Ca' Foscari University of Venice
VENICE SCHOOL OF MANAGEMENT

# Working Papers Series

**Marco Corazza, Giovanni Fasano and Raffaele Pesenti**

## Basics on Atomic Swaps in blockchains for financial applications


**Working Paper n. 4/2026**
**February 2026**

**ISSN: 2239-2734**

# Basics on Atomic Swaps in blockchains for financial applications

Marco Corazza[1], Giovanni Fasano[2], Raffaele Pesenti[2]

[1]Department of Economics, University Ca' Foscari, San Giobbe Cannaregio 873, Venice, 30121, Italy.

[2]Venice School of Management, University Ca' Foscari, San Giobbe Cannaregio 873, Venice, 30121, Italy .

Contributing authors: corazza@unive.it; fasano@unive.it; pesenti@unive.it;

**Abstract**

In this paper we investigate both the motivations and the potential use of the Atomic Swap protocols, as defined to handle interoperability among private and public blockchains. We consider a specific perspective, that is suggested by applications in financial markets, where ensuring and possibly forcing both trustability and fairness among actors plays a keynote role.

**Keywords:** Blockchain, Interoperability among Blockchains, Atomic Swap, Smart Contracts

# 1 Basics on the Blockchain

Events and transactions among interacting agents can be recorded in a number of different ways through a suitable database, that may be categorized as *centralized* or *decentralized* one. In particular, rather than keeping a single, centralized database or ledger, blockchains replicate an append–only chain of blocks across many independent nodes that form a peer–to–peer (P2P) network. Each block contains, in addition to the data associated with specific transactions among agents, a *cryptographic hash* of the previous block and a *timestamp*, so that each block actually forms *a ring* of a chain. The data in a given block cannot be altered without correspondingly altering the subsequent blocks. Hence, the structure of the blockchain portion already created is immutable. This ensures the immutability and unchangeability of the data recorded in the created blocks, making the blockchain suitable, for example, for recording a bunch of data to be preserved in the future (*e.g.*, trading of financial transactions along with their timestamp sequence). The nodes where a blockchain can be stored, typically coordinate to validate new data and keep the replicas in sync [1, 2]. Bitcoin and its ledger were released in 2009 by Satoshi Nakamoto, and represents the first working blockchain endowed with a decentralized protocol for minting and transferring a native token, namely *bitcoin* (BTC) [1].

Ethereum is a later, general–purpose blockchain inspired by Bitcoin, but designed to run smart contracts stateful programs, compiled and executed by the Ethereum Virtual Machine (EVM). In the current paper Ethereum is mentioned as a prototype of a blockchain where the so called *smart contracts* may be defined: the last contracts represent the foundation for properly motivating the use of atomic swaps. Solidity represents a high–level programming language designed to implement smart contracts in the EVM [3–5]. Participants interact within Ethereum protocols through accounts that are identified pseudonymously, by addresses derived from public–key cryptography. A specific token, namely *eth*, is adopted in the Ethereum blockchain to finalize the creation of newly added blocks. Furthermore, Ethereum blockchain handles two types of accounts:

- *Externally Owned Accounts* (EOAs), that serve for basic transactions and *eth* management; their ownership is through a private key (typically a seed phrase), and can be created by a software or hardware. They are controlled by private keys and hold a balance;
- *Smart Contract Accounts* (SCAs), that offer more versatile functions and integrations on the Ethereum network w.r.t. EOAs. They are basically governed by the code they deploy with on–chain. Hence, unlike EOAs they also hold code that runs on the EVM [3, 6].

The finalized transactions on the Ethereum blockchain are messages that modify the chain state, *e.g.*, transferring the token *eth* or invoking a contract execution. Read–only calls that merely query the state (*e.g.*, *eth* call) do not create a transaction or spend *eth* [7, 8]. To account for computation and prevent abuse, every EVM operation

consumes the so called *gas*, *i.e.*, a quantity of *eth* that is required to finalize any transaction, whose amount depends on the purpose and nature of the transaction. A transaction also carries a gas limit, so that the user can in principle control the maximum amount of the token required to add the transaction to a block. In case the execution runs out of gas. *i.e.*, the gas limit is outreached, then state changes revert but the gas used up to that point remains paid [8, 9].

As already mentioned, in any blockchain the finalized transactions are packaged into blocks, and the last are appended to the blockchain via a consensus mechanism. To this end Bitcoin selects miners through the Proof–of–Work (PoW) mechanism [1], *i.e.*, claiming for competing special nodes (validators) whose computational efforts on complex and efficient hardware architectures are required. Similarly to Bitcoin, Ethereum originally used PoW but, following the *Merge* update in September 2022, now it uses Proof–of–Stake (PoS) with validators proposing and attesting the blocks [10, 11].

