No More Hacked Bridges: Exploring the new xERC20 Token Standard for Cross-chain Assets

As modular blockchains (such as Ethereum L2s) have grown in popularity, so have bridges—protocols that allow users to move tokens and data among these chains. Unfortunately, this rise in the popularity of bridges has brought a rise in the number of *hacks* of these bridges—hacks that have cost protocols, and users, millions of dollars.

If you’re a developer or team trying to make your token available on multiple chains, you already know these problems.

One alternative that aims to solve these current vulnerabilities is the new [xERC20 token standard](https://docs.connext.network/usecases/xerc20)proposed by Connext. xERC20 is a standard that defines a better way to create cross-chain tokens: ones that are deployed and controlled by the token issuer, transferable across chains with no slippage, and still fully secured by the Layer-1 chain.

Let’s look at this new xERC20 token standard in detail, including how it works, how it can help your team, how it compares to OFT, and a few of the projects already implementing it.

But first, a little background.

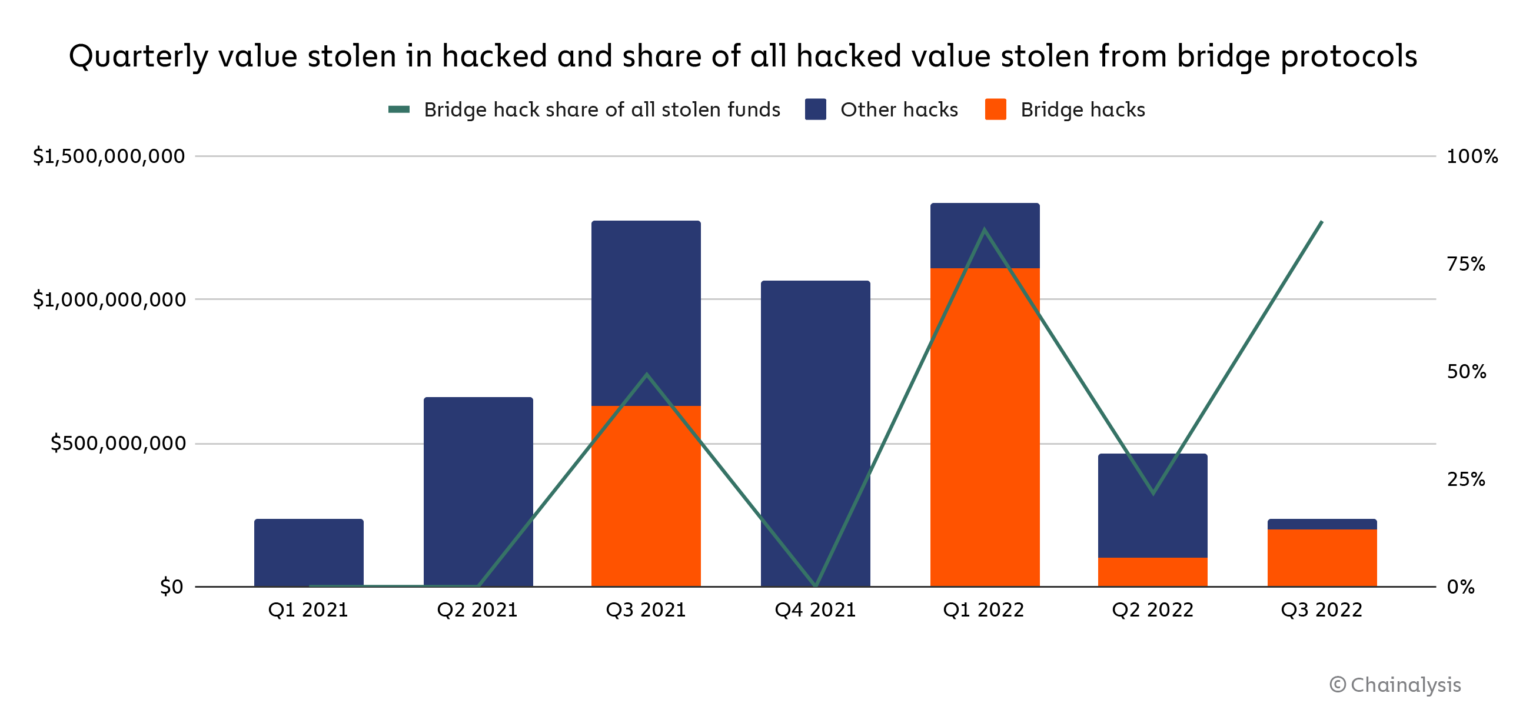
**What are bridges and why are they hacked so often?**

A **bridge** is a “gateway” between blockchains that allows you to transfer tokens and data between those chains.

