THE DUBLR WHITEPAPER:

The Dublr ERC20 token, and the Dublr decentralized exchange

Hiroshi Yamamoto
hiroshi.yamamoto.dublr@protonmail.com

2022-10-14

Abstract: Dublr is an ultra-secure fungible token smart contract that implements the ERC20 token standard, as well as several extensions to improve usability and to mitigate a number of known ERC20 security vulnerabilities. As a consequence of close attention to security, Dublr may be the most secure and functional of all currently-available ERC20 tokens. Additionally, Dublr implements its own built-in decentralized exchange (DEX) for decentralized finance (DeFi), so that Dublr tokens can be bought and sold using the Dublr smart contract itself: Dublr has the unique property that it is both a token and its own DEX. Uniquely, the total supply of Dublr tokens is generated on-demand, rather than by ICO. New tokens are minted whenever buyer demand exceeds seller supply on the DEX below the current mint price (the mint price sets an upper bound on how fast the price of the token can grow). The mint price increases exponentially on a compound interest curve, with a doubling time of 90 days, hence the name "Dublr". The economics of this minting behavior should give rise to interesting market dynamics for this token.

Background

Security and usability problems with ERC20

Since the ERC20 fungible token standard was ratified, thousands of ERC20 tokens have been created on EVM-compatible blockchains. Reference implementations of the ERC20 standard, such as the OpenZeppelin implementation, have made it trivially easy to create and launch a new fungible token. However, the ERC20 standard has a number of innate security problems: a double-spend race condition allows the allowance mechanism to be abused in frontrunning attacks against the ERC20 standard itself, and a tendency for some decentralized applications (dapps) to ask the user to grant unlimited allowances has led to hundreds of millions of dollars' worth of tokens being siphoned out of users' accounts, across many different ERC20 tokens, usually due to vulnerabilities in dapps rather than the token implementation. Additionally, hundreds of millions of dollars have been lost by users attempting to transfer tokens to the address of a contract that is not designed to receive incoming token transfers.

Fig 1: The icon for the DUBLR ERC20 token

As a result of these issues, there is a need for a better ERC20 implementation that protects users from bad actors, and from their own mistakes. *The Dublr smart contract implements a greatly improved ERC20 implementation that can mitigate these issues through API extensions, and/or by optionally breaking compatibility with ERC20 to prevent these issues*. The Dublr ERC20 implementation is contained in a new token library known as OmniToken (not to be confused with the OMNI ERC20 token).

SEC regulatory problems with ERC20 tokens

Additional issues arise around the standard model of how ERC20 token supply is generated and distributed, which is the ICO (Initial Coin Offering). An ICO is a process or event in which a company (especially a start-up) attempts to raise capital by selling a new cryptocurrency, which investors may purchase in the hope that the value of the cryptocurrency will increase, or to later exchange for services offered by that company. In this process, the founders usually set aside some large number of tokens for themselves, effectively generating profit out of thin air. The US Securities and Exchange Commission has argued that the ICO model of token distribution satisfies the Howey test to determine whether an asset is a security, since an ICO can lead investors to believe they will generate profits through the efforts of others. If a cryptocurrency is declared to be a security, then the distributors of that cryptocurrency are bound by US law to properly implement KYC (Know Your Customer), collecting and verifying passport information from all users, in order to comply with AML (Anti-Money Laundering) laws. KYC is effectively impossible for basically every cryptocurrency deployed today, which effectively renders illegal any cryptocurrency that has been declared by the SEC to be a security. Many countries defer to the US SEC over these matters, therefore this is an issue with international impact.

To mitigate this problem, Dublr implements its own decentralized exchange (DEX), which generates all token supply on demand, when demand exceeds supply below a current "mint price". No tokens were generated on smart contract deployment, and no tokens are distributed via ICO.

Additionally, the SEC has argued that the possession by owners of assets of the same class being distributed to investors could be construed as sufficiently motivating of owners' future efforts that they are likely to trigger the Howey test, causing the asset to be declared to be a security.

