Cross-Chain Bridges: Attack Taxonomy, Defenses, and Open Problems

Mengya Zhang
The Ohio State University
Columbus, OH, United States
zhang.9407@osu.edu

Xiaokuan Zhang
George Mason University
Fairfax, VA, United States
xiaokuan@gmu.edu

Yinqian Zhang*
Southern University of Science and Technology
Shenzhen, China
yinqianz@acm.org

Zhiqiang Lin
The Ohio State University
Columbus, OH, United States
zlin@cse.ohio-state.edu

ABSTRACT

Cross-chain bridges play a pivotal role in enabling token and data exchanges between disparate blockchains. Despite their growing popularity, these bridges are still in their infancy and have been the target of numerous attacks, leading to significant financial losses. Current literature lacks a comprehensive examination of the security landscape surrounding cross-chain bridges, with existing incident reports dispersed and unconsolidated. Addressing this gap, this paper presents a systematic investigation into the security challenges facing cross-chain bridges. We begin by outlining the key features of current cross-chain bridges, including their applications, verification processes, communication models, and a novel three-fold categorization. From this foundation, we identify 12 potential attack vectors and develop a taxonomy of cross-chain bridge attacks observed over the past three years, classifying them into 10 unique categories. Each category is detailed with corresponding vulnerabilities, illustrated through Solidity code examples. Furthermore, we explore existing defense mechanisms, propose potential security solutions, and highlight crucial open questions and avenues for future research. This paper aims to illuminate the path towards more secure cross-chain bridge designs and stimulate further investigation into fortifying the cross-chain bridge ecosystem.

CCS CONCEPTS

• Security and privacy → Distributed systems security.

KEYWORDS

Cross-chain Bridge, Attack Surface, Defense

---

*Yinqian Zhang is affiliated with the Research Institute of Trustworthy Autonomous Systems (RITAS) and the Department of Computer Science and Engineering at Southern University of Science and Technology.

---

1 INTRODUCTION

In recent years, blockchain applications have surged in popularity, enabling users not just token transfers but also leveraging smart contracts for computations. Smart contracts are self-executing programs commonly found in blockchains like Ethereum [65] and serve as backends for Decentralized Applications (DApps). DappRadar [56] reports that all DApps combined had 5.3 million daily Unique Active Wallets (dUAW) as of Feb 2024. Despite the popularity, blockchain applications are restricted to a single blockchain domain; blockchains cannot communicate or share data since they are isolated from each other. For instance, Ethereum's native token, *ETH*, restricted to Ethereum, often requires users to sell it for fiat currency before trading it for tokens on other blockchains. These steps are time-consuming, complex, and prone to risks.

To meet the increasing demand for conducting cross-chain transactions, cross-chain bridges are created to facilitate communications between different blockchains. For example, AnySwap [17] is a cross-chain bridging protocol that allows users to move assets between various blockchains such as Solana [28] and Ethereum; ChainBridge [38] supports bridging between EVM and Substrate-based chains, where Substrate is a framework for creating custom blockchains [128]. As of April 2024, there are around 80 cross-chain bridges [71], which support various user demands.

Due to the growing popularity and Total Value Locked (TVL) of cross-chain bridges, bridges increasingly attract the attention of attackers. For instance, the first reported attack on bridges was the ChainSwap attack [110] on Jul-10-2021, resulting in a loss of $8 million. For another instance, the attacker stole around $190 million worth of tokens in the Nomad hack [83] that occurred on Aug-02-2022. As of April 2024, cross-chain bridges have been compromised for approximately $2.83 billion, accounting for nearly 36.42% of the total blockchain value hacked, as reported by DefiLlama [60].

Despite the prosperity of cross-chain bridges and the severity of attacks, research on the security of cross-chain bridges is still in the early stage. Few recent works [4, 8, 147] focus on detecting existing bridge attacks, while most existing works focus on measuring the interoperability between blockchains [9, 25, 142], summarizing current cross-chain protocols [145], or proposing new protocols [98,

130, 135, 143]. *Lee* et al.[97] examine eight attack surfaces across bridge components using eight attacks. Additionally, *Belchior* et al.[8] define 45 cross-chain vulnerabilities and analyze 14 bridge hacks. However, they did not provide a holistic view of the security issues on cross-chain bridges, such as a systematic characterization of cross-chain bridges, their vulnerabilities, and all reported attacks.

To clear the fog that surrounds the dark side of cross-chain bridges, this paper conducts a systematic study to fill the knowledge gap. First, we present the characteristics of bridges by introducing four types of verification methods (*external verification*, *local verification*, *optimistic verification*, and *native verification*), two types of communication models (*lock-and-mint; burn-and-release* model and *liquidity-pool-based* model), three categorization methodologies (*trust mechanism*, *layer connection*, and *functionalities*), and usages (*tokens transfer*, *governance*, *lending and borrowing*, and *staking*). We also summarize 30 popular cross-chain bridges with describing their characteristics. We find most bridges use the *external verification* as their underlying protocol and deploy the *lock-and-mint; burn-and-release* communication model. Moreover, based on these characteristics, we examine the workflow of cross-chain bridges and identify 12 potential pitfalls that attackers might exploit.

Next, we provide a taxonomy of existing bridge attacks by investigating all 35 bridge attacks from April 2021 to April 2024 [35, 59, 87] into 10 distinct types of vulnerabilities. The fundamental vulnerabilities that led to the attacks are categorized into four categories: *permission issue (PI)*, *logic issue (LI)*, *event issue (EI)*, and *front-end issues (FI)*. For each category of attacks, we give concrete examples in Solidity code. Furthermore, we summarize existing defenses to these attacks and give recommendations to bridge developers for both on-chain backends (*i.e.*, smart contracts) and off-chain servers[1] to help them build more secure services. Finally, we discuss open problems in the cross-chain bridge ecosystem, and propose potential directions for future research.

Contributions. In short, we make the following contributions:

- We give an overview of characteristics of cross-chain bridges, including verification mechanisms, communication models, and classifications based on different properties, including trust mechanism, layer connection, and functionalities (Sec. 4);
- We identify 12 potential attack surfaces based on the characteristics of cross-chain bridges (Sec. 5);
- We provide a taxonomy on all the reported cross-chain bridge attacks (Sec. 6);
- We summarize the existing defenses of reported bridge attacks and provide recommendations to developers (Sec. 7);
- We discuss open questions and future works on cross-chain bridge research (Sec. 8).

Scope. Our primary focus lies in the study of bridges between public blockchains, to 1) summarize the characteristics of existing cross-chain bridges, 2) identify potential attack surfaces, 3) analyze all existing bridge attacks happened in the wild, and 4) provide lessons learned from these incidents. While we discuss defense mechanisms in the paper, a systematic study on how to build mitigations or detection tools is not the goal of this paper. A comprehensive study

of cross-chain protocols (*e.g.*, [98, 130, 135, 143, 145]) is also out of the scope.

2 BACKGROUND

Blockchain is a decentralized ledger built on top of a Peer-to-peer (P2P) network [50] consisting of distributed peers. A blockchain can be considered as a continuously growing list of blocks. Each block contains transactions that either transfer tokens in Bitcoin-like blockchains or perform more complex operations based on the computation logic in Ethereum-like blockchains.

Smart contracts were first invented in Ethereum. They are programs that are stored and automatically executed according to the agreements between different parties [64], such as users and developers. Once smart contracts are deployed to blockchains, they become immutable. Thus, bugs in smart contracts are hard to fix due to their immutability. While smart contracts can suffer from attacks, they have been the foundations of the Decentralized Finance ecosystem, and have been adopted to many widely used blockchains, such as BNB [14] and Fantom [72].

Transactions are data structures recording cryptocurrency information in the blockchains, which are either calls to the smart contracts or simple token transfers to the blockchain users [93]. Once transactions are mined to blockchains, the changes to the blockchain will be permanent and cannot be reverted. Moreover, a transaction normally needs gas fee to pay for its computation cost. For example, Ethereum needs users to pay the transaction fee using the native token *ETH* [63]. In particular, the fee consists of gas used, *i.e.*, the total units of the cost and gas price, *i.e.*, the value users would like to pay for every unit. Furthermore, the higher the gas price is, the quicker a transaction usually is mined since miners are motivated to maximize their profits.

Tokens are digital assets and a form of smart contracts that reside on blockchains. Tokens have their own addresses; however, *ETH*, the native cryptocurrency of Ethereum, is not considered a token. Moreover, every user can create his/her own token by implementing the smart contract standards required by blockchains, such as ERC20 [62] and ERC721 [122] in Ethereum. Tokens can be mainly classified into three categories:

- **Fungible tokens** are not unique, divisible, and interchangeable tokens that represent some values (similar to fiat currency such as USD). ERC20 tokens are one popular type of fungible tokens.
- **Non-fungible tokens** (NFTs) are non-divisible assets that are unique to each other. ERC721 tokens are one type of NFTs.
- **Wrapped tokens** represent cryptocurrencies pegged to the value of the related original tokens [46], including both fungible and non-fungible tokens. For example, *WETH* is the wrapped version of *ETH*.

DApps are decentralized applications relying on blockchains [12]. Compared with traditional applications (e.g., websites or mobile apps), there is no difference in the frontends, but the backends of DApps are usually composed of multiple smart contracts in blockchains. In particular, cross-chain bridges are an increasingly popular type of DApps.

---

[1]Cross-chain bridge server facilitates communication and coordination between different blockchains, ensuring that the assets are transferred securely and accurately.

Paper	Research Focus	#Attack Surface	#Real-world Attack	#Vul Type	Code Example?	Open Problems?
Lee *et al.* [97]	Bridge Component	8	8	8	No	No
Augusto et al. [4]	Interoperability	45	14	45	No	No
This Paper	Communication Model	11	31	10	Yes	Yes

Table 1: Summary of comparison with related works.

Liquidity pools are the places that lock the tokens in smart contracts [5]. Liquidity pools are designed to encourage users to provide liquidity by depositing tokens to the pools. Such users are called liquidity providers. Moreover, liquidity pools will distribute rewards to liquidity providers in terms of crypto tokens, based on the liquidity they have supplied. While liquidity pools facilitate token exchanges, liquidity providers are exposed to impermanent losses caused by the token price decrease.

3 RELATED WORKS

In this section, we discuss related works that study bridge attacks. Additional related works are discussed in Appendix A, including blockchain interoperability, cross-chain protocols, atomic swaps, and side chains. Different from all the existing works, our work provides a systematization of attacks on cross-chain bridges, and further discusses open problems and future directions.

Bridge attack systemization. As shown in Table 1, Lee *et al.* [97] conducted an analysis of bridge attacks from four components, including custodians, debt issuers, communicators, and token interfaces. They only summarized eight possible attack surfaces based on eight real-world examples and discussed potential mitigation strategies. Moreover, *Augusto et al.* [4] analyze 14 notable bridge hacks. McCorry *et al.* [101] review bridge literature, detailing roles and components, especially validating bridge contracts, but their threat models are theoretical without real-world attacks. In contrast, our research takes a different approach. We systematically proposed 12 potential attack surfaces, drawing from two widely used bridge communication models. Our work offers a comprehensive taxonomy that encompasses reported 35 real-world cross-chain bridge attacks from April 2021 to April 2024, which are classified into 10 vulnerability types. For each vulnerability, we provide source code examples, an explanation of related real-world attacks, and potential solutions. We also discuss open problems.

Bridge attack detection. *Zhang et al.* [147] introduce a tool to uncover bridge attacks originating from transactions, and assess its effectiveness across four cross-chain bridges. Notably, the tool identifies vulnerabilities in cross-chain bridges, encompassing issues such as unrestricted deposit emissions, inconsistent event parsing, and unauthorized unlocking—elements integral to our condensed attack surface analysis. *Belchior et al.* [8] propose a cross-chain model generator, aiding in the detection of outliers and malicious behaviors. Our work primarily centers on categorizing real-world attacks and their associated attack surfaces, as opposed to presenting a methodology for their detection; we consider detection mechanisms as potential avenues for future research.

4 CHARACTERIZING CROSS-CHAIN BRIDGES

In this section, we first examine the bridge usages. Second, we illustrate four types of verification mechanisms. Third, we introduce the

two prominent communication models in cross-chain bridges: *lock-and-mint; burn-and-release* model and *liquidity-pool-based* model. Furthermore, we categorize the bridges based on their trust mechanism, layer connection, and functionalities. We also summarize 30 popular bridges with describing their characteristics in Table 2.

4.1 Bridge Usages

Bridges exchange data between multiple blockchains, including messages and tokens. We observe that bridges primarily facilitate cross-chain token transfers. We also describe other potential usages, including governance, lending and borrowing, and staking (see Appendix B for details).

Cross-chain token transfer. As shown in Fig. 1, $\mathbb{A}_S$ can submit a transaction *TxLock* in Ethereum to transfer 10.0 *ETH* to the bridge account, and then the bridge submits the transaction *TxMint* to transfer wrapped tokens (10.0 *WETH*) to $\mathbb{A}_D$ in BNB.

4.2 Verification Mechanisms

Each bridge utilizes a unique protocol for monitoring blockchains, achieving consensus, and relaying messages. Cross-chain bridge verification methods fall into four categories: external, optimistic, local, and native verification [48].

4.2.1 External Verification. External Verification (EV) relies on a single or set of external validators, who are responsible for verifying the validity of cross-chain transactions. Bridges operating under this categorization assume that *most validators are honest*, and typically adopt multi-signature or multi-party computation (MPC). With the multi-signature mechanism, a threshold is set for the number of validators required to sign the transaction, and each validator has a complete private key. In contrast, MPC requires the validators to jointly generate a private key. Despite being simple to implement and widely used, this mechanism is exposed to considerable risks if the external validators are not trusted or are hacked. For instance, if the private keys of these validators are compromised, the bridges may be taken over by attackers. We illustrate two protocols, including Chainlink's cross-chain interoperability protocol (CCIP) and an omnichain interoperability protocol (LayerZero) in Appendix C.1.

