Security difficulties and barriers in building decentralized blockchain bridges, solution methods*

Andrii Bondarchuk[1,†], Viktoriia Onyshchenko[2,†], Andrii Hashko[3,†], Andrii Strazhnikov[3,†], and Nataliia Korshun[1,*,†]

[1] Borys Grinchenko Kyiv Metropolitan University, 18/2 Bulvarno-Kudriavska str., 04053 Kyiv, Ukraine
[2] University of Warmia and Mazury in Olsztyn, 2 M. Oczapowskiego str., 10-719 Olsztyn, Poland
[3] State University of Information and Communication Technologies, 7 Solomyanska str., 03110 Kyiv, Ukraine

Abstract

The use of distributed ledger technologies (DLTs) and blockchains is rapidly expanding across multiple sectors, including finance and governance. Although numerous blockchain frameworks and networks exist to support different use cases, a major challenge remains: enabling seamless communication between these diverse frameworks, protocols, and ledgers. As blockchain adoption accelerates, the need for effective interoperability solutions is becoming increasingly critical to deliver greater value to users. This study explores the design of current blockchain bridges and evaluates common security threats and mitigation strategies within the realm of interoperability. A threat model is introduced to examine the key components, vulnerabilities, risks, and corresponding safeguards involved in blockchain interoperability. Security concerns—such as excessive trust centralization and flaws in smart contracts—are identified and categorized based on the type of bridge component, along with recommended countermeasures. Different interoperability approaches—including relays, Hash Time-Locked Contracts (HTLCs), notary schemes, and smart contract-based solutions—are analyzed in detail. Ultimately, this research aims to help developers better understand the security challenges in blockchain interoperability and highlights the importance of establishing standardized practices in this area.

Keywords

smart contracts, decentralized ledger system, cross-network communication, security, data forwarders, attestation methods, blockchain, solidity

1. Introduction

The decentralized wallet system is a digital ledger framework that is both distributed and decentralized, allowing multiple parties to share and update a common database without depending on any central authority. Initially developed as the backbone for cryptocurrencies like Bitcoin, this technology has since expanded its reach into industries such as finance, healthcare, supply chain management, voting systems, and identity verification [1].

As decentralized wallet systems gain widespread adoption, challenges and risks continue to emerge—particularly regarding interoperability and system integration. Interoperability refers to the ability of decentralized wallet systems or blockchains to communicate, exchange information, and interact either with each other or with external systems. In simpler terms, it enables one ledger to trigger actions or respond to events occurring on another ledger, a process known as interoperation. To achieve interoperability, several approaches are commonly used, including Bridges, Direct Connections, Connectors, and others. Each approach serves different technical requirements and use cases, but all share the goal of enabling seamless communication while preserving the decentralized nature of the systems [2–4].

* CPITS-II 2025: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, October 26, 2025, Kyiv, Ukraine
* Corresponding author.
† These authors contributed equally.
a.bondarchuk@kubg.edu.ua (A. Bondarchuk); viktoriia.onyshchenko@uwm.edu.pl (V. Onyshchenko); a.gashko1111@gmail.com (A. Hashko); andrew.strazh@gmail.com (A. Strazhnikov); n.korshun@kubg.edu.ua (N. Korshun)
0000-0001-5124-5102 (A. Bondarchuk); 0000-0002-3126-2260 (V. Onyshchenko); 0009-0000-4103-8425 (A. Hashko); 0009-0000-3492-2968 (A. Strazhnikov); 0000-0003-2908-970X (N. Korshun)

On a more detailed level, mechanisms such as asset locking, burning, and wrapping are often applied to improve interoperability. Asset locking involves placing assets in an unusable state on the source ledger so they can be recreated on another network. Wrapping replicates the asset on the destination ledger with the same theoretical value as the original. Burning, meanwhile, destroys assets on the source ledger to allow identical assets to be issued on another network [5, 6].

Despite these solutions, achieving complete interoperability is still difficult. Key concerns include ensuring atomicity in cross-ledger transactions, maintaining decentralization in communication channels like bridges, and addressing vulnerabilities in smart contracts [7].

Although numerous Blockchain frameworks have been developed, no single network can satisfy the diverse requirements of all applications. This has created a pressing need for mechanisms that enable different Blockchains to communicate and share data effectively. However, the fundamental design of Blockchain technology makes seamless interaction between networks inherently challenging [8, 9].

Even with existing solutions such as bridges and other integration mechanisms, cross-Blockchain data exchange often depends heavily on the reliability and security of these systems. With the total market capitalization of Blockchain assets surpassing $3 trillion as of August 2022—and growing demand for asset transfers across multiple chains—it has become essential to identify the common security risks linked to Blockchain bridges and to design robust countermeasures.

Despite the availability of various frameworks and protocols, knowledge about achieving effective interoperability remains scattered. The severity of security threats was made evident by the high-profile attack on the Wormhole Blockchain bridge, which enables asset transfers between the Ethereum and Solana networks. Exploiting a vulnerability in the bridge contract, hackers stole assets worth $690 million, demonstrating the significant risks posed by bridge-based interoperability [3].

As the phrase goes, “Interoperability is key to survivability.” Therefore, it is vital that interoperability solutions—particularly bridges—adhere to stringent security standards to safeguard assets and preserve user trust.

This study focuses specifically on interoperability between two Blockchains, assessing existing interoperability frameworks to uncover security vulnerabilities and risks.

2. Research Methodology

A threat model analysis is employed to evaluate system complexity, pinpoint potential attack vectors, and prioritize threats based on their severity. A six-step case flow analysis is then used to rank threats and propose appropriate mitigation strategies.

3. The goals of this research

- provide a clear review of current Blockchain interoperability needs and associated risks.
- perform threat analysis and evaluate the security risks in Blockchain bridge design patterns.
- recommend countermeasures to mitigate identified security risks.
- examine the limitations and potential challenges of the proposed methods.

4. Interoperability and Cross-Chain Communication

Interoperability refers to the ability of different software systems to exchange information and use it effectively. Vernadat described interoperability as the capability of two or more systems to provide or accept services from one another and accurately share data. In the context of distributed ledger systems, interoperability can also be defined as the ability of one ledger to process transactions originating from another, even if they use different identity management systems, cryptographic methods, consensus mechanisms, or smart contract functionalities.

Cross-chain communication, on the other hand, is the process through which two independent blockchains interact to maintain synchronized and consistent data. Essentially, it allows different types of blockchains to communicate at the interchain level. One example is the Interledger Protocol, which enables value transfer between two separate chains. However, designing a cross-chain communication protocol is challenging because each blockchain has unique structural and operational characteristics [10].