The creator of Dublr has minted zero tokens for their own use. This is in sharp contrast to the launch of most cryptocurrency tokens, where the creator(s) of the token usually mint a large number of tokens for themselves during token deployment, with the purpose of self-enrichment. Pre-minting undermines their users' trust in the token, because users know that they are buying into an ecosystem dominated by a few "whales" (holders of an enormous number of tokens) who spent nothing to obtain their holdings. To avoid this, and the securities implications of token ownership by creators, the total supply of Dublr tokens will be community-owned. In fact, it is difficult to verify, but Dublr may be the only ERC20 token contract that has ever been deployed without minting tokens for the contract creator(s).

SEC regulatory problems with Ethereum

SEC Chairman Gary Gensler recently declared that the entire Ethereum ecosystem may constitute a security, because the switch from proof of work to proof of stake triggers the Howey test, since tokens are staked with the expectation of profit. Additionally, he stated that the entire Ethereum ecosystem should be considered to fall under US jurisdiction, because the majority (40.1%) of Ethereum validator nodes are currently hosted on US-based IP addresses. This could mean that any token launched on Ethereum could be construed to be a security, effectively making the token illegal. The SEC has already retroactively sued one creator of a cryptocurrency that launched in 2018, four years before "the merge" (the switch to proof of stake). This lawsuit was launched immediately after the merge.

Consequently, due to the legal and regulatory threats to the entire Ethereum ecosystem, *the Dublr smart contract was not launched on Ethereum, but rather on the Polygon blockchain.* Polygon is closely associated with Ethereum, and is fully EVM-compatible, but it operates its own servers and has its own block validation algorithm. Critically, Polygon does not have a core set of fixed validator nodes, and does not rely solely on proof of stake for validation, but rather validators are voted in and out from a worldwide pool, so that the set of validators is constantly changing. The validation algorithm is a hybrid that is not a pure proof of stake algorithm.

(Note that while every effort has been made to ensure that Dublr never functions as a security, the SEC can make any declaration that they want, and they tend to make the majority of their rulings through lawsuits, to establish legal precedent, rather than by going through the proper legislative process to democratically establish securities laws and rulings. Therefore, no guarantee can be made that Dublr will never be construed to be a security by the US government.)

The Dublr dapp

The frontend of the Dublr exchange is the Dublr decentralized application (dapp), which is hosted using GitHub Pages, at https://dublr.github.io/ (Fig 2). The Dublr dapp is the primary means for interacting with the Dublr DEX. The source code of the dapp is available on GitHub under an MIT license.

The Dublr dapp needs to be connected to a cryptocurrency wallet such as MetaMask to function. Any wallet that supports the WalletConnect protocol should work.

Dublr tokens (ticker: DUBLR) can only be bought using network currency, i.e. MATIC, since Dublr is deployed on the Polygon network. Users need to transfer native Polygon MATIC tokens (*not* wrapped ERC20 MATIC tokens on the Ethereum network) into their cryptocurrency wallet before they can buy DUBLR tokens on the Dublr DEX.

The parameters of the `buy` and `sell` functions are exposed in the dapp frontend, allowing users to easily buy tokens and list them for sale without having to manually call the backend API (Fig 2).

The dapp frontend's "Buy" tab fully simulates the Dublr DEX's `buy` function, in order to be able to generate an accurate count of DUBLR tokens that would be bought given some amount of MATIC tokens, whether by buying sell orders or via minting. This simulation is used to calculate the minimum number of tokens that would be expected to be bought with a given maximum amount of slippage, expressed as a percentage. This allows the Dublr DEX to prevent frontrunning attacks, wherein a malicious actor could observe a buy request being submitted, and then submit their own competing buy request with a higher gas price, allowing the malicious transaction to force the original transaction to buy tokens at a higher price. It also prevents inadvertent slippage due to race conditions between multiple legitimate buyers.

The Dublr DEX

The Dublr decentralized exchange (DEX) implements a handful of functions for buying and listing for sale Dublr tokens. Only the sell side of the market is implemented in the exchange, i.e. buyers can buy orders listed by sellers, but they can't list their own buy orders on the exchange. All buys are market orders.