4.2.2 Optimistic Verification. Optimistic Verification (OV) operates on the assumption that a transaction is valid and initiates a challenge period to verify its accuracy. If the transaction is found to be invalid during the challenge period, the transaction will be reversed, and malicious actors will be penalized. However, bridges under this categorization often have a longer latency due to the challenge period. Moreover, such bridges *require at least one honest watcher* to verify updates and detect fraudulent activity. We illustrate an optimistic inter-chain communication protocol in Appendix C.2.

4.2.3 Local Verification. Local Verification (LV) requires only the parties involved in the transaction to verify it, making it a two-party verification mechanism that maintains the security of the underlying blockchain. This mechanism *inherits the security of the underlying blockchain* and is popularly used for atomic swaps, which ensure transaction atomicity using Hash Time Lock Contracts (HTLC). HTLC uses hashlock and timelock to lock assets

Bridge	Verification	Comm. Model	Classifications			Blockchains
			Trust Mechanism	Layer	Functionalities	
Polygon Bridge	External	Lock-and-Mint	Semi-Trustless	L1 - L2	Chain-Specific	Ethereum, Polygon
Arbitrum Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Chain-Specific	Ethereum, Arbitrum
Optimism Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Chain-Specific	Ethereum, Optimism
Portal Token Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Chain-Specific	30 blockchains [22]
Orbit Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Generalized	21 blockchains [20]
Rainbow Bridge	Native	Lock-and-Mint	Trustless	L1 - L1	Chain-Specific	Ethereum, Near
xDAI Bridgex	External	Lock-and-Mint	Trusted	L1 - L2	Asset-Specific	Ethereum, Gnosis
Satellite by Axelar	Native	Lock-and-Mint	Trustless	L1 - L2	Generalized	14 blockchains [26]
Synapse Protocol	External	Lock-and-Mint	Semi-Trustless	L1 - L2	Application-Specific	19 blockchains [136]
Hop.Exchange	Local	Lock-and-Mint	Trustless	L1 - L2	Application-Specific	7 blockchains [92]
Across Protocol	Optimistic	Liquidity-Pool-Based	Trustless	L1 - L2	Application-Specific	5 blockchains [123]
Voltage Bridge	Local	Liquidity-Pool-Based	Trustless	L1 - L2	Chain-Specific	Ethereum, Fuse, Bnb
Hot Cross Multi-Chain Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Application-Specific	Ethereum, Avalanche, Bnb
BoringDAO Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Application-Specific	14 blockchains [15]
Multichain	External	Lock-and-Mint	Trusted	L1 - L2	Application-Specific	26 blockchains [107]
ioTube Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Chain-Specific	Ethereum, IoTeX, Bnb, Polygon
ChainPort Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Application-Specific	17 blockchains [41]
Wrap Protocol	External	Lock-and-Mint	Trusted	L1 - L1	Chain-Specific	Ethereum, Tezzo
Avalanche Bridge	External	Lock-and-Mint	Trusted	L1 - L1	Chain-Specific	Ethereum, Avalanche
ThunderCore Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Chain-Specific	4 blockchains [139]
RenBridge	External	Lock-and-Mint	Trusted	L1 - L1	Application-Specific	Ethereum, Bitcoin
Cross-Chain Bridge	External	Liquidity-Pool-Based	Trusted	L1 - L2	Application-Specific	5 blockchains [18]
Sovryn Bridge	External	Liquidity-Pool-Based	Trusted	L1 - L2	Chain-Specific	Ethereum, RSK, Bnb
Hyphen Bridge	External	Liquidity-Pool-Based	Trusted	L1 - L2	Application-Specific	7 blockchains [19]
Celer cBridge	Native	Lock-and-Mint	Trustless	L1 - L2	Application-Specific	34 blockchains [29]
RSK Token Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Chain-Specific	Ethereum, RSK
Celo Optics Bridge	Optimistic	Lock-and-Mint	Trustless	L1 - L2	Generalized	Ethereum, Polygon, Celo
Nomad	Optimistic	Lock-and-Mint	Trustless	L1 - L2	Application-Specific	4 blockchains [114]
Allbridge	External	Lock-and-Mint	Trusted	L1 - L2	Application-Specific	21 blockchains [3]
SOY Bridge	External	Lock-and-Mint	Trusted	L1 - L2	Application-Specific	5 blockchains [42]

Table 2: Summary of 30 popular cross-chain bridges. *Lock-and-Mint* represents the *Lock-and-Mint; Burn-and-Release* communication model.
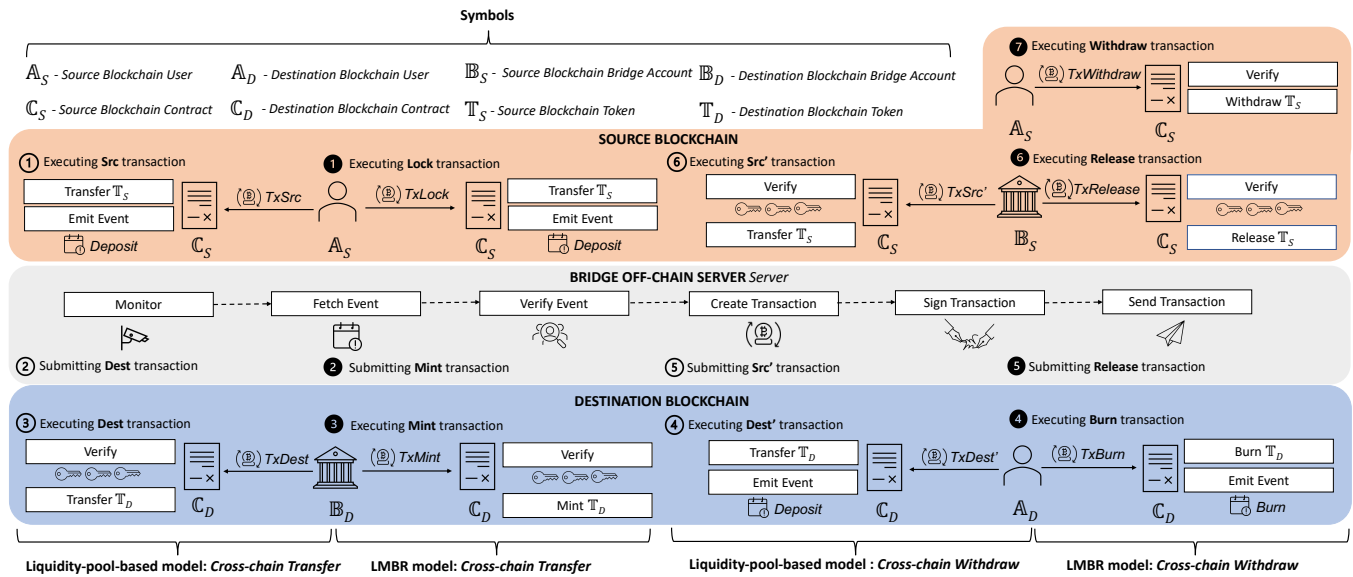

Figure 1: The workflows of two cross-chain bridge communication models. *Lock-and-Mint; Burn-and-Release (LMBR)* model consists of ❶ - ❼, and *Liquidity-pool-based* model consists of ① - ⑥.

within a time window and requires consensus from both parties to complete the payment. The receiver of the payment can either accept it before it expires or forfeit it. However, LV has limitations and cannot be easily extended to other applications in addition to token transfers. We illustrate Connext's NXTP in Appendix C.3.

4.2.4 Native verification. Native verification (NV) requires the destination blockchain to run a light client of the source blockchain to verify the related events or states. Light clients store the minimum information required to verify the status change of blockchains in a self-verifiable manner. They keep block headers to confirm cross-chain messages and stay updated with any changes in blockchains to verify the relayed cross-chain transactions based on stored state information. Although this mechanism is the most secure, as it *has the same security assumptions as the public chain*, it is expensive to maintain the light clients for all supported blockchains for bridges. Moreover, we illustrate one MAP Protocol in Appendix C.4.

Note in Table 2, the majority of bridges employ external verification (22), while a small number opt for optimistic verification (3), native verification (3), or local verification (2).

4.3 Communication Models

There are two models commonly used by the cross-chain bridges, which are (1) **lock-and-mint; burn-and-release** model and (2) **liquidity-pool-based** model, respectively. Specifically, model (1) will mint new wrapped tokens (e.g., *wBTC*) in the destination chain, which will be burnt to release tokens locked in the source blockchain, while model (2) only includes native assets (e.g., *BTC*). To make it clear, we illustrate the workflow of these two models with common usages: cross-chain transfer and withdraw in Fig. 1.

4.3.1 Lock-and-Mint; Burn-and-Release Model. In Fig. 1, Step ❶ to ❸ are the *cross-chain transfer* and Step ❹ to ❼ are the *cross-chain withdraw*, which is the reverse operation to the *cross-chain transfer*.

Cross-chain transfer. Assume user $\mathbb{A}_S$ wants to send money from one chain to his/her own account $\mathbb{A}_D$ in another chain, which is usually implemented via the *cross-chain transfer*:

❶ *Executing **Lock** transaction.* The first step is to submit a transaction *TxLock* from $\mathbb{A}_S$ to $\mathbb{C}_S$, performing a token transfer from $\mathbb{A}_S$ to the bridge account. Meanwhile, the *Deposit* event is emitted for *Server* to listen and record.

❷ *Submitting **Mint** transaction.* In step two, the *Deposit* event containing detailed information (e.g., token amount) is verified by *Server*. If the event is parsed and verified to be correct (e.g., emitted from the $\mathbb{C}_S$), the related transaction *TxMint* will be created, signed by validators, and then sent to call $\mathbb{C}_D$.

❸ *Executing **Mint** transaction.* In step three, *TxMint* is sent from $\mathbb{B}_D$. During execution, the signatures from validators are verified. After verification, related tokens $\mathbb{T}_D$ (wrapped tokens of $\mathbb{T}_S$) are minted. In particular, tokens are transferred from the bridge account or *zero* address [68] to $\mathbb{A}_D$, which is appointed by $\mathbb{A}_S$ in *TxLock*.

Cross-chain withdraw. When user $\mathbb{A}_S$ needs to take his/her tokens that are locked in the source blockchain back, $\mathbb{A}_S$ conducts a *cross-chain withdraw*. The cross-chain withdraw can only be used after the cross-chain transfer.

❹ *Executing **Burn** transaction.* In step four, $\mathbb{A}_D$ should submit a transaction *TxBurn* calling $\mathbb{C}_D$, to burn tokens $\mathbb{T}_D$, by sending the tokens to the bridge account or the *zero* address. Meanwhile, the *Burn* event is emitted from $\mathbb{C}_D$ for *Server* to monitor.

❺ *Submitting **Release** transaction.* This step is similar to step ❷. If the *Burn* event is verified, the transaction *TxRelease* will be created and signed by enough validators.

❻ *Executing **Release** transaction.* In step six, **Release** is sent from $\mathbb{B}_S$, signatures from bridge validators are verified in $\mathbb{C}_S$, and tokens locked before will be released, after verification. However, tokens are not sent directly back to $\mathbb{A}_S$ yet. Instead, a proof showing that the tokens are available to withdraw is sent to $\mathbb{A}_S$.

❼ *Executing **Withdraw** transaction.* In step seven, $\mathbb{A}_S$ submits the transaction *TxWithdraw* to $\mathbb{C}_S$ with the proof, to withdraw tokens. If the proof is valid, the bridge will transfer tokens back to $\mathbb{A}_S$.

4.3.2 Liquidity-Pool-Based Model. For the liquidity-pool-based model in Fig. 1, Step ① to ③ are the *cross-chain transfer* and Step ④ to ⑥ are the *cross-chain withdraw*.

Cross-chain transfer. Assume user $\mathbb{A}_S$ wants to send money from one chain to his/her own account $\mathbb{A}_D$ in another chain, which is usually implemented via the *cross-chain transfer*:

① *Executing **Src** transaction.* To begin with, the bridge creates the liquidity pool that only contains token $\mathbb{T}_S$. Different from normal liquidity pools, cross-chain bridge liquidity pools only have single type of tokens, to prevent impermanent losses for liquidity providers. Then, the liquidity providers add liquidity to the pool by depositing tokens $\mathbb{T}_S$. Accordingly, the first step is to execute a transaction *TxSrc* from $\mathbb{A}_S$ to $\mathbb{C}_S$, performing a token transfer and emitting the *Deposit* event for *Server* to listen and record.

② *Submitting **Dest** transaction.* This step is similar to the step ❷ in Lock-and-Mint; Burn-and-Release model.

③ *Executing **Dest** transaction.* Before step three, the liquidity pools in the destination blockchain should be prepared as well. Similarly, the pool only containing tokens $\mathbb{T}_D$ is created by the bridge and the liquidity providers add liquidity to the pool. In particular, tokens $\mathbb{T}_S$ and $\mathbb{T}_D$ represent the same tokens in different blockchains. Meanwhile, the transaction *Dest* sent from $\mathbb{B}_D$ to $\mathbb{C}_D$ will perform the verification and then transfer the tokens $\mathbb{T}_D$ from the liquidity pool to $\mathbb{A}_D$.

Cross-chain withdraw. Cross-chain withdraw is similar to *Cross-chain transfer*, except that the direction (from destination to source) is opposite from the direction of *Cross-chain transfer* (from source to destination).

④ *Executing **Dest'** transaction.* To withdraw the tokens deposited in the source blockchain, user $\mathbb{A}_D$ initializes a transaction *TxDest'* that sends the tokens $\mathbb{T}_D$ back to the bridge $\mathbb{C}_D$.

⑤ *Submitting **Src'** transaction.* This step is also similar to the step ❷ in Lock-and-Mint; Burn-and-Release model.