As a distinguishing feature of many blockchains, we have that a number of transactions recorded in blocks involves assets and tokens different from the native token of the corresponding blockchain (*i.e.*, bitcoin, *eth*, etc.). In this regard, in Ethereum blockchain rather than in Bitcoin blockchain, both specific and versatile smart–contracts can be executed on the EVM, to duly consider operations among several tokens. Common standards for these tokens include

- ERC–20: for *fungible tokens*,
- ERC–721: for *non–fungible tokens* (NFTs),
- ERC–1155: for contracts that manage many fungible or non–fungible types under one interface,
- ERC–223: which is to some extent a standard similar to the ERC–20 (unlike ERC–20 the standard ERC–223 defines also the logic for transferring tokens from sender to recipient) [12–15].

A number of different blockchains have been introduced in the literature, apart from Bitcoin and Ethereum, with the special purpose of preserving keynote features of the resulting structure like *scalability*, *costs for finalized transactions*, *security*, *decentralization*, *anonimity of transactions information*, etc.

As mentioned above, among the additional tools that the Ethereum (and several other) blockchain(s) provide to their users we find SCAs, or *smart contracts* [16] for short. A smart contract may be described as a software that encodes the terms of an agreement, and runs on a blockchain. When predefined conditions are satisfied, the program executes the agreed actions – without manual intervention of any operator – such as transferring digital assets, issuing receipts, or updating records. For instance, an escrow smart contract can hold funds and automatically release payment once delivery is confirmed. By shifting execution and verification to the blockchain network, smart

contracts can streamline business workflows and make enforcement of on–chain obligations automatic. We urge to recall, as regards the smart contracts, some relevant issues that need further specifications because of their impact on inter–blockchains operations:

- **Transparency & auditability**: Code and transaction history are typically public on open blockchains, aiding audits but reducing privacy unless privacy techniques are used.

- **Composability**: Contracts can call each other like building blocks, enabling complex products (*e.g.*, DeFi protocols) but also creating dependency risk.

- **Oracles for real–world data**: If a contract needs off–chain facts (*e.g.*, prices, delivery events, etc.), it relies on oracles–trusted bridges whose reliability and incentives matter.

- **Costs & scalability**: Each operation consumes network resources (fees) and may face throughput limits; designs should minimize on–chain computation.

- **Upgrades & governance**: Because code is hard to change, many systems use upgradeable patterns (*e.g.*, proxies) along with on–chain governance (conversely they both add flexibility and new attack surfaces).

- **Legal context**: Smart contracts enforce on–chain outcomes. Their recognition as legal contracts varies by jurisdiction; pairing code with traditional legal terms is commonly sought for clarity.

## 2 Basics on Atomic Swaps

From a historical viewpoint, the early idea of Atomic Swaps may refer to the seminal paper [17], and since the publication of the last reference, the early idea of an atomic protocol has been largely discussed. Nevertheless, the foundations for a more systematic definition of the notion of an atomic transaction was borrowed from the literature in database systems. Namely, the idea of a composite operation that must commit in its entirety or be rolled back, so that no partial effects remain, is at the basis of an Atomic Swap. Hence, this "all–or–nothing" property has long been formalized in the database literature [18, 19].

Before more extensively describing both features and drawbacks of Atomic Swaps, including possible alternatives and integrations, we want to give evidence about the historical background that motivated their introduction: namely, the *Herstatt crisis*. On June 26th, 1974 the Herstatt Bank, a privately owned bank in Cologne (Germany) went bankrupt, so that German regulators forced its liquidation. In that day, a relevant number of US banks had sent payments using Duetsche Marks (Euro currency was introduced only in 2001) to the Herstatt Bank, in exchange for USDs, to be delivered in New York. Unfortunately, German authorities suspended Herstatt Bank activities

at 4.30 pm (Berlin local time) of June 26th, when at New York it was 10.30 am. Hence, the German bank suspended its operations before the counterpart US banks could receive their USD payments. This raised an international issue (known as the *Herstatt Risk*) between private and public financial institutions, which was uniquely related to the *timing* of contracts closing operations, and not to the clauses contained therein. This case claimed for the necessity of an "all–or–nothing" contract in finance, *i.e.*, the importance of an Atomic Swap based on smart contracts became evident. Their introduction was studied in the subsequent years, and both their practical application and effectiveness were indirectly tested during the Lehman Brothers crisis of 2008.