One of the most significant challenges, known as the cross-blockchain proof problem, arises when one blockchain needs to verify the state or transaction records of another chain. It is often difficult—if not impossible—to confirm data recorded on one chain simply by observing information shared from another. This means the receiving chain cannot always fully verify the intended changes or authenticate the transaction. Consequently, blockchains cannot communicate with each other directly [11].

To overcome this limitation, trusted third-party solutions have been developed to facilitate data transfers between blockchains. However, relying on third parties contradicts one of blockchain technology's core principles—decentralization—as such solutions can introduce single points of failure. Therefore, interoperability frameworks must ensure that information and value are exchanged in a secure and trustless manner.

Researchers have observed that cross-blockchain transactions often do not directly alter the state of the other chain. Instead, they trigger specific functions on the target chain, which may subsequently change its state. Interoperable solutions can be categorized based on integration type, consensus mechanism, level of decentralization, or design rationale.

Buterin's classification of cross-chain solutions falls into three main categories:

1. Centralized or Multi-signature Attestation Methods—A group of trusted parties verifies and acts on the destination chain when triggered by the source chain.
2. Data Forwarders—Systems where one blockchain can read and interpret data from another.
3. Hash-locking Method—Two blockchains interact using a cryptographic hash trigger to verify and execute transactions.

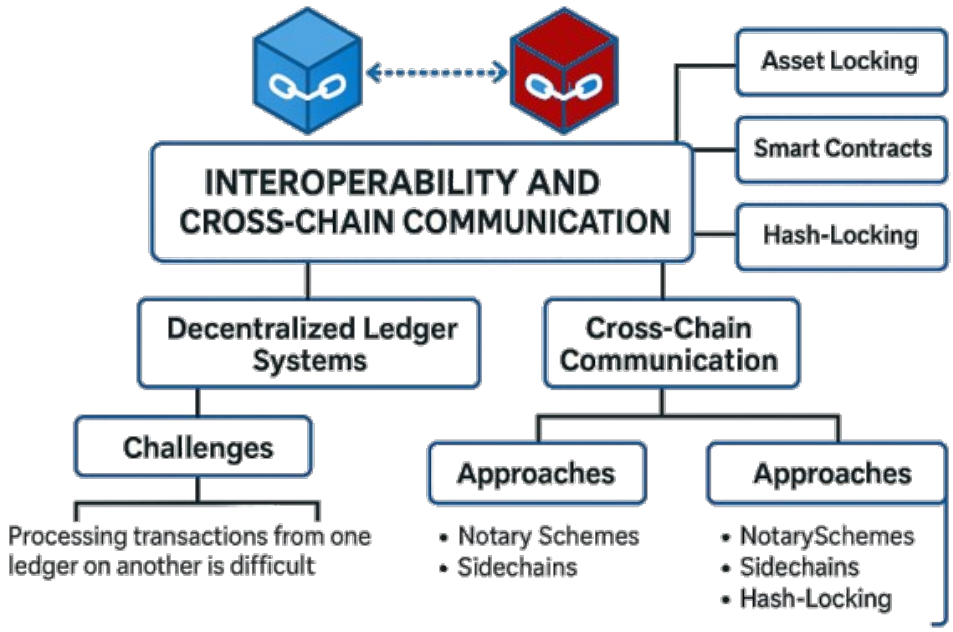

Figure 1: Model of the bridge solution

Further studies also discuss concepts like Blockchain of Blockchains (BoB) and Hybrid Connectors, along with variations of sidechains, such as centralized two-way pegs, federated two-way pegs, and simplified payment verification proofs. Bridge solutions may combine multiple mechanisms to achieve seamless cross-chain communication.

Let's mathematically describe the method of interaction between Decentralized Ledger Systems (DLS). I will formulate this as an interaction model suitable for cross-chain communication [9].

Let's say we have a set of decentralized ledgers (blockchains):

$$L = \{L_1, L_2, \dots, L_n,\}$$

where L_i is the ith decentralized registry.

Each register L_i is defined as:

$$L_i = (S_i, T_i, C_i)$$

where: S is a set of states, T is a set of transactions, C is a consensus mechanism.

Interaction between two registries L_i and L_j can be presented as a reflection:

$$f_{ij}: T_i \longrightarrow T_j$$

where f_{ij} is a function that transforms a transaction from one register to the format of another.

Correctness Condition

For a transaction from L_i to be accepted in L_j, the following must hold:

$$C_j(f_{ij}(t)) = True, \forall \quad t \in T_i$$

Where C_j is the consensus mechanism of ledger L_j, which validates the transaction after transformation.

Cross-Chain Communication Model

To represent data exchange, we can define the operation

$$L_i \oplus L_j = \{f_{ij}(t) / t \in T_i, C_j(f_{ij}(t)) = True\}$$

Hash-Time Lock Contracts (HTLC)

For security, a hash function H and a timelock τ are added:

$$HTLC(t) = \{f_{ij}(t), if\ H(s) = h \wedge t < \tau, \varnothing \quad otherwise\}$$

This ensures that the transaction will only be executed if the secret sss is revealed (such that $H(s) = h$) and within the allowed time window.

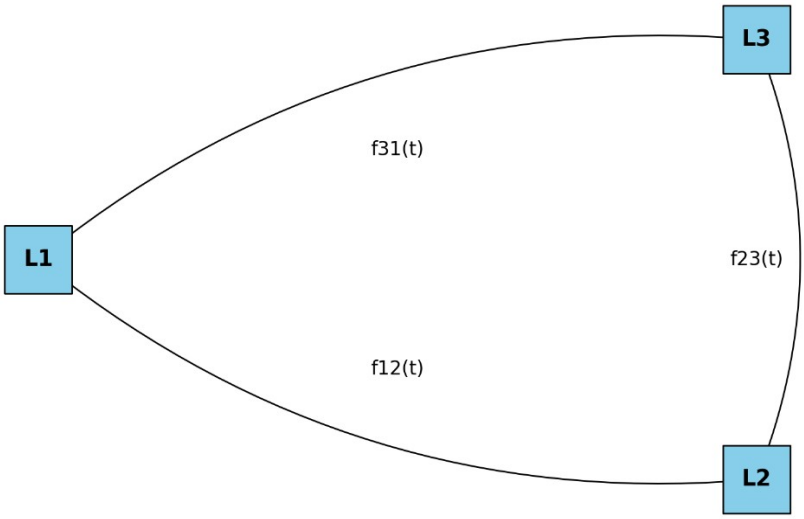

Figure 2: Interaction between Decentralized Ledger Systems (DLS)