⑥ *Executing **Src'** transaction.* Transaction *TxSrc'* is sent to $\mathbb{C}_S$, verifying the data (e.g., proof) and transferring tokens $\mathbb{T}_S$ back to $\mathbb{A}_S$ then.

Note in Table 2, 25 bridges use *lock-and-mint; burn-and-release* communication model and 5 bridges use *Liquidity-Pool-Based* communication model.

4.4 Categorization

We classify the cross-chain bridges based on different standards, including 1) trust mechanisms, 2) layer connections, and 3) functionalities.

4.4.1 Trust Mechanism Classification. Considering the trust mechanisms, we classify the bridges into three categories: trusted, semi-trustless, and trustless.

Trusted bridges typically have to trust a single or several validators, to perform related operations (e.g., signing and submitting transactions). The security relies on the reputations of these validators since they do not deposit collateral as penalty. For instance, the bridge Muiltichain [17] is powered by Fusion's DCRM technology [74] to ensure the security.

Semi-trustless bridges require the validators to deposit assets for punishment if anything goes wrong. Such bridges may suffer hacks from the inappropriate operations from validators. For instance, if the private keys of enough validators are compromised, the bridge will be under attack. Moreover, the deposited assets from validators are required. Taking Synapse [23] as an example, it is secured by cross-chain multi-party computation (MPC). In particular, if two-thirds of all validators sign the transactions, the consensus will be achieved and the further related transactions will be submitted to blockchains.

Trustless bridges have the highest security, equivalent to the underlying blockchains, in which anyone can become a validator. These bridges usually perform operations using smart contracts or algorithms. For example, Celer cBridge [31] depends on the decentralized State Guardian Network (SGN) [30] to monitor transactions and faithfully perform operations, such as passing the information and storing the locked tokens. In partciular, SGN is based on the Proof-of-Stake (PoS) consensus mechanism, which requires validators to stake some tokens to participate in the network.

According to Table 2, among the top 30 bridges, 20 are trusted, 2 are semi-trustless, and 8 are trustless.

4.4.2 Layer Connection Classification. Blockchains can be divided into layer 1 (L1) and layer 2 (L2). L1 is the base blockchain protocol, such as Ethereum. L2 represents a third-party solution integrated with L1, allowing for better scalability, such as Polygon [118]. Considering the connections of different blockchains, bridges can also be classified into two categories: L1 - L1 and L1 - L2.

L1 - L1 bridges connect different L1 blockchains. For example, Near Rainbow Bridge [109] connects Ethereum and Near.

L1 - L2 bridges establish a connection between L1 and various L2 blockchains, while L2 blockchains are also interconnected with each other. Anyswap [17], for instance, facilitates connectivity between 35 blockchains, including Ethereum, Polygon, and others.

Note in Table 2, among the top 30 bridges, only 4 bridges are L1 - L1 and 26 bridges are L1 - L2.

4.4.3 Bridge Functionality Classification. Based on functionality, bridges can be classified into asset-specific, chain-specific, application-specific, and generalized bridges.

Asset-specific bridges only allow a specific asset transfer. For example, xDAI bridge [144] supports *DAI* and *xDAI* transfer between Ethereum and Gnosis.

Chain-specific bridges are dedicated to certain blockchains. These bridges are usually developed by major blockchains, such as Polygon Bridge [117], which allows users to transfer assets between Ethereum and Polygon.

Application-specific bridges have limited functionality and are designed to handle only specific applications, particularly asset exchanges. For example, AnySwap [17] does not support transmitting state data or validation information.

Generalized bridges enable the transfer of various information types, such as assets, contract calls, proofs, and states. They are not restricted to particular blockchains or applications, and can instead facilitate communication between a diverse set of disconnected networks. For example, Chainlink [39] supports cross-chain services, including the Programmable Token Bridge, other bridge implementations, and even cross-chain DApps.

Note in Table 2, among the top 30 bridges, 1 is asset-specific, 12 are chain-specific, 14 are application-specific, and 3 are generalized.

5 BRIDGE ATTACK SURFACES

In this section, we illustrate 12 possible attack surfaces of cross-chain bridges, according to the two bridge models (M1: *lock-and-mint; burn-and-release* model and M2: *liquidity-pool-based*) in Sec. 4.3, as shown in Table 3. Initially, we outline the shared attack surfaces that exist in both communication models, followed by an introduction to the distinctive attack surfaces.

Models	Attack Surface	Step/Target	Example
Both	A1 (Front-end phishing)	*Server*	deceiving users
	A2 (Inaccurate deposit)	❶; ①, or ④	incorrect bridge account
	A3 (Mishandling events)	❷ or ❺; ② or ⑤	incorrect Deposit event
	A4 (Mismatched transactions)	❷ or ❺; ② or ⑤	*TxMint* not sent
	A5 (Single points of failures)	*Server*	private key leakage
	A6 (Rugpull)	Bridge	bridge running away
	A7 (Vulnerable Contracts)	$\mathbb{C}_S$ or $\mathbb{C}_D$	reentrancy
M1	A8 (Problematic mint)	❸	invalid mint permission
	A9 (Fake burn)	❹	invalid burn approval
	A10 (Incorrect release)	❺	invalid verification
	A11 (Replayed withdraw)	❻	problematic proof
M2	A12 (Inconsistent transfer)	③ or ⑥	incorrect transfer amount

Table 3: Summary of 12 possible attack surfaces according to two bridge communication models (Sec. 4.3).

5.1 Common Attack Surfaces

A1-A7 can be applied to both the lock-and-mint; burn-and-release model and liquidity-pool-based model.

A1: Front-end phishing. Regardless of the communication model used, this type of attack is possible in *Server*. If an attacker gains access to the front-end of bridges, they can deceive users into transferring their funds, resulting in the theft of user funds.

A2: Inaccurate deposit. This attack can happen in step ❶, ①, or ④ in Fig. 1. Users need to deposit tokens to the bridge account. If the bridge account address is set incorrectly, the deposited tokens will be lost. Moreover, if the related events (e.g., *Deposit*) are constructed and emitted incorrectly, users will not receive the tokens in the destination blockchain.

A3: Mishandling events. This attack can occur in step ❷, ❺, ②, or ⑤ in Fig. 1. *Server* will fetch and parse the events from the transactions. If these events are handled incorrectly, bridges might perform inappropriate actions, such as sending more money to users than desired. For example, if a *Deposit* event recording a 10

ETH transfer in the source blockchain is mistakenly treated as a 100 *ETH* transfer, bridges will send 100 wrapped *ETH* to the user in the destination blockchain.

A4: Mismatched transactions. This attack can occur in step ❷, ❺, ②, or ⑤ in Fig. 1. Cross-chain bridges are heavily relying on the off-chain bridge server (*Server*), for monitoring, parsing information, and submitting transactions. If the related transactions (e.g., *TxMint* to mint wrapped tokens for users) are not sent at all, user will not receive their tokens. Moreover, if the transactions are constructed incorrectly (e.g., the receiver address is given with a wrong value), attacks can also happen.

A5: Single points of failures. This attack can occur in *Server* in Fig. 1. Some bridges rely on external validators that can be compromised or engage in malicious behaviors, as follows.

- Compromised bridge validators. If the private keys of enough validators are compromised, attackers can take over the bridges to drain all the assets or mint tokens without limitations.
- Malicious validators. Validators might be dishonest if the values in the bridges are far more than the values validators stake, as punishments for behaving maliciously.

A6: Rugpull. This attack can occur if the bridge itself is the attacker. Rugpulls are scams in which the token founders suddenly vanish and drain all the assets. For example, users of Luna Yield, a farming project on the Solana blockchain, suffered from the rugpull attack on Aug-19-2021, in which the creators suddenly deleted the website and disappeared [47]. Additionally, the investors of an NFT game DApp, BlockVerse, withdrew all the money and disappeared on Jan-25-2022 [75]. Users should watch out for the possibility of rugpulls in cross-chain bridges, especially on newly created bridge projects.

A7: Vulnerable bridge smart contracts. This attack can happen in $\mathbb{C}_S$ or $\mathbb{C}_D$ in Fig. 1. Common vulnerabilities in smart contracts (e.g., access control) are possible in bridge smart contracts. We illustrate some potential attacks, taking the liquidity pool smart contract code as an example (see Appendix D), including incorrect initialization, inappropriate function permission, unchecked balance, miscalculated token price, and inconsistent event. Existing vulnerabilities in DeFi can also exist in cross-chain bridges. Current cross-chain bridges only support simple financial services, which are mainly token transfers. However, we believe the attacking surface is much larger than the existing bridge attacks.

- Price manipulation attack. If the bridge relies on unstable price oracles that could be maliciously manipulated by attackers, the bridges will suffer from price manipulation attacks.
- Reentrancy attack. Reentrancy attack happens when the victim contract calls a malicious contract who calls back to the original function in the victim contract, to steal money. In both functions *addLiquidity* and *removeLiquidity*, the reentrancy attacks occur.

5.2 Unique Attack Surfaces

5.2.1 Lock-and-Mint; Burn-and-Release Model. A8-A11 are only applicable to the lock-and-mint; burn-and-release model.

A8: Problematic mint. This attack can happen in step ❸ in Fig. 1. Once the tokens are locked in the source blockchain, the bridges will mint wrapped tokens in the destination blockchain. However,

if the proof is not validated appropriately, attackers can mint more tokens than desired.

A9: Fake burn. This attack can happen in step ❹ in Fig. 1. To get the tokens back in the source blockchain, users should burn the wrapped tokens in the destination blockchain. If there are some permission check issues that allow users to pass the burn, attacks can happen due to fake burns in which attackers burn nothing but get burn proofs. Moreover, if the *Burn* event is not emitted correctly, there will be problems.

A10: Incorrect release. This attack can occur in step ❻ in Fig. 1. The related data need to be verified before creating the release proof. If the verification process is not handled correctly, the release proofs might be forged by attackers.

A11: Replayed/Unlimited withdraw. This attack can happen in step ❼ in Fig. 1. The release proof needs validation before allowing users to withdraw tokens. However, if the proof can be used multiple times or forged, users can steal tokens until the assets are drained.

5.2.2 Liquidity-Pool-Based Model. A12 is only applicable to the liquidity-pool-based model.

A12: Inconsistent cross-chain transfer. This attack can occur in step ③ or ⑥ in Fig. 1. Assume that a user $\mathbb{A}_S$ transfers *ETH* tokens from Ethereum (source blockchain) to BNB (destination blockchain). It is possible to suffer from the inconsistencies of cross-chain token transfer, in which the amount of transferred token in *TxDest* or *TxDest'* is incorrect (e.g., less than expected).

6 A TAXONOMY OF CROSS-CHAIN BRIDGE ATTACKS AND VULNERABILITIES

In this section, we systematically categorize reported cross-chain bridge attacks and vulnerabilities into four categories, including Permission Issue (PI), Logic Issue (LI), Event Issue (EI), and Front-end Issue (FI), and summarize them in Table 4. We also introduce the attacks caused by front-end phishing and replay attacks. In particular, we summarize the knowledge of cross-chain bridge attacks from reliable resources, including official bridge documentations (e.g., Wormhole [24]), attack reports from blockchain companies (e.g., Certik [35]), and blogs or posts about cross-chain bridges from technical organizations (e.g., HALBORN [87]).

6.1 Permission Issue

Permission issues are quite common in both regular smart contracts and cross-chain bridges; 20 out of the total 35 attacks fall into this category. We divide this category into five types, including unchecked intermediary permission, misused proof permission, problematic approval permission, invalid signature permission, and leaked key permission.

6.1.1 V1. Unchecked Intermediary Permission. This type of attacks can happen in either $\mathbb{C}_S$ or $\mathbb{C}_D$, as shown in Fig. 1. A smart contract can be assigned a set of managers for permission check when some fundamental changes are made, such as owner modification or money transfer. If the functions in the smart contract are restricted with these permission checks, they can only be called when the caller belongs to one of the managers. However, if the caller utilizes the manager as the intermediate contract to call the permissioned

Type	Sub-type	Attack Surface	Victim Bridge	Attack Date	Loss ($million)	Victim Blockchains	Defense
PI	V1. Unchecked intermediary permission	$A7(\mathbb{C}_S \text{ or } \mathbb{C}_D)$	Poly Network	Aug-10-2021	600	Ethereum	Au, BB
	V2. Misused proof permission	$A11(\mathbb{C}_S)$ $A10(\mathbb{C}_S \text{ or } \mathbb{C}_D)$	Plasma Binance Bridge	Oct-05-2021 Oct-06-2022	850 566	Polygon Bnb	BB BB, HF
	V3. Problematic approval permission	$A7(\mathbb{C}_S \text{ or } \mathbb{C}_D)$	Multichain LI.FI Rubic Socket Protocol	Jan-18-2022 May-20-2022 Dec-25-2022 Jan-16-2024	3 0.6 1.4 3.3	Ethereum Ethereum Ethereum Ethereum	UVC, RIA UVC UVC, Au UVC, Au
	V4. Invalid signature permission	$A8(\mathbb{C}_D)$	ChainSwap Wormhole Multichain	Jul-10-2021 Feb-02-2022 Feb-15-2023	8 320 0.1	Ethereum Solana, Ethereum Ethereum	Au, BB BB UVC
	V5. Leaked key permission	$A5(Server)$	Ronin Network WonderHero Horizon Bridge QAN Platform Bridge Rubic pNetwork Poly Network Multichain Heco Bridge Orbit Bridge	Mar-29-2022 Apr-07-2022 Jun-24-2022 Oct-11-2022 Nov-02-2022 Nov-04-2022 Jul-01-2023 Jul-06-2023 Nov-22-2023 Dec-31-2023	625 0.3 100 2 1.2 108 10.2 126 86.28 82	Ethereum Bnb Ethereum, Bnb Ethereum, Bnb Ethereum Bnb Ethereum Ethereum et al. Ethereum Ethereum	Au, IVD Au, UVC, BB Au Au Au UVC UVC N/A N/A N/A
LI	V6. Incorrect balance logic	$A7(\mathbb{C}_S \text{ or } \mathbb{C}_D)$	Polkabridge Poolz Finance	Nov-22-2021 Mar-15-2023	0.6 0.6	Ethereum Ethereum, Bnb, Polygon	UVC Au
	V7. Inaccurate initialization logic	$A7(\mathbb{C}_S \text{ or } \mathbb{C}_D)$	Nomad Allbridge Hypr OP Stack Bridge	Aug-02-2022 Apr-01-2023 Dec-13-2023	190 0.57 0.22	Ethereum, Moonbeam Bnb Ethereum	BB UVC N/A
EI	V8. Incorrect event emission	$A2(\mathbb{C}_S)$	Qubit Meter.io Omni	Jan-27-2022 Feb-05-2022 Sep-16-2022	80 4.2 4.2	Ethereum, Bnb Ethereum, Bnb, Moonriver Ethereum PoW	BB BB BB
	V9. Fake event emission	$A2(\mathbb{C}_S)$ and $A3(Server)$	THORChain THORChain THORChain pNetwork Cennznet	Jun-29-2021 Jul-16-2021 Jul-23-2021 Sep-19-2021 May-08-2022	0.35 8 8 13 0.4	THORChain THORChain THORChain Bnb Ethereum	Au Au Au UVC, BB UVC, EOS
FI	V10. Front-end phishing	$A1(Server)$	EVODeFi Celer cBridge	Mar-22-2022 Aug-17-2022	0.3 0.2	Bnb Ethereum	BB BB

