned Token.

Problem: How to ensure that a token is unspendable once it is redeemed?

Forces:
- *Double spending.* A token should be redeemed only once.
- *Complexity.* There are a large number of redeemed tokens that need to be maintained and tracked.
- *Vulnerability.* Vulnerability might exist if the token contract or the token registry is not appropriately managed or contains bugs, e.g., the private key is stolen.

Solution: Potential misuse of a token can be prevented by either destroying or locking the token once redeemed (aka., removed from circulation). This process is referred to as *token burning* and could be achieved in several ways. Fig. 6 illustrates three ways of achieving this. In the first approach, the seller can transfer the tokens to an unusable/invalid address (aka., burn address). Such an address should not have a corresponding private key that can control the tokens it receives. The second way is to delete a token by calling the delete/burn function on the token contract used to create it or the token registry that keeps track of balances. The third method is to deploy a smart contract that immediately self-destructs as soon as it receives a token. For archiving, burned tokens could be moved to the token owner's read-only token wallet that any party cannot operate on the seller side.

Consequences:

Benefits:
- *Redeem once.* Double spending is alleviated as the token is no longer spendable or even exist.
- *Integrity.* The blockchain ensures the integrity of token status.

Drawbacks:
- *Cost.* If a public blockchain is used, extra cost incurs when calling the smart contract functions to burn the token.

- *Traceability.* The burned tokens cannot be tracked. On the seller side, this can be tracked by moving to the read-only wallet.

Related patterns: Once a token is redeemed, *token template* and *token registry* patterns can use the burned token pattern to mark a token as unspendable.

Known uses:
- *Perth Mint Gold* token [1] holders can redeem their tokens to either get Perth Mint gold or fiat currency. Before redeeming, the token owner needs to call the burn function on the smart contract to burn the token. Once the burned token is confirmed, the mint issues the gold or fiat currency.
- *Smart Money*[19] developed by TietoEVRY maintains burned tokens in a separate wallet.
- In *Making Money Smart*[20] project, sellers can redeem smart tokens for payment in Australian Dollars.

4.5 Seller Credential

Summary: The buyer can validate the seller's business qualifications through seller credentials.

Context: In blockchain-based payment applications, different policies stipulate rules for the usage of tokens. Some of these rules specify eligible sellers and their products or services regarding their qualifications or qualities.

Problem: How can the buyer verify the eligibility of the seller before transferring a token to buy a product or use a service?

Forces:
- *Immutability.* A payment cannot be reversed if a token is later found to be ineligible to spend for the chosen product, service, or its seller.
- *Complexity.* A seller's qualifications or qualities might be complex and difficult to codify.
- *Privacy.* The disclosed eligibility information should contain minimum data necessary to identify relevant aspects of oneself without revealing additional information.
- *Authentication.* The authenticity of proof of eligibility documents must be ensured as the documents might be counterfeit.

Solution: When a token stipulates rules for its usage in term of the eligibility of a seller and its products or services, the sequence of activities illustrated in Fig. 7 can be used to prove the eligibility to a potential buyer. First, the seller applies for a credential from the token issuer. Once approved, the credential is stored in the seller's wallet and the credential registry (i.e., a registry that tracks the seller's attributes using the *token registry* pattern). Third, when a token requires a specific credential from the seller to be validated before spending, the seller can present the credential to the token owner (i.e., buyer) as proof of eligibility. Finally, the buyer then validates the credential from the credential registry.

A token could stipulate rules for its usage in term of the eligibility of a seller or its products/services using the *policy contract* pattern. Similarly, the seller's credentials, qualifications, and qualities could be specified as a set of attributes (e.g., codified as a JSON schema)

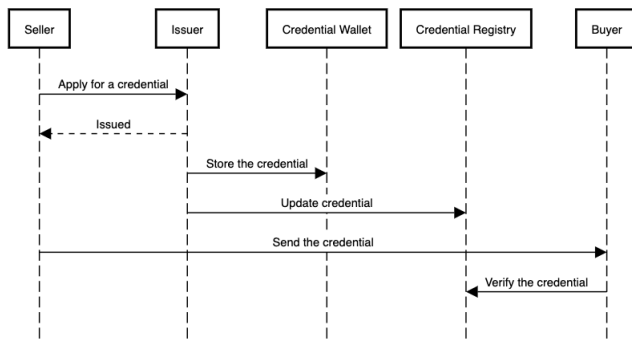[19]See footnote 16
[20]See footnote 12

Figure 7: Sequence diagram of seller credential.

or digital certificates signed by the issuer. The credential registry could be implemented similar to the *token registry* pattern. Specific use cases also require credentials of the buyer to be validated by the seller. For example, the seller may want to validate the buyer's age or whether the buyer has the right to spend the given token. In such cases, the pattern can be applied by interchanging the role of seller and buyer. Rather than sending the credential to the seller's wallet, the issuer could send an identifier/link to the credential stored in the credential registry. Either way, the seller should share the credential (or relevant parts of its) or its identifier with the potential buyer. This can be handled off-chain, e.g., using the sharing functionality provided by the seller's wallet or attaching the identifier/link to the invoice. Even if the credential is shared, the buyer may want to check the credential registry to ensure that the issuer has not updated or revoked the credential presented by the seller.

Consequences:

Benefits:

- *Immutability.* As the seller's credential is verified before initiating the payment, need for payment reversal does not arise.
- *Authenticity.* Digital signatures can guarantee the authenticity of credentials. Also, only the issuer is authorised to add credentials to the credential registry.
- *Reusability.* The seller credential can be used for all related tokens associated with the same policy.
- *Cost.* No cost is involved in sharing the credential to multiple buyers as it is handled off-chain.
- *Upgradability.* The credential details can be updated once it is issued to a seller by issuing a new credential and updating the credential registry.

Drawbacks:

- *Security.* A seller may present a revoked or old credential; hence, the buyer needs to validate with the credential registry.
- *Cost.* A transaction fee needs to be paid on a public blockchain to add a credential to the credential registry. Further, transaction fees need to be paid when the credential verification process involved more complex operations than read-only ledger data access.