Token holders may list their tokens for sale on the Dublr DEX using the `sell(price, amount)` function. The parameters specify the list price, and the number of DUBLR tokens that should be listed for sale. The price is given in "NWC per DUBLR", where NWC is a generic ticker for the network currency (MATIC). Note

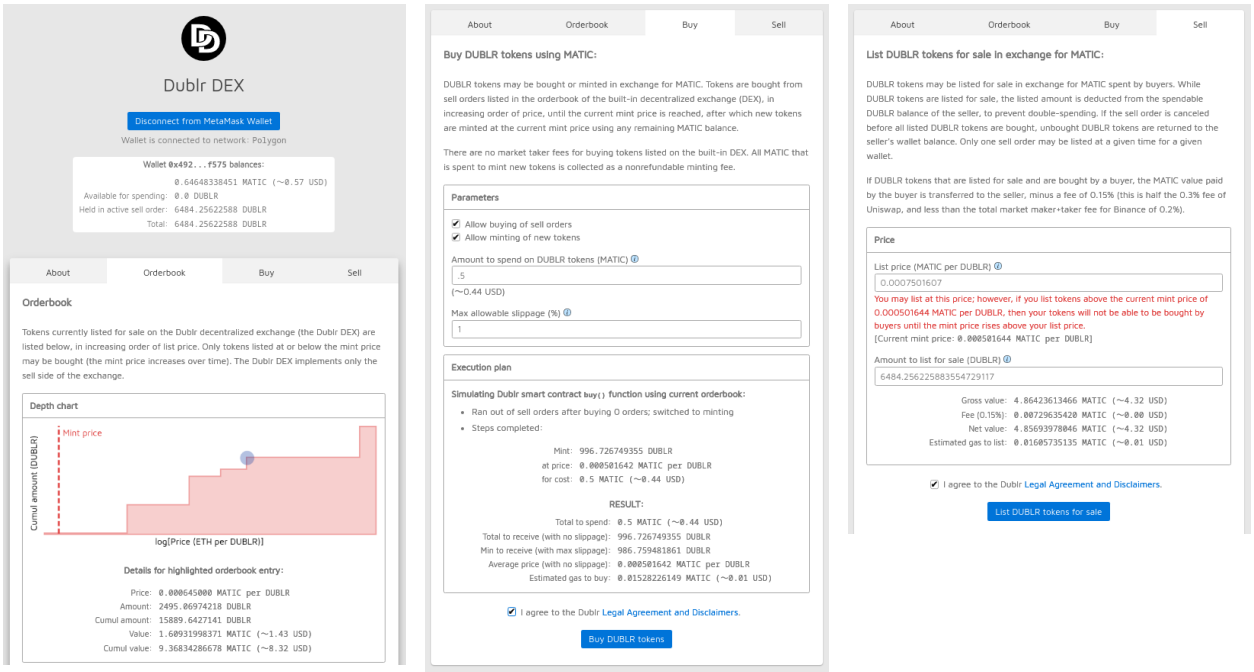

Fig 2: The DUBLR dapp, showing the depth chart from the orderbook, the buy function, and the sell function

that this same price would be listed in the form DUBLR/NWC on most exchanges, since exchanges list the base currency last, but the actual price ratio is the reciprocal of this notation.

Buyers may buy tokens on the exchange using the `buy(minTokens, allowBuying, allowMinting)` payable function. Some amount of NWC (MATIC) must be attached to the function when it is called, and the DEX will proceed to buy sell orders in increasing order of price, until either the buyer's attached payment runs out, or there are no more sell orders listed on the DEX, or until the current *mint price* is reached. If the sell orders run out or the mint price is reached, then new tokens are minted for the buyer at the current mint price using the remaining balance of the buyer's payment, increasing total supply. Any balance that is insufficient to purchase one DUBLR token is then sent back to the buyer as change.

The `minTokens` parameter allows the user to specify how many tokens they expect to receive for their payment, at a minimum, in order to limit slippage and to prevent frontrunning attacks. The `allowBuying` and `allowMinting` parameters allow the user to ignore the sell orders on the exchange (if they have a reason to do that), or ensure that they do not purchase tokens that are minted at an exorbitant price (the mint price grows exponentially, so will eventually become exorbitant).