Table 4: A taxonomy of cross-chain bridge attacks and vulnerabilities. Type: PI: Permission Issue; LI: Logic Issue; EI: Event Issue; OI: Other Issue; FI: Front-end Issue. Attack Surface: A_i is the index of possible attacks defined in Sec. 5; $\mathbb{C}_S$, $\mathbb{C}_D$, and *Server* are involved identities in Fig. 1. Defense (presented in Sec. 7): Au: Auditing; BB: Bug Bounty; HF: Hard Fork of Blockchain; UVC: Upgrading Vulnerable Contract; RIA: Revoking Infinite Approval; IVD: Increasing Validator Decentralization; EOS: Examining Off-chain Server.

functions, the permission check will be bypassed and the attack will happen. Moreover, we illustrate this type with 1 example of Solidity code here and 1 real-world attack in Appendix E.1.1.

Example. We illustrate the pattern of this type via the example in Fig. 2, in which the contract B is the manager of contract A. We explain some primitives needed in this example, as follows.

- *address* is a type within Solidity smart contracts, representing an Ethereum account.
- *bytes4* and *bytes* are both arrays of bytes, and *bytes4* is a fixed-length array of four bytes.
- *modifier* restricts the behaviors of the functions, and can be considered as a compile-time code roll-up.
- *function* is a combination of lines of code. In particular, there are four types of functions: external, internal, public, and private. External functions can be called from outside smart contracts and internal functions can only be called internally within the

```solidity
1  contract A {
2    address owner;
3    address manager;
4
5    modifier onlyManager {
6      require(msg.sender == manager);
7    }
8
9    function changeOwner(address account) public
          onlyManager {
10     owner = account;
11   }
12 }
13
14 contract B {
15   function call(address callee, bytes4 funcSig,  bytes
          parameters) public {
16     callee.call(funcSig, parameters);
17   }
18 }
```

Figure 2: Unchecked intermediary permission.

current contract. Moreover, public functions can be called without restrictions and private functions are only visible in the contract they are defined.

- *function signature* represents the unique identifier for a specific function within a smart contract. It is generated by hashing the function's name and parameter types.
- *call* is a default low-level function for interacting with public and external smart contract functions.

In this example, the function *changeOwner()* is to modify the owner of the smart contract A to be the parameter *account*. However, not anyone is allowed to call this function, unless the caller *msg.sender* is the manager of the contract, as defined by the modifier *onlyManager*. In smart contract B, the function *call()* interacts with any callable function in smart contract *callee*, as long as the correct function signature *funcSig*, parameters *parameters*, and the contract address *callee* are provided. If the data of function *call()* in contract B is constructed correctly, anyone can bypass the permission check in contract A, to call *changeOwner()* and transfer the ownership to its account, since contract B is the manager of contract A.

6.1.2 V2. Misused Proof Permission. This attack type can occur in step ❼ inside $\mathbb{C}_S$ or $\mathbb{C}_D$, as shown in Fig. 1. In cross-chain bridges, unique proofs should be provided to prove the deposit or burn actions from users, for future minting or releasing the related tokens to users. If the smart contract is not designed carefully to deal with proofs, the previous valid proof could be replayed multiple times. As a result, the money will be stolen. We also illustrate this type with 1 code example here and 2 real-world attacks in Appendix E.1.2.

```
1  contract A {
2    function verifyProof(bytes data) external {
3      if isValid(data) return throw;
4      receiver, tokenAddr, tokenAmount = decode(data);
5      tokenAddr.transfer(receiver, tokenAmount);
6    }
7  }
```

Figure 3: Misused proof permission.

Example. We illustrate this type with one example in Fig. 3. In particular, if the proof *data* is verified to be valid via the function *isValid()*, the money will be sent back to the user according to the decoded parameters via the function *decode()*. However, the *data* could be constructed maliciously multiple times from one valid proof related data, if the function *isValid()* does not check the uniqueness of the value *data* correctly.

6.1.3 V3. Problematic Approval Permission. This type of attacks can happen in $\mathbb{C}_S$ or $\mathbb{C}_D$, as shown in Fig. 1. To save on transaction gas fees, users must give their permissions to DApps in advance, allowing bridges to transfer their tokens. Such behaviors are usually implemented by the standard ERC20 function *approval()*, which is insecure when the DApp smart contract cannot handle the permission check correctly. Moreover, we illustrate this type with 1 code example here and 4 real-world attacks in Appendix E.1.3.

Example. We illustrate this type with the example from Fig. 4, in which the function *transferWithPermit()* is not handled correctly. Here, we assume some users have given infinite approvals to the

```
1  contract A {
2    function transferWithPermit(address from, address to
          , address token, uint amount, uint deadline,
          uint8 v, bytes32 r, bytes32 s) external {
3      address _underlying = DERC20(token).underlying();
4      IERC20(_underlying).permit(from, address(this),
            amount, deadline, v, r, s);
5      _underlying.call(abi.encodeWithSelector(0x23b872dd
            , from, token, amount));
6    }
7  }
```

Figure 4: Problematic approval permission.

DApp in advance. The vulnerability can be exploited via the following steps. (1) Calling to an unverified smart contract. In line 3, the attacker can set the address *token* to a malicious contract, which is usually unknown and unverified. Then the attacker can control the returned value, which is not checked by contract A. (2) Entering a fallback function. In line 4, the function *permit()* intends to ensure the ability of the current contract with the supplied transaction (v,r,s), which allows transferring users' tokens. However, *permit()* does not exist in the attacker-controlled malicious contract, and the fallback function will be called. If the fallback function is empty and the transaction continues to execute, permit checking will be bypassed. (3) Stealing money. Tokens will be transferred from the victim user to the attacker in this step. In line 5, the transfer is made via the low-level call function by constructing the appropriate malicious data, specifying the standard function signature (0x23b872dd) and parameters of *transferFrom()*.

6.1.4 V4. Invalid Signature Permission. This type of attack can occur at step ❸ inside $\mathbb{C}_D$, as shown in Fig. 1. In cross-chain bridges, signatures of validators are required to sign transactions, in order to transfer tokens back to users. In particular, the signatures should be checked. If signature verification is not properly processed, attacks can occur. Moreover, we illustrate this type with 1 code example here and 3 real-world attacks in Appendix E.1.4.

```
1  contract A {
2    function mintToken(bytes[] signatures, address to,
          address token, uint amount) public {
3      bool ret = verify_signatures(signatures);
4      require(ret, "Signatures are not valid!");
5      mint(to, token, amount);
6    }
7  }
```

Figure 5: Invalid signature permission.

Example. From the example in Fig. 5, *verify_signatures()* function in line 3 incorrectly checks signatures, and the permission check is bypassed by the attacker. Then, the attacker can mint tokens without any cost since he/she has the permission due to the forged but passed signatures.

6.1.5 V5. Leaked Key Permission. This type of attack can occur in *Server*, as shown in Fig. 1. Attackers usually compromise the private key of admin bridge accounts or enough private keys of bridge

validators. We demonstrate 1 real-world attack resulting from this vulnerability, while the remaining 9 are detailed in Appendix E.1.5.

Ronin Network attack. Ronin Network [111] suffered from the attack on Mar-29-2022, which was caused by the compromised private keys [84]. In particular, the attacker accessed four validators controlled by Sky Mavis [100] and the fifth validator from Axie DAO [54]. Due to the 5-of-9 multi-signature mechanism, the attack succeeded and caused a $625 million loss.

6.2 Logic Issue

Logic issues are caused by the logical mistakes in smart contracts. We classify this type into incorrect balance logic and inaccurate initialization logic; 5 of the total 35 attacks belong to this category.

6.2.1 V6. Incorrect Balance Logic. This type of attack can occur in either $\mathbb{C}_S$ or $\mathbb{C}_D$, as shown in Fig. 1. A liquidity pool is a collection of locked tokens. Assume there are two types of tokens in a liquidity pool, the quantities of those assets are called reserves. Generally, $x \times y = c$ (x and y are the reserves of the two types of tokens; c is a constant number). However, the value paid by the user should be slightly higher than the expected amount every time the tokens are deposited or withdrawn to pay the fee. Thus, to ensure the correctness of such mechanisms, the balances of these two tokens should be no less than the initial constant. Otherwise, any user can steal money from the pool. Moreover, we illustrate this type with 1 code example here and 2 real-world attacks in Appendix E.2.1.

```
1  contract A {
2    function balanceCheck() public {
3      _reserve0, _reserve1 = getReserves();
4      uint balance0Adjusted = balance0.mul(1000).sub(
         amount0In.mul(3));
5      uint balance1Adjusted = balance1.mul(1000).sub(
         amount1In.mul(3));
6      require(balance0Adjusted.mul(balance1Adjusted) >=
         uint(_reserve0).mul(_reserve1).mul(1000**2),
         'Balance check failed!');
7    }
8  }
```

Figure 6: Incorrect balance logic.

Example. An example is shown in Fig. 6, which is the representative source code of the balance check for a liquidity pool. The vulnerability exists if line 6 is removed.

6.2.2 V7. Inaccurate Initialization Logic. This type of attack can occur in $\mathbb{C}_S$ or $\mathbb{C}_D$, as shown in Fig. 1. The initialization process sets some privileged addresses as managers or owners of the smart contracts. Similarly, some non-trivial variables (e.g., verification variable) will be assigned with initial values. If these fundamental variables are not defined appropriately, bridges will be attacked. Moreover, we illustrate this type with 1 code example here and 3 real-world attacks in Appendix E.2.2.

Example. Fig. 7 gives an example in which line 11 sets the owner of the contract as an incorrect address, or line 13 defines the key variable with incorrect value.

```
1  contract A {
2    address initializer_addr;
3    address owner;
4    bool canTransfer;
5
6    modifier initializer {
7      msg.sender = initializer_addr;
8    }
9
10   function initialize() public initializer {
11     owner = 0x;
12     // or
13     canTransfer = true;
14   }
15 }
```

Figure 7: Inaccurate initialization logic.

6.3 Event Issue

To gather the necessary information from the transactions, bridges usually retreive emitted events from smart contracts and interpret these events to perform future steps. For example, if some tokens are sent from users to bridges, the related *Deposit* events will be emitted, serving as the deposit proof. However, events can be problematic and money will be stolen from bridges. In particular, this type includes incorrect event emission and fake event emission. Moreover, 8 of the total 35 attacks belong to this category.

6.3.1 V8. Incorrect Event Emission. This type of attacks can happen in step ❶ inside $\mathbb{C}_S$, as shown in Fig. 1. These two functions *deposit()* and *depositETH()* emit the same *Deposit* event, representing that users deposit *ETH* tokens to the bridge. Then, *BOS* will process the events, minting *ETH* tokens to users no matter what tokens are deposited. Attackers could utilize this vulnerability to deposit less valuable tokens (e.g., USDT [90]) and earn *ETH* tokens back. Moreover, we illustrate this type with 1 code example here and 3 real-world attacks in Appendix E.3.1.

```
1  contract A {
2    function deposit(address from, address to, address
         token, uint amount) external payable {
3      token.transferFrom(from, to, amount);
4      emit Deposit(from, to, amount);
5    }
6
7    function depositETH() external payable {
8      transferETH(msg.sender,address(this),msg.value);
9      emit Deposit(msg.sender,address(this),msg.value);
10   }
11 }
```

Figure 8: Incorrect event emission.

Example. In the example shown in Fig. 8, the attacker transferred some less valuable tokens (e.g., USDT [90]) via the *deposit()* function. The attacker can get wrapped *ETH* tokens. Moreover, the attacker can even construct malicious data to transfer nothing, but obtain a *Deposit* event. For instance, the token address can be some invalid addresses and the transaction will not fail, since the return value of function *transferFrom()* is not checked. Then, the attacker can use the event to steal tokens.

6.3.2 V9. Fake Event Emission. This type of attack can only occur when both $\mathbb{C}_S$ and *Server* have bugs in the first two steps (❶ and ❷), as shown in Fig. 1. In particular, the *Deposit* event is being emitted from the attacker controlled smart contract, not from the bridge-related contract. However, the bridge off-chain server is not aware of the problem and mistakenly accepts fake events. Thus, the attackers can deposit nothing, fake the event, and steal tokens. Moreover, we illustrate this type with 1 code example here and 5 real-world attacks in Appendix E.3.2.

```
1   contract Bridge {
2     function withdraw(address recipient, uint256 amount)
            external payable {
3       bool sent, _ = recipient.call{value:amount}("");
4       require(sent, "Sent failed!");
5     }
6   }
7
8   contract Attacker {
9     function() external payable {
10      emit Deposit(msg.sender,address(this),msg.value);
11    }
12  }
```

Figure 9: Fake event emission.