Related patterns:

The *token template* pattern specifies tokens that require seller validation, and the *policy contract* pattern specifies the seller credentials to be validated. The *token registry* pattern can be used to build the credential registry. When seller credential involves external/physical attributes, those can be verified using the *oracle* pattern.

Known uses:

- *Smart money*[21] developed by TietoEVRY defines spending rules for tokens. Credentials are created and issued to the registered merchants. Credentials need to be checked before making the payment.
- *Polymesh*[22] specify credentials such as seller's KYC status and whether the seller is affiliated, accredited, or exempted from a specific jurisdiction.
- *Sovrin*[23] provides a blockchain-based self-sovereign identity infrastructure to issue credentials.
- *uPort Serto*[24] provides a blockchain-based credential management platform to achieve self-sovereign identity.

4.6 Escrow

Summary: Before making a transaction, tokens are transferred to a third-party smart contract called the *escrow*. The escrow holds the deposited tokens until the payment conditions are satisfied.

Context: The parties involved in the transaction need to ensure that both the agreed product/service is delivered and payment is made. One party should not be able to default the transaction at the expense of the other party.

Problem: How to ensure that the buyer gets the desired product/service while ensuring the seller gets the payment?

Forces:

- *Trust.* It is hard for the parties involved in a transaction to completely trust a third party with their private information and funds.
- *Efficiency.* It is time-consuming to establish a trust account and regulate the funds in a centralised payment system.
- *Cost.* Centralised payment systems often charge high fees for trust account services.
- *Security.* When the funds are controlled from a single point, there is a possibility of funds could be stolen or mistakes are made.
- *Volume.* Buyers and sellers may be engaged in a large number of transactions.

Solution: As seen in Fig. 8, a smart contract can play the role of an escrow that holds the fund until the payment conditions are fulfilled. First, specify the settlement procedure and conditions as a smart contract. This smart contract could be specified and deployed by either the seller or buyer. Second, the buyer transfers the token(s) to the escrow smart contract. Third, when token release conditions are met by providing the desired product/server, the respective event is informed to the escrow smart contract. Finally, the escrow

---

[21]See footnote 16
[22]See footnote 18
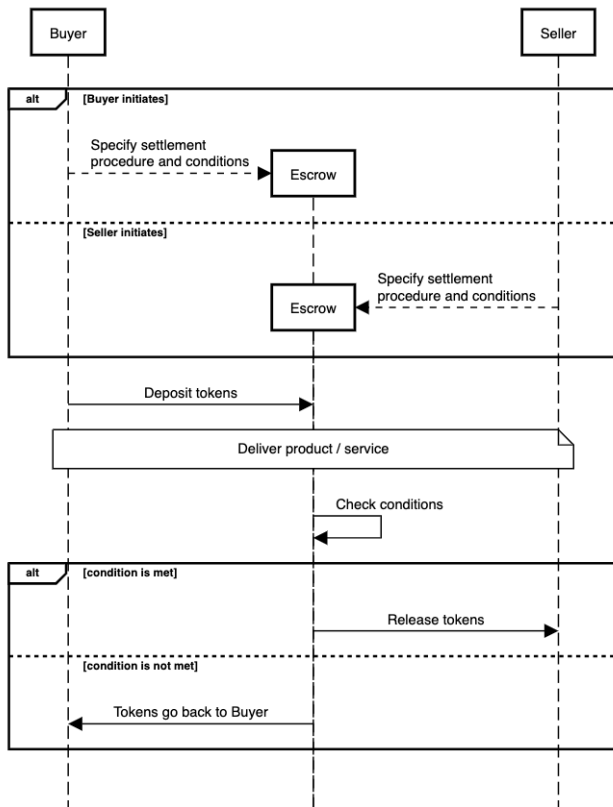[23]https://sovrin.org
[24]https://serto.id

Figure 8: Sequence diagram of escrow.

validates the pre-defined conditions and releases the tokens to the seller. If the respective event is not informed to the escrow within the stipulated time or the event indicates that the product/service was not delivered as per the agreed terms, then the tokens are sent back to the buyer.

If the payment conditions depend only on on-chain data, then a delegated call can be made to the escrow contract to inform the delivery of the product/service. If the payment conditions depend on external data like the shipment of a product, an *oracle* pattern can be used to provide desired data to the escrow. The security of escrow functionality implemented by the smart contract can be ensured as the smart contract code is immutable once deployed on the blockchain. This gives the parties involved in the transaction confidence that they will not be cheated during the trade. Care must be taken to ensure that the specified settlement conditions are unambiguous. For example, if the delivery is to be performed within 7-days, smart contract should also specify from when clock should start and whether it is sufficient for the seller to ship the product or whether it needs to be received by the buyer by the deadline. Further, it is recommended to specify the time in terms of a block number, as the block timestamp is not precise or could be crafted by a miner within a specific window.

Consequences: Benefits:

- *Trust and security.* An escrow smart contract reduces the risk of fraud by acting as a neutral party and ensuring proper escrow logic execution.
- *Transparency.* Operations happening in the system are transparent as relevant transactions are accessible to all blockchain participants.
- *Efficiency.* Blockchain eliminates the need for third parties, which in turn helps to reduce transaction cost and enhances service efficiency.

Drawbacks:

- *Cost.* Transaction fees need to be paid to deploy and execute escrow smart contract on public blockchains.
- *Privacy.* As all blockchain transactions are transparent, escrow transactions can potentially leak sensitive business information, e.g., the rate of disputes with customers.

Related patterns: Tokens can be transferred to/from the token contract generated using the *token template* pattern. If the *token registry* pattern is used, seller's and buyer's balances need to be updated on the registry. *Oracle* pattern can be used to provide the data required to validate physical product or service delivery. Alternatively, the buyer and seller may sign a transaction using the *multisignature* pattern to inform the escrow about the successful delivery of the product or service. *Stealth address* pattern can be used to generate one-time addresses to transact with the escrow to enhance privacy.