The most common use of bridges is to transfer ERC20 tokens (such as WETH, DAI, USDT, etc.). Protocols such as [Across](https://across.to/), [Connext](https://www.connext.network/), [and Stargate](https://stargate.finance/) are some of the [largest](https://defillama.com/bridges) bridges by volume that currently allow ERC20 transfers between Ethereum, Polygon, BNB, and others.

However, even though bridges are commonly used, *transferring ERC20 tokens using bridges is a highly vulnerable process and often results in hacks or loss of funds*.

According to [Chainalysis](https://blog.chainalysis.com/reports/cross-chain-bridge-hacks-2022/), attacks on bridges account for 69% of total funds stolen in 2022.

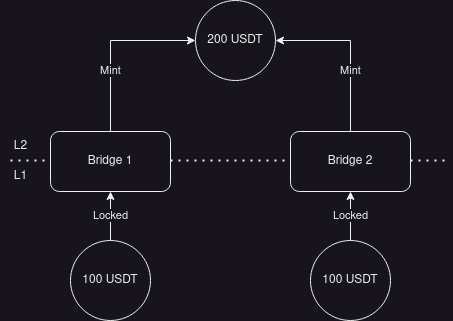


Chainalysis estimates that attacks on bridges account for 69% of total funds stolen in 2022.

But today, teams don’t really have a choice. If they want to make their token available on multiple chains, they are dependent on bridges and one of two flawed models:

1. **Provide liquidity to the bridge on each chain**. In this model, when a user requests a transfer, the tokens enter the pool on chain A, and the user withdraws tokens from chain B. Unfortunately, for this case, not only will you (as the token creator) have to provide large amounts of liquidity to both pools, but your users end up bearing slippage fees and pool rebalancing fees.
2. **Mint a new token through the bridge**. In this model, when tokens are transferred, tokens on the source chain are locked in the bridge contract, and an equivalent amount of tokens are minted on the destination chain. This not only locks you (as the token creator) into the bridge’s model forever, but is a security risk as a bug in the code could give hackers the ability to mint a high (or even unlimited) number of unbacked tokens.

Bridges also create other issues, such as **liquidity fragmentation**.



Imagine a scenario where a user sends 100 USDT through the Optimism bridge (Bridge 1) and an additional 100 USDT through Connext (Bridge 2) to Optimism, resulting in a total of 200 USDT on Optimism (as shown in the figure above). If the user then attempts to bridge back the 200 USDT through the Optimism bridge, they can only unlock the first 100 USDT. The remaining 100 USDT will be stuck.

Why does this happen? The issue lies in the fact that the liquidity on Layer 1 (L1) is divided between the Optimism bridge and Connext. Because of this fragmentation, the user can't retrieve the full amount through a single bridge, leading to limitations in the usability of these bridging systems.

**The xERC20 token standard: the solution to problematic bridges**

To solve these problems, Connext has introduced an open standard - the [**xERC20**](https://docs.connext.network/usecases/xerc20) **token design**.

In this design, xERC20 tokens are:

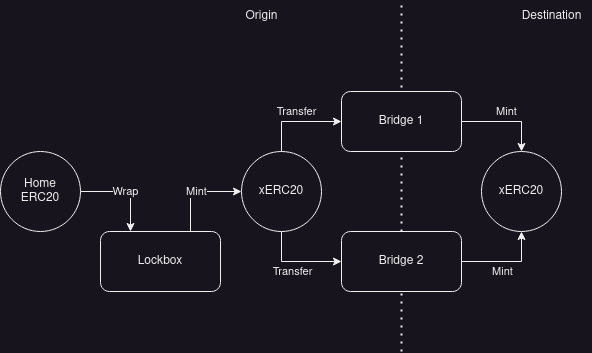
* Transferable across chains with no slippage.
* Deployed and fully controlled by the token issuer.
* Still secured fully by Ethereum L1 and the canonical bridges of each chain.

How does this work? The power of the xERC20 token standard lies in standardization of the way that tokens are made “bridge-able” and the possibility for any token to be compatible with this standard.

As we discuss further below, the xERC20 standard introduces a new interface for tokens. This interface allows the token issuer to define

* What bridges can transfer their tokens from one chain to another
* And in what amounts

By utilizing a simple smart contract on the source chain, these limits can be enforced and the safety of the token issuance guaranteed.



With xERC20 tokens, creating multi-chain tokens, and bridging them, is easy.

Let’s assume that a user wants to bridge your token (let’s call it MY\_TOKEN) from Ethereum to Optimism. With xERC20, the steps are as follows:

1. The MY\_TOKEN on Ethereum is "wrapped" (similar to how ETH is wrapped to WETH) on a smart contract named Lockbox into xMY\_TOKEN.
2. The wrapped tokens are sent to a bridge, which burns/locks the wrapped token.
3. The bridge sends a message through Optimism's canonical bridge stating that the tokens have been received on the mainnet and mints an equivalent amount of xMY\_TOKEN on Optimism.