An atomic swap is a specialized atomic transaction whose overall effect is the exchange of assets among two or more parties. In blockchain settings, atomic swaps are studied as a trust–minimizing mechanism for interoperability across ledgers–popularized in early community discussions, and are given a precise formal model as by Herlihy [17, 20]. An Atomic Swap protocol was originally designed to last for a limited time interval $\Delta t$, and to guarantee three basic properties to the parties [20, 21]:

1. *Completion if all parties comply:* if every participant follows the protocol, all agreed transfers occur. Equivalently, if all parties comply with the protocol, then all assets are exchanged among the parties, within a given time limit $\Delta t$;

2. *No loss for complying parties:* if some coalition (or sub–coalition) deviates, no conforming participant loses assets. Similarly, even if some parties deviate from the protocol, no party conforming to the protocol suffers a loss when the time limit $\Delta t$ is outreached;

3. *No incentive to deviate:* there is no advantage or incentive, for any coalition (or sub–coalition), to deviate from the expected behavior as by the agreement in the contract.

## 2.1 The mechanism behind a *basic Atomic Swap*

Note that point 2. in the previous section ensures that the protocol prevents possible losses, but it does not ensure the full enforcement or successful completion of the contract underlined by the Atomic Swap. In the worst–case scenario, a party will not lose its original asset, but the intended overall swap will not be forced, resulting in a missed gain or opportunity (and possibly the waste of time $\Delta t$). In addition, assets may also be temporarily locked up during such failures, leading to inconvenience or additional unexpected losses. In [21] we also find the next definitions, whose relevance consists in suitably embedding the concept of Atomic Swap into a broader literature. We remark that the definition in the last reference may differ w.r.t. other authors. Let us now more accurately address the steps that characterize an Atomic Swap based on a timelock, as introduced in [20] (hereafter *basic Atomic Swap*). The basic Atomic Swap requires the following three basic elements:

- The appointment of a *party* which initiates the procedure designed to complete the swap;

- The communication channels among the $k$ parties involved in the Atomic Swap, so that they may (informally and preliminarily) agree on the details of their deal. This phase is generally finalized using wallets, exchanges, etc., on the blockchains where the parties decide to lock/swap their assets;
- A number of Hashed Timelock Contracts (HTLCs), which are special smart contracts between couples of parties, that require the next two basic elements:

  1. **hashlock** ($h$): it is a hashed (or a cryptographically scrambled version) of a key (*i.e.*, a *secret* $s$) generated by the party that initiated the swap procedure. The secret $s$ is then used by all other parties, involved in the Atomic Swap, to unlock their assets.
  2. **timelock** ($\Delta t$): it indicates the time interval in which the assets will be locked into the HTLCs. In other words, the locked assets will be released by the HTLCs (each created by a party), when at the specific time limit $\Delta t$ some conditions related to $h$ and $s$ will be met. This mechanism ensures that assets cannot remain indefinitely locked within smart contracts, protecting participants from their funds being frozen indefinitely.

Observe that the next assumption is necessary to formally define any Atomic Swap.

**Assumption 1.** *The parties involved in any Atomic Swap share the information contained in the HTLCs they create. Moreover, they agree about using:*

- the same clock *which is adopted to associate time stamps to all the steps of the Atomic Swap;*
- the same Hash function $H()$ *in all the steps of the Atomic Swap;*
- *the sequence of propagation of the information among the parties, i.e., the sequence of parties involved in the Atomic Swap, including the 'initiator' party;*
- *the timelocks generated and used by the parties.*

In Figure 1 we find a scheme summarizing a basic Atomic Swap (see *e.g.*, [20]) among three parties, respectively Alice, Bob and Carol, Alice being the "initiator" party. Of course, w.l.o.g. the number of the parties can be any positive integer larger or equal to two. The numbers in the scheme correspond to actions in a time sequence detailed hereafter. Furthermore, arrows identify the flow of information with the following distinction: continuous arrows identify the first (clockwise) round of the basic Atomic Swap, while dashed arrows identify its second (counter clockwise) round.

1. Alice generates a secret $s_A$ and the corresponding hashlock $h_A = H(s_A)$. Then, she selects a value $t > 0$ and a timelock $t_A = 6t$, so that she can create on her blockchain an HTLC (namely $HTLC_A()$) using $h_A$ and $t_A$. This HTLC will use the keylock $h_A$ and the interval $6t$ as necessary conditions to expire, in the following way. It ensures that if $s_A$ is provided, by Assumption 1 the function $H()$ is known, so that the contract can be unlocked at the time limit $6t$;
2. Alice sends the hashlock $h_A$ to Bob;
3. Bob can verify that $HTLC_A()$ was created by Alice, by using the hashlock $h_A$ that is associated to $HTLC_A()$. However, Bob has not access to the assets in $HTLC_A()$