Known uses:

- *Kleros Escrow*[25] is a blockchain-based trustless dispute resolution platform that provides escrow services for cross-chain asset swaps.
- *Counos*[26] is a blockchain platform based in Switzerland, which offers financial and payment services, including multisignature-based escrow for cryptocurrencies.
- *IBC Group*[27] is a blockchain financial services company that offers smart-contract-based cryptocurrency escrow services charging 1% fee in fiat currency.
- *Merchant Token* [6] brings consumer protection concepts from the traditional card payment industry to blockchains by locking away a buyer's payment to a seller with less reputation until delivery of the product/service is confirmed, or disputes are resolved.

4.7 Payment Channel

Summary: A two-way pathway (aka., payment channel) is established between transacting parties to perform transactions off-chain while recording opening and final settlement transactions on the blockchain.

Context: A set of low-value payments are to be made frequently, e.g., a small payment paid every time a WiFi service is used.

Problem: If a public blockchain is used, micro-payments are too expensive to make as the transaction fee might be higher than the monetary value within the transaction. Also, it takes time to achieve

---

[25]https://kleros.io/en/escrow
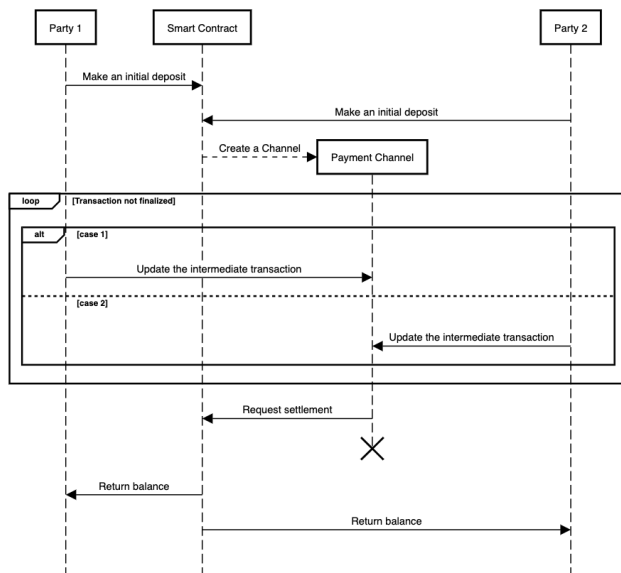[26]https://counos.io
[27]https://ibcgroup.io

Figure 9: Sequence diagram of payment channel.

finality on a blockchain. Therefore, how can frequent payments be processed without waiting for a long time and incurring high transaction fees?

Forces:

- *Latency.* It might take a long time to confirm a transaction on the blockchain, although users usually expect instantaneous settlement of payments.
- *Cost.* If a public blockchain is used, the transaction fee might be higher than the monetary value of the payment.

Solution: An off-chain, two-way pathway (aka., *payment channel*) can be established between two parties to handle frequently occurring transactions. Fig. 9 depicts the sequence of activities involved in a payment channel. First, an initial deposit is made by one or both parties (e.g., buyer and seller) to a smart contract that locks up tokens as a security. The buyer and seller can then engage in a sequence of transactions where both parties digitally sign each intermediate transaction. The signed transactions are recorded off-chain. When the settlement is to be made, each party's closing balance is informed to the smart contract. Finally, the smart contract returns the tokens as per the closing balance of each party.

The buyer and seller may also interchange the roles such that tokens flow both ways. Each intermediate transaction should carry a sequence number to correctly calculate the settlement balance and ensure no transaction is missed. Recorded intermediate transactions may also be sent to the smart contract as proof, especially if there is any dispute. Settlement on the blockchain can also be performed periodically depending on the agreement between the two parties. The payment channel is discontinued once the settlement is requested.

Consequences:
Benefits:

- *Performance.* The off-chain micro-payment transactions can be settled instantaneously without waiting for finality on the blockchain.
- *Throughput.* Off-chain payment channels can achieve much higher throughput as they are not constrained by inter-block time.
- *Privacy.* Intermediate transactions processed by payment channels are not publicly available.
- *Cost.* If a public blockchain is used, only the initial deposit and final settlement transactions incur transaction fees.

Drawbacks:

- *Data Integrity.* The data integrity of the intermediate state of payment channels cannot be ensured because intermediate transactions are not stored on-chain and vulnerable to transaction malleability. However, disputes can be usually resolved by presenting all the intermediate transactions to the smart contract as evidence.
- *Lack of Liquidity.* To establish a payment channel, tokens from one or both parties of the channel needs to be locked in the smart contract during the payment channel's lifecycle.

Related patterns: *Token registry* pattern could be used by the smart contract to update the balance of the transacting parties at the time of payment channel initiation and settlement.

Known uses:

- *Bitcoin Lightning Network*[28] adopts an off-chain protocol to enable micro-payments. It only broadcasts the final version of the funding transaction to settle the payment in the Bitcoin network.
- *Ethereum Raiden Network*[29] uses a network of off-chain payment channels that enables secure money transfer.
- *Ethereum State Channel*[30] provides a channel protocol that supports exchanging state for general-purpose blockchain applications.

4.8 Stealth Address

Summary: Use a one-time address (aka., stealth address) for the transaction to protect the the privacy of the parties involved in the payment.

Context: Sellers and buyers are engaged in multiple transactions and are concerned about their privacy.

Problem: How can transacting parties protect their privacy when engaging in a large number of transactions?

Forces:

- *Privacy.* While a blockchain transaction is pseudonymous, it is possible to correlate transactions and reveal the real-world identity by linking transactions with other data (e.g., IP address). For example, by analysing the transaction graph, flow of tokens, and other external information, third parties could correlate the transactions and observe buyers' and sellers' behaviour and identify the entity behind an address.