Here is a mathematical model of the interaction between Decentralized Ledger Systems (DLS).
Mathematically, it can be described as follows:
Let L_i is a decentralized ledger.
$f_{ij}(t)$ is the transformation function of transactions between $L_i$and L_j, at time t.
For correct interaction, the following is performed:

$$L_j(t+1) = f_{ij}(L_i(t))$$

which means that the state of book L_j is updated based on events in book L_i.
Let's add mathematical equations and formalism for the security and atomicity of cross-chain transactions between Decentralized Ledger Systems (DLS) [12].
Condition for correctness of transaction acceptance:
Transaction T_i with L_i after transformation must be validated in L_j:

$$C_j(f_{ij}(T_i)) = True.$$

That is, only if the consensus L recognizes the converted transaction as valid will its state be applied.
Proof verification (Merkle/Proof verification) when L_j receives proof p about the existence of a transaction/state in L_i:
$VERIFI\ j(p\ ROOT\ i) = True$ is a valid proof (Merkle/POW/POA) for $ROOT\ i$.
Without correct $VERIFI\ j - L_j$should not be equal to L_i:
Atomicity: formalization (general condition):
Let's call the commit/rollback boolean variables:

$$COMMIT_i, COMMIT_j \in \{0,1\}.$$

The atomicity of a cross-chain operation requires:

$$COMMIT_i \Longleftrightarrow COMMIT_j.$$

That is means, the operation must either be performed on both books or on neither.

HTLC (Hash Time-Lock Contract)—formalism for atomicity

HTLC provides conditional commit via the secret sss and timeout τ\tauτ.

Formally:

$$HTLC_i(T_i; h, \tau) = \begin{cases} execute\ T_i, if\ s\ is\ presented \wedge\ H(s) = h\ before\ \tau \\ refund\ T_i, if\ \tau\ has\ passed \wedge\ s\ was\ not\ provided. \end{cases}$$

HTLC atomicity for a paired swap $L_i \leftrightarrow L_j$:

1. $L_i$locks asset A under h until τi
2. L_j locks asset B under the same h until τj (with $\tau j < \tau i$ for safety).
3. If a party reveals $s$on L_j before τj , then L_i can verify s and redeem B. Otherwise, after τ a refund occurs.

Formally, atomicity:

$$(H(s) = h \wedge\ t < \tau j) \Longrightarrow (COMMIT_i ¿ C\ OMMIT_j = 1).$$

Otherwise, if $t \geq \tau j$, both refund:

$$t \geq \tau j \Longrightarrow COMMIT_i ¿\ COMMIT_j = 0.$$

Notaries (multi-sig / validator threshold)—trust-minimized formalism

Validator/notary model: a set V with weights w_v. Threshold Θ.

Acceptance condition for a message/attestation:

$$\sum_{v \in V_{approve}} w_v \geq \Theta \longrightarrow accept.$$

This formalizes: if there are enough signatures/votes, the action is executed.

Security requires that an attacker cannot gather the threshold:

$$Pr\left(\sum_{v \in V_{malicious}} w_v \geq \Theta\right) is\ small.$$

Protection against reorgs and block confirmations

For blockchains with probabilistic finality (e.g., PoW), the required number of confirmations k is:

$$k \geq \left\lceil \frac{\ln(\epsilon)}{\ln(1 - p)} \right\rceil$$

where p is the probability that a single block is reorganized (roughly dependent on attacker share), and ϵ is the target risk level (e.g.,10^{-9}). In practice, k is set based on a risk assessment, after which a transaction is considered final once k blocks have confirmed it.

Safety and liveness conditions

Safety: under the assumed conditions (e.g., <50% adversarial validator weight or sufficient confirmations), no interaction leads to an invalid state:

$$t : C_j(f_{ij}(T_i)) = True \wedge \text{ (the transaction was fraudulent).}$$

Liveness: if participants follow the protocol, the operation completes (commit or refund) within a finite time Tmax.

Composite model—summary equation

The full condition for a correct and atomic cross-chain operation between L_i and L_j:

$$VERIFY_j(p_i, ROOT_i) = True \wedge C_j(f_{ij}(T_i)) = True$$

$\wedge$ (an atomicity mechanism is specified, e.g., HTLC or threshold notary)
$\wedge$ (sufficient confirmations k are ensured for L_i and L_j)

$$\Rightarrow COMMIT_i \Leftrightarrow COMMIT_j$$

Recommended practical parameters (as formulas):

- HTLC timeouts: $\tau j < \tau i$ and $\tau i - \tau j \geq \Delta$, where Δ is a safety margin to complete refund paths.
- Threshold schemes: choose Θ such that $\Theta >$ maximum plausible adversarial weight.
- Confirmations: select k considering the attack model and desired ϵ.

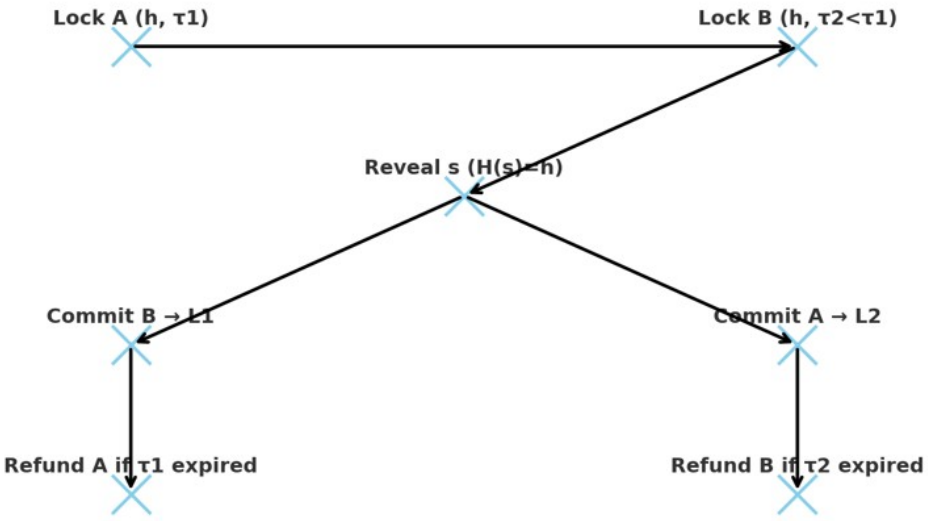

Figure 3: diagram of the atomic interaction

This is the first diagram showing the atomic interaction between two Decentralized Ledger Systems (DLS) using Hash Time-Lock Contracts (HTLC) [11]:

- Ledger $L1$ locks asset A under hash h until timeout τ_1.
- Ledger $L2$ locks asset B under the same hash h, but with a shorter timeout $\tau_2 < \tau_1$.
- To complete the exchange, one participant reveals the secret s such that $H(s) = h$.