Finally the DEX includes functions for fetching the orderbook, finding the current cheapest sell order, and canceling the current sell order. These are all accessible via the dapp frontend.

The Dublr DEX source code is available on GitHub under an MIT license.

Dublr DEX Fees

When a buyer buys a sell order, 0.15% of the value of the sell order is collected as a market fee, and the remaining MATIC balance is sent from the buyer to the seller, while the purchased DUBLR tokens are transferred from the seller to the buyer. This 0.15% fee is lower than Uniswap's standard 0.30% fee, and lower than Binance's current 0.2% total market fee (the sum of the market maker of 0.1% and the market taker fee of 0.1%).

If new tokens are minted, the amount spent by the buyer to mint new tokens is collected as a minting fee, in exchange for the DUBLR tokens they receive.

The Dublr ERC20 implementation

Dublr's ERC20 implementation is known as OmniToken. The Dublr DEX extends OmniToken to provide exchange functionality, however OmniToken may be used standalone as a powerful and secure token implementation. OmniToken implements the following APIs:

- ERC20, for standard token ownership, balance-checking, and transfer of tokens.
- ERC1363, for preventing token loss due to a token being sent to a contract that cannot receive the tokens, by requiring the receiver to implement a given interface, and for notifying the spender of tokens that the tokens have been spent.
- ERC4524, for preventing token loss due to a token being sent to a contract that cannot receive the tokens, by requiring the receiver to implement a given interface if it is not an EOA or non-contract wallet (this is a simpler version of ERC1363, and also allows sending to wallet addresses rather than just contracts).
- EIP2612, the ERC20 permit extension for signed approvals of allowances, via EIP712 secp256k1 signatures.

Extra OmniToken security measures

OmniToken has a very strong security model that improves over the ERC20 standard in several ways, by the addition of secure allowance management APIs, and by optionally deviating from the ERC20 standard in order to increase security. Note that all deviations from the ERC20 standard are disabled by default, but may be enabled by the contract owner after deployment, if the need arises due to a security incident.

The OmniToken implementation of EIP2612 is more secure than most other implementations, because the `chainId` is inserted dynamically into `DOMAIN_SEPARATOR` every time it is requested, which prevents sidechain replay attacks. In most other implementations, `DOMAIN_SEPARATOR` is implemented as a constant, calculated from a fixed `chainId`, rather than as a dynamic function, as it is implemented in OmniToken.

The APIs added by OmniToken for mitigating the approval double-spend race condition vulnerability in ERC20 comprise the following:

- The `permitting` extension proposed and adopted by OpenZeppelin, consisting of the following API functions:
  ```
  increaseAllowance(amount)
  decreaseAllowance(amount)
  ```
- The atomic compare-and-set permit approval mechanism proposed for preventing the approve/transferFrom vulnerability in ERC20, consisting of the following API function, which rejects the approval if the current approval value is not equal to `currentValue`:
  ```
  approve(spender, currentValue,
      newValue)
  ```
  Note that this proposal adds two new events, `Transfer` and `Approval`, which were renamed to `TransferInfo` and `ApprovalInfo` in the OmniToken implementation, since events of this name already exist in the ERC20 standard, and the Ethers library cannot handle contracts that have multiple events of the same name but different parameter types.
- A modified version of the time-limited token allowances mechanism proposed for reducing the chance that a forgotten allowance could be sent to an attacker by means of a clickjacking attack or compromised dapp. Note that OmniToken deviates from this proposal as it uses seconds rather than number of blocks for expiration time, since inter-block time intervals can vary. Allowances may be set specifically each time an allowance is granted, using the following API function:
  ```
  approveWithExpiration(spender,
      value, expirationTime)
  ```
  The expiration time of an allowance (and the allowance amount) may be read using the following API function:
  ```
  allowanceWithExpiration(owner,
      spender)
  ```