Example. An example is shown in Fig. 9. If the attacker withdraws money from the contract Bridge, the fallback function in the appointed smart contract *recipient* will be called. Then the attacker can fake the *Deposit* event in the fallback function within the Attacker contract, to steal tokens.

6.4 Front-end Issue

Some bridges suffer from front-end hijackings, in which websites or smart contracts are redirected to fake websites or malicious contracts. Specifically, 2 of the total 35 attacks belong to this category.

6.4.1 V10. Front-end Phishing. Front-end phishing refers to a type of cyberattack where the attacker manipulates the user interface or the frontend of the bridge to deceive users into revealing sensitive information or taking malicious actions unwittingly. We demonstrate 1 real-world attack resulting from this vulnerability, while the remaining 1 is detailed in Appendix E.4.1.

EVODeFi attack. On Mar-22-2022, an attacker gained unauthorized access to the EVODeFi domain by exploiting a bug in Vercel, which hosts the bridge's front-end [7]. The attacker was able to bypass the domain transfer confirmation process, which should have been required from the legitimate owner, EVODeFi. As a result of the security breach, the attacker was able to steal $320,000.

7 DEFENSES AND RECOMMENDATIONS

7.1 Existing Defenses

We summarize the existing defenses from real-world attacks, categorize the defenses into four types: general defenses and defenses of three types of attacks, including PI, LI, EI, FI (defined in Sec. 6), and illustrate the defenses with examples.

7.1.1 General Defenses. The general defenses include auditing, bug bounty, and upgrading vulnerable contracts, which can be applied to any attack.

Auditing (Au). DApps typically are audited by trusted third parties (e.g., Certik [35]) before being deployed. The more comprehensively a DApp is audited, the more secure the project will be.

Bug Bounty (BB). Bug bounty programs can not only motivate white hats to disclose vulnerabilities before being attacked, but also serve as rewards for attackers to send the stolen funds back. For example, Wormhole offered a $10m bug bounty reward in exchange for lost funds from attackers [61].

Hard Fork (HF). To mitigate severe attacks (e.g., DAO attack on Ethereum), the blockchain can be upgraded by a hard fork that requires all nodes and users to update.

Upgrading Vulnerable Contracts (UVC). DApps can fix smart contract bugs by upgrading contracts. For example, Multichain removed the vulnerabilities in the token liquidity pool and upgraded new contracts [106].

7.1.2 Defenses of PI Attacks. Permission issue (PI) attacks are caused by incorrect permission checks or problematic protocols (e.g., multiple signatures).

Revoking Infinite Approval (RIA). Developers should avoid asking for unlimited user approvals, as it can be dangerous. Instead, they should request only the necessary approvals. For example, Multichain advised its users to revoke approvals on the same day of the attack (Jan-18-2022) [137].

Increasing Validator Decentralization (IVD). To mitigate the vulnerability of key permissions that have leaked, bridges can improve their security by increasing the number of validators or improving their decentralization. For example, Ronin Network increased the number of validators to 21 after being attacked [121].

7.1.3 Defenses of LI Attacks. Smart contract vulnerabilities stemming from logical errors can lead to logic issue attacks. For example, PolkaBridge [116] attack happened due to the incorrect calculation of the token price. Bridges typically employ standard defensive measures to combat such attacks.

7.1.4 Defenses of EI Attacks. Bridges off-chain server monitors on-chain activities by recording events and performing related actions. If such events are incorrect, event issue (EI) attacks can occur.

Examining Off-chain Server (EOS). Bridges extract information from on-chain transactions, store them in off-chain datasets, and perform operations such as submitting transactions to transfer tokens back to users. It is common to have vulnerabilities on the off-chain server since they are not public and may not be audited by others. Thus, it is non-trivial to examine the implementation of off-chain server closely and cooperate with the trusted third parities for auditing, to avoid risks.

7.1.5 Defenses of FI Attacks. Front-end issue (FI) attacks usually arise when malevolent websites redirect user transfers to attackers. Bridges employ standard defensive strategies, such as bug bounties, to prevent or mitigate such attacks.

7.2 Recommendations to Developers

Based on our study, this section offers recommendations for developers building on-chain backends comprising smart contracts (for blockchain transactions) and off-chain servers (for off-chain operations).

7.2.1 On-chain Backend. For on-chain backend security, developers should pay particular attention to smart contracts.

Ensuring proper permission handling code. Developers must thoroughly review all possible execution paths regarding permission checks. Additionally, they can use tools such as Slither [70] to assess the correctness of the code.

Avoiding unverified smart contract calls. To prevent attacks where attackers deploy malicious smart contracts and hijack the control flow to their code, developers should avoid calling unverified smart contracts, particularly user-specified contracts, unless absolutely necessary.

Checking return values of external calls. Developers must be mindful of return values of functions, especially external function calls, as attackers can leverage inconsistencies caused by unchecked return values to launch exploits.

Avoiding directly copying code. While copying smart contract code from popular projects may save time, it can also result in code with bugs or vulnerabilities. Therefore, developers should closely examine the code and understand its logic.

Setting longer withdraw latency. Increasing the latency on token withdrawal can provide developers with additional time to mitigate risks. However, it may also affect bridge performance, and developers must weigh the trade-off between security and speed.

Providing insurance. Bridges can establish insurance for assets to compensate users in case of an attack, similar to real-world financial services. While precautions can reduce attack risks, insurance can mitigate users' financial losses in the event of an attack.

7.2.2 Off-chain Server. For off-chain server security, developers can monitor bridge server activities.

Money monitoring. DApps must vigilantly monitor their transactions to prevent losses from copycat or 1-day attacks. For instance, Ronin [84] was aware of the attack several days after it occurred, avoiding further attacks. DApps must keep a close eye on transactions with unusual behaviors, such as large transfers of tokens. It is recommended to receive early alerts for such transactions.

8 OPEN PROBLEMS AND FUTURE DIRECTIONS

8.1 Security Property Concerns

Q1: Future of bridges: will it be multi-chain? With so many severe attacks on bridges, it raises doubts on the security of cross-chain bridges. There are suggestions of replacing existing cross-chain bridges with the multi-chain solutions, which are blockchains of blockchains (e.g., Cosmos). According to Vitalik Buterin, the founder of Ethereum, multi-chain is likely to dominate the future considering it is more secure [138]. To this end, a systematic study measuring and comparing the cross-chain bridges and multi-chain protocols should be performed.

Q2: Trust models: what is the real security level of existing bridges? According to bridge documentations with trusted or semi-trustless models (e.g., Horizon Bridge [89]), bridges are always secured with the guardian from trusted third parties. It is unclear what the *true* security level of these bridges is. For example, bridges with individual validators are more secure than bridges with validators controlled by a single party. Moreover, trustless bridges claim to have the same security as the underlying public blockchain. We should be aware of the *true* security level. To answer this question, systematic work studying the security of real-world cross-chain bridges with different trust levels is needed.

8.2 Bridge Related Issues

Q3: Bridge usages: any financial crime issues? Cross-chain bridges are now often used for cross-chain transfers and swaps. It is not clear whether these bridges are used for any illegal purposes, such as money laundering. To avoid being traced and deanonymized, attackers usually mix their stolen assets; the assets will be exchanged between different users, making it difficult to trace their original accounts. In particular, many attacks move the stolen funds to bridges in the end. For example, attackers in Hamster rugpull profited 1,730 *BNB* tokens along with the Cheese rugpull [44], in which 900 *BNB* tokens were sent to the Wormhole bridge. After cross-chain transfer via bridges, the behaviors of attackers are hard to analyze and the assets are difficult to trace. With the recent sanction of Tornado Cash, which is a popular token mixer [45], we believe that the use of bridges as cash mixers will become more and more popular. Thus, work to develop systematic frameworks to analyze illegal activities is definitely in need.

Q4: Inconsistencies: are the bridges as efficient and convenient as they claim to be? Except for the security property, bridges describe their costs (including both fee and time cost) in their official documentations. Are these really the same as what they claim to be? For example, if the bridges claim that the cross-chain transactions take no more than 20 minutes, can all transactions be completed within that timeframe? There might be some inconsistencies between the real-world situations and the ones bridges officially claim. To understand the bridges deeper and identify those inconsistencies, bridge-related transactions should be studied.

8.3 Attack Detection and Prevention

Q5: Attack detection: How to detect attacks? Many works [73, 77, 78, 94, 129, 141, 148] focus on detecting blockchain attacks from the perspectives of smart contracts and transactions. However, it is yet unclear how these bridge-related attacks can be detected. So far, there has been only one work [147] detecting cross-chain bridge attacks; Zhang *et al.* propose a tool to discover security violations, detect bridge attacks from transactions, and evaluate the tool on four cross-chain bridges. However, only three attacks presented in Sec. 6 can be detected by the proposed tool. With so many severe attacks, we believe that there are urgent needs to propose a framework that can detect bridge attacks from transactions or smart contracts.

Q6: Attack prevention: how to prevent attacks? Most of existing bridge attacks are due to contract bugs, and there will be more

attacks in the future. Detecting these attacks in a post-mortem manner is not enough, since it cannot prevent attacks from happening. Solutions to prevent attacks or principled methods to reduce attack surfaces are also important to protect the cross-chain bridges.

9 CONCLUSION

In this paper, we provide the *first* systematic study to understand the security issues of cross-chain bridges. We first present the models of existing cross-chain bridges, then identify potential attack surfaces. We further analyze all the 35 bridge attacks over the period of April 2021 to April 2024, study the fundamental vulnerabilities that lead to the attacks, and classify them into three categories based on the vulnerabilities exploited by the attacker. We summarize existing defenses and provide recommendations for developers to build a secure bridge ecosystem. We also discuss open problems and future research directions. Our work provides a holistic view of security issues of cross-chain bridges, and the insights are valuable for facilitating future research in the field.