- This allows the other ledger to verify the transaction and finalize the exchange.
- If the timeout expires before the secret is revealed, a refund (return of assets) occurs.

Thus:

$$COMMIT_i \Longleftrightarrow COMMIT_j$$

meaning that the transaction is executed on both chains simultaneously, or not executed at all.

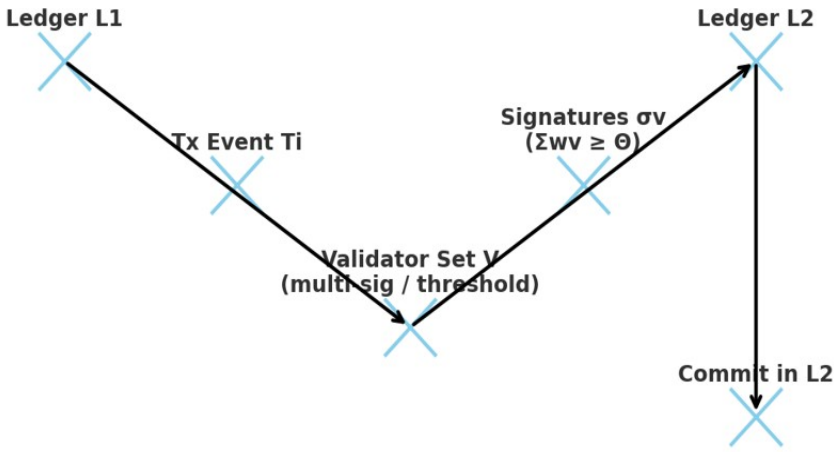

Figure 4: diagram of the notary model

This is the second diagram—it illustrates the notary model of interaction (multi-sig / threshold) between two Decentralized Ledger Systems (DLS):

- A transaction event $T_i$ppears in Ledger $L1$.
- This event is verified by a set of notaries/validators V. Each notary provides a signature σv.
- If the threshold condition is met:

$$\sum_{v \in V_{approve}} w_v \geq \Theta ,$$

then the transaction is considered confirmed [13].

- After that, a commit occurs in Ledger $L2$

Thus, without a sufficient number of honest validators, the transaction cannot proceed, which ensures secure atomicity in the notary model.

Look at the comparative diagram of the two approaches:

HTLC (Hash Time-Lock Contracts) are trustless mechanisms where assets are locked on both ledgers with a common hash h and timeouts (τ_1, τ_2), one participant reveals a secret s such that $H(s)= h$to trigger symmetric execution on both ledgers, and if the secret is not revealed within the time limits, assets are refunded—offering the advantage of no third-party reliance but the disadvantage of practical complexity due to timeouts and risks of delays.

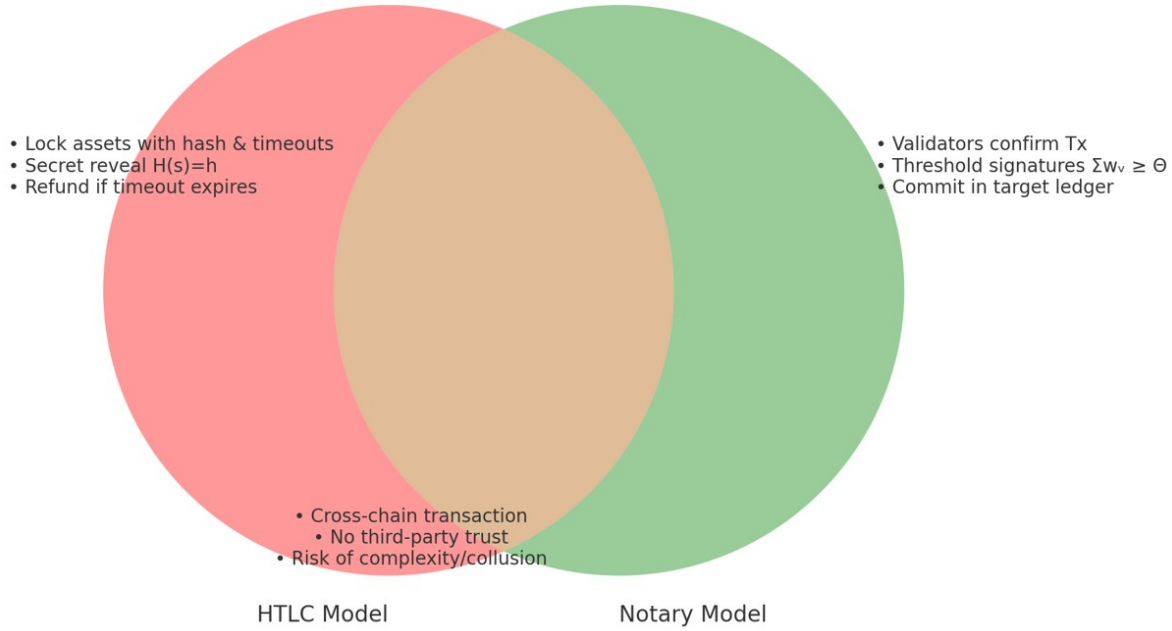

Figure 5: Comparative diagram

In the Notary Model (Multi-sig / Threshold Validators), a transaction event on Ledger $L1$ is verified by a group of notaries. If the number of signatures or votes exceeds the threshold Θ, the transaction is confirmed and a corresponding record (commit) is created in Ledger $L2$. This model offers simpler implementation but carries the drawback of centralization of trust, with potential risks of collusion among notaries [14].

Comparison Table: HTLC vs Notary Model

Criterion	HTLC (Hash Time-Lock Contracts)	Notary Model (Multi-sig / Threshold Validators)
Trust Level	Trustless, fully cryptographically secured	Semi-centralized—relies on trusted notaries
Security	High (based on hash functions and time locks)	Depends on the honesty of the majority of notaries
Atomicity	Guaranteed mathematically: either execution on both ledgers or none	Guaranteed if threshold of signatures Θ is reached
Execution Speed	Slower due to timeouts and block confirmations	Faster, depends on collecting notary signatures
Implementation Complexity	High: requires smart contracts, time locks, coordination	Lower: only signature collection and verification needed
Flexibility	Works with any blockchain that supports smart contracts	Works where reliable validators/notaries are available
Drawbacks	Risk of delays, complex setup	Centralization of trust, risk of collusion among notaries
Use Cases	Atomic swaps (e.g., BTC ↔ ETH)	Cross-chain bridges with validators (e.g., Cosmos, Polkadot)

So we can conclude that HTLC is best suited for scenarios where decentralization and trustlessness are critical but Notary Model is more practical when speed and simplicity are preferred, even if partial trust centralization is introduced [11].