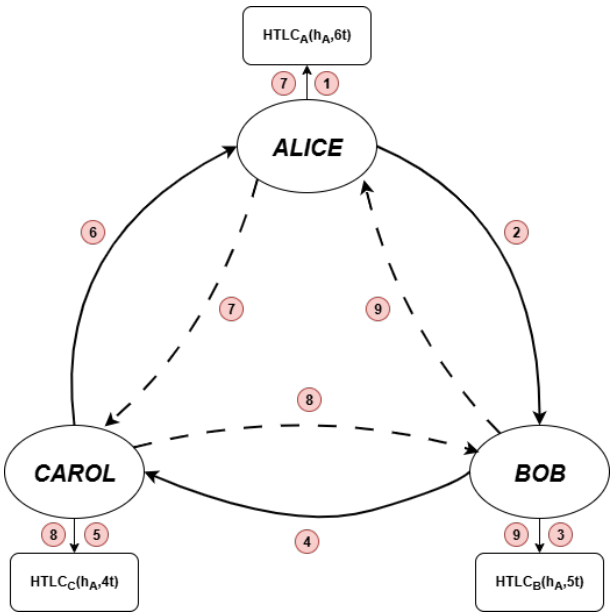


**Fig. 1**: Scheme of a *basic Atomic Swap* [20], relying on HTLCs among three parties, where Alice is the leading party (initiator) of the Atomic Swap.

since he has not got the secret $s_A$. Similarly to Alice, Bob creates his HTLC (namely $HTLC_B()$) on his blockchain, using $h_A$ and $t_B = 5t$. This HTLC will use both $h_A$ and $5t$ as necessary conditions to expire. Hence, if $s_A$ is provided, again by Assumption 1 this contract will be unlocked at the time limit $5t$;

4. Bob propagates the hashlock $h_A$ to Carol;
5. Now it is Carol to verify that both $HTLC_A()$ was created by Alice and $HTLC_B()$ was created by Bob, by using the hashlock $h_A$ that is associated to $HTLC_A()$ and $HTLC_B()$. Then, Carol creates an HTLC (namely $HTLC_C()$) on her blockchain, again using $h_A$ along with $t_C = 4t$, similarly to $HTLC_A()$ and $HTLC_B()$;
6. Carol sends the hashlock $h_A$ to Alice, proving that all the parties have participated in the first round of the Atomic Swap;
7. Alice now performs two distinct operations:
   - she uses $s_A$ to first unlock $HTLC_A()$. Indeed, sending $s_A$ to $HTLC_A()$, by Assumption 1, allows $HTLC_A()$ to experience the necessary unlocking condition $h_A = H(s_A)$. However, $HTLC_A()$ does not allow yet the withdrawal of the assets locked therein, until $t_A$ is outreached;
   - she sends the secret $s_A$ to Carol (starting the counter clockwise round), in order to allow the other parties to unlock their HTLCs;
8. Similarly to the previous step, now Carol performs two distinct operations:

- she uses $s_A$ to first unlock her HTLC, namely $HTLC_C()$, similarly to the unlocking procedure of $HTLC_A()$ by Alice in the previous step;
- she propagates the secret $s_A$ to Bob;

9. Finally, now Bob performs two operations:
   - he uses $s_A$ to unlock $HTLC_B()$, similarly to the unlocking procedures adopted for $HTLC_A()$ (by Alice) and $HTLC_C()$ (by Carol);
   - he propagates the secret $s_A$ to Alice.

Observe that in the end of the procedure 1.–9. above, all $HTLC_A()$, $HTLC_B()$ and $HTLC_C()$ have been unlocked (as regards the hashlock), and only if (necessary condition) the timelock $6t$, starting from step 1., has been outreached, all the parties (*i.e.*, Alice, Bob and Carol) can claim and swap the assets contained in the HTLCs they created on their respective blockchains.

Moreover, in case either of the parties does not claim the assets in due time, *i.e.*, if the assets in an HTLC are not claimed within the corresponding timelock, then the assets cannot be indefinitely trapped in that HTLC, and are automatically returned to the party that issued it.

Beyond basic Atomic Swaps, HTLCs may also enhance escrow services and multi–signature wallets. In particular, their versatility helps secure Decentralized Applications (DApps) in Decentralized Finance (DeFi), enabling more complex conditional payment scenarios safely and transparently (see *e.g.*, [22, 23]).

We also remark on the role played by the initiator in a basic Atomic Swap, since she/he generates the unique secret $s_A$ which is then shared with all the other parties.