ACKNOWLEDGMENTS

REFERENCES

[1] Aave. 2023. Aave Document Hub. https://docs.aave.com/hub/.
[2] Aave. 2023. Aave Governance Cross-Chain Bridges. https://github.com/aave/governance-crosschain-bridges.
[3] Allbridge. 2023. Allbridge Docs. https://docs.allbridge.io/allbridge-overview/networks-and-tokens.
[4] André Augusto, Rafael Belchior, Miguel Correia, André Vasconcelos, Luyao Zhang, and Thomas Hardjono. 2023. Sok: Security and privacy of blockchain interoperability. *Authorea Preprints* (2023).
[5] B2Broker. 2022. How Do Liquidity Pools Work in Crypto? https://b2broker.com/news/how-do-liquidity-pools-work-in-crypto/.
[6] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. 2014. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains* 72 (2014), 201–224.
[7] BanklessTimes. 2022. Crypto platform EVODeFi loses $320K in hack. https://www.banklesstimes.com/news/2022/03/22/crypto-platform-evodefi-loses-dollar320k-in-hack/.
[8] Rafael Belchior, Peter Somogyvari, Jonas Pfannschmidt, André Vasconcelos, and Miguel Correia. 2023. Hephaestus: Modeling, Analysis, and Performance Evaluation of Cross-Chain Transactions. *IEEE Transactions on Reliability* (2023).
[9] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. 2021. A survey on blockchain interoperability: Past, present, and future trends. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–41.
[10] Bitcoin.com. 2022. Binance-Backed Blockchain Completes Hard Fork to Mitigate Future Cross-Chain Bridge Hacks. https://news.bitcoin.com/binance-backed-blockchain-completes-hard-fork-to-mitigate-future-cross-chain-bridge-hacks/.
[11] The Block. 2022. Hacker drains $2 million from QAN platform bridge, token slumps 94%. https://www.theblock.co/post/176118/hacker-drains-1-million-from-qanplatform-bridge-token-slumps-94.
[12] Blockchainhub. 2023. Decentralized Applications – dApps. https://blockchainhub.net/decentralized-applications-dapps/.
[13] LI.FI Blog. 2022. LI.FI Smart Contract Vulnerability Post Mortem. https://blog.li.fi/20th-march-the-exploit-e9e1c5c03eb9.
[14] BNB. 2023. Binance Smart Chain. https://github.com/bnb-chain/whitepaper/blob/master/WHITEPAPER.md.
[15] BoringDao. 2023. BoringDao Docs. https://docs.boringdao.com/.
[16] Michael Borkowski, Daniel McDonald, Christoph Ritzer, and Stefan Schulte. 2018. Towards atomic cross-chain token transfers: State of the art and open questions within tast. *Distributed Systems Group TU Wien (Technische Universit at Wien), Report* 8 (2018).
[17] Anyswap Bridge. 2023. Anyswap DEX User Guide. https://anyswap-faq.readthedocs.io/en/latest/index.html.
[18] Cross-Chain Bridge. 2023. Cross-Chain Bridge Docs - Conneted Networks. https://docs.crosschainbridge.org/connected-networks.
[19] Hyphen Bridge. 2023. Hyphen Bridge. https://hyphen.biconomy.io/bridge.
[20] Orbit Bridge. 2023. Orbit Bridge FAQs. https://bridge.orbitchain.io/questions.
[21] Plasma Bridge. 2023. Plasma Bridge. https://docs.polygon.technology/docs/develop/ethereum-polygon/plasma/getting-started/.
[22] Portal Token Bridge. 2023. Portal Token Bridge - Introduction. https://docs.wormhole.com/wormhole/.
[23] Synapse Bridge. 2023. Welcome to Synapse. https://docs.synapseprotocol.com.
[24] Wormhole Bridge. 2023. Wormhole Bridge. https://wormholebridge.com/.
[25] Vitalik Buterin. 2016. Chain interoperability. *R3 Research Paper* 9 (2016).
[26] Satellite by Alexar. 2023. Satellite by Alexar Docs. https://docs.axelar.dev/resources/mainnet.
[27] Horizon by Harmony. 2023. Harmony ONE-ETH Bridge. https://bridge.harmony.one/busd.
[28] Solana Cashio. 2023. Solana Cashio. https://www.solanacash.io.
[29] Celer cBridge. 2023. Celer cBridge. https://cbridge.celer.network/1/56/USDC.
[30] Celer cBridge. 2023. State Guardian Network. https://cbridge-docs.celer.network/introduction/state-guardian-network.
[31] Celer cBridge. 2023. Welcome to cBridge. https://cbridge-docs.celer.network.
[32] Cennznet. 2023. BLOCKCHAIN FOR THE OPEN METAVERSE. https://cennz.net.
[33] CertiK. 2021. PolyNetwork Attack Analysis. https://certik.medium.com/polynetwork-hack-analysis-a86513f2a730.
[34] Certik. 2022. Nomad Bridge Exploit Incident Analysis. https://www.certik.com/resources/blog/28fMavD63CpZJOKOjb9DX3-nomad-bridge-exploit-incident-analysis.
[35] Certik. 2023. CertiK Blockchain Security Leaderboard. https://www.certik.com/.
[36] Cross chain Loans. 2022. Cross-chain Loans - Decentralized lending marketplace across blockchains. https://crosschain.loans/.
[37] Chainalysis. 2022. Lessons from the Wormhole Exploit: Smart Contract Vulnerabilities Introduce Risk; Blockchains' Transparency Makes It Hard for Bad Actors to Cash Out. https://blog.chainalysis.com/reports/wormhole-hack-february-2022/.
[38] ChainBridge. 2023. ChainBridge Overview. https://chainbridge.chainsafe.io.
[39] Chainlink. 2023. Chainlink: Blockchain Oracles for Hybrid Smart Contracts. https://chain.link.
[40] Chainlink. 2023. Chainlink: Blockchain Oracles for Hybrid Smart Contracts. https://chain.link/cross-chain.
[41] ChainPort. 2023. ChainPort Docs. https://docs.chainport.io/chainport/chainport-features.
[42] Chainspot. 2023. Soy Bridge. https://chainspot.io/bridge/soy-bridge.
[43] ChainSwap. 2023. The cross-chain hub for all ecosystems. https://chainswap.com.
[44] CoinCodeCap. 2022. Hamster Coin Rugged: Reportedly 1,730 BNB Lost. https://coincodecap.com/hamster-coin-rugged-reportedly-1730-bnb-lost.
[45] Coindesk. 2022. US Treasury's Tornado Cash Sanctions Are 'Unprecedented,' Warns Congressman. https://www.coindesk.com/policy/2022/08/18/treasury-tornado-cash-sanctions-are-unprecedented-warns-us-congressman/.
[46] Coindesk. 2023. What Are Wrapped Tokens? https://www.coindesk.com/learn/what-are-wrapped-tokens/.
[47] CoinGeek. 2021. Solana sees first rug pull: Luna Yield disappears with $6.7M in digital currency. https://coingeek.com/solana-sees-first-rug-pull-luna-yield-disappears-with-6-7m-in-digital-currency/.
[48] CoinYuppie. 2022. In-depth analysis of four types of cross-chain bridges and their risks. https://coinyuppie.com/in-depth-analysis-of-four-types-of-cross-chain-bridges-and-their-risks/.
[49] Connext. 2021. nxtp: A simpler xchain protocol. https://blog.connext.network/nxtp-a-simpler-xchain-protocol-88760697ea04.
[50] Blockchain Council. 2023. Blockchain & Role of P2P Network. https://www.blockchain-council.org/blockchain/blockchain-role-of-p2p-network/.
[51] CryptoBriefing. 2022. BNB Chain's $566M Hack: Binance Network's Major Bridge Attack Unpacked. https://cryptobriefing.com/bnb-chain-566m-hack-binance-networks-major-bridge-attack-unpacked/.
[52] Cryptonomist. 2022. Rubic DEX loses $1 million in crypto to hacker attack. https://en.cryptonomist.ch/2022/11/03/dex-rubic-loses-1-million-crypto/.
[53] Coin Culture. 2021. White Hat Saves Polygon from $850 Million Hack. https://coinculture.com/au/tech/white-hat-saves-polygon-from-850million-hack/.
[54] Axie DAO. 2023. We are the Axie DAO. https://axiedao.org/.
[55] DappRadar. 2022. Staking - RADAR Token - DappRadar. https://dappradar.com/token/staking.

[56] DappRadar. 2024. Dapp Industry Hits Record 5.3 Million dUAW. https://dappra dar.com/blog/dapp-industry-hits-record-5-3-million-duaw/.

[57] Rekt Database. 2023. Bridge Attacks. https://de.fi/rekt-database.

[58] REKT Database. 2023. Multichain Attack. https://de.fi/rekt-database/multichai n.

[59] Rekt Database. 2024. Biggest Crypto Hacks & Scams - DeFi REKT Database. https://de.fi/rekt-database.

[60] DefiLlama. 2024. DefiLlama - Hacks. https://defillama.com/hacks.

[61] DEFIYIELD.App. 2022. Wormhole exploit: the second-largest DeFi hack ever. https://blog.defiyield.app/wormhole-exploit-the-second-largest-defi-hack-ever-237ed5c81670.

[62] Ethereum. 2023. ERC-20 TOKEN STANDARD. https://ethereum.org/en/develo pers/docs/standards/tokens/erc-20/.

[63] Ethereum. 2023. GAS AND FEES. https://ethereum.org/en/developers/docs/ga s/.

[64] Ethereum. 2023. INTRODUCTION TO SMART CONTRACTS. https://ethereum .org/en/developers/docs/smart-contracts/.

[65] Ethereum. 2023. Welcome to Ethereum. https://ethereum.org/en/.

[66] Etherscan. 2022. Ethereum Transaction Hash (Txhash) Details. https://ethersca n.io/tx/0xa5fe9d044e4f3e5aa5bc4c0709333cd2190cba0f4e7f16bcf73f49f83e4a 5460.

[67] Etherscan. 2023. Address. https://etherscan.io/address/0xa8c83b1b30291a3a1a1 18058b5445cc83041cd9d.

[68] Etherscan. 2023. Null Address. https://etherscan.io/address/0x000000000000000 0000000000000000000000000.

[69] Binance Feed. 2022. Rubic Lost More Than $1.4 Million Due To The Hack. https://www.binance.com/en/feed/post/134659.

[70] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: A Static Analysis Framework for Smart Contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE. https://doi.org/10.1109/wetseb.2019.00008

[71] Dezentralized Finance. 2024. Cross Chain Bridges. https://dezentralizedfinance .com/cross-chain-bridges/.

[72] Fantom Foundation. 2023. Intro to Fantom. https://fantom.foundation/intro-to-fantom/.

[73] Joel Frank, Cornelius Aschermann, and Thorsten Holz. 2020. {ETHBMC}: A Bounded Model Checker for Smart Contracts. In *29th USENIX Security Symposium (USENIX Security 20)*. 2757–2774.

[74] Fusion. 2023. Distributed Control Rights Management. https://www.fusion.org /tech/dcrm.

[75] United Gamers. 2022. RUG PULL OR FUD? BLOCKVERSE DISAPPEARS, REAPPEARS, CONFUSES US ALL. https://www.unitedgamers.gg/news/blockverse-rupull-fud/.

[76] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. 2019. Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 139–156.

[77] Gustavo Grieco, Will Song, Artur Cygan, Josselin Feist, and Alex Groce. 2020. Echidna: effective, usable, and fast fuzzing for smart contracts. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*.

[78] Shelly Grossman, Ittai Abraham, Guy Golan-Gueta, Yan Michalevsky, Noam Rinetzky, Mooly Sagiv, and Yoni Zohar. 2017. Online detection of effectively callback free objects with applications to smart contracts. *Proceedings of the ACM on Programming Languages* (2017).

[79] Joël Gugger. 2020. Bitcoin-monero cross-chain atomic swap. *Cryptology ePrint Archive* (2020).

[80] Hacken. 2023. Heco Bridge Hack Explained. https://hacken.io/insights/heco-bridge-hack-explained/.

[81] Halborn. 2021. EXPLAINED: THE MULTICHAIN HACK (JANUARY 2022). https://halborn.com/explained-the-multichain-hack-january-2022/.

[82] Halborn. 2021. EXPLAINED: THE PNETWORK HACK (SEPTEMBER 2021). https://halborn.com/explained-the-pnetwork-hack-september-2021/.

[83] HALBORN. 2022. EXPLAINED: THE NOMAD HACK (AUGUST 2022). https: //halborn.com/explained-the-nomad-hack-august-2022/.

[84] HALBORN. 2022. EXPLAINED: THE RONIN HACK (MARCH 2022). https: //halborn.com/explained-the-ronin-hack-march-2022/.

[85] HALBORN. 2022. Harmony's Horizon Bridge Attack: How $100M was Siphoned Off By a Hacker. https://hackernoon.com/harmonys-horizon-bridge-attack-how-dollar100m-was-siphoned-by-a-hacker.

[86] Halborn. 2023. EXPLAINED: THE ORBIT BRIDGE HACK (DECEMBER 2023). https://www.halborn.com/blog/post/explained-the-orbit-bridge-hack-december-2023.

[87] Halborn. 2023. Halborn Blockchain Security Firm: Ethical Hackers, Infosec. https://halborn.com/.

[88] Halborn. 2024. EXPLAINED: THE SOCKET PROTOCOL HACK (JANUARY 2024). https://www.halborn.com/blog/post/explained-the-socket-protocol-hack-january-2024.

[89] Harmony. 2023. Horizon Bridge - Harmony. https://docs.harmony.one/home/ general/bridges.

[90] Coinbase Help. 2023. Tether (USDT) - Coinbase Help. https://help.coinbase.co m/en/coinbase/getting-started/crypto-education/usdt.

[91] Maurice Herlihy. 2018. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*. 245–254.

[92] Hop. 2023. Hop Exchange. https://bridge.connext.network/.

[93] IBM. 2023. What is blockchain technology? https://www.ibm.com/topics/what-is-blockchain.

[94] Bo Jiang, Ye Liu, and WK Chan. 2018. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 259–269.

[95] Aggelos Kiayias and Dionysis Zindros. 2020. Proof-of-work sidechains. In *Financial Cryptography and Data Security: FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*. Springer, 21–34.

[96] Numen Cyber Labs. 2023. Poolz Finance Attacked for $665,000. https://medium .com/@numencyberlabs/poolz-finance-attacked-for-665-000-56084cacae53.

[97] Sung-Shine Lee, Alexandr Murashkin, Martin Derka, and Jan Gorzny. 2023. Sok: Not quite water under the bridge: Review of cross-chain bridge hacks. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE.

[98] Xiubo Liang, Yu Zhao, Junhan Wu, and Keting Yin. 2022. A Privacy Protection Scheme for Cross-Chain Transactions Based on Group Signature and Relay Chain. *International Journal of Digital Crime and Forensics (IJDCF)* 14, 2 (2022), 1–20.

[99] LI.FI. 2023. Advanced Bridge & DEX Aggregation. https://li.fi.

[100] Sky Mavis. 2023. Blockchain: Gamified. https://www.skymavis.com/.

[101] Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. 2021. Sok: Validating bridges as a scaling solution for blockchains. *Cryptology ePrint Archive* (2021).

[102] Meter. 2023. THE FUTURE IS MULTI-CHAIN. https://meter.io.

[103] Meter.io. 2022. Community, unfortunately Meter Passport was hacked a few hours ago. https://twitter.com/Meter_IO/status/1490045486606139392.

[104] Mahdi H Miraz and David C Donald. 2019. Atomic cross-chain swaps: development, trajectory and potential of non-monetary digital token swap facilities. *arXiv preprint arXiv:1902.04471* (2019).

[105] Moonscan. 2022. Moonbeam Transaction Hash (Txhash) Details. https://moon scan.io/tx/0xcca9299c739a1b538150af007a34aba516b6dade1965e80198be021e3 166fe4c.

[106] Multichain. 2022. Multichain Contract Vulnerability Post Mortem. https: //medium.com/multichainorg/multichain-contract-vulnerability-post-mortem-d37bfab237c8.

[107] Multichain. 2023. Multichain Supported Chains. https://docs.multichain.org/ge tting-started/introduction/supported-chains.

[108] Neptune Mutual. 2022. Decoding Omni Bridge's Call Data Replay Exploit. https://medium.com/neptune-mutual/decoding-omni-bridges-call-data-replay-exploit-f1c7e339a7e8.

[109] Near. 2023. ETH <> NEAR Rainbow Bridge - NEAR Protocol. https://near.org/b ridge/.

[110] Razor Network. 2021. ChainSwap Exploit Post-Mortem. https://medium.com/r azor-network/chainswap-exploit-post-mortem-d73f5d15ce3c.

[111] Ronin Network. 2023. Ronin Network. https://bridge.roninchain.com/.

[112] Crypto News. 2022. pNetwork Clears the Air After Rumored Over $1B Breach on Their Platform. https://crypto.news/pnetwork-clears-the-air-after-rumored-over-1b-breach-on-their-platform/.

[113] Nomad. 2023. Nomad | Bridge. https://app.nomad.xyz/.

[114] Nomad. 2023. Nomad Docs. https://docs.nomad.xyz/token-bridge/how-to-bridge.

[115] Optics. 2023. Bridges By Optics v2. https://optics.app/.

[116] PolkaBridge. 2023. First Cross-Chain & MultiChain AMM. https://polkabridge. org.

[117] Polygon. 2023. Getting started with Polygon Proof of Stake chain. https: //wallet.polygon.technology.

[118] Polygon. 2023. Polygon - Ethereum's Internet of Blockchains. https://polygon. technology/lightpaper-polygon.pdf.

