Gas Usage & Costs

The Factory/Proxy pattern has a high setup cost and a low repeating cost. The total cost of onboarding **a user is $7.88 on Ethereum mainnet and $0.42 on Polygon zkEVM** networks. A **token mint costs $2.68 on Ethereum and $0.16 on Polygon zkEVM**.

Prices calculated for ETH: $2,000 USD

Step	**Gas**	**Ethereum** 25 gwei	**zkEVM** 1.5 gwei	**Repeats**
Dcentral Infrastructure Deployment	4,549,081	**$227.45**	**$13.65**	**Per chain**
Project Setup	157,675	**$7.88**	**$0.42**	**Per project**
Token Mint	53,510	**$2.68**	**$0.16**	**Per token**

Breakdown by contract

Dcentral Factory Contract Launch	2,440,494
ERC1155 Singleton Contract Launch	1,950,912
Proxy Contract Launch	140,209
Gas used for mint:	53,510

Contract Overview

The contract code is short and relies on several advanced smart contract properties. The effort took approximately 2 days to write, one day to unit test, and relies on the OpenZeppelin library, CREATE2 Factory pattern and Proxy pattern

The main components:

DcentralFactory.sol: This contract is responsible for creating the ERC1155Base contract and ERC1155Proxy contracts. It uses the create2 assembly function to deploy a new logic contract (ERC1155Base) with a unique address computed based on a provided salt and the creation code of the ERC1155Base contract. The deployed ERC1155Base contract's address is stored in the logicContract state variable. The contract also provides a function to deploy a new ERC1155Proxy contract, which requires the logic contract to be deployed.

ERC1155Base.sol: This contract implements the ERC1155 token standard, which allows for creating multiple token types within a single contract. It includes functions for minting and burning tokens and transferring tokens from one address to another. The contract is also Ownable, meaning it has an owner address with special permissions, such as the ability to mint and burn tokens. The contract includes an initialization function that can only be called once, which transfers ownership from the zero address to the address that calls the function.

ERC1155Proxy.sol: This contract acts as a proxy for an implementation contract provided during deployment. Any calls made to the proxy contract are forwarded to the implementation contract using a delegatecall, which means that the call is executed in the context of the proxy contract, allowing the proxy to use the implementation's code while maintaining its state. If the delegatecall fails, the proxy contract reverts the transaction.

Slither Contract Security Audit

Slither command: $ slither --print human-summary .

Number of optimization issues: 0
Number of informational issues: 28
Number of low issues: 0
Number of medium issues: 14
Number of high issues: 0

Use: Openzeppelin-Ownable
ERCs: ERC165

Command: `$ slither .`

High Risk

No High Risk issues reported by Slither

Medium Risk

No Medium Risk issues report by Slither

Low Risk

Assembly Usage

Severity: Informational
Risk: The use of assembly is error-prone and should be avoided.
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

1. **DcentralFactory.sol#30-43 uses assembly**

Mitigation: The assembly used is standardized create2 inline assembly.

2. **ERC1155Proxy.sol#16-35 uses assembly**

Mitigation:
It is based on OpenZeppelin's Transparent, Upgradeable Proxy with the following changes:

Implementation Differences:
Both functions are using low-level assembly code to delegate calls to an implementation contract. This is a common pattern in upgradeable contracts where the logic is separated from the data. Dcentral does not need an upgradeable pattern and will rely on the proxy to reduce gas needs.

Here's a breakdown of the differences:

Function Signature:
_delegate(address implementation): This is an internal function that takes an address as an argument. This address is the contract to which the call will be delegated.
fallback() external: This is a fallback function that gets executed if no other function matches the call. It doesn't take any arguments. It uses the state variable _impl as the contract to which the call will be delegated.

Memory Management:
In _delegate, the memory slot 0 is used to store the calldata. This is a bit risky because Solidity might use this slot for other things in the future, but it's currently safe.

In fallback, the memory slot pointed by 0x40 is used to store the calldata. This is the idiomatic way to get a free memory pointer in Solidity.

Implementation Address:
In _delegate, the implementation address is passed as an argument to the function.
In fallback, the implementation address is read from a state variable.

Error Handling:

Both functions use the same error handling. If the delegatecall fails, they revert the transaction and pass along the error message.

Both functions are doing the same thing in slightly different ways. The _delegate function is more flexible because it can delegate the call to any address, while the fallback function always delegates the call to the address stored in the state variable.

Different versions of Solidity are used

Severity: Informational
Risk: Different versions of Solidity may have different bugs or features.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

- ^0.8.0 (contracts/DcentralFactory.sol#3)
- ^0.8.0 (contracts/ERC1155Base.sol#3)
- ^0.8.0 (contracts/ERC1155Proxy.sol#3)

Mitigation: Prior to deployment, review the latest Solidity changes and compile at the latest stable release of Solidity recommended for deployment.