Decentralized Bridge Smart Contracts

Bridge smart contracts are pieces of code that run on blockchains to enable communication between different chains. They handle tasks like locking, burning, and unlocking tokens so that assets can be transferred securely from one chain to another.

Because they control important state changes in the system, vulnerabilities in their code can be dangerous. One example is a reentrancy vulnerability. In this case, an attacker exploits the time gap during a state change to repeatedly call the contract before the final state is settled, allowing them to manipulate the process.

Bridge contracts often lock tokens on one chain while creating or unlocking them on another. But if the contract responsible for unlocking or redeeming the tokens is destroyed (for example, if the Ethereum Virtual Machine executes a SELFDESTRUCT), those funds may be stuck permanently. This means the tokens remain locked forever, with no way to recover them.

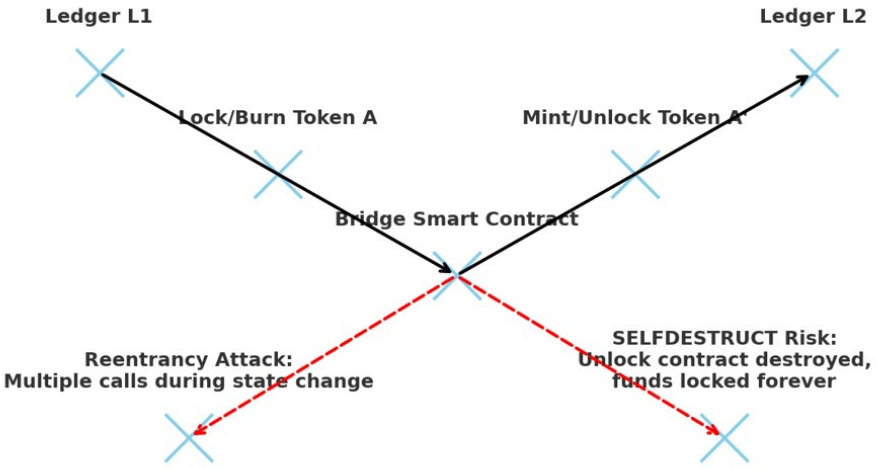

Figure 6: Diagram of the Bridge Smart Contract

The diagram illustrates how a Bridge Smart Contract operates between two ledgers ($L1$ and $L2$), enabling cross-chain flows such as lock $\rightarrow$ mint/unlock. However, it also highlights key vulnerabilities, including reentrancy attacks that exploit intermediate states to repeatedly call the contract. Another risk is SELFDESTRUCT, where destroying the unlocking contract could leave tokens permanently locked.

The second diagram showing a safe design pattern (e.g., using validator thresholds or additional checks).

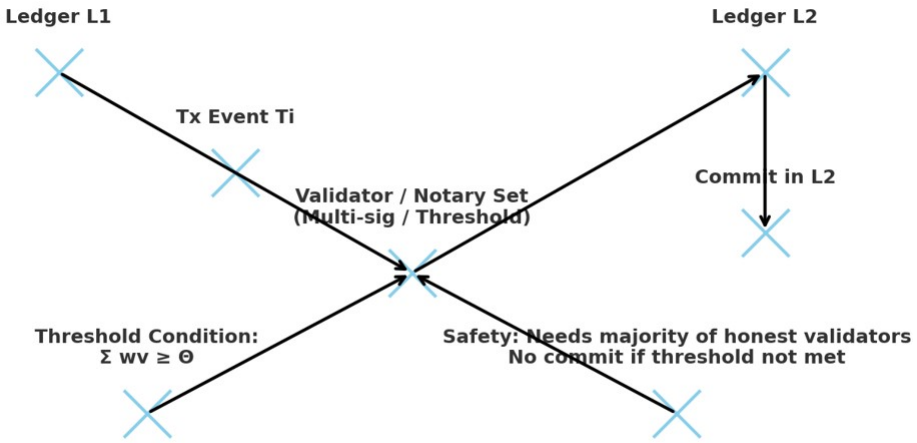

Figure 7: Diagram of the safe pattern design

The diagram illustrates how a Bridge Smart Contract operates between two ledgers ($L1$ and $L2$), enabling cross-chain flows such as lock $\longrightarrow$ mint/unlock. However, it also highlights key vulnerabilities, including reentrancy attacks that exploit intermediate states to repeatedly call the contract. Another risk is SELFDESTRUCT, where destroying the unlocking contract could leave tokens permanently locked.It is assumed that the blockchains being discussed already have strong security foundations and reliable consensus mechanisms. For the case study, the focus is placed on an everyday or common use case, which serves as the basis for examining potential security risks.

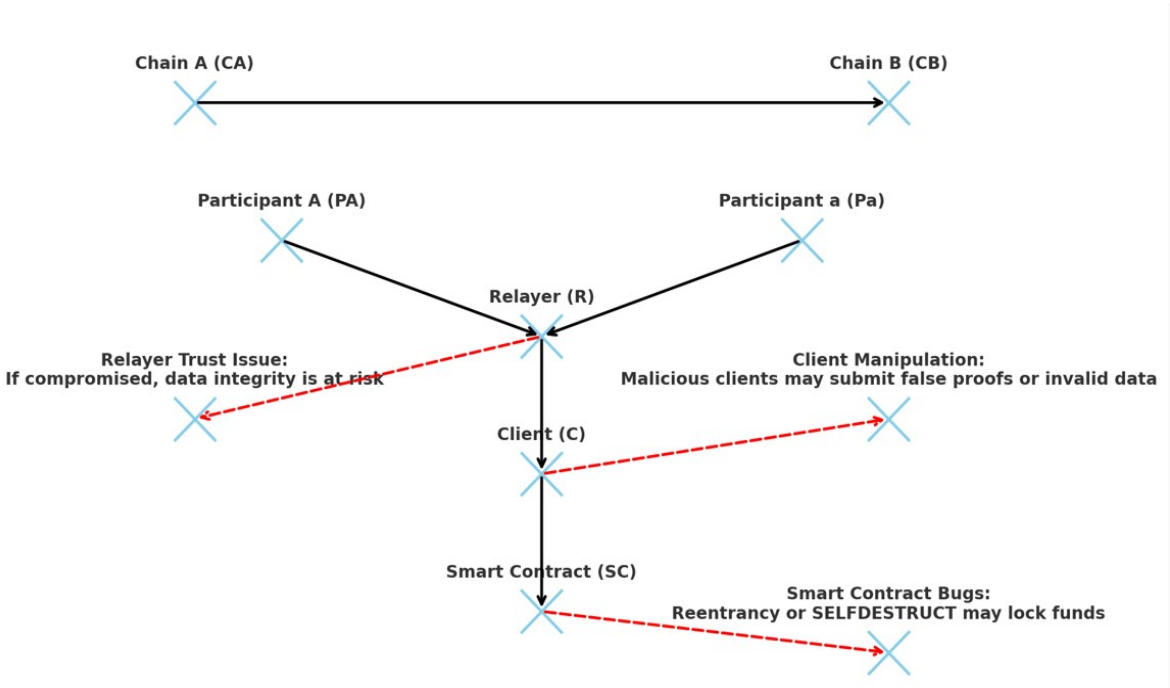

Figure 8: Participants diagram

The diagram shows participants, the flow of a cross-chain transfer, and key vulnerabilities:

In a cross-chain asset transaction model, the main components include Chain A and Chain B (where transfers occur), participants on each chain, a relayer that forwards events, a client interacting with smart contracts, and the smart contracts themselves that execute locking/unlocking logic. This setup faces vulnerabilities such as relayer trust issues, malicious clients submitting false proofs, and smart contract bugs like reentrancy or SELFDESTRUCT, all of which expose potential attack surfaces. To enable secure transfers, each cross-chain transaction must specify the type of transaction, the amount being moved, and an embedded proof of lock showing that assets are secured on the source chain.Relay Messages [15].

Messages sent through the relayer do not transfer assets directly but instead carry information needed to validate cross-chain transactions. Each relay message includes the chainId, blockNumber, and transaction data. In this way, a message acts as proof passed from one chain to another, enabling the receiving chain to validate the client's submission. Threat Model

To understand possible vulnerabilities, we use a threat model. This allows us to analyze the risks involved for a given transaction flow and network setup, and then test whether proposed countermeasures can effectively mitigate these risks. The goal is to improve security and sustainability for blockchain projects.

The transaction flow begins with a call to a smart contract requesting an asset transfer, followed by the contract validating the submitted data. Subsequent steps involve the relayer, client, and corresponding smart contract logic to complete the cross-chain process. A practical Solidity example demonstrates how a bridge can lock Ether on the source chain and emit proofs for relayers or validators before releasing assets on the destination chain.

1) Minimal "Event-Based Locker" (single relayer/agent releases on target chain)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

/// @notice Educational-only Ether locker that emits a lock event for off-chain relayers.
contract EthLocker {
    event Locked(
        address indexed sender,
        uint256 indexed dstChainId,
        address indexed dstRecipient,
        uint256 amount,
        bytes32 lockId
    );

    mapping(bytes32 => bool) public consumed; // prevent replay of the same lockId

    /// @dev User locks ETH on the source chain; relayer listens to the event.
    function lock(uint256 dstChainId, address dstRecipient, bytes32 lockId) external payable {
        require(msg.value > 0, "No ETH sent");
        require(!consumed[lockId], "lockId used");
        consumed[lockId] = true; // prevent duplicates on the source chain

        // ---- LOCK HAPPENS HERE ----
        // ETH stays held by this contract's balance until an admin-controlled refund flow occurs.

        emit Locked(msg.sender, dstChainId, dstRecipient, msg.value, lockId);
    }
```

```
    /// @notice Simple owner-based refund (for demonstrations only; real bridges avoid single admin trust).
    address public owner = msg.sender;
    function refund(address payable to, uint256 amount) external {
        require(msg.sender == owner, "not owner");
        (bool ok, ) = to.call{value: amount}("");
        require(ok, "refund failed");
    }

    receive() external payable {}
}
```

What it shows:

1. The lock is simply ETH held by the contract (msg.value).
2. A Locked event provides data (lockId, dstChainId, dstRecipient, amount) that a relayer can use on the destination chain to mint/unlock the corresponding asset.

HTLC (Hash Time-Lock Contract) for trustless ETH locking.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

/// @notice Simple ETH HTLC: funds can be claimed with a preimage s (H(s)=h) before deadline, or refunded after.
contract EthHTLC {
    struct Lock {
        address sender;
            address receiver; // often a bridge agent or the user's address on the target chain representation
        uint256 amount;
        bytes32 hashlock; // h = keccak256(s)
        uint256 timelock; // UNIX time; after this sender can refund
        bool withdrawn;
        bool refunded;
    }

    mapping(bytes32 => Lock) public locks;

     event Locked(bytes32 indexed id, address indexed sender, address indexed receiver, uint256 amount, bytes32 hashlock, uint256 timelock);
    event Withdrawn(bytes32 indexed id, bytes preimage);
    event Refunded(bytes32 indexed id);

    /// @dev Create a lock: send ETH + specify hashlock h and timelock.
       function lock(bytes32 id, address receiver, bytes32 hashlock, uint256 timelock) external payable {
        require(msg.value > 0, "no ETH");
        require(locks[id].amount == 0, "id used");
        locks[id] = Lock(msg.sender, receiver, msg.value, hashlock, timelock, false, false);
        emit Locked(id, msg.sender, receiver, msg.value, hashlock, timelock);
    }
```

```
    /// @dev Claim with preimage s before timelock.
    function withdraw(bytes32 id, bytes calldata preimage) external {
        Lock storage L = locks[id];
        require(!L.withdrawn && !L.refunded, "done");
        require(block.timestamp < L.timelock, "timed out");
        require(keccak256(preimage) == L.hashlock, "bad preimage");
        L.withdrawn = true;
        (bool ok, ) = payable(L.receiver).call{value: L.amount}("");
        require(ok, "send failed");
        emit Withdrawn(id, preimage);
    }

    /// @dev Refund after timelock if not withdrawn.
    function refund(bytes32 id) external {
        Lock storage L = locks[id];
        require(!L.withdrawn && !L.refunded, "done");
        require(block.timestamp >= L.timelock, "not yet");
        L.refunded = true;
        (bool ok, ) = payable(L.sender).call{value: L.amount}("");
        require(ok, "refund failed");
        emit Refunded(id);
    }
}
```

The example shows that ETH remains locked until either the preimage is revealed for withdrawal or the timeout passes for a refund, ensuring atomicity when paired with an HTLC on the destination chain. To make bridge lockers secure, developers must guard against reentrancy with proper patterns, avoid reliance on selfdestruct-dependent contracts, enforce replay protection with unique lockIds, and wait for sufficient block confirmations before release. Additional safeguards include using EIP-712 signature domain separation, setting transaction limits, and implementing a pausable mechanism for incident response [16].