## 2.2 Possible drawbacks of the basic Atomic Swap

Among the advantages of using HTLCs (for trustless transactions), we undoubtedly mention that they eliminate the need for trust between transaction participants, through enforcing cryptographic conditions that must be satisfied before any funds are exchanged. Hence, basic Atomic Swaps provide assurance and fairness, ensuring that neither party is disadvantaged or defrauded. As a consequence, the security offered by both cryptographic hashing and timelock, along with the predefined conditions specified in Assumption 1, greatly reduce the possibility of unauthorized access or manipulation with contracts.

Nevertheless, the conditions specified in Assumption 1 are required and may create some limitations of use. In particular, we list below a number of possible criticalities which are strictly related to possible weakness for the fulfillment of the conditions in Assumption 1.

- **Griefing Attacks.** They represent a notable vulnerability of HTLCs and exploit vulnerabilities in smart contracts, usually related to business logic, though without providing no direct profit for the attackers. In a griefing attack malicious parties exploit the timelock they can set, deliberately delaying or blocking transactions by

potentially locking funds temporarily. As an example, in Figure 1 Bob and Carol may have an advantage to abort the second round of the overall Atomic Swap (*e.g.*, if Carol does not propagate $s_A$ to Bob).

- **Scalability Concerns.** As blockchain networks increasingly utilize HTLCs and/or the number of parties increases, scalability may become an issue. Indeed, it is evident by Figure 1 that the larger the concurrent HTLC transactions, the larger the timelock $t_A$ required by the initiator, potentially causing congestion, elevated transaction fees (when introduced) and delay. Observe that successfully addressing scalability often requires significant innovation in blockchains infrastructure.
- **Complexity.** Implementing HTLCs is technically sophisticated, demanding substantial knowledge of cryptography and blockchain programming. This complexity may present barriers to widespread adoption and could lead to potential misuse or misconfiguration, highlighting the need for user–friendly interfaces and educational resources.
- **Disagreement.** Not all the parties may share the same (say *correct*) information about a unique hash function $H()$ and/or the time elementary interval $t$.

## 2.3 Resilient Atomic Swap: a first proposal of alternative to the basic Atomic Swap

One possible alternative to the basic Atomic Swap in Figure 1, in order to possibly attenuate the risks of griefing attacks, is given in Figure 2. Basically, a couple of rounds are performed in order to securely share information among parties, and the main differences between the schemes in Figure 1 and Figure 2 can reduce to the following (see also the blue larger numbered bullets in Figure 2):

- Each of the parties creates a *couple* of HTLCs when the hashlock $h_A$ is known (*i.e.*, first Alice creates two HTLCs, then does Bob and finally Carol), where the second HTLC (namely $HTLC_{A2}(c_A, h_A, 6t)$, $HTLC_{B2}(c_B, h_A, 5t)$, $HTLC_{C2}(c_C, h_A, 4t)$, respectively) contains the same parameters of the corresponding HTLC in Figure 1, but additionally it needs the positive quantities $c_A$, $c_B$, $c_C$. They can be associated to *fictitious costs* that each party is committed to lock, in the corresponding second HTLC, in order to convince the other parties about resilience of the Atomic Swap on griefing effects. In other words, any of the parties will have to lock an amount $c_A$, $c_B$, $c_C$, that can be redeemed only if (necessary condition) the Swap becomes Atomic, *i.e.*, only if all the parties have swapped their assets in due time.
- The *Dummy Party*[1] represents a counterpart w.r.t. all the other parties, and has the unique role of possibly supervising the correct creation of $HTLC_{A2}(c_A, h_A, 6t)$, $HTLC_{B2}(c_B, h_A, 5t)$ and $HTLC_{C2}(c_C, h_A, 4t)$. This explains why Alice at step 2. of the procedure has got to send the hashlock $h_A$ also to the Dummy Party. To possibly reward the Dummy Party, when the Swap becomes Atomic, then the parties are able to redeem the quantities $\gamma_A c_A$, $\gamma_B c_B$, $\gamma_C c_C$, with $\gamma_A, \gamma_B, \gamma_C \in [0, 1]$, while the

[1] We remark that in principle the Dummy Party can coincide with any of the other parties already involved in the Atomic Swap. Conversely, the Dummy Party may even be unaware of any of the other parties involved in the Atomic Swap, apart from the initiator (*i.e.*, Alice in Figure 2).

