**A guideline to create a TRC-20 Token on TRON (Nile Testnet)**

This comprehensive guide demonstrates how to deploy and interact with an upgradeable TRC-20 token on the TRON Nile testnet using a proxy pattern. The token will feature standard ERC20 capabilities, including minting, burning, and initial supply management.

**Project Setup**

**Prerequisites**

* **Node.js (v20+ recommended)**
* **TronBox**
* **TronLink wallet** (configured with Nile testnet)

**Installation**

Clone the repository and install dependencies:

*git clone https://github.com/aziz1975/trc20-proxy.git*

*cd trc20-proxy*

*npm install*

Create a .env file in the project root which contains the following:

*PRIVATE\_KEY\_NILE=your\_private\_key*

*FULL\_NODE\_NILE=https://nile.trongrid.io*

*PROXY\_ADDRESS=your\_deployed\_proxy\_address (after deployment)*

**Smart Contracts**

**Proxy.sol**

A lightweight proxy contract delegating all calls to a separate implementation contract:

* Stores implementation contract address
* Delegates all calls, enabling upgradeability without losing data

**MyToken.sol**

Token implementation based on OpenZeppelin’s ERC20Upgradeable:

* **initialize(name, symbol, initialSupply):** Initializes the token
* **mint(to, amount):** Creates new tokens
* **burn(from, amount):** Destroys tokens

**Deployment**

Deploy the logic and proxy contracts to Nile:

*npx tronbox migrate --network nile*

This deploys:

* **MyToken.sol** (logic implementation)
* **Proxy.sol** initialized with parameters ("AHM TRC20 Token", "AHM", "1,000,000 tokens")

**Testing**

Test your deployed token contract using the provided script:

1. Update your .env with your created proxy token address
2. *node testToken.js*

This script performs:

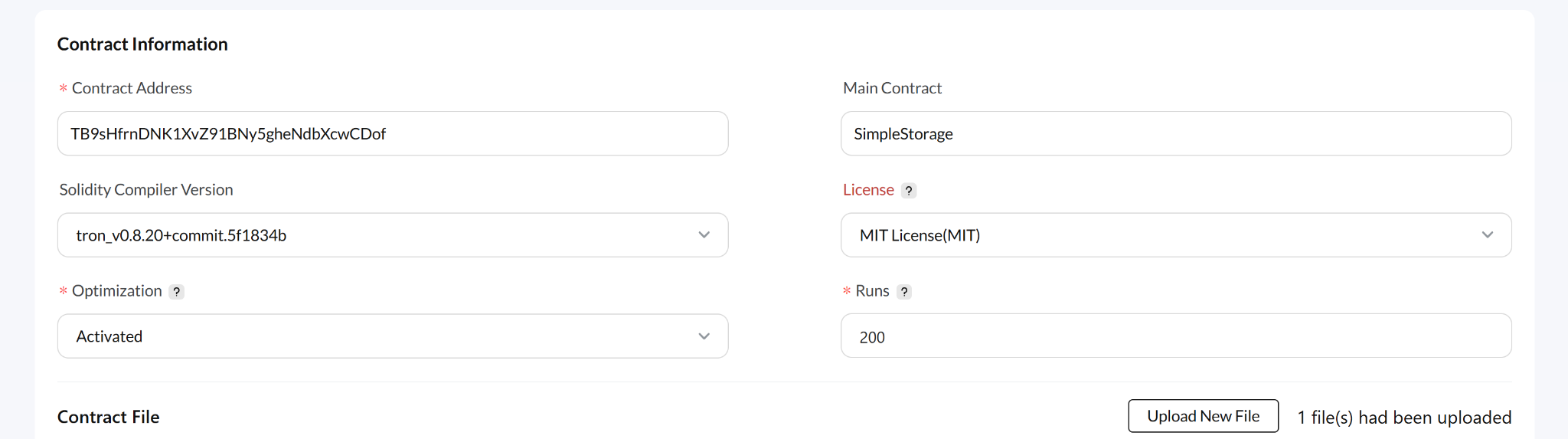
* Token metadata retrieval (name, symbol, decimals)
* Total supply and owner balance verification
* Token transfer, minting, and burning operations

**Contract Verification on Tronscan**

You need to perform two verifications: one for the proxy contract and another for the token implementation contract.

Go to **Contract Verification** (for nile it is “https://nile.tronscan.org/#/contracts/verify”)

1. **Contract Address**: Enter the deployed contract’s address.
2. **Main Contract**: If the source file contains multiple contracts (e.g., library code), select the **Main Contract** name that matches the deployed contract.
3. **Solidity Compiler Version**: Choose the exact compiler version used to compile your contract (match the version used during deployment).
4. **License**: Select the appropriate SPDX **License** identifier (e.g., MIT, Unlicense) as in your source code​.
5. **Optimization**: Set **Optimization** to “Activated” if you enabled it during compilation (and enter the same **Runs** value used); if you did not use optimizer, select “Not Activated” and enter “0” in the **Runs** field.
6. **Upload contract file(s):** Upload (.sol file).
7. **Verify And Publish**: Click the **Verify And Publish** button. TronScan will compile the code with the given parameters and compare it to the on-chain bytecode – if everything matches exactly what was used at deployment, the contract will be marked as verified​



## **Proxy Contract Verification**

* If your contract has any “**import**” statement, you do need to flatten them using the following command:

*npx tronbox flatten contracts/SmartContract.sol > flattened-SmartContract.sol*

* **You should absolutely have exactly one SPDX license identifier as the very first line of your *flattened* Solidity file.**
* Then follow the same steps as mentioned above for the standard contract.