Pragma allows older versions of Solidity

Severity: Informational
Risk: Different versions of Solidity may have different bugs or features.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Mitigation: At deployment, set the Solidity version pragma to a static version.

Literals with too many digits

Severity: Information
Risk: Literals with many digits are difficult to read and review.
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

DcentralFactory.sol#11-20
DcentralFactory.sol#14-19
DcentralFactory.sol#22-28

Mitigation: Accept the risk. Humans will not be manually crafting/reviewing bytecode or salt values.

Contract Architecture Diagrams

Command: `$ slither -print call-graph .`

Architecture diagrams are too large to fit in this document:
Dcentral Factory
ERC1155 Base
ERC1155 Proxy

Deployment Instructions

1. Create and sign a transaction to deploy the DcentralFactory contract.
2. Rotate the r, s, v values of the transactions to random values.
3. Perform an ecrecover function on the r, s, v values.
4. Fund the address recovered in step 3 with enough eth to deploy the DcentralFactory contract.
5. Broadcast the transaction with the rotated r, s, v values in step 2.

The ecrecover function will always recover a valid address from r, s, v values but the recovered address is *highly unlikely* to be controlled by anyone. This is perfect, as we want to guarantee the first transaction from that account is always the DcentralFactory contract. This will keep the factory contract address consistent across EVM chains.

When the factory receives the salt + bytecode for the ERC1155Base logic contract will always compute the same create2 contract address.

CREATE addresses are created using:
```
new_address = hash(sender, nonce)
```

CREATE2 addresses are created using:
```
new_address = hash(0xFF, sender, salt, bytecode)
```

This guarantees Dcentral Infrastructure deploys to the same address across all EVM blockchains.

6. Create an arbitrary message. “Dcentral Infrastructure”
7. Sign the message, hash the value.
8. Deploy the ERC1155Base contract, `deployLogicContract`, using a salt cryptographically tied to a set of keys Dcentral controls.

Now, anyone may deploy the Dcentral singleton Factory and deploy the Dcentral Infrastructure to any chain, however we have a guarantee that the contracts deployed on that chain are exclusively tied to Dcentral. If the bytecode is ever changed or modified or redeployed it will have a completely different address.

If a contract deploys at the same address on all chains, it's easier to verify that the contract code is the same across all chains. This helps prevent scams where a malicious actor deploys a contract that looks similar but behaves differently on a different chain.

Knowing that a contract is at the same address on all chains can make it easier to interact with end users. They don't need to look up the address on each chain; they can use the same address everywhere.

Users and developers can have more confidence in a contract deployed at the same address on all chains. It shows that the contract's developers have taken the time to ensure consistency across chains, which can signify a well-designed and well-maintained contract.

Auditing contracts deployed at the same address on all chains is easier. Auditors can compare the contract code across chains to ensure it's the same, and they can also reach the state of the contract on different chains to look for any discrepancies.

If a contract deploys at the same address on all chains, then any off-chain systems or services that interact with the contract (like oracles, indexing services, etc.) only need to be set up once rather than once per chain. This can save time and resources.