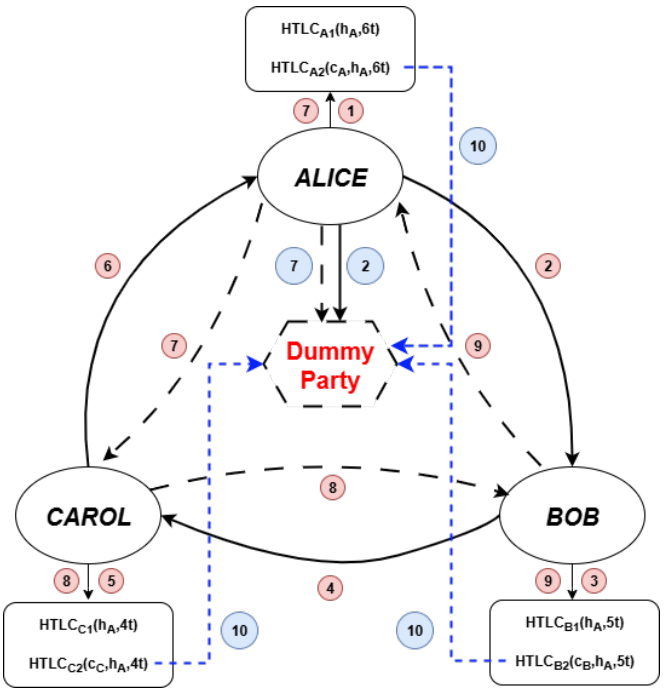


**Fig. 2**: Scheme of a Resilient Atomic Swap, an alternative to the *basic Atomic Swap*, relying on HTLCs among three parties, where griefing attacks may be possibly prevented.

amounts $(1-\gamma_A)c_A$, $(1-\gamma_B)c_B$, $(1-\gamma_C)c_C$ can be kept by the Dummy Party (see the dashed blue arrows in Figure 2 identified by the step 10.).

- The Dummy Party is formally unable to unlock $HTLC_{A2}(c_A, h_A, 6t)$, $HTLC_{B2}(c_B, h_A, 5t)$, $HTLC_{C2}(c_C, h_A, 4t)$ and withdraw the corresponding amounts $(1-\gamma_A)c_A$, $(1-\gamma_B)c_B$ and $(1-\gamma_C)c_C$, before knowing also the secret $s_A$. This last piece of information is indeed communicated to the Dummy Party by Alice only at step 7. in Figure 2 (see the dashed arrow identified by the step 7.), when all the parties have already agreed to join the Atomic Swap. In any case, the knowledge of $s_A$ by the Dummy Party is only a becessary condition to unlock the created HTLCs, since in any case the time interval $t_A = 6t$ represents the second necessary condition.

We remark that since committing the amounts $c_A$, $c_B$, $c_C$ for a longer time equivalently may imply additional costs, we may assume that $\gamma_A = \gamma_A(t)$, $\gamma_B = \gamma_B(t)$ and $\gamma_C = \gamma_C(t)$. This suggests that in case some parties are simultaneously involved in different Atomic Swaps, then a multilevel optimization problem may arise in order to suggest preferable/optimal values for the parameters $t$, $\gamma_A(t)$, $\gamma_B(t)$ and $\gamma_C(t)$.

# 3 Alternative formulations for Atomic Swaps

Some alternatives perspectives when describing Atomic Swaps have been proposed in the literature. For instance, in [24] the authors prefer to first define an *atomic transaction*, as originally specified in the context of Database Management System (DBMS), to identify "*a single and indivisible macro–operation, consisting of multiple smaller operations of different types such that all, or none of them, are performed*". Then, in [24] the authors assimilate the Atomic Swap to "*a particular case of an atomic transaction, where the whole operation involves the exchange of one or more assets between multiple actors*". Though the last definition appears surely consistent with the remaining literature, we prefer to follow the taxonomy in [21], that in our opinion helps these authors to follow a more rigorous approach in the current paper, and requires some basic notions on graphs.

Formally, an Atomic Swap can be modeled as a *directed graph* $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of vertices and each vertex represents an actor in the swap, while $\mathcal{A}$ is the set of the directed arcs. Then, if $(u, v) \in \mathcal{A}$, with $u, v \in \mathcal{V}$, the arc $(u, v)$ represents a transfer from the actor $u$ to the actor $v$ (the weight of the arc being the amount to transfer). In particular, we identify in an Atomic Swap the so called *leader set* $\mathcal{L} \subseteq \mathcal{V}$, as a subset of nodes in $\mathcal{V}$ such that removing any vertex from $\mathcal{L}$ leaves $\mathcal{D}$ *acyclic*. The nodes in $\mathcal{L}$ are in charge for initializing the Atomic Swap protocol, *i.e.*, for creating the proper smart contracts. Herlihy et al. [20] prove that for any pair $(\mathcal{D}, \mathcal{L})$, an atomic protocol exists if $\mathcal{D}$ is a strongly–connected directed graph,

An atomic cross-chain swap is a procedure involving assets located on at least two blockchains B1 and B2, while an atomic intra-chain swap is an atomic swap involving assets located on the same blockchain B1.

