# Security and best practices

## #

Security and best practices

Before diving into development on zkSync Era, it's crucial to consider the following recommendations. These best practices will help you optimize your code, ensure security, and align with the unique characteristics of zkSync Era.

## #

Usecall over.send or.transfer

Avoid usingpayable(addr).send(x) /payable(addr).transfer(x) because the 2300 gas stipend may not be enough for such calls, especially if it involves state changes that require a large amount of L2 gas for data. Instead, we recommend usingcall .

Instead of:

payable ( addr) . send ( x) // or payable ( addr) . transfer ( x) Use:

( bool s, ) = addr. call{ value: x} ( "" ) ; require ( s) ; This converts thesend /transfer functionality tocall and avoids potential security risks outlined here.open in new window .

Be aware of reentrancy

While.call offers more flexibility compared to.send or.transfer , developers should be aware that.call does not provide the same level of reentrancy protection as.transfer /.send . It's crucial to adhere to best practices like the checks-effects-interactions pattern and/or use reentrancy guard protection to secure your contracts against reentrancy attacks. It can help ensure the robustness and security of your smart contracts on the zkEVM, even under unexpected conditions.

## #

Use the proxy pattern at the early stage of the protocol

zkSync Era is based on the zk-friendly VM. Thus, we offer a dedicated compiler responsible for transforming conventional Solidity and Vyper code into zkEVM bytecode.

While we have extensive test coverage to ensure EVM compatibility, issues may still appear. We will implement the patches for these in a timely manner.

To integrate a compiler bug fix, you need to recompile and upgrade your smart contract. We recommend using the Proxy pattern for a few months after your first deployment on zkSync Era, even if you plan to migrate to an immutable contract in the future.

zkSync Upgradeable plugin

- The hardhat-zksync-upgradeable plugin
- is now available to help you create proxies.

## #

Do not rely on EVM gas logic

zkSync Era has a distinctive gas logic compared to Ethereum. There are two main drivers:

- We have a state-diff-based data availability, which means that the price for the execution depends on the L1 gas price.
- zkEVM has a different set of computational trade-offs compared to the standard computational model. In practice, this means that the price for opcodes is different to Ethereum. Also, zkEVM contains a different set of opcodes under the hood and so the “gas” metric of the same set of operations may be different on zkSync Era and on Ethereum.

Note

Our fee model is being constantly improved and so it is highly recommendedNOT to hardcode any constants since the fee model changes in the future might be breaking for this constant.

## #

gasPerPubdataByte should be taken into account in development

Due to the state diff-based fee model of zkSync Era, every transaction includes a constant calledgasPerPubdataByte .

Presently, the operator has control over this value. However, in EIP712 transactions, users also sign an upper bound on this value, but the operator is free to choose any value up to that upper bound. Note, that even if the value is chosen by the protocol, it still fluctuates based on the L1 gas price. Therefore, relying solely on gas is inadequate.

A notable example is a Gnosis Safe'sexecTransaction method:

// We require some gas to emit the events (at least 2500) after the execution and some to perform code until the execution (500) // We also include the 1/64 in the check that is not send along with a call to counteract potential shortcomings because of EIP-150 require ( gasleft ( )

> = ( ( safeTxGas* 64 ) / 63 ) . max ( safeTxGas+ 2500 ) + 500 , "GS010" ) ; // Use scope here to limit variable lifetime and prevent stack too deep errors { uint256 gasUsed= gasleft ( ) ; // If the gasPrice is 0 we assume that nearly all available gas can be used (it is always more than safeTxGas) // We only subtract 2500 (compared to the 3000 before) to ensure that the amount passed is still higher than safeTxGas success= execute ( to, value, data, operation, gasPrice== 0 ? ( gasleft ( ) - 2500 ) : safeTxGas) ; gasUsed= gasUsed. sub ( gasleft ( ) ) ; // ... } While the contract does enforce the correctgasleft() , it does not enforce the correctgasPerPubdata , since there was no such parameter on Ethereum. This means that a malicious user could call this wallet when thegasPerPubdata is high and make the transaction fail, hence making it spend artificially more gas than required.

This is the case for all relayer-like logic ported directly from Ethereum and so if you see your code relying on logic like "the user should provide at X gas", then thegasPerPubdata should be also taken into account on zkSync Era.

For now, zkSync Era operators use honest values for ETH L1 price andgasPerPubdata , so it should not be an issue if enough margin is added to the estimated gas. In order to prepare for the future decentralization of zkSync Era, it is imperative that you update your contract.

## #

Use native account abstraction overecrecover for validation

Use zkSync Era's native account abstraction support for signature validation instead of this function.

We recommend not relying on the fact that an account has an ECDSA private key, since the account may be governed by multisig and use another signature scheme.

Read more aboutzkSync Era Account Abstraction support .

## #

Use local testing environment

For optimal development and testing of your contracts, it is highly recommended to perform local testing before deploying them to the mainnet. Local testing allows you to test your contracts in a controlled environment, providing benefits such as reduced network latency and cost.

We providetwo different testing environments designed for local testing purposes. These tools allow you to simulate the zkSync network locally, enabling you to validate your contracts effectively.

By incorporating local testing into your development workflow, you can effectively verify the behavior and functionality of your contracts in a controlled environment, ensuring a smooth deployment process to the mainnet.

For detailed instructions on configuring the local testing environment and performing tests using Mocha and Chai, refer to the dedicatedTesting page.

[ ]Edit this pageopen in new windowLast update: Contributors: [[Mickaël Schoentgen ,][albicodes ]]

Prev Interact with zkSync Era Next Useful Addresses