As long as the token design has been adopted by the token issuer and both bridges, they bridges can work together to ensure the availability of the token on Optimism and on both bridges.

Note: For tokens that are already upgraded or support the xERC20 standard, the transfer process starts directly at step 2.

And by implementing the xERC20 standard, you as the token issuer **can set specific limits on how many tokens can be minted by a specific bridge in a specified time period**. This, coupled with the chronology mentioned above, ensures that even if a bridge is hacked, the hacker cannot mint an infinite number of MY\_TOKEN.

This mechanism also safeguards the user in case the bridge is hacked. Assuming the xERC20 token standard is accepted widely, the user would be able to retrieve their MY\_TOKEN on the mainnet whenever they wish by simply using another supporting bridge.

**Current alternatives - why are they not enough ?**

Before the xERC20 standard, there were no **open** standards that could provide the same functionality as xERC20. The only alternatives were:

* trusting a third-party bridge to be bug-free and allow it to mint tokens on the destination chain - *it is not as safe*
* providing liquidity on source and destination chain to allow transfer - *the opportunity cost is too high*
* using the OFT standard - *it is not an open standard*

The Omnichain Fungible Token (OFT) standard is a developer-dependent token standard developed and used by the LayerZero team. The OFT standard allows the token issuer to transfer tokens across chains, but not all chains. The OFT standard can work only with LayerZero smart contracts and hence work only with chains that are supported by the LayerZero bridge.

The main difference between xERC20 and OFT is that xERC20 is an **open standard** that is not anyhow dependent on the developers of xERC20. For a token issuer that wants to make their token compatible with the xERC20 standard, they would either use an already [audited](https://t.me/c/1814026758/157) code of the implementation or write one of their own (though there is no significant advantage of the latter option over the former if functionality remains the same). The Lockbox contract (which can be independently deployed on any chain) also does not have any connection to the deployer (the only connection would be the deployment transaction sent by the deployer).

|  | xERC20 | OFT |
| --- | --- | --- |
| Cross-bridge interoperability? | Yes, since this would be an ERC standard, all bridges could become compatible with it. | No, works only with LayerZero smart contract and 14 chains that LayerZero supports. |
| Control on tokens issued? | Yes, allows to set bridge-specific max. Limits on the amount of tokens issued in a period of time. | No. |

Even though the OFT standard allows for an alternative method of cross-chain bridging of tokens, the ownership of the LayerZero contracts *remains with the LayerZero team*.

So, while xERC20 allows for broad cross-bridge compatibility and retains control over token issuance, the OFT standard is specific to LayerZero, works only on LayerZero-supported-chains, relinquishes control over tokens, and locks them with LayerZero until the tokens remain on the destination chain.

## **How to create an xERC20 token**

There are a couple paths for making your ERC20 token compatible with the xERC20 standard.

The simplest option (if the token contract is upgradeable) is to upgrade your contracts to be compatible with the standard by implementing the relevant functions.

If your token contract is not upgradeable (or you don’t want to upgrade), two other options are available.

The first option is to move the liquidity of the original token into a newly deployed token that implements the xERC20 standard.

The second option is to keep your original token, but use the Lockbox contract to wrap the original token into a new token compatible with the xERC20 standard. Since the wrapping logic is reliable and well-tested (the most prominent example being the ETH-WETH pair), this option is considered proven and safe.

For instance, if a token issuer has a standard ERC20 contract ready for deployment and they wish for their token to be compatible with xERC20, the major changes they would have to do are:

1. Create a contract xERC20 which inherits from ERC20:

| contract XERC20 is ERC20 {  // ... code goes here ... } |
| --- |

1. Create and maintain in code a mapping for limits associated for each bridge

| mapping(address => Bridge) public bridges; // Bridge can be a struct representing bridges with any additional configs that the token issue may want to maintain |
| --- |

1. Implement the minting and burning logic

| function \_mint(address account, uint256 amount) internal virtual {  require(account != address(0), "ERC20: mint to the zero address");   \_beforeTokenTransfer(address(0), account, amount);   \_totalSupply = \_totalSupply.add(amount);  \_balances[account] = \_balances[account].add(amount);  emit Transfer(address(0), account, amount); }  function \_burn(address account, uint256 amount) internal virtual {  require(account != address(0), "ERC20: burn from the zero address");   \_beforeTokenTransfer(account, address(0), amount);   \_balances[account] = \_balances[account].sub(amount, "ERC20: burn amount exceeds balance");  \_totalSupply = \_totalSupply.sub(amount);  emit Transfer(account, address(0), amount); } |
| --- |