Clean, drop-in formulas plus concrete BTC↔ETH and BTC↔Solana examples of Latency Models

$C_X(k)$ = confirmation/finality wait on chain X for k blocks (or finality epochs).

I_X = inclusion delay for a single tx on chain X (≈ one block).

S_{quorum} = notary/validator quorum signing time.

δ_{net} = cross-chain relay/propagation delay (usually seconds).

ϕ_{proof} = proof gen/verification overhead (SPV/merkle proof etc., HTLC-style bridges).

1) One-way transfer (Lock on A → Receive on B)

HTLC-style (proof-based)

$$T_{\text{HTLC, A}\to\text{B}} = C_A(k_A) + \phi_{\text{proof}} + I_B + \delta_{\text{net}}$$

Notary / Threshold-signer

$$T_{\text{Notary, A}\to\text{B}} = C_A(k_A) + S_{\text{quorum}} + I_B + \delta_{\text{net}}$$

2) Atomic swap (A↔B, both chains must lock at safety depth)

Time until you (the side receiving on B) get assets on B:

$$T_{\text{HTLC-swap, receive on B}} = C_A(k_A) + C_B(k_B) + I_B + \delta_{\text{net}}$$

Claim on A by the counterparty happens after, and doesn't delay your receipt on B.

Parameter Cheatsheet (typical)

Bitcoin (BTC): $I_{\text{BTC}} \approx 10$ minper block; common $k_A = 3 \sim 6 \Rightarrow C_{\text{BTC}} \approx 30 \sim 60$ min.

Ethereum (ETH): $I_{\text{ETH}} \approx 12$ s; common policies:

- "light" confirmations ~12 blocks $\Rightarrow C_{\text{ETH}} \approx 2.4$ min, or
- full finality $\approx 12 - 13$ min.

Solana (SOL): $I_{\text{SOL}} \approx 0.4 - 0.6$ s; finality $C_{\text{SOL}} \approx 2 - 5$ s.

Overheads: $\delta_{\text{net}} \sim 1 - 5$ s; $S_{\text{quorum}} \sim 1 - 3$ s(well-peered validators); ϕ_{proof}can be seconds to a minute+ depending on proof size/verification gas and relayer cadence.

Worked Examples

A) BTC ⟶ ETH

Notary model (one-way lock & mint)

Choose $k_{\text{BTC}} = 3$(exchange-class fast path): $C_{\text{BTC}} \approx 30$ min

$$S_{\text{quorum}} \approx 2 \text{ s}$$

$I_{\text{ETH}} \approx 12 - 36$ s(1–3 blocks)

$$\delta_{\text{net}} \approx 2 \text{ s}$$

$$T_{\text{Notary}} \approx 30 \text{ min} + 2 \text{ s} + 24 \text{ s} \approx 30.5 \text{ min}$$

With conservative $k_{\text{BTC}} = 6$: ≈ 60.5 min.

HTLC-style (proof-based one-way)

C_{BTC}same as above.

ϕ_{proof}(BTC SPV relay + on-chain verify) ≈ 20–90 s (design-dependent).

$I_{\text{ETH}} \approx 12 - 36$ s, $\delta_{\text{net}} \approx 2$ s.

$$T_{\text{HTLC}} \approx 30 - 60 \text{ min} + (0.5 - 1.5 \text{ min}) \approx 30.5 - 61.5 \text{ min}$$

Takeaway (BTC→ETH): Notary and HTLC are both dominated by BTC confirmations; Notary typically shaves off the proof overhead, saving ~0.5–1.5 minutes and some gas.

B) BTC ⟶ Solana

Notary model (one-way lock & mint)

$C_{\text{BTC}} = 30 - 60$ min(3–6 confs)

$$S_{\text{quorum}} \approx 2 \text{ s}$$

$I_{\text{SOL}} \approx 0.4 - 0.6$ s, finality $2 - 5$ s

$$\delta_{\text{net}} \approx 2 \text{ s}$$

$$T_{\text{Notary}} \approx 30 - 60 \text{ min} + \text{a few seconds}$$

HTLC-style (proof-based one-way)

$$C_{\mathrm{BTC}} = 30-60 \text{ min}$$

ϕ_{proof}(BTC proof relay → Solana verifier) ≈ 10–60 s
$I_{\mathrm{SOL}} \approx < 1$ s, $\delta_{\mathrm{net}} \approx 2$ s

$$T_{\mathrm{HTLC}} \approx 30-60 \text{ min} + 10-60 \text{ s}$$

Takeaway (BTC→SOL): Again dominated by BTC. Solana's sub-second inclusion means the Notary advantage is mostly the missing proof cost.

C) Atomic swap sketches (for completeness)

BTC↔ETH (receive on ETH):

$$T \approx C_{\mathrm{BTC}}(3-6) + C_{\mathrm{ETH}}(\text{policy}) + I_{\mathrm{ETH}}$$

$\approx 30-60$ min $+ 2.4-13$ min $+ 12-36$ s $\Rightarrow 32-73$ min.

BTC↔SOL (receive on SOL):

$T \approx 30-60 \text{ min} + 2-5 \text{ s} + ¿1 \text{ s} \Rightarrow 30-60$ min + seconds.

Quick Design Notes (why Notary feels "faster")

Both models must trust BTC finality; that's the wall-clock bottleneck.

Notary avoids ϕ_{proof}(heavy proofs/on-chain verification), replacing it with tiny S_{quorum}.

On fast destination chains (ETH w/ light confirms, Solana), Notary often "wins by seconds–a minute," not by tens of minutes—because BTC dominates.

Scenario	Model	Total Seconds	Total Minutes	Notes
BTC - ETH (3 conf)	Notary	1828	30.46666667	BTC 3 conf (~30m), quorum ~2s, ETH inclusion ~24s
BTC - ETH (3 conf)	HTLC	1886	31.43333333	BTC 3 conf (~30m), proof ~60s, ETH inclusion ~24s
BTC - ETH (6 conf)	Notary	3628	60.46666667	BTC 6 conf (~60m), quorum ~2s, ETH inclusion ~24s
BTC - ETH (6 conf)	HTLC	3686	61.43333333	BTC 6 conf (~60m), proof ~60s, ETH inclusion ~24s
BTC - SOL (3 conf)	Notary	1805	30.08333333	BTC 3 conf (~30m), quorum ~2s, SOL inclusion ~1s
BTC - SOL (3 conf)	HTLC	1833	30.55	BTC 3 conf (~30m), proof ~30s, SOL inclusion ~1s
Atomic BTC - ETH (recv on ETH)	HTLC-swap	1970	32.83333333	BTC 3 conf + ETH ~2.4m + ETH inclusion ~24s
Atomic BTC - SOL (recv on SOL)	HTLC-swap	1806	30.1	BTC 3 conf + SOL ~3s + inclusion ~1s