---

[28]https://blockstream.com/lightning/
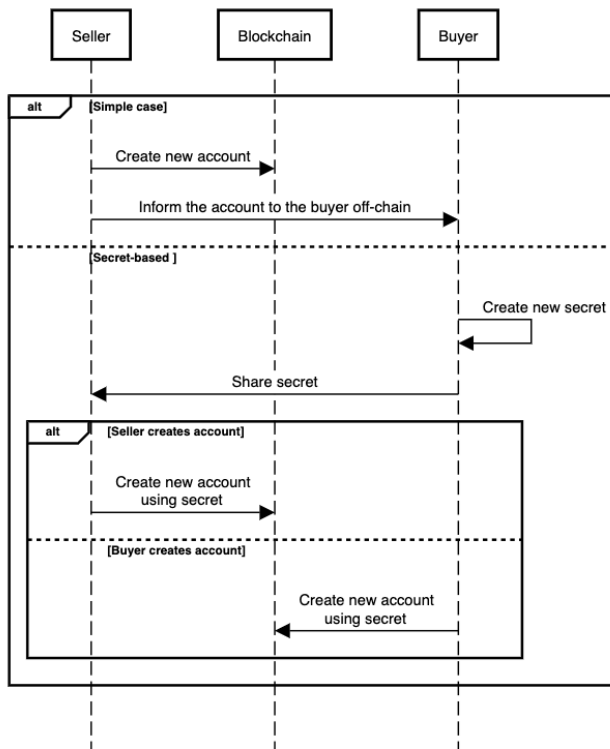[29]https://raiden.network/
[30]http://jeffcoleman.ca/state-channels/

Figure 10: Sequence diagram of stealth address.

- *Volume.* Buyers and sellers may engage in a large number of transactions.
- *Traceability.* Transacting parties and regulators expect to track the flow of tokens.
- *Auxiliary information.* Depending on the use case, buyers' and sellers' information may be available on-chain (e.g., *seller credential*) or off-chain (e.g., trade reports and census data).

Solution: Buyer's and seller's anonymity can be enhanced by using a one-time address referred to as the *stealth address*. Fig. 10 illustrates the sequence of activities required to generate a stealth address and share it between transacting parties using two techniques. Note that the buyer and seller roles in Fig. 10 are interchangeable. In the first technique, the seller creates a one-time address/account for each transaction on behalf of the buyer such that different transactions sent to the same recipient are unlinkable. The seller could inform the new account to the buyer off-chain, e.g., an invoice can contain the new address. The second technique shares a key/secret off-chain between the two parties that can be used to generate a new account for each future transaction. For example, a one-time secret can be shared between the parties to use as the seed to create a public-private key pair of a new account. Then the hash of that key can be used as a seed to create the next account. This could be extended to a hash chain, where each round of hashing produces a datum that can be used as the seed to create a new public-private key pair. However, this would mean both seller and buyer can spend the token as the private key for the account are known to both. This could be overcome by modifying the public-private key generation

process as presented in the BIP-32 standard[31]. A couple of other solutions could also be used to share the secret on-chain, e.g., reuse a public key, reuse a signature nonce, and the seller's addressing requiring to have a specific prefix[32].

Consequences:

Benefits:
- *Privacy.* As a new address is used for each transaction, it is not straightforward to correlate and trace transactions on the blockchain. This enhances the protection of the real-world identity of the recipient.

Drawbacks:
- *Traceability.* As new addresses are created based on an off-chain secret, it is difficult to track transactions targeted to the same physical recipient. Such a lack of traceability could be leveraged to engage in illegal activities.
- *Volume.* The recipient of payments will find it difficult to link, reconcile, redeem, and manage transactions as each transaction is linked to a different address. Moreover, transactions may still contain additional details such a product/service types and quantities, when aggregated, could reveal transacting parties behaviour and identity. Furthermore, if the recipient wants to later aggregate tokens without revealing its identity, it has to use token mixing services, which incur additional costs.

Related patterns: Stealth addresses can be used with *token registry* and *escrow* patterns to hide the address of sellers and buyers.

Known uses:
- *Basic Stealth Address Protocol* (BSAP)[33] adopts the Elliptic Curve Diffie-Hellman (ECDH) protocol to deal with Bitcoin transactions. *Improved Stealth Address Protocol* (ISAP)[34] is an upgraded version of BSAP with an additional key creation feature. The recipient is the only entity that can compute the private key for the temporary address to receive Bitcoin.
- To address overuse of private spending key, Dual-Key Stealth Address Protocol (DKSAP) is designed for a wallet solution *ShadowSend*[35] uses two pairs of keys, a scan key pair and a spend key pair to provide decentralised anonymous currency.

4.9 Oracle

Summary: To examine the fulfilment of payment conditions, an oracle is used to introduce the state of external systems into the blockchain execution environment.

Context: The payment is conditional on the physical properties of a product/service or completion of a real-world action, e.g., quality attributes of a product or shipment of a product. Typically such data are provided or certified by third parties.

Problem: How to fetch external state/data into a smart contract to decide on a payment condition?

Forces:

---

[31]https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
[32]https://sourceforge.net/p/bitcoin/mailman/message/31813471/
[33]https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html
[34]https://bytecoin.org/old/whitepaper.pdf
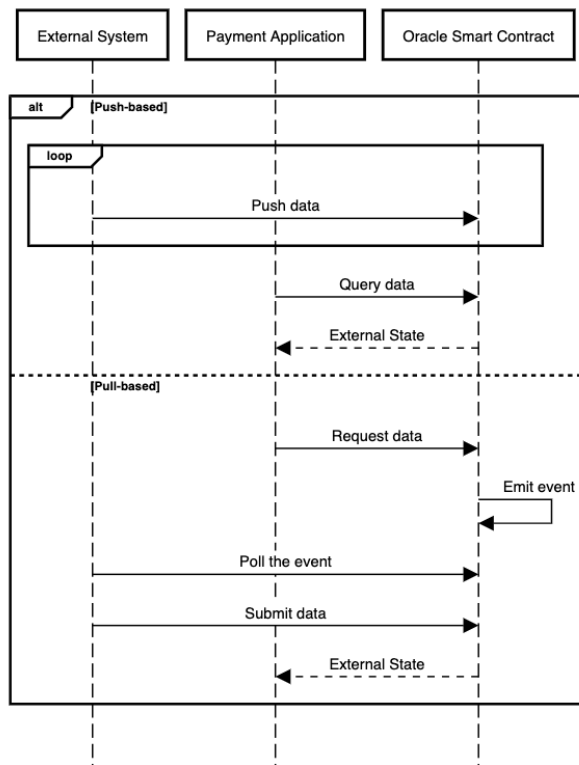[35]https://github.com/shadowproject

Figure 11: Sequence diagram of oracle.