Additionally, OmniToken implements a number of nonstandard behaviors, which increase security. These behaviors are disabled by default, because they decrease ERC20 compatibility, but they may be each be selectively enabled by the contract owner:

- OmniToken can be configured to prevent tokens from ever being sent to a contract address, so that tokens can only ever be sent to an EOA wallet address, preventing accidental burning of tokens by sending to an address that cannot accept tokens.
- OmniToken can be configured to force users to set an allowance to zero before it is able to be set to a nonzero value, when switching the allowance between two nonzero values. This prevents the allowance race condition attack.
- A default expiration time may be configured for allowances granted through the standard ERC20 API.
- OmniToken can be configured to reject unlimited allowances of value 2^{256}-1. This value is used by a number of dapps to grant unlimited allowances to a spender. This leaves the entire balance of the accountholder vulnerable to theft, if the dapp is insecure.

Collectively, these security extensions and vulnerability mitigation measures make OmniToken an extremely secure and versatile foundation for token implementation.

The OmniToken source is available on GitHub under an MIT license.

Security

The security of both the Dublr DEX layer and the underlying OmniToken ERC20 library has been subjected to extreme levels of scrutiny, to identify any and all bugs or vulnerabilities:

- The Checks-Effects-Events-Interactions pattern is used everywhere in Dublr and OmniToken, without exception, to prevent reentrancy attacks. This is enforced via function modifiers: `stateUpdater` for functions that modify core account state, and `extCaller` for functions that call other contracts. A `stateUpdater` cannot be called deeper in the call stack than an `extCaller`.
- The Dublr and OmniToken code is extensively unit-tested.
- The Dublr and OmniToken code has been fully audited by two different 3rd-party auditing companies, SolidProof and Omniscia, and has

passed both audits with the highest standard of security rating.

- The Dublr and OmniToken code and API are copiously documented using the NatSpec rich comment syntax, so that all functions, function parameters, events, and event parameters are explained in EtherScan and in the source code. This reduces confusion about how to properly and safely call the API. The vast majority of Solidity contracts are developed without even a fraction of the number of comments. Additionally, the code within each function is liberally commented.
- Function parameter values and invariants are checked carefully, and transactions will revert if expectations are not met.
- All token APIs (ERC20, ERC1363, ERC4524, and EIP2612) can be individually enabled or disabled by the contract owner/deployer, in case a security problem is discovered with one of the APIs.
- Several classes of vulnerability are prevented by using a recent version of the Solidity compiler (0.8.17) to compile Dublr:
    - Solidity 0.5.0 prevented short address attacks.
    - Solidity 0.6.0 prevented "phantom function calls", where a contract does not define a required function, and the fallback function is called instead. Phantom function calls have been responsible for hundreds of millions of dollars of lost tokens in other contracts. The `receive` and `fallback` payable functions are not defined by OmniToken, to prevent triggering phantom function call issues in other contracts.
    - Solidity 0.8.0 prevented overflow and underflow attacks by utilizing checked arithmetic by default.

Economics of the mint price

The Dublr smart contract mints new tokens when insufficient tokens are listed for sale below the mint price to fulfill a buyer's request to buy DUBLR tokens. Minting tokens is an inherently deflationary activity, which is why this behavior effectively sets an upper bound on how fast the price of DUBLR tokens may grow. Note that there is no lower bound on price, because tokens may be listed for sale at any time at any price.

The mint price is the price at which new tokens are minted. The mint price grows exponentially, on a compound interest curve, with a doubling time of 90 days. The following polynomial approximation of the exponential function is used to calculate the current mint price, given the number of seconds since contract deployment, using fixed point algebra.

$$e^x = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n$$

The value n=10 is sufficient to give an accuracy to within 3% per doubling period.

Total supply is permanently fixed by the smart contract after 30 doubling periods, ~7.5 years at 90 days per doubling period.

The behavior of the mint price should give rise to some interesting market dynamics, from an economics point of view.

Note that nothing in this whitepaper should be construed to be a promise of future profits -- no future profits are promised or even suggested -- or to be a prediction of future market behavior. Despite the fact that the mint price curve is fixed, The exact shape of the future price curve of the DUBLR token is unknown and unknowable, and is subject to market forces.