[119] PolyNetwork. 2023. Enhancing connections between ledgers by providing interoperability in Web 3.0. https://poly.network/.

[120] Metaverse Post. 2022. WonderHero Token Collapses After Hack. https://mpost. io/wonderhero-token-collapses-after-hack/.

[121] Crypto Potato. 2022. Ronin Network Announces Bridge Restart Date Three Months After $625M Hack. https://cryptopotato.com/ronin-network-announces-bridge-restart-date-three-months-after-625m-hack/.

[122] Ethereum Improvement Proposals. 2023. EIP-721: Non-Fungible Token Standard. https://eips.ethereum.org/EIPS/eip-721.

[123] Across Protocol. 2023. Across Protocol. https://across.to/bridge.

[124] Map Protocol. 2023. Map Protocol.: Secure Omnichain Layer for Web3 & dApps. https://www.mapprotocol.io.

[125] Qubit. 2023. Driving growth with personalization. Make ecommerce More personal. https://www.qubit.com.

[126] RazorNetwork. 2023. Truly decentralized oracle network for decentralized. https://razor.network.

[127] REKT. 2023. HYPR NETWORK - REKT. https://rekt.news/hypr-network-rekt/.

[128] Log Rocket. 2021. Substrate blockchain development: Core concepts. https://blog.logrocket.com/substrate-blockchain-framework-core-concepts/.

[129] Michael Rodler, Wenting Li, Ghassan Karame, and Lucas Davi. 2019. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. In *Proceedings of the 26th Network and Distributed System Security Symposium*.

[130] Narges Shadab, Farzin Houshmand, and Mohsen Lesani. 2020. Cross-chain transactions. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–9.

[131] Amritraj Singh, Kelly Click, Reza M Parizi, Qi Zhang, Ali Dehghantanha, and Kim-Kwang Raymond Choo. 2020. Sidechain technologies in blockchain networks: An examination and state-of-the-art review. *Journal of Network and Computer Applications* 149 (2020), 102471.

[132] SlowMist. 2021. SlowMist: Analysis of Three Consecutive Attacks on THOR-Chain (Released in 2021). https://slowmist.medium.com/slowmist-analysis-of-three-consecutive-attacks-on-thorchain-6223f1c691be.

[133] SlowMist. 2022. Truth Behind the Celer Network cBridge cross-chain bridge incident: BGP hijacking. https://medium.com/coinmonks/truth-behind-the-celer-network-cbridge-cross-chain-bridge-incident-bgp-hijacking-52556227e940.

[134] Stargate. 2023. Welcome to the omnichain future. https://stargate.finance.

[135] Hong Su, Bing Guo, Jun Yu Lu, and Xinhua Suo. 2022. Cross-chain exchange by transaction dependence with conditional transaction method. *Soft Computing* 26, 3 (2022), 961–976.

[136] Synapse. 2023. Synapse. https://synapseprotocol.com/.

[137] Coin Telegraph. 2022. Multichain asks users to revoke approvals amid 'critical vulnerability'. https://cointelegraph.com/news/multichain-asks-users-to-revoke-approvals-amid-critical-vulnerability.

[138] Coin Telegraph. 2022. Vitalik Buterin gives thumbs down to cross-chain applications. https://cointelegraph.com/news/vitalik-buterin-gives-thumbs-down-to-cross-chain-applications.

[139] ThunderCore. 2023. ThunderCore Docs. https://docs.developers.thundercore.com/product-protocol/bridges/interact-with-thundercore-bridge.

[140] Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. 2022. Universal atomic swaps: Secure exchange of coins across all blockchains. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1299–1316.

[141] Christof Ferreira Torres, Julian Schütte, and Radu State. 2018. Osiris: Hunting for integer bugs in ethereum smart contracts. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 664–676.

[142] Gang Wang. 2021. Sok: Exploring blockchains interoperability. *Cryptology ePrint Archive* (2021).

[143] Wenqi Wang, Zhiwei Zhang, Guoren Wang, and Ye Yuan. 2022. Efficient Cross-Chain Transaction Processing on Blockchains. *Applied Sciences* (2022).

[144] xDAI Bridge. 2023. xDAI Bridge - Gnosis Chain. https://bridge.gnosischain.com.

[145] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. 2021. Sok: Communication across distributed ledgers. In *International Conference on Financial Cryptography and Data Security*. Springer, 3–36.

[146] Ryan Zarick, Bryan Pellegrino, and Caleb Banister. 2021. Layerzero: Trustless omnichain interoperability protocol. *arXiv preprint arXiv:2110.13871* (2021).

[147] Jiashuo Zhang, Jianbo Gao, Yue Li, Ziming Chen, Zhi Guan, and Zhong Chen. 2022. Xscope: Hunting for Cross-Chain Bridge Attacks. *arXiv preprint arXiv:2208.07119* (2022).

[148] Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. 2020. TXSPECTOR: Uncovering Attacks in Ethereum from Transactions. In *USENIX Security Symposium*.

A RELATED WORKS

In Sec. 3, we have covered related studies on cross-chain bridge attacks. Here, we delve into topics such as blockchain interoperability, cross-chain protocols, atomic swaps, and side chains.

Blockchain interoperability. The capability of different blockchains to communicate with each other is known as blockchain interoperability. According to *Buterin* [25], there are three primary types of blockchain interoperability, which are centralized or multi-signature notary schemes, side chains or relays, and hash-locking. *Belchior et al.* [9] conduct a literature review on blockchain interoperability and classify the studies into three categories: public connectors, blockchain of blockchains, and hybrid connectors. *Wang* [142]

performs a systematic study on blockchain interoperability and classifies it into three types (chain-based, bridge-based, and DApp-based) based on current research. *Augusto et al.* [4] systematically categorizes 56 interoperability solutions. These works focus on measuring the current state of blockchain interoperability, while our work explores the applications of blockchain interoperability on cross-chain bridges.

Cross-chain protocols. Cross-chain protocols enable data sharing among multiple blockchains, similar to blockchain interoperability. *Zamyatin et al.* [145] measure current cross-chain communication protocols and classify them into two categories: exchange protocols for token exchange and migration protocols for asset transfer between blockchains. *Shadab et al.* [130] propose a uniform 3-phase protocol for general cross-chain transactions that complies with uniformity requirements and adds a third end-to-end constraint. *Wang et al.* [143] design and implement a new cross-chain model based on cross-chain scheduling, while *Su et al.* [135] propose an asynchronous cross-chain model that ensures cross-chain atomicity based on transaction dependence. *Liang et al.* [98] propose a cross-chain privacy protection scheme to address identity privacy leakage. These works measure or propose cross-chain protocols, while our work summarizes the cross-chain protocols used in current cross-chain bridges, such as CCIP from Chainlink.

Atomic swaps. Atomic swap can facilitate local verification, which is one type of four cross-chain bridge verification mechanisms (see Sec. 4.2). They do not require any intermediary party to enable the exchange. In 2018, *Herlihy* [91] introduced the atomic swap protocol, which guarantees that the exchange is successfully completed or not executed at all. *Thyagarajan et al.* [140] proposes a universal protocol for swapping tokens across blockchains, utilizing adaptor signatures and time-lock puzzles. *Gugger* [79] describes a protocol for atomic swaps between Monero and Bitcoin, which can be generalized to similar blockchains. *Miraz et al.* [104] discusses atomic swaps, including their workflow, current state, future trends, and possible applications. *Borkowski et al.* [16] presents an overview of the current state-of-the-art in cross-chain atomic swaps, as well as prominent blockchains and relevant ongoing and operational projects. Our work briefly mentions atomic swaps as a potential cross-chain bridge underlying protocol and classify them to the local verification categorization.

Side chains. Side chains can increase the main network's scalability by processing and batching large amounts of transactions before submission on the main blockchain. Pegged side chains were first introduced by *Back et al.* [6] in 2014, allowing for the transfer of assets between different blockchains, such as Bitcoin. *Singh et al.* [131] provide a comprehensive review of various state-of-the-art side chains and platforms, analyzing their impact and identifying their limitations. *Kiayias et al.* [95] propose the side chain construction that facilitates communication between Proof-of-Work blockchains without trusted intermediaries, while *Gaži et al.* [76] developed a side chain construction suitable for Proof-of-Stake systems. These works focus on side chains primarily for scalability concerns, while our work concentrates on cross-chain bridges that enhance blockchain interoperability.

Classification	Usages	Example
Token transfer	token transfer	Anyswap
Token withdraw	native token withdraw	Anyswap
Token swap	token exchanges	Anyswap
Governance	extend governance	Aave
Lending and borrowing	borrow and deposit	Cross-chain Loans
Staking	stake and claim rewards	RADAR

Table 5: Summary of potential bridge usages.

B BRIDGE USAGES

We have discussed the most popular usage, cross-chain token transfer, in Sec. 4.1. Table 5 presents these usages.

Cross-chain token withdraw. Continuing with the above example, $\mathbb{A}_D$ submits a transaction *TxBurn* in BNB and burns the wrapped tokens (10.0 *WETH*). Then the bridge submits the related transaction *TxRelease* to release 10.0 *ETH* back to the user.

Cross-chain token swap. Similar to the token transfer, cross-chain token swap is another major usage of bridges. For instance, a user can swap 10.0 *USDC* from Ethereum to BNB for 10.0 *USDT*.

Governance is a popular way to manage and implement changes. In some DApps, users who hold the native governance tokens are allowed to submit and vote for proposals, to participate in the future of DApps. For instance, Aave [1] holders of *AAVE* and/or *stkAAVE* tokens are given the rights of proposal and vote. To realize the governance across chains, Aave proposes bridges to extend its governance on Ethereum to other blockchains, such as Polygon [2].

Lending and borrowing have always been fundamental services provided by DeFi platforms. Cross-chain lending and borrowing can bring flexibility to DeFi users. For example, Crosschain Loans [36] supports the cross-chain lending between EVM compatible blockchains. Users can borrow Ethereum tokens by depositing assets on other blockchains as collateral.

Staking locks up a certain amount of assets, serves as a proof of work, and earns passive incomes. Previously, staking is limited to the single native blockchain. With cross-chain staking, users can stake across different blockchains and claim staking rewards on any blockchain, regardless of which blockchain the staked assets are deposited. For example, RADAR Cross-Chain Token Staking [55] allows users to stake and claim rewards on any blockchain.

C VERIFICATION MECHANISMS

C.1 External Verification

Cross-Chain Interoperability Protocol (CCIP) facilitates standardized and secure communication and information exchange among various blockchain networks. To develop their CCIP, Chainlink utilizes Decentralized Oracle Networks (DONs) and the Anti-Fraud Network [40]. DONs provide off-chain services and data, such as random number generation, and enable secure validation of cross-chain transactions through the consensus of multiple oracle nodes. The Anti-Fraud Network, on the other hand, detects and halts malicious cross-chain activity to safeguard users. Chainlink's CCIP supports various cross-chain services, such as programmable token bridges, cross-chain bridges, and cross-chain DApps. Specifically, Chainlink's CCIP enables a smart contract on the source blockchain to use their messaging router to send a secure message to a destination chain's messaging router. The destination messaging router then validates and forwards the message to the destination smart contract.

Omnichain interoperability protocol (LayerZero) facilitates seamless communication between multiple blockchains. Unlike other protocols, Omnichain has no wrapping or intermediary layers. LayerZero is one of such protocols [146], which offers trustless messaging and can operate on any blockchain network, such as Ethereum and Bitcoin. LayerZero leverages an Oracle and a Relayer to securely pass messages between different domains off-chain. The Oracle sends a block header from the source blockchain to the destination blockchain, while the Relayer passes a transaction proof. If the block header and proof match and are validated, the cross-chain message is sent to the destination blockchain. In practice, LayerZero leverages the Chainlink's DONs to ensure trustless delivery of messages between disparate chains, as an Oracle. For example, Stargate [134] is a fully composable cross-chain bridge protocol built on LayerZero that enables native asset transfers between different blockchain networks.

C.2 Optimistic Verification

Optimistic Interchain Communication Protocol (Optics) is a protocol that enables cost-efficient message transfer between consensus systems [115]. It relies on channels to send messages as raw bytes from one chain to another. Once a channel is established, any application on a chain can use it to send messages to any other chain, without requiring a light client or extra gas for verifying remote chain block headers. Optics contains initial implementations of on-chain contracts in Solidity code and off-chain system agents in Rust code. Applications that use Optics need to implement their messaging protocol and are facilitated by Router contracts. For instance, Nomad [113] extends the Optics protocol. During a 30-minute window, challengers can question the validity of transactions.

C.3 Local Verification

Connext's NXTP provides a straightforward method for cross-chain transfers and contract calls that is fully noncustodial [49]. The protocol involves three phases: route selection, preparation, and fulfillment. During route selection, routers bid to complete the transfer. In the preparation phase, users initiate a transaction on the source blockchain through the NXTP contracts. In the fulfillment phase, the message is signed to complete the transaction on the destination blockchain. At any point, either party can cancel the transaction if necessary. Once the transaction has expired, both parties can cancel the transaction.

C.4 Native verification

MAP Protocol is an interoperability protocol that facilitates cross-chain transactions, supported by the dedicated MAP blockchain [124]. The protocol consists of three layers: the MAP Protocol Layer, the MAPO Services Layer, and the MAPO Application Layer. The MAP

Protocol Layer is responsible for verifying cross-chain assets and data through independent, self-verifying light clients deployed on each blockchain. The MAPO Services Layer serves as the execution layer for assets and data and supports developers to build applications for the MAPO Application Layer. Overall, MAP Protocol is designed to enable DeFi and other applications to reach their full potential while ensuring the safety of cross-chain assets.