- *Connectivity.* Blockchain-based payment applications may require external state data, e.g., the ambient temperature of a container. However, blockchain is a self-contained execution environment. Smart contracts in blockchain-based payment applications cannot read the state data of external devices or systems.
- *Trust.* Both the transacting parties need to agree on the state of a product and its attributes. When such data are provided by a third party, the provider needs to be trustable to transacting parties, including any regulators.
- *Availability.* The external devices or systems that provide data may change or even disappear.

Solution: A third-party hosted smart contract can be used to provide data to the blockchain from external servers, websites, and IoT devices. These external devices/systems call functions on the smart contract to record the data on the blockchain using their private key attesting the data origin. The system consisting of the smart contract and external elements are referred to as an *oracle*. As illustrated in Fig. 11, an oracle could allow other smart contracts to either push or pull data from off-chain components. In the first case, the external system sends the desired data to the oracle smart contract using a transaction. This transaction should invoke a method that accepts data as an input parameter. When the payment application needs to access the data, it can make a delegated call to the oracle smart contract to read the data. In the second case, the payment application indicates its need to access

external data to the oracle using a delegated call. In response, the oracle emits a blockchain event indicating the desired data. External system should poll for such events to determine what data are requested by the oracle. As the blockchain cannot directly interact with external systems, events provide a passive communication mechanism between a smart contract and external systems that poll for relevant events. Once the requested data are available, the external application submits the data to the blockchain using a transaction. Consequently, the oracle makes a delegated call to the smart contract to deliver the requested data.

When the smart contract calls the oracle, if it does not have the desired or latest data, a subscription could be made to the oracle by specifying the smart contract address and method to be called. Once the data are available, the oracle can call the smart contract using a delegated call. A new transaction could be issued every time the external state changes. Even if there is no change in data, some applications may require the external system to submit a transaction indicating no change in data. There are two types of oracles: centralised and decentralised oracles. A centralised oracle introduces a single trusted third party into the system, which might become a single point of failure. Alternatively, a decentralised oracle is designed based on multiple external servers and multiple independent data sources, overcoming the single point of failure. A decentralised oracle can use multisignature or voting to reach a consensus on the validity of data injected into the oracle. Similarly, at the hardware level, redundant IoT devices could be deployed to deal with adversary IoT devices that produce malicious data. Then the oracle smart contract can cross-check the consistency of data collected by different IoT devices.

Oracles can be used to access the physical attributes of a product to avoid frauds in trading. A physical item's integrity can be verified by connecting the physical item with its digital representation (i.e., *digital-physical parity*). The physical attributes of an item (e.g., chemical composition, geo-location, and visual features) can be extracted via various techniques such as genomics analysis, isotope analysis, and machine learning. The captured attributes can be digitally recorded on the blockchain through oracles to ensure immutability and transparency.

Consequences:
Benefits:

- *Connectivity.* Payment application's smart contracts can access external states through the oracle.
- *Trust.* A decentralised oracle increases trust as it relies on trust on a majority of third parties.
- *Availability* A decentralised oracle can remove the risk of a single point of failure and increase availability.

Drawbacks:

- *Trust.* A centralised oracle is a third party integrated into the system, which needs to be trusted by all stakeholders involved in the payment.
- *Data Integrity.* Validators can only check the digital signature of the oracle but cannot check the correctness or originality of the injected external data.
- *Availability* A centralised oracle may lead to a single point of failure.

- *Cost.* There is a cost in injecting data into oracle smart contract and looking up data from the oracle. Further, the cost of injecting data from the external world increases with the number of oracles being used and the rate of data change.
- *Time.* It could take a longer time for the decentralised oracle to reach a consensus over the state data to be injected.

Related patterns: *Policy contract* and *seller credential* patterns can use an oracle to verify physical attributes about a seller or its products. The *multisignature* pattern can be used to sign the data submitted to a decentralised oracle.

Known uses:

- *Oracle* in Bitcoin[36] is a server outside the Bitcoin blockchain, which can evaluate user-defined condition expressions for payment based on the external state data.
- *Provable*[37] is an oracle provider, which utilises trusted hardware to directly collect data from the external environment, which can be used to check payment conditions.
- *Orisi*[38] is an oracle framework for Bitcoin smart contracts that enables the transaction participants to select a set of oracles and apply multisignature to confirm the status.
- *Merchant Token* [6] uses a decentralised oracle to confirm a product/service delivery or resolve disputes before releasing funds locked in the escrow to the seller.

4.10 Multisignature

Summary: Given a pool of n addresses that could authorise a payment, get a subset m of them to authorise a payment by signing the respective transaction ($m \leq n$).

Context: A transaction needs to be authorised by multiple parties (i.e., blockchain addresses), e.g., to release tokens held by an escrow, both buyer and seller need to confirm successful delivery of product or service.

Problem: How can more than one party approve a transaction?

Forces:

- *Dynamism.* A transaction needs to be authorised by a subset of authorities pre-defined in a pool of addresses. However, some of the authorities may not be available, unreachable, or related private key may be compromised when issuing the transaction. Also, the address pool needs to be changed as the payment application evolves, e.g., the addition of a new authority or revocation of an existing authority.
- *Tolerance of lost key.* Losing a private key on the blockchain means permanent loss of control over a blockchain account and associated smart contracts as blockchain does not provide any key recovery mechanism.

Solution: A transaction could indicate multi-party authorisation using a digital signature scheme that allows a group of addresses to sign the transaction. Such a signature is known as a *multisignature*. Depending on the chosen multisignature algorithm, we can enforce that the transaction is signed by all or a subset of the authorised
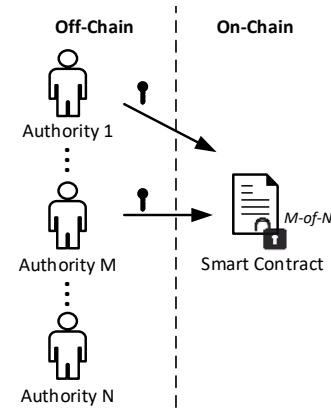
---

[36]https://en.bitcoin.it/wiki/Contract#Example_4:_Using_external_state
[37]https://provable.xyz/
[38]http://orisi.org/

Figure 12: Multisignature.

addresses. As depicted in Fig. 12, to enable flexible binding of pre-defined authorities for a payment transaction, a multisignature mechanism can be designed to require m out of n private keys to authorise a transaction, in which m is the threshold of authorisation.