One canonical construction for the mechanims of an Atomic Swap uses Hashed Time-Lock Contracts (HTLCs). HTLCs represent a class of payments that need to use both *hashlocks* and *timelocks* to be finalized. Broadly speaking, in an HTLC the receiver of a payment (or possibly a supply) must generate a cryptographic proof (the hashlock) to unlock the payment, prior to a deadline (the timelock), acknowledging that he/she is indeed the correct recipient of the payment. In case the deadline is not respected, then the payment can no more be claimed and the amount of the payment (or the amount of the supply) will be automatically returned to the payer.

In case an Atomic Swap were modeled as a directed graph $\mathcal{D}$, the time to complete an Atomic Swap is proportional to the diameter of $\mathcal{D}$, *i.e.*, the resulting protocol has time complexity which is indeed proportional to the graph diameter. On the other hand, the length of the sequence of bits published on the blockchains by the Atomic Swap depends on a couple of parameters: (1) the number/amount of the assets to be transferred, (2) the number of the leaders in the set $\mathcal{L}$ [20]. We may classify Atomic Swaps according with the following categories:

- *Cross–chain case*. In this case, suppose the nodes $u, v \in \mathcal{V}$ want to finalize the swap associated with the arc $(u, v) \in \mathcal{A}$, and they intend to swap the assets $a_u$ and $a_v$, respectively. The leader $u \in \mathcal{V}$ (or a sub–coalition) creates a smart contract [16].

The smart contract we refer to, when dealing with an Atomic Swap, requires at least a couple of parameters:

- a timelock $t_u$, indicating the maximum time for which the asset $a_u$ will be locked and available for the counterpart $v$;
- the hashlock $h_u$ associated to a random secret $s_u$, so that $h_u = H(s_u)$, being $H()$ an hash function.

The initiator (*i.e.*, the node $u$) escrows the asset $a_u$ in an HTLC locked smart constract, with deadline $t_u$. The counterparty $v$ similarly escrows $a_v$ *under the same hashlock* $h$, but with a strictly earlier deadline $t_v$ w.r.t. $t_u$. Revealing $s_u$ to the counterpart $v$ before $t_v$ allows the initiator to claim the asset $a_v$, and the revelation lets the counterparty learn $s_u$, so that $v$ will be able to claim for $a_u$, before the deadline $t_u$. In case any of these deadlines $t_u$ or $t_v$ are missed, then refund paths return funds $a_u$ and $a_v$ after the respective timeouts. This decreasing–deadline pattern is standard in practice [20].

- *Intra–chain case.* When both assets live on the same blockchain, a single time–bounded escrow contract can enforce the swap atomically without a hashlock: the contract holds both deposits, executes the exchange via a single transaction's state transition (claim), or refunds both after a deadline if conditions are unmet (refund). This design relies on the fact that smart–contract calls on a single chain execute atomically (again, the "all or nothing" strategy is pursued) [25].

We are now ready to propose the taxonomy on Atomic Swaps in [21], that gives a perspective based on Nash Equilibrium. We are persuaded that in a future work it can be a starting point, to further investigate the relationship between Atomic Swaps in blockchains and swaps in quantitative finance.

**Definition 1.**

1. *A swap protocol $P$ is* uniform *if it guarantees the followings:*
   - *If all parties follow $P$, all arcs in $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ are triggered.*
   - *Even if there are parties arbitrarily deviating from $P$, every party $v$ following $P$ gets all assets of entering arcs to $v$ or regains all assets of leaving arcs from $v$.*
2. *A swap protocol $P$ is* Nash equilibrium *if no party can get more benefits by deviating from $P$ than by following $P$, unless he/she joins a coalition.*
3. *A swap protocol $P$ is* strong Nash equilibrium *if it guarantees that no party and no coalition can get more benefits by deviating from $P$ than by following $P$.*
4. *A swap protocol $P$ is* atomic *if it guarantees to be uniform and strong Nash equilibrium.*

**Acknowledgnments.** This paper was developed within the projects funded by: Next Generation EU - "Just Energy Transition - JET: Stochastic and machine learning methods for the evaluation, mitigation and geographical hedging of involved natural risks (with climate in view)", National Recovery and Resilience Plan (NRRP), Mission 4, C2, Intervention 1.1 - Notice: Progetti di Rilevante Interesse Nazionale (PRIN) 2022, DD No. 104 dated 2022/02/02/, Prot. P2022XTLM2, CUP H53D23008460001.

G. Fasano thanks the National Research Council–Marine Technology Research Institute (CNR–INSEAN), Italy. G. Fasano also thanks the working group GNCS of IN$\delta$AM (Istituto Nazionale di Alta Matematica), Italy, for the support he received.

