Smart Contracts

All of our smart contracts are available on GitHub:

- World ID Smart Contracts
- State Bridge Smart Contracts

If you're interested in using World ID and verifying proofs on-chain, see ourOn-Chain Verification guide .

Supported Chains

Chain Testnet Role Identity Availability Ethereum Goerli Canonical ~60 minutes after verification Optimism Optimism Goerli Bridged ~5 Minutes after Ethereum Polygon Mumbai Bridged ~40 Minutes after Ethereum Base (Goerli Only) Base Goerli Bridged ~5 Minutes after Ethereum Goerli

Find our smart contractaddress book here .

Architecture

This section offers a high-level overview of the various smart contracts that make up World ID. This structure (including state bridging) is replicated on testnets -- currently Goerli, Optimism Goerli, Polygon Mumbai, and Base Goerli.

Identity Managers:WorldIdIdentityManager

Identity Managers are only deployed on Ethereum. The Identity Manager contracts are responsible for managing the Semaphore instance. Worldcoin's signup sequencers call the Identity Manager contracts to add or remove identities from the merkle tree.

State Bridges:OpStateBridge /PolygonStateBridge

One State Bridge contract is deployed on Ethereum for each bridged chain. It publishes the root of the merkle tree to its configured chain's World ID contract, allowing proofs to be verified on that chain.

Bridged World ID:OpWorldId /PolygonWorldId

One World ID contract is deployed on each bridged chain, with an associated State Bridge contract on Ethereum. It is responsible for receiving merkle roots from its State Bridge contract, and verifying World ID proofs against those roots.

You can deploy your own State Bridge contract on Ethereum and Bridged World ID contract to any chain to bridge World ID to that chain permissionlessly.

World ID Router:WorldIdRouter

This is the contract you should interact with.

The World ID Router will route your call to the correct Identity Manager contract (Ethereum) or World ID contract (L2 Chains) based on thegroupId argument. This contract is proxied, so you will not need to update your code if the underlying contracts are upgraded.

Only Orb credentials are supported on-chain, so thegroupId must be1 .

Usage

TheverifyProof method of theWorld ID Router is used to verify proofs on-chain.

Arguments

- Name
- root
- Type
- uint256
- Required
- REQUIRED
- Description
- The World ID root to verify against.
- Name

- groupId
- Type
- uint256
- Required
- REQUIRED
- Description
- Determines which Credential Type to verify against. As only Orb credentials are supported on-chain, this must be1
- .

- ### Orb-Only groupId

- uint256
- internal
- immutable
- groupId
- =
- 1
- ;
- Copy
- Copied!
- Name
- signalHash
- Type
- uint256
- Required
- REQUIRED
- Description
- Thekeccak256
- hash of the signal to verify.

- ### signalHash

- abi.
- encodePacked
- (signal).
- hashToField
- ();
- Copy
- Copied!
- Name
- nullifierHash
- Type
- uint256
- Required
- REQUIRED
- Description
- The root of the merkle tree to verify against. This is obtained from the IDKit widget as a hex stringnullifier_hash
- , and must be converted to auint256
- before passing it to theverifyProof
- method.
- Name
- externalNullifierHash
- Type
- uint256
- Required
- REQUIRED
- Description
- Thekeccak256
- hash of theexternalNullifier
- to verify. TheexternalNullifier
- is computed from theapp_id
- andaction
- .

- ### externalNullifierHash

- externalNullifier
- =

- abi.
- encodePacked
- (abi.
- encodePacked
- (appId).
- hashToField
- ()
- ,
- action)
- externalNullifierHash
- =
- externalNullifier.
- hashToField
- ();
- Copy
- Copied!
- Read more about the External Nullifier inProtocol Internals
- .
- Name
- proof
- Type
- uint256[8]
- Required
- REQUIRED
- Description
- The zero-knowledge proof to verify. This is obtained from the IDKit widget as a hex stringproof
- , and must be converted to auint256[8]
- before passing it to theverifyProof
- method.

- **Unpacking Proof**

- viem
- ethers.js
- import
- { decodeAbiParameters }
- from
- 'viem'
- const
- unpackedProof
- =
- decodeAbiParameters
- ([{ type
- :
- 'uint256[8]'
- }]
- ,
- proof)[
- 0
-]
- Copy
- Copied!

Sybil resistance

While the World ID protocol makes it very easy to make your contracts sybil resistant, this takes a little more than just calling theverifyProof function. To make your contract sybil-resistant, you'll need to do the following:

- Store thenullifierHash
- of each user that has successfully verified a proof.
- When a user attempts to verify a proof, check that thenullifierHash
- is not already in the list of usednullifierHash
- es.

Here's an example function doing the above. You can also use theWorld ID starter kits to get started with sybil resistance.

/// @param root The root (returned by the IDKit widget). /// @param groupId The group ID /// @param signal An arbitrary input from the user, usually the user's wallet address /// @param nullifierHash The nullifier hash for this proof, preventing double signaling (returned by the IDKit widget). /// @param proof The zero-knowledge proof that demonstrates the claimer is registered with World ID (returned by the IDKit widget). function

verifyAndExecute (address signal , uint256 root , uint256 nullifierHash , uint256 [8] calldata proof) public { // First make sure this person hasn't done this before if (nullifierHashes[nullifierHash]) revert

InvalidNullifier ();

// Now verify the provided proof is valid and the user is verified by World ID worldId. verifyProof (root , groupId , abi. encodePacked (signal). hashToField () , nullifierHash , externalNullifierHash , proof);

// Record the user has done this, so they can't do it again (proof of uniqueness) nullifierHashes[nullifierHash] =

true ;

// Finally, execute your logic here, for example issue a token, NFT, etc... } Copy Copied!