D CROSS-CHAIN BRIDGE ATTACK SURFACES

A7: Vulnerable bridge smart contracts. Common vulnerabilities in smart contracts (e.g., access control) are possible in bridge smart contracts. We illustrate some potential attacks as follows, taking the liquidity pool smart contract code as an example in Fig. 10.

```
1   contract LiquidityPool {
2     address bridgeAdmin;
3     mapping(address => uint256) balances;
4     uint256 liquidReserves;
5
6     modifier onlyAdmin {
7       require(msg.sender == bridgeAdmin);
8     }
9
10    constructor(address _bridgeAdmin) {
11      bridgeAdmin = _bridgeAdmin;
12    }
13
14    function changeAdmin(address _newAdmin) public
          onlyAdmin {
15      bridgeAdmin = _newAdmin;
16    }
17
18    function addLiquidity(uint256 tokenAmount) public
          payable {
19      require(balances[msg.sender] >= tokenAmount);
20      uint256 lpTokenAmount = tokenAmount /
          exchangeRatio();
21      mint(msg.sender, lpTokenAmount);
22      liquidReserves += tokenAmount;
23      emit AddLiquidity(tokenAmount, lpTokenAmount,
          msg.sender);
24    }
25
26    function removeLiquidity(uint256 lpTokenAmount)
          public {
27      uint256 tokenAmountReturn = lpTokenAmount *
          exchangeRatio();
28      require(liquidReserves >= tokenAmountReturn);
29      burn(msg.sender, lpTokenAmount);
30      liquidReserves -= tokenAmountReturn;
31      emit RemoveLiquidity(tokenAmountReturn,
          lpTokenAmount, msg.sender);
32    }
33  }
```

Figure 10: An example of liquidity pool smart contract.

- Incorrect initialization. Bridge accounts will be set as the admin/owner addresses of the liquidity pool during the pool contract creation. If these privileged addresses are initialized with incorrect values in Line 10-11, the total assets in the smart contract can be drained.

- Inappropriate function permission. Similarly, there are some special functions that change privileged addresses or control contracts, such as killing or pausing contracts when the caller is an admin/owner. If these functions are created with inappropriate permissions in Line 14-15 (e.g., allowing anyone to call them), the contracts will be hacked.

- Unchecked balance. To add liquidity to the pools, the liquidity providers should transfer their tokens. If the balance of tokens is not checked in Line 19, attackers can deposit nothing or tokens with less value than the value of returned tokens (e.g., LP tokens). Similarly, there will be problems if balance check is not performed correctly in Line 28.

- Miscalculated token price. If the amount of LP tokens is given to the liquidity providers more than desired due to the miscalculated token price in Line 20, the pool will lose money. In particular, the function *exchangeRatio* calculates the token price. Similarly, in the liquidity remove process, if the amount of original tokens returned to the liquidity providers is more than desired in Line 27, there will be financial losses too.

- Inconsistent event. The events will be emitted for off-chain bridge server to monitor, when non-trivial actions happen. For example, *AddLiquidity* event is emitted in Line 23 when the function *addLiquidity* is called. If the *tokenAmount* representing the amount of deposited tokens is wrong, the bridge will get incorrect information and there will be inconsistencies. Similarly, the event *RemoveLiquidity* in Line 31 should be correctly recorded.

E REAL-WORLD ATTACKS

E.1 Permission Issue

E.1.1 V1. Unchecked Intermediary Permission. Among all the 35 real-world attacks, 1 falls under this specific vulnerability type.

Poly Network attack. On Aug-10-2021, Poly Network [119] suffered from an attack resulting in $600M loss. In particular, the bridge failed to set the appropriate permission in the manager smart contract and allowed anyone to call the key function in the contract controlled by the manager [33]. Thus, the attacker could change the owner to be his/her account and drained all the tokens from the hacked smart contract.

E.1.2 V2. Misused Proof Permission. Among all the 35 real-world attacks, 2 fall under this specific vulnerability type.

Plasma attack. Plasma [21] is one of the popular bridges on Polygon, supporting token deposit and withdraw. On Oct-05-2021, a vulnerability was found and exploited by white hats rescuing around $850 million [53]. In this exploit, the previous valid proof could be replayed multiple times, to exit the burn transaction and steal tokens. Fortunately, no loss was caused due to the rescue.

Binance Bridge attack. On Oct-06-2022, attackers stole around $566 million from Binance Bridge [51]. The root cause is a smart contract bug that allows attackers to forge proofs to pass the validation so that the fraudulent withdraw can succeed. To mitigate this attack, Binance-backed blockchains launched a hard fork on Oct-12-2022 [10].

E.1.3 V3. Problematic Approval Permission. Among all the 35 real-world attacks, 4 fall under this specific vulnerability type.

Multichain attack. Multichain [138] is a cross-chain bridge that supports financial services on multiple blockchains. On Jan-18-2022, Multichain was attacked, in which several blockchains were affected [81]. It caused around $3 million loss by multiple group of attackers. In particular, users approved the infinite number of tokens to bridge-related smart contracts in advance. The attacker exploited the incorrect permission check to pass the validation and drain the tokens from users.

LI.FI attack. On May-20-2022, an attacker exploited the vulnerable swapping feature of LI.FI bridge [99]'s smart contract [13]. Instead of swapping, attackers steal money from users by directly transferring their tokens. As a result, anyone who gave infinite approval to the vulnerable contract could be attacked.

Rubic attack. On Dec-25-2022, Rubic suffered a hack which resulted in a loss of $1.4 million [69]. The main reason behind the attack was the incorrect addition of USDC tokens to the whitelist. This led to the theft of USDC tokens from users to the Rubic contract.

Socket protocol attack. On Jan-16-2024, a security incident occurred within the Socket protocol, resulting in infinite approvals granted to Socket contracts by users, due to incomplete input validation [88]. Exploiting this, attackers stole funds of $3.3 million from approximately 712 affected users.

E.1.4 V4. Invalid Signature Permission. Among all the 35 real-world attacks, 3 fall under this specific vulnerability type.

ChainSwap attack. ChainSwap [43] is a cross-chain asset bridge, which allows users to bridge tokens between blockchains seamlessly. ChainSwap was exploited on Jul-10-2021 and many projects using ChainSwap were affected, including Razor Network [126]. The attacker stole tokens worth $8 million and sold the tokens on various exchanges [110].

Wormhole attack. On Feb-02-2022, an attacker launched this attack with bypassing the verification process in the Wormhole bridge [24] and minted Wormhole *ETH* (*wETH*) tokens [37]. In particular, the attacker injected a faked account and successfully generated a malicious message to mint 120,000 *wETH* worth $320 million.

Multichain attack. On Feb-15-2022, multichain attack took place again by the insufficient signature verification [58]. The attacker managed to transfer the approved tokens to the victim's contract, resulting in a profit of 87 ETH.

E.1.5 V5. Leaked Key Permission. Among all the 35 real-world attacks, 10 fall under this specific vulnerability type. We discussed the Ronin Network attack in Sec. 6.1.5 and will cover the remaining 9 here.

WonderHero attack. On Apr-07-2022, WonderHero, a NFT-based game, claimed that it was attacked by attackers who obtained signatures of validators and then minted 80 million *WND* tokens [120]. The loss was approximately $0.3 million.

Horizon Bridge attack. Horizon Bridge [27] supports connections between any Proof-of-Work (PoW) and Proof-of-Stake (PoS)

blockchain. On Jun-24-2022, the bridge was hacked and lost around $100 million. The attackers compromised private keys of two validators and successfully stole tokens with the 2-of-4 multi-signature mechanism [85].

QAN platform bridge attack. On Oct-11-2022, the QAN platform bridge experienced a security breach that was caused by the compromise of a private key [11]. The attacker gained access to the private key of the bridge address and withdrew tokens from both the Ethereum and BNB blockchains. The attack resulted in a total loss of approximately $2 million at the time it occurred.

Rubic attack. Attackers compromised the private key of an admin wallet address belonging to Rubic on Nov-02-2022 [52]. This wallet was responsible for managing the RBC/BRBC bridge and staking rewards. The attackers were able to transfer 34 million RBC out of the wallet, which was valued at $1.2 million at the time.

pNetwork attack. On Nov-04-2022, more than $1 billion of pGALA tokens on Bnb were minted out of thin air. This occurred because the private key was leaked on Github [112]. To prevent further attacks, pNetwork minted a substantial number of tokens themselves, trying to the drainage of the liquidity pool. However, this action was not handled correctly and introduced losses caused by arbitrage opportunities.

Poly Network attack. On Jul-01-2023, attackers were suspected to take control of the private keys associated with Poly Network wallets, granting themselves the ability to create signatures without limit. This exploit allowed the hacker to generate 57 tokens that span 10 different blockchains, including Ethereum and BNB Chain, among others. As a result of this attack, a total of $10,201,612 in losses was incurred [57].

Multichain attack. On Jul-06-2023, Multichain experienced a significant breach, resulting in substantial cross-chain fund losses ($231,129,033). Funds were consolidated into a single EOA across 9 chains, suggesting full control by the attacker and raising internal operation suspicions [58].

Heco bridge attack. On Nov-22-2023, Heco bridge, facilitating cross-chain transfers between Ethereum and Heco chain, suffered an access control exploit [80]. The attacker compromised the private keys of a privileged address, enabling withdrawal from the bridge's smart contract. The attack resulted in the loss of assets valued at $86,284,430.

Orbit bridge attack. On Dec-31-2023, Orbit Bridge encountered a private key breach [86]. The attacker drained funds from the bridge contract, converting them to ETH and DAI, and dispersed them among various EOA addresses. The attacker received funding via TornadoCash and compromised 7 out of 10 multisig signers' wallets.

E.2 Logic Issue

E.2.1 V6. Incorrect Balance Logic. Among all the 35 real-world attacks, 2 fall under this specific vulnerability type.

PolkaBridge attack. PolkaBridge [116] was attacked on Nov-22-2021 and lost around $0.6 million. The attack was caused by a smart contract bug after the balance check was removed mistakenly.

Poolz Finance attack. Poolz Finance was attacked on May-15-2023, in which attackers triggered an integer overflow vulnerability and stole around $665,000 on several blockchains [96].

E.2.2 V7. Inaccurate Initialization Logic. Among all the 35 real-world attacks, 3 belong to this specific vulnerability type.

Nomad attack. Nomad [113] suffered from an attack on Aug-02-2022 with around $190 Million loss. For example [34], an attacker [67] sent 0.01 *WBTC* to Nomad in Moonbeam transaction [105] and then Nomad transferred 100 *WBTC* back to the attacker in Ethereum transaction [66] This attack was caused by an initialization error, in which the trusted root was set to an incorrect value, the permission check could be passed by anyone, and the money was then stolen by attackers [83].

Allbridge attack. On Apr-01-2023, Allbridge fell victim to a flash loan attack triggered by an error in the price calculation logic. This exploit resulted in approximately $570,000 worth of Bnb being lost [57].

Hypr OP Stack bridge attack. On Dec-13-2023, the Hypr OP Stack bridge suffered a compromise [127]. The exploit stemmed from a vulnerability allowing contract reinitialization and fund drainage by the attacker. The total loss amounted to about $215,193, equivalent to 2,570,000 HYPR tokens.

E.3 Event Issue

E.3.1 V8. Incorrect Event Emission. Among the 35 real-world attacks, 3 fall under this specific vulnerability type.

Qubit attack. Qubit [125] exploit happened on Jan-27-2022, in which the attacker used the vulnerable functions and obtained an ETH *Deposit* event with a ERC20 token transfer. The incorrect event allowed the attacker to mint around 77,162 *qxETH*, which was worth around $80 million.

Meter.io attack. On Feb-05-2022, the Meter.io bridge [102] project was exploited, causing a $4.2 million loss [103]. In the attack, the attacker obtained *Deposit* events by using the function *deposit()* targeting *WETH/WBNB* tokens without real deposit.

Omni bridge replay attack. On Sep-16-2022, the Omni Bridge replay attack was due to a chainID permission check vulnerability [108]. Replay attacks can occur during blockchain protocol upgrades or when a vulnerability is exploited across multiple blockchains. The attacker in this case first transfer 200 *WETH* to Ethereum Proof-of-Stake (PoS) blockchain and then used an outdated chainID to replay the same transaction on the Ethereum Proof-of-Work (PoW) blockchain. Note that there are some users who did not switch to PoS and are still on the Ethereum PoW blockchain.

E.3.2 V9. Fake Event Emission. Among the 35 real-world attacks, 5 fall under this specific vulnerability type.

THORChain attack. In a series of unfortunate events, THORChain experienced three separate fake deposit attacks in quick succession. The first attack occurred on Jun-29-2021, causing a loss of approximately $350,000. Following closely, a second attack struck on Jul-16-2021, resulting in a staggering loss of nearly $8 million. Just a week later, on Jul-23-2021, THORChain was targeted yet again, resulting

in another substantial loss of nearly $8 million. These incidents underscored the vulnerability of the THORChain network to such fraudulent deposit exploits during this period [132].

pNetwork attack. On Sep-19-2021, the attacker generated counterfeit events to access cryptocurrency in the Bitcoin blockchain [82]. The events were executed without confirmation of their legitimacy from the pNetwork contracts. Thus, the attacker successfully stole 277 *BTC*, equivalent to more than $13 million by falsifying the events.

Cennznet attack. On May-08-2022, an attacker hacked the Cennznet bridge [32] by emitting fake *Deposit* events. The bridge captured the fake events and sent ETHs to the attacker. In total, $0.4 million (around 155 *ETH*) was stolen.

E.4 Front-end Issue

E.4.1 V10. Front-end Phishing. Among all the 35 real-world attacks, 2 fall under this specific vulnerability type. We discussed the EVODeFi attack in Sec. 6.4.1 and will cover the remaining 1 here.

Celer cBridge attack. On Aug-17-2022, Celer cBridge suffered from a domain name system (DNS) poisoning attack, resulting in a loss of $240,000 [133]. Some users were redirected to malicious smart contracts while using the bridge, and their approved tokens were drained by the attacker.