# References

[1] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008). Accessed 2025-08-29

[2] Technical Intro to Ethereum. https://ethereum.org/en/developers/docs/intro-to-ethereum/. Accessed 2025-08-29 (2025)

[3] Wood, G.: Ethereum: A Secure Decentralised Generalised Transaction Ledger. https://ethereum.org/content/developers/tutorials/yellow-paper-evm/yellow-paper-berlin.pdf. Berlin version PDF; Accessed 2025-08-29 (2014)

[4] Ethereum Virtual Machine (EVM). https://ethereum.org/en/developers/docs/evm/. Accessed 2025-08-29 (2025)

[5] Solidity Documentation. https://docs.soliditylang.org/. Accessed 2025-08-29 (2025)

[6] Ethereum Accounts. https://ethereum.org/en/developers/docs/accounts/. Accessed 2025-08-29 (2025)

[7] Ethereum JSON-RPC API. https://ethereum.org/en/developers/docs/apis/json-rpc/. See `eth_call`; Accessed 2025-08-29 (2025)

[8] Transactions on Ethereum. https://ethereum.org/en/developers/docs/transactions/. Accessed 2025-08-29 (2025)

[9] Buterin, V., Conner, E., Dudley, R., Slipper, M., Norden, I., Bakhta, A.: EIP-1559: Fee Market Change for ETH 1.0 Chain. https://eips.ethereum.org/EIPS/eip-1559. Accessed 2025-08-29 (2019)

[10] Kalinin, M., Ryan, D., Buterin, V.: EIP-3675: Upgrade Consensus to Proof-of-Stake. https://eips.ethereum.org/EIPS/eip-3675. Accessed 2025-08-29 (2021)

[11] Proof-of-Stake (PoS) on Ethereum. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/. Accessed 2025-08-29 (2024)

[12] Vogelsteller, F., Buterin, V.: ERC-20: Token Standard. https://eips.ethereum.org/EIPS/eip-20. Accessed 2025-08-29 (2015)

[13] Entriken, W., Shirley, D., Evans, J., Sachs, N.: ERC-721: Non-Fungible Token Standard. https://eips.ethereum.org/EIPS/eip-721. Accessed 2025-08-29 (2018)

[14] Radomski, W., Cooke, A., Castonguay, P., Therien, J., Binet, E., Sandford, R.: ERC-1155: Multi Token Standard. https://eips.ethereum.org/EIPS/eip-1155. Accessed 2025-08-29 (2018)

[15] Dexaran: ERC-223: Token with Transaction Handling Model. https://eips.ethereum.org/EIPS/eip-223. Draft proposal; Accessed 2025-08-29 (2017)

[16] What is a smart contract? https://blockchain-observatory.ec.europa.eu/what-smart-contract_en. Accessed 2025-09-05 (2025)

[17] TierNolan: Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.0. BitcoinTalk forum post, May 2013 (2013)

[18] Haerder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Computing Surveys **15**(4), 287–317 (1983) https://doi.org/10.1145/289.291

[19] Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, CA (1993)

[20] Herlihy, M.: Atomic cross-chain swaps. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC '18) (2018). https://doi.org/10.1145/3212734.3212736 . https://arxiv.org/abs/1801.09515

[21] Soichiro, I., Yuichi, S., Hirotsugu, K., Toshimitsu, M.: Atomic cross-chain swaps with improved space, time and local time complexities. Information and Computation **292**(105039), 1–11 (2023) https://doi.org/10.1016/j.ic.2023.105039

[22] Westerkamp, M., Küpper, A.: Instant function calls using synchronized cross-blockchain smart contracts. IEEE Transactions on Network and Service Management **20**(3), 2136–2150 (2023) https://doi.org/10.1109/TNSM.2023.3236437

[23] Yuechen, T., Bo, L., Baochun, L.: On atomicity and confidentiality across blockchains under failures. IEEE Transactions on Knowledge and Data Engineering **36**(2), 766–780 (2024) https://doi.org/10.1109/TKDE.2023.3255842

[24] Lisi, A., De Salve, A., Mori, P., Ricci, L.: Practical application and evaluation of atomic swaps for blockchain-based recommender systems. In: Proceedings of

the 2020 3rd International Conference on Blockchain Technology and Applications. ICBTA '20, pp. 67–74. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3446983.3446993 . https://doi.org/10.1145/3446983.3446993

[25] An Analysis of Atomic Swaps on and between Ethereum Blockchains using Smart Contracts. https://rp.os3.nl/2017-2018/p42/report.pdf. Accessed 2025-09-03 (2018)