1. Implement functions relating to the bridge limits (these shall update the mapping introduced in Step 2)

| function setLimits(address \_bridge, uint256 \_mintingLimit, uint256 \_burningLimit) external onlyOwner {  \_changeMinterLimit(\_mintingLimit, \_bridge);  \_changeBurnerLimit(\_burningLimit, \_bridge); } |
| --- |

| function mintingMaxLimitOf(address \_bridge) public view returns (uint256 \_limit) {  \_limit = bridges[\_bridge].minterParams.maxLimit;  } |
| --- |

| function burningMaxLimitOf(address \_bridge) public view returns (uint256 \_limit) {  \_limit = bridges[\_bridge].burnerParams.maxLimit;  } |
| --- |

With the functions mentioned above (and any other functions which are finalized to be there if the [EIP-7128](https://ethereum-magicians.org/t/erc-7281-sovereign-bridged-tokens/14979) is accepted) any token issuer would allow their token to be a part of the xERC20 standard (and hence other bridges would be able to interact with it).

**Integrating your own xERC20 with Connext**

The xERC20 being an open token standard, allows multiple teams to be working on their own implementation of the interfaces required by it. For example, token issuers can already use Connext to deploy their xERC20 tokens and allow seamless cross-chain transfer without worrying about providing liquidity on multiple chains. In order to use xERC20 with Connext bridges (remember that xERC20 standard is permissionless and can work with canonical bridges as well but select bridges must be approved by the issuer before transfers), the steps are given [here](https://docs.connext.network/usecases/xerc20/requirements).

The way this process works right now can be summarized as the following:

1. An implementation of the token on a “home” chain must be deployed
2. A mintable and burnable representation of the token must be deployed on “destination” chains (the bridge must have permission to burn and mint this representation)
3. The token representation addresses must be added to the mainnet [allowlist config](https://github.com/connext/monorepo/blob/main/packages/deployments/contracts/src/cli/init/config/mainnet/production.ts) file and the [Chaindata mappings](https://github.com/connext/chaindata/blob/main/crossChain.json) by submitting a PR.
4. Once the PRs are approved by the Connext team, the tokens can be transferred across chains using the [bridge](https://www.connext.network).

**Teams are already implementing xERC20 tokens**

Multiple teams are already collaborating with the developers behind the [EIP](https://github.com/ethereum/EIPs/pull/7281) to create the first implementations of the xERC20 token standard.

**Alchemix**

Alchemix has recently [announced](https://cointelegraph.com/news/connext-alchemix-cross-chain-token-standard-reduce-bridge-exploit-losses) the support of the xERC20 standard and is onboarding projects and helping them to safely bring their tokens to every chain. They have been working with the Connext team to bring [$alUSD](https://twitter.com/search?q=%24alUSD&src=cashtag_click), [$alETH](https://twitter.com/search?q=%24alETH&src=cashtag_click), and [$ALCX](https://twitter.com/search?q=%24ALCX&src=cashtag_click) tokens toArbitrum andOptimism using the xERC20 token standard.



**Defi Wonderland**

[Defi Wonderland](https://twitter.com/DeFi_Wonderland) is closely researching the implementation of tokens compatible with the xERC20 EIP. They have published a suggested implementation on their [GitHub](https://github.com/defi-wonderland/xTokens/tree/52d1e5c2e8671a05f8020ab61f5e204250d945d5/solidity/contracts), which recently was audited by members of [Creed](https://thecreed.xyz/), Shayan Eskandari, and Dominik Muhs.

This implementation has an [interface](https://github.com/defi-wonderland/xTokens/blob/52d1e5c2e8671a05f8020ab61f5e204250d945d5/solidity/interfaces/IXERC20.sol) for the xERC20 contract with eight core functions that the token issuer must implement. These are functions related to setting the LockBox contract (setLockbox), issuance limits for bridges (setLimits, mintingMaxLimitOf, burningMaxLimitOf, etc.), and the core mint and burn functions.

Other teams researching and developing the standard in collaboration with Connext include DappRadar and TokenSoft.

**Conclusion**

In the evolving world of interoperability between chains, a solution for moving data and assets among these chains is desperately needed. While bridges have filled the gap for the last couple years, they come with a variety of problems. The xERC20 standard is worth looking into as a promising solution.

To learn more about developments regarding the EIP, stay connected on the Ethereum Magicians [discussion](https://ethereum-magicians.org/t/erc-7281-sovereign-bridged-tokens/14979), or join the "xERC20 Builders" Telegram [group](https://t.me/+FSzAqZvIC_w2YTZh) to stay updated.