Depending on the context, either the token issuer or buyer can pre-define a group of addresses that can authorise a payment and set the minimal number of authorisations required to approve a payment. While some blockchain platforms accept transactions containing a multisignature, a multisignature is more commonly validated using smart contracts. In the simplest form, each authorised address can submit a signed transaction to the multisignature smart contract that keeps track of the number of transactions with valid signatures. Once the number of valid transactions reaches the pre-specified threshold, the smart contract can perform the token payment. Alternatively, more efficient signature schemes, such as threshold signatures, could be used to generate a short signature off-chain and submit to the smart contract using a single transaction. In either case, the pool of authorised addresses needs to be pre-specified before issuing the transaction. When the membership change, threshold signature schemes usually require regeneration and distribution of all the keys/addresses.

Consequences:
Benefits:

- *Dynamism.* Multisignature allows flexible binding of authorities for approving a transaction.
- *Tolerance of lost keys.* The risk of losing control over smart contracts is reduced as one participant can have more than one blockchain address in case a private key is lost.

Drawbacks:

- *Risk of losing keys.* There is still a risk of losing control over smart contracts when m private keys among the n private keys are lost.
- *Cost.* If a public blockchain is adopted, storing multiple addresses, updating the list of addresses, and issuing multiple transactions cost money.

Related patterns: The *escrow* pattern can use a multisignature to get confirmation from both buyer and seller about the successful

delivery of a product. A decentralised *oracle* can use multisignature to confirm the physical state of a product from multiple attesters.

Known uses:

- *MultiSignature*[39] is offered by Bitcoin to authorise transactions.
- Multisignature wallet is provided in the *Ethereum Mist DApp* browser[40] to approve transactions.
- A multisignature wallet is used in *TokenMarket* security token to confirm critical activities that affect the shareholder balances.
- Data filed in the *ERC-1400*[41] standard can be used to submit a multisignature signed off-chain using a single transaction.

4.11 Token Swap

Summary: Token swap allows users to trade directly between two types of tokens as an atomic transaction.

Context: A blockchain-based payment application supports different types of tokens, e.g., different asset classes.

Problem: How can users buy and sell tokens for other types of tokens without the risk of other-party not transferring their tokens?

Forces:

- *Liquidity.* Users need to convert tokens to other types of tokens.
- *Atomicity.* Exchange of token should be atomic based on a rate/amount agreed by both parties of the transaction.
- *Trust.* It is hard for the parties involved in a transaction to trust each other completely.

Solution: By making the token swap atomic, we can ensure both parties will either receive the respective tokens or none of the tokens. A *token swap* is an agreement between two parties that exchange different token types (say token *A* and token *B*). In a token swap, one party will pay a certain amount of token *A* to the other party and receive the agreed amount of token *B* in return. The token swap process is usually managed using a Hashed Timelock Contract (HTLC) deployed on the blockchain. Multiple HTLCs are used to swap tokens in payment channels and across blockchains with the help of an application-layer integrator. Fig. 13 illustrates the steps involved in atomically swapping two tokens using a HTLC. First, a HTLC smart contract is deployed by one of the parties on-chain. HTLC specifies a hash generated from an off-chain secret and a timeout specified as a block number. Second, each party transfers the agreed amount of tokens to the HTLC. For example, party 1 transfers token *A* while party 2 transfers token *B*. Next, to swap the tokens, the off-chain secret is sent to the HTLC using a transaction. Finally, if the revealed off-chain secret generates the same hash embedded in the HTLC, the contract exchanges tokens by transfers token *A* to party 2 and token *B* to party 1. If the off-chain secret is not revealed before the timeout, tokens are sent back to their owners.

In payment channels and inter-blockchain token swap, two or more intermediate HTLC contacts are deployed, forming a chain
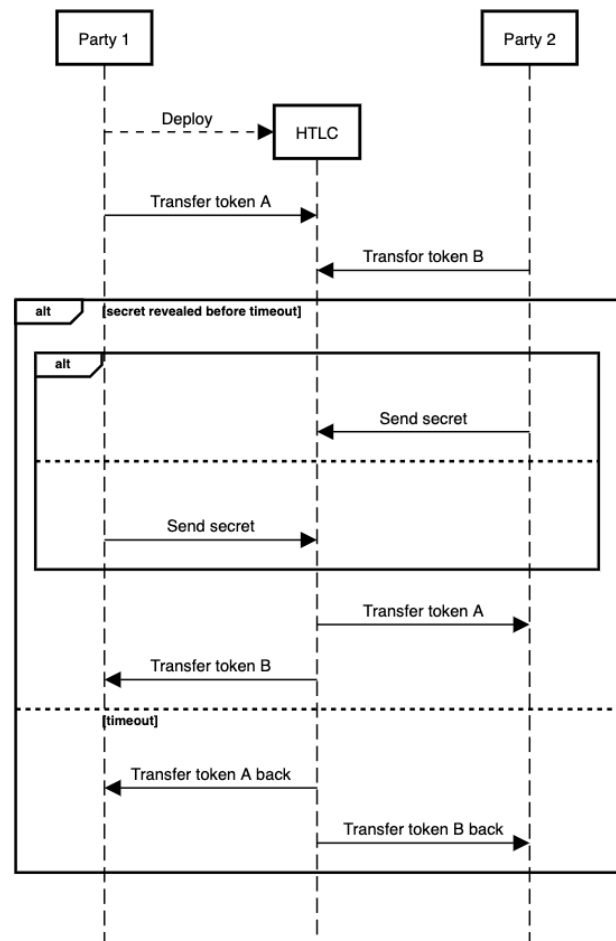
---
[39]https://en.bitcoin.it/wiki/Multisignature
[40]https://github.com/ethereum/mist
[41]See footnote 17

Figure 13: Sequence diagram of token swap.

of HTLCs. While each HTLC specifies the same hash generated from the off-chain secret, timeout decreases as we move along the HTLC chain. Then the off-chain secret is revealed from the end of the HTLC chain, leading to a chain reaction that enables each HTLC to swap the tokens it holds. In practice, one HTLC does not automatically transfer the off-chain secret to the next HTLC on the chain. Instead, each party interested in receiving tokens locked by a particular HTLC continues to look for a transaction that reveals the off-chain secret. Once it is observed, the off-chain secret is extracted from the transaction and submitted in a new transaction to the respective HTLC to unlock the tokens. The off-chain secret must be revealed to a HTLC before its timeout.