Nevertheless, it is straightforward to conclude that if there is any initial growth in token price, it must be fueled by buyer demand for tokens outstripping the supply of tokens available from sellers, causing new tokens to be minted at the current mint price; whereas eventually, the mint price will become exorbitant, and/or many tokenholders may list their tokens for sale, effectively fixing total supply, because no new tokens will be minted. If or when that point is reached, DUBLR will trade just like any other ERC20 token, with a floating price dependent upon demand vs. supply. This transition in market behavior of the DUBLR token, from minting new token supply at the growing mint price to trading over a fixed supply, will probably occur long before the total supply becomes forcibly fixed by the smart contract, after 30 doubling periods.

The foregoing postulation about market dynamics does not constitute financial guidance or a financial prospectus, and must not be taken to be a reliable prediction of future market performance.

Contract deployment information

Dublr DEX deployment address (Polygon mainnet):
`0x3D739A2db4F42632ca090a7a6713a9A62DB994C0`
(`Deployment transaction`)

Deployed from git commit:
`eb50917365bbbb0d948efe656610c5abe06aa3d8`
(The source code of the deployed Dublr smart contract can be verified to be the same as the source code in the Dublr GitHub repository using PolygonScan)

Deployment date:
`2022-10-14, 22:06 UTC`
(This starts the clock on the mint price schedule)

Initial mint price:
`0.0005 MATIC per DUBLR`
(`initialMintPriceNWCPerDUBLR_x1e9 == 500000`)

Initial supply:
Zero DUBLR tokens were minted by the creator of Dublr, in order for DUBLR to not be considered a security. All supply is minted on demand by buyer requests.

Deployment proxy:
The Dublr smart contract is intentionally deployed as a non-proxied contract, so the code is not changeable or upgradeable by the creator of Dublr after deployment. Non-proxied contracts are far more secure than proxied contracts, as long as the code has passed extensive testing and thorough third-party security audits, because non-proxied contract can operate in a trustless way.

Conclusion

The Dublr smart contract implements both an extremely secure ERC20 token, DUBLR, via a new token library, OmniToken, and an extremely secure decentralized exchange, the Dublr DEX, for buying and selling DUBLR tokens. When the DEX does not have enough supply of tokens for sale below the current mint price, new tokens are minted for the buyer at the current mint price. The mint price grows exponentially with a doubling time of 90 days. A frontend for the Dublr DEX, the Dublr dapp, is hosted on GitHub Pages at https://dublr.github.io/ . This dapp allows DUBLR tokens to be easily bought or sold for Polygon MATIC tokens.

Developer

The Dublr core developer is Hiroshi Yamamoto, who has a PhD in computer science from MIT, and 35 years of software development experience.

Legal Disclaimers

- The name "Dublr" describes only the growth of the mint price, not the profitability of DUBLR tokens, or the growth of any fair market value of DUBLR tokens.
- The growth of the mint price sets a hard upper bound on how fast the price of DUBLR can grow relative to the network currency (MATIC), enforced by increasing total supply of tokens to meet demand whenever the demand outstrips the supply of tokens for sale below the mint price. There is no lower bound on price, and minting is an inherently deflationary activity, so there are no guarantees or promises, express or implied, about the profitability of purchasing DUBLR tokens.
- The purchasing, sale, and use of DUBLR tokens is entirely at the purchaser's own risk. DUBLR tokens may not be able to be sold without incurring loss, or may not be able to be sold at all if there is insufficient demand.
- DUBLR Tokens may not be used for any illegal purpose, including money laundering.
- Collected fees will not be used to fund ongoing development, marketing, or any other action beneficial to DUBLR token holders, and cannot be used to fund ongoing maintenance or improvement of the Dublr smart contract code, since no changes can be made to the deployed Dublr contract code after deployment. Therefore, any MATIC spent to mint or sell DUBLR tokens does not constitute investment in a common enterprise.
- By buying, selling, or using DUBLR tokens, you signify that you agree to the full Dublr Legal Agreement and Disclaimers.