When verifying contracts, you might encounter the error:

*“verification failed. Please confirm the correct parameters and try again*.”

**Solution:**

* Ensure the correct compiler version (v0.8.23) and optimizer settings (200 runs) match exactly.
* **Proxy.sol** requires two constructor parameters:
  + **Implementation address** (MyToken logic contract address)
  + **Initialization data** (encoded ABI data from deployment script)
* **MyToken.sol** has no constructor arguments.

If verification issues persist:

* Visit: [Contact Tronscan Support](https://nile.tronscan.io/#/tools/contactUs)
* Raise a ticket through **Others** or contact the [Telegram developer group](http://t.me/TronOfficialDevelopersGroupEn).

**Adding a Logo to Your Token**

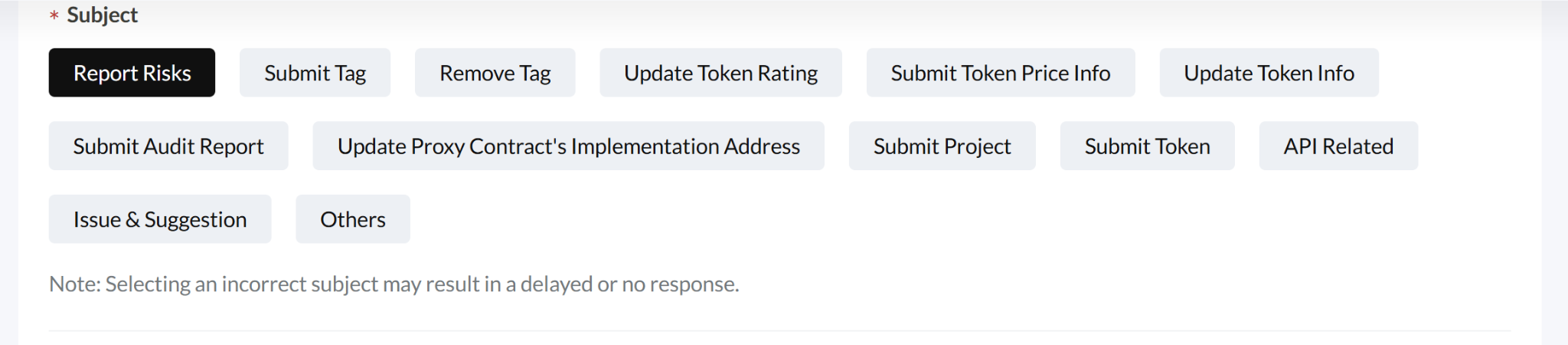
1. Visit [Token Update Page](https://nile.tronscan.io/#/wallet/tokensCreate)
2. Select your token and update your logo.

**Updating Token Metadata**

Update metadata (name, description, website, etc.):

1. Visit [Token Update Page](https://nile.tronscan.io/#/wallet/tokensCreate)
2. Select your token and update relevant information.

If you encounter any issues while updating the token, [raise a ticket](https://nile.tronscan.org/#/tools/contactUs) in Tronscan and select the appropriate Subject:



**TronLink Integration**

To add the token in TronLink:

* Wait approximately 5–10 minutes for TronLink synchronization.
* If synchronization issues occur, seek help in the [Telegram developer group](http://t.me/TronOfficialDevelopersGroupEn).

**Viewing Your Token on Tronscan**

To view your token:

* Use the search feature on [Tronscan homepage](https://nile.tronscan.io/) and enter the token name or contract address.

**Adding Tokens to Other Wallets (e.g., TrustWallet)**

Wallet integration rules vary:

* Contact the specific wallet’s customer service or documentation to determine their integration process.

**Resources**

* [TRON Nile Testnet Explorer](https://nile.tronscan.io/)
* [OpenZeppelin Upgradeable Contracts](https://docs.openzeppelin.com/upgrades-plugins/writing-upgradeable)
* [TronBox Official Documentation](https://tronbox.io/docs/quickstart)

**Troubleshooting**

* Verify .env configuration values (private keys, RPC URLs)
* Ensure correct compiler and optimizer settings for contract verification
* Consult TRON's [developer Telegram group](http://t.me/TronOfficialDevelopersGroupEn) for ongoing issues

Following these steps, you'll have successfully deployed an upgradeable TRC-20 token on TRON's Nile Testnet, verified your contracts, and integrated your token seamlessly into various platforms.

Explore the full working example in our [TRC-20 repository](https://github.com/aziz1975/trc20-proxy).