Consequences:

Benefits:

- *Liquidity.* A user can use his/her tokens to buy other types of tokens, increasing liquidity.
- *Data Integrity.* The data integrity of the swapped tokens can be ensured because the token swap process and respective transactions are stored on-chain.

- *Atomicity.* Smart contracts guarantee the atomicity of token swap.
- *Cost.* As smart contracts can manage the token swap process, no third party service fee is incurred, e.g., payment to an escrow.
- *Interoperability.* Interoperability is increased through cross-chain token swap.

Drawbacks:

- *Privacy.* Token swap transactions are publicly visible.
- *Cost.* There might be additional cost due to the exchanged rate. Also, if a public blockchain is used, there is a cost to deploy and use HTLC smart contract.
- Lack of flexibility. If a party does not withdraw tokens out in time by submitting the off-chain secret, the deposited tokens will be locked in the HTLC or sent back to the payer.

Related patterns: When exchanging on-chain assets like tokens, this pattern could be used over the *escrow* pattern, as no external confirmation of delivery of assets is needed. The *token registry* pattern can be used to send/receive tokens to/from HTLC.

Known uses:

- *Metamask*[42] supports token swap feature to compare and swap tokens directly within the wallet and browsers.
- *Kaileido*[43] offers a simple process to swap tokens using HTLCs securely.
- The *Bitcoin Lightning Network* [16] supports multi-hop payment channels by linking existing payment channels through a chain of HTLCs. This enables parties to transact beyond their payment channel.
- *AirSwap*[44] supports atomic token swap with a guaranteed price. It also supports swap via Metamask wallet and custom swaps with parties trading tokens.

4.12 Authorised Spender

Summary: An authorised third-party spender can spend a certain amount of tokens owned by the approver/buyer.

Context: In blockchain applications, buyers need to transfer tokens to sellers to buy products or services.

Problem: How can a buyer delegate a third party to pay on behalf of him/her?

Forces:

- *Security.* The approval process needs to be secure in case there is a dishonest delegate. For example, the delegate should be able to spend only within the approved allowance.
- *Complexity.* It is usually difficult to delegate a third party to pay on behalf of token owners.

Solution: As seen in Fig. 14, a token owner can approve a third-party user or a smart contract to transfer a certain number of tokens on his/her behalf. First, an approval step can be designed to allow the spenders to use a certain amount of tokens. For example, an approve method can be added to the token template contract
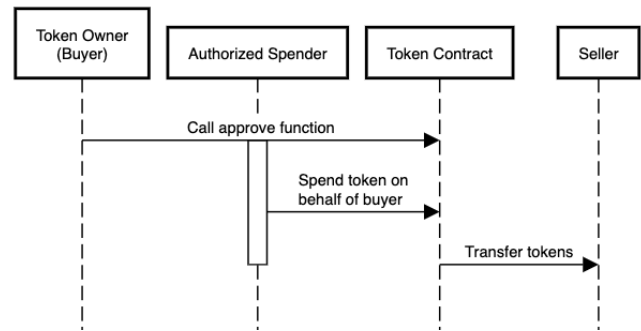
---

[42]https://metamask.io/
[43]https://www.kaleido.io/blockchain-platform/token-swap
[44]https://www.airswap.io/

Figure 14: Sequence diagram of authorised spender.

with third-party spender address and allowance as parameters. Alternatively, an authorised spender and allowance can be specified on the token registry. Next, when the third party needs to perform a payment on behalf of the buyer, it can send a transaction to the token contract requesting tokens to be transferred from the buyer to the sellers updating the balance of both accounts. Simultaneously, approved balance needs to be updated.

Consequences:

Benefits:

- *Flexibility.* Buyers can find a delegate to pay on their behalf within the allocated allowance.
- *Automation.* The buyers can automate payments by relying on the approved third party to perform the payment.
- *Security.* The tokens are transferred securely according to the conditions predefined in smart contracts.

Drawbacks:

- *Cost.* If a public blockchain is used, an additional cost is incurred to call the approval function.
- *Security.* There might be a bug in the smart contract that results in unexpected transfers by the deligate (e.g., wrong address or spending beyond the allowance).

Related patterns: The *token template* and *token registry* patterns are needed to support setting an authorised spender and an allowance for the delegate to act on behalf of the buyer.

Known uses:

- Both *ERC-20*[45] and *ERC-721*[46] use an approve function to enable a third party or a smart contract to transfer up to a specified number of tokens.
- *Ethex*[47] supports setting a custom allowance for token trading to ensure the decentralised exchange cannot spend more than the allowance even if its smart contract is compromised.
- *Saturn Network*[48] offers an approve feature that allows users to submit a transaction to approve Saturn Network to trade tokens on their behalf.

---

[45]See footnote 2
[46]See footnote3
[47]https://ethex.market
[48]https://www.saturn.network

5 CONCLUSION

To systematically organise knowledge on designing payment applications using blockchain, we collected 12 patterns covering critical aspects of blockchain-based payment applications. We identify the lifecycle of a token, in which the state transitions are associated with the collected patterns. The lifecycle along with the annotated patterns, provide a payment-focused systematic view of system interactions and a guide to the effective usage of the patterns in software architecture design. In future, we plan to extend our decision model to select blockchain-based patterns [21] to include payment patterns.