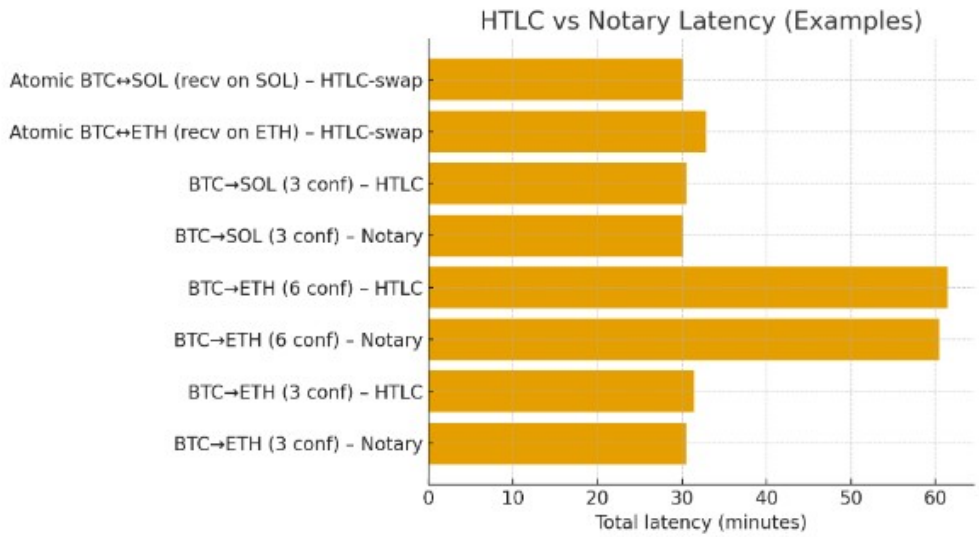

Figure 9: HTLC vs Notary Latency (Examples)

5. Conclusion

The research provides a detailed analysis of blockchain bridge frameworks, including relay schemes, HTLCs, notary models, and smart contracts, highlighting their risks, vulnerabilities, and mitigation strategies. A general threat model and case studies are presented to show how weaknesses can be identified and countered, helping developers make informed design choices and strengthen bridge security. Practically, the study offers developers concrete steps to mitigate risks and gives users clearer insight into how cross-chain solutions function and the safeguards applied.

The research highlights the importance of standardizing frameworks for blockchain interoperability and developing a unified cross-chain communication model. It also stresses the need to define recommended cryptographic protocols and create unified templates for building bridges. Together, these steps would help establish a more secure and reliable blockchain ecosystem.

The paper applies a component-based threat analysis along with a case study but suggests that future research could also use methodologies like STRIDE and DREAD. Incorporating these approaches would expand the scope of security evaluation for blockchain bridges. Such efforts would foster collaboration between academia and industry, ultimately enhancing the protection and efficiency of cross-chain transactions.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] P. Petriv, I. Opirskyy, N. Mazur, Modern Technologies of Decentralized Databases, Authentication, and Authorization Methods, in: Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3826, 2024, 60–71.

[2] I. Hanhalo, et al., Adaptive Approach to Ensuring the Functional Stability of Corporate Educational Platforms under Dynamic Cyber Threats, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3991 (2025) 481–491.

[3] V. Zhebka, et al., Methodology for Choosing a Consensus Algorithm for Blockchain Technology, in: Digital Economy Concepts and Technologies Workshop, DECaT, vol. 3665 (2024) 106–113.
[4] I. Opirskyy, et al. (2025): Modern Methods of Ensuring Information Protection in Cybersecurity Systems using Artificial Intelligence and Blockchain Technology. In: O. Harasymchuk (ed.) Kharkiv: Technology Center PC. doi:10.15587/978-617-8360-12-2
[5] M. Adamantis, V. Sokolov, P. Skladannyi, Evaluation of State-of-the-Art Machine Learning Smart Contract Vulnerability Detection Method, Advances in Computer Science for Engineering and Education VII, vol. 242 (2025) 53–65. doi:10.1007/978-3-031-84228-3_5
[6] V. Astapenya, et al., Conflict Model of Radio Engineering Systems under the Threat of Electronic Warfare, in: Cybersecurity Providing in Information and Telecommunication Systems, CPITS, vol. 3654 (2024) 290–300.
[7] P. Skladannyi, et al., Model and Methodology for the Formation of Adaptive Security Profiles for the Protection of Wireless Networks in the Face of Dynamic Cyber Threats, in: Cyber Security and Data Protection, vol. 4042 (2025) 17–36.
[8] K. W. Prewett, G.L. Prescott, K. Phillips, Blockchain Adoption is Inevitable—Barriers and Risks remain, J. Corp. Account. Finance, 31(2) (2020) 21–28. doi:10.1002/jcaf.22415
[9] L. E. Whitman, D. Santanu, H. Panetto, An Enterprise Model of Interoperability, IFAC Proc., 39(3) (2006) 609–614. doi:10.3182/20060517-3-FR-2903.00311
[10] B. Pillai, K. Biswas, Z. Hóu, V. Muthukkumarasamy, Burn-to-Claim: An Asset Transfer Protocol for Blockchain Interoperability, Comput. Netw., 200 (2021) 108495. doi:10.1016/j.comnet.2021.108495
[11] G. Wang, M. Nixon, SoK: Tokenization on Blockchain, in: 14th IEEE/ACM Int. Conf. Utility and Cloud Comput. Companion, 2021, 1–9. doi:10.1145/3492323.3495577
[12] M. Zhang, et al., SoK: Security of Cross-Chain Bridges: Characteristics, Attack Surfaces, Defenses, and Open Problems, 2023. https://arxiv.org/html/2312.12573v1
[13] A. Shantyr, et al., Prediction of Software Quality Indicators with Applied Modifications of Integrated Gradient Methods, Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska, 15(2) (2025) 139–146.
[14] N. Belenkov, et al., SoK: A Review of Cross-Chain Bridge Hacks in 2023, 2025. doi:10.48550/arXiv.2501.03423
[15] H.O. Sevim, A survey on Trustless Cross-Chain Interoperability Solutions in On-Chain Finance, Catholic Univ. Sacred Heart, Milano, Italy, 2024. https://dlt2024.di.unito.it/wp-content/uploads/2024/05/DLT2024_paper_14.pdf
[16] EIP-5164 (Cross-Chain Execution): Standard Dispatcher/Executor Interface for Cross-Chain Calls + Community Discussions, 2025. https://eips.ethereum.org/EIPS/eip-5164