Appendix - Open Zeppelin, RAW Slither Callouts

Slither is unable to discern accepted/mitigated risks. It will call out libraries like OpenZeppelin even if a human has already determined the risk to be acceptable or mitigated.

```
slither .
hayek@rothbard
'npx hardhat clean' running (wd: /home/hayek/proj/proxy-1155)
'npx hardhat clean --global' running (wd: /home/hayek/proj/proxy-1155)
'npx hardhat compile --force' running
Downloading compiler 0.8.20
Generating typings for: 15 artifacts in dir: typechain-types for target: ethers-v6
Successfully generated 42 typings!
Compiled 15 Solidity files successfully

INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256)
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication
on the result of a division:
	- denominator = denominator / twos
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)
	- inverse = (3 * denominator) ^ 2
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#116)
Math.mulDiv(uint256,uint256,uint256)
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication
on the result of a division:
	- denominator = denominator / twos
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)
	- inverse *= 2 - denominator * inverse
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#120)
Math.mulDiv(uint256,uint256,uint256)
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication
on the result of a division:
	- denominator = denominator / twos
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)
	- inverse *= 2 - denominator * inverse
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256)
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication
on the result of a division:
	- denominator = denominator / twos
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)
```

```
        - inverse *= 2 - denominator * inverse
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256)
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication
on the result of a division:
        - denominator = denominator / twos
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)
        - inverse *= 2 - denominator * inverse
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256)
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication
on the result of a division:
        - denominator = denominator / twos
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)
        - inverse *= 2 - denominator * inverse
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256)
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication
on the result of a division:
        - denominator = denominator / twos
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#101)
        - inverse *= 2 - denominator * inverse
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256)
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) performs a multiplication
on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#104)
        - result = prod0 * inverse
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[]
,bytes).response (node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#478)
is a local variable never initialized
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).r
esponse (node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#456) is a
local variable never initialized
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[]
,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#483) is
a local variable never initialized
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).r
eason (node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#460) is a local
variable never initialized
```

```
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes)
(node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#447-466) ignores
return value by IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data)
(node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#456-464)
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[]
,bytes) (node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#468-489)
ignores return value by
IERC1155Receiver(to).onERC1155BatchReceived(operator,from,ids,amounts,data)
(node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#477-487)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Address._revert(bytes,string)
(node_modules/@openzeppelin/contracts/utils/Address.sol#231-243) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#236-239)
Strings.toString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#19-39) uses
assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts/utils/Strings.sol#25-27)
	- INLINE ASM (node_modules/@openzeppelin/contracts/utils/Strings.sol#31-33)
Math.mulDiv(uint256,uint256,uint256)
(node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-134) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts/utils/math/Math.sol#62-66)
	- INLINE ASM (node_modules/@openzeppelin/contracts/utils/math/Math.sol#85-92)
	- INLINE ASM (node_modules/@openzeppelin/contracts/utils/math/Math.sol#99-108)
DcentralFactory._deployLogicContract(bytes32,bytes) (contracts/DcentralFactory.sol#30-43)
uses assembly
	- INLINE ASM (contracts/DcentralFactory.sol#35-40)
ERC1155Proxy.fallback() (contracts/ERC1155Proxy.sol#16-35) uses assembly
	- INLINE ASM (contracts/ERC1155Proxy.sol#20-34)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity are used:
	- Version used: ['^0.8.0', '^0.8.1']
	- ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/IERC1155.sol#4)
	- ^0.8.0
(node_modules/@openzeppelin/contracts/token/ERC1155/IERC1155Receiver.sol#4)
	- ^0.8.0
(node_modules/@openzeppelin/contracts/token/ERC1155/extensions/IERC1155MetadataURI.s
ol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
```

```
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SignedMath.sol#4)
        - ^0.8.0 (contracts/DcentralFactory.sol#3)
        - ^0.8.0 (contracts/ERC1155Base.sol#3)
        - ^0.8.0 (contracts/ERC1155Proxy.sol#3)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-us
ed
INFO:Detectors:
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows
old versions
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol#4) allows old versions
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts/token/ERC1155/IERC1155.sol#4) allows old versions
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts/token/ERC1155/IERC1155Receiver.sol#4) allows old
versions
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts/token/ERC1155/extensions/IERC1155MetadataURI.s
ol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) allows old
versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old
versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old
versions
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4) allows old versions
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#4) allows
old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SignedMath.sol#4)
allows old versions
Pragma version^0.8.0 (contracts/DcentralFactory.sol#3) allows old versions
Pragma version^0.8.0 (contracts/ERC1155Base.sol#3) allows old versions
Pragma version^0.8.0 (contracts/ERC1155Proxy.sol#3) allows old versions
solc-0.8.20 is not recommended for deployment
```

```
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256)
(node_modules/@openzeppelin/contracts/utils/Address.sol#64-69):
	- (success) = recipient.call{value: amount}()
(node_modules/@openzeppelin/contracts/utils/Address.sol#67)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string)
(node_modules/@openzeppelin/contracts/utils/Address.sol#128-137):
	- (success,returndata) = target.call{value: value}(data)
(node_modules/@openzeppelin/contracts/utils/Address.sol#135)
Low level call in Address.functionStaticCall(address,bytes,string)
(node_modules/@openzeppelin/contracts/utils/Address.sol#155-162):
	- (success,returndata) = target.staticcall(data)
(node_modules/@openzeppelin/contracts/utils/Address.sol#160)
Low level call in Address.functionDelegateCall(address,bytes,string)
(node_modules/@openzeppelin/contracts/utils/Address.sol#180-187):
	- (success,returndata) = target.delegatecall(data)
(node_modules/@openzeppelin/contracts/utils/Address.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
DcentralFactory.getLogicContractAddress(bytes32) (contracts/DcentralFactory.sol#11-20) uses
literals with too many digits:
	- _computeAddress(salt,address(this),type()(ERC1155Base).creationCode)
(contracts/DcentralFactory.sol#14-19)
DcentralFactory.deployLogicContract(bytes32) (contracts/DcentralFactory.sol#22-28) uses
literals with too many digits:
	- _deployLogicContract(salt,type()(ERC1155Base).creationCode)
(contracts/DcentralFactory.sol#27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:. analyzed (15 contracts with 85 detectors), 42 result(s) found
```