REFERENCES

[1] 2021. *Perth Mint Gold Token.* Technical Report. Trovio Operating Pty Ltd. https://pmgt.io/static/assets/pmgt_whitepaper.pdf

[2] Lennart Bader, Jens Christoph Bürger, Roman Matzutt, and Klaus Wehrle. 2018. Smart contract-based car insurance policies. In *2018 IEEE Globecom Workshops (GC Wkshps).* 1–7. https://doi.org/10.1109/GLOCOMW.2018.8644136

[3] HMN Dilum Bandara, Xiwei Xu, and Ingo Weber. 2020. Patterns for Blockchain Data Migration. In *Proceedings of the European Conference on Pattern Languages of Programs 2020* (Virtual Event, Germany) *(EuroPLoP '20).* Association for Computing Machinery, New York, NY, USA, Article 7, 19 pages. https://doi.org/10.1145/3424771.3424796

[4] Massimo Bartoletti and Livio Pompianu. 2017. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In *Int. Conf. on Financial Cryptography and Data Security.* Springer, 494–509.

[5] Alexandra Bratanova et al. 2019. *Blockchain 2030: A look at the future of blockchain in Australia.* Technical Report. Data61, CSIRO, Brisbane, Australia. https://www.researchgate.net/publication/332298704_Blockchain_2030_A_Look_at_the_Future_of_Blockchain_in_Australia

[6] John Cavebring, Axel Arvidsson, and Tricia Buenvenida. 2021. *DeFi Merchant Payment Protocol with consumer protection for Web 3.0.* Technical Report. https://static.hips.com/pdf/white_paper/merchant_token_white_paper.pdf

[7] Jacob Eberhardt and Stefan Tai. 2017. On or off the blockchain? Insights on off-chaining computation and data. In *European Conf. on Service-Oriented and Cloud Computing.* Springer, 3–15.

[8] Steven Goldfeder, Joseph Bonneau, Rosario Gennaro, and Arvind Narayanan. 2017. Escrow protocols for cryptocurrencies: How to buy physical goods using Bitcoin. In *Financial Cryptography and Data Security,* Aggelos Kiayias (Ed.). Springer International Publishing, Cham, 321–339.

[9] Yue Liu, Qinghua Lu, Hye-Young Paik, and Xiwei Xu. 2020. Design patterns for blockchain-based self-sovereign identity. In *25th European Conf. on Pattern Languages of Programs (EuroPLoP'20).*

[10] Yue Liu, Qinghua Lu, Xiwei Xu, Liming Zhu, and Haonan Yao. 2018. Applying design patterns in smart contracts. In *Blockchain – ICBC 2018,* Shiping Chen, Harry Wang, and Liang-Jie Zhang (Eds.). Springer International Publishing, Cham, 92–106.

[11] Qinghua Lu et al. 2020. Integrated model-driven engineering of blockchain applications for business processes and asset management. *Software: Practice and Experience* (2020), 1–21.

[12] Qinghua Lu and Xiwei Xu. 2017. Adaptable blockchain-based systems: A case study for product traceability. *IEEE Software* 34, 6 (November 2017), 21–27.

[13] Qinghua Lu, Xiwei Xu, Yue Liu, Ingo Weber, Liming Zhu, and Weishan Zhang. 2019. uBaaS: A unified blockchain as a service platform. *Future Generation Computer Systems* 101 (2019), 564–575. https://doi.org/10.1016/j.future.2019.05.051

[14] Doble J. Meszaros and Jim Doble. 1997. A pattern language for pattern writing. In *Proc. Intl. Conf. on Pattern Languages of Program Design.*

[15] Roman Mühlberger, Stefan Bachhofner, Eduardo Castelló Ferrer, Claudio Di Ciccio, Ingo Weber, Maximilian Wöhrer, and Uwe Zdun. 2020. Foundational Oracle Patterns: Connecting Blockchain to the Off-Chain World. In *Business Process Management: Blockchain and Robotic Process Automation Forum,* Aleksandre Asatiani, José María García, Nina Helander, Andrés Jiménez-Ramírez, Agnes Koschmider, Jan Mendling, Giovanni Meroni, and Hajo A. Reijers (Eds.). Springer International Publishing, Cham, 35–51.

[16] Joseph Poon and Thaddeus Dryja. 2016. *The Bitcoin Lightning Network: Scalable off-chain instant payments.* Technical Report. http://lightning.network/lightning-network-paper.pdf

[17] Daniel Royal, Nicole Lim, Mark Staples, Paul Rimba, and Sophie Gilder. 2018. Making money smart: Indicative data analytics that could be supported by the blockchain proof of concept. https://data61.csiro.au/en/Our-Research/Our-Work/SmartMoney

[18] TietoEvry and Kela. 2019. Smart money specification document v1.0. https://www.tietoevry.com/en/campaigns/2020/smartmoney--a-conditional-digital-payment-guarantee/#Smart-money-51723

[19] Maximilian Wohrer and Uwe Zdun. 2018. Smart contracts: Security patterns in the Ethereum ecosystem and Solidity. In *2018 Int. Workshop on Blockchain Oriented Software Engineering (IWBOSE).* IEEE, 2–8.

[20] M. Wöhrer and U. Zdun. 2018. Design Patterns for Smart Contracts in the Ethereum Ecosystem. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData).* 1513–1520. https://doi.org/10.1109/Cybermatics_2018.2018.00255

[21] Xiwei Xu, H. M. N. Dilum Bandara, Qinghua Lu, Ingo Weber, Len Bass, and Liming Zhu. 2021. A decision model for choosing patterns in blockchain-based applications. In *18th IEEE Int. Conf. on Software Architecture (ICSA 2021).*

[22] Xiwei Xu, Qinghua Lu, Yue Liu, Liming Zhu, Haonan Yao, and Athanasios V. Vasilakos. 2019. Designing blockchain-based applications a case study for imported product traceability. *Future Generation Computer Systems* 92 (2019), 399–406. https://doi.org/10.1016/j.future.2018.10.010

[23] Xiwei Xu, Cesare Pautasso, Liming Zhu, Qinghua Lu, and Ingo Weber. 2018. A pattern collection for blockchain-based applications. In *23rd European Conf. on Pattern Languages of Programs.* 1–20.

[24] Peng Zhang, Jules White, Douglas C Schmidt, and Gunther Lenz. 2017. Applying software patterns to address interoperability in blockchain-based healthcare apps. *arXiv preprint arXiv:1706.03700* (2017).