Blockchain Meets Securities: A Scalable Tokenization Framework

REINA KE XIN LI, Department of Electrical and Computer Engineering, University of Toronto, Canada
SRISHT FATEH SINGH, Department of Electrical and Computer Engineering, University of Toronto, Canada
ANDREAS PARK, Rotman School of Management, University of Toronto, Canada and Department of Management, University of Toronto Mississauga, Canada
ANDREAS VENERIS, Department of Electrical and Computer Engineering and Department of Computer Science, University of Toronto, Canada

This paper presents a securities tokenization solution that brings the accessibility, transparency, efficiency, and innovation of blockchain and decentralized finance to real-world securities. Tokenization in principle seems straightforward—an intermediary holds assets and issues 1:1 tokens—but decentralized finance applications (DeFi) introduce significant complications. Even basic DeFi mechanisms, such as liquidity pools, pose challenges for tokenizing stocks and bonds because when assets are pooled in smart contracts, ownership becomes unclear, hindering asset owners to access their entitlements, such as dividends, coupons, or voting rights. Existing solutions often fail to address these challenges and are typically limited to specific security types. Our solution, by contrast, generalizes to any security and any holding rights through fungible tokens and using separate smart contracts for shareholders to redeem their entitlements. To address the decentralized ownership issue, our solution employs off-chain accounting with additional logic for liquidity pools. We implement this on Ethereum, demonstrating that it is 27% cheaper in gas costs than current alternatives. We also analyze the liquidity logic of over 90% of Ethereum's liquidity pools, confirming compatibility with our solution. Finally, we demonstrate its use for dividend-paying stocks, common stock, mergers, and coupon-paying bonds.

CCS Concepts: • **Applied computing** → **Digital cash**; Economics; • **Networks** → *Application layer protocols*; • **Information systems** → *Information systems applications*.

Additional Key Words and Phrases: Tokenization, Blockchains, Finance, Smart Contracts, Decentralized Applications

ACM Reference Format:
Reina Ke Xin Li, Srisht Fateh Singh, Andreas Park, and Andreas Veneris. 0. Blockchain Meets Securities: A Scalable Tokenization Framework. *Distrib. Ledger Technol.* 0, 0, Article 0 (0), 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Blockchain technology is a powerful tool promoting fairness, transparency, accessibility, efficiency, and security, properties desirable in finance. These advantages have led to the development and enthusiastic adoption of various *decentralized finance* (*DeFi*) protocols[1] leveraging blockchain with the goal of democratizing finance,

[1]The market capitalization of popular DeFi protocols can be viewed at https://coinmarketcap.com/view/defi/.

Authors' addresses: Reina Ke Xin Li, reinakx.li@mail.utoronto.ca, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada; Srisht Fateh Singh, srishtfateh.singh@mail.utoronto.ca, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada; Andreas Park, andreas.park@rotman.utoronto.ca, Rotman School of Management, University of Toronto, Toronto, Canada and Department of Management, University of Toronto Mississauga, Mississauga, Canada; Andreas Veneris, veneris@eecg.toronto.edu, Department of Electrical and Computer Engineering and Department of Computer Science, University of Toronto, Toronto, Canada.

ACM 2769-6472/0/0-ART0
https://doi.org/XXXXXXX.XXXXXXX

removing intermediaries, and promoting innovation. As such, today's DeFi ecosystem provides viable on-chain alternatives for traditional financial systems and infrastructures.

In a agenda-setting speech on July 31, 2025, S.E.C. Chairman David Atkins announced "the launch of Project Crypto—a Commission-wide initiative to modernize the securities rules and regulations to enable America's financial markets to move on-chain."[2] A central component of this vision is the *tokenization* of real-world securities so that investors can take advantage of the accessibility and transparency of blockchains, with the potential to eliminate intermediaries in the custody, management, and trading of the underlying assets. In fact, Citi's annual survey anticipates that by 2030, tokens and digital assets will drive 10% of market turnover [31]. As per the January 23, 2025 United States Executive Order,[3] "individual citizens and private-sector entities [must be able to] maintain self-custody of digital assets." In this broader context where the United States' financial markets move on-chain and where users can hold tokenized assets in self-custody, it is highly desirable that users also have access to secondary markets and on-chain liquidity through DeFi protocols.

The process of tokenization is akin to the issuance of American Depositary Receipts (ADRs), exchange-tradable certificates issued by a U.S. depositary bank representing a share of a foreign company's stock [30]. Similarly, a tokenization scheme entails a deposit of shares with a "trusted" custodian *issuer*, who issues on-chain tokens representing these shares so that they may be traded on blockchain markets, creating an on-chain representation of a traditional financial instrument.

On-chain financial markets would naturally have to enable the trading of tokenized publicly traded stocks using the on-chain analog of traditional stock exchanges: *decentralized exchanges*. Decentralized exchanges are online, open, and eliminate the need for brokers, clearinghouses, custodians, market makers, and other intermediaries necessary for traditional stock trading [2]. Thus, with stock tokenization, the functions of traditional stock exchanges can be translated seamlessly on-chain while improving accessibility, transparency, and efficiency for investors. In fact, the authors in [23] argue that investors could save 30% of trading costs if stock trading was organized using optimally designed, blockchain-based automated market makers.

However, even though the process of creating a token is well-established, the workings of DeFi applications create significant barriers. A major challenge faced by stock tokenization in DeFi ecosystems is *ownership attribution*. A stock is an investment contract that provides the owner with different rights, such as the right to vote in shareholder meetings and the right to dividends [4, 6]. The issuer of tokenized shares of a stock must be able to distribute the rights associated with the stock to the correct owners. Although tokens on blockchains have well-defined holding accounts, in some cases, the account may be a smart contract which is only holding the tokens on behalf of their owners [24]. Furthermore, some DeFi protocols, such as lending protocols and decentralized exchanges, hold assets in liquidity pools, and the ownership of each token is not well-defined. Previous works on securities tokenization fail to adequately address this challenge, as they either ignore assets held by smart contracts [3, 8, 12, 20, 29, 32, 35], or restrict the tokenized asset's use to custom permissioned DeFi protocols with additional functionalities [26], thereby eliminating key advantages of DeFi such as the inter-operability with most decentralized applications. Meanwhile, existing security token standards face the same problems and focus only on compliance enforcement and regulated transfers [10, 13, 19].

This paper extends and generalizes the results in [21] that present a solution for tokenizing stocks on Ethereum that translates the stock's holding rights on-chain and addresses the challenges of ownership accounting. The solution involves three parts: *(i)* a *Stock Token Contract* that provides a liquid tokenized representation of the stock on-chain; *(ii)* the *Off-Chain Accounting* procedure that calculates the ownership of tokenized shareholdings; and, *(iii)* a *Rights Redemption Contract* that allows tokenized shares to be redeemed for rights on the chain.

[2] "American Leadership in the Digital Finance Revolution", delivered July 31, 2025.
[3] "Strengthening American Leadership in Digital Financial Technology", issued January 23, 2025.

A Stock Token Contract implements a fungible and liquid token, making it operable with general DeFi protocols. The Off-Chain Accounting eliminates the gas costs involved in the accounting, allowing for the complex calculations required for dealing with smart contracts such as liquidity pools and for supporting an unbounded number of on-chain shareholders. It works by calculating both shareholders' wallet balances and their contributions to DeFi pools at a particular block using blockchain event queries. Furthermore, it is adaptable to the evolving regulatory landscape as the off-chain nature allows the accounting procedure to be easily upgraded, in case, for example, it is decided that shareholders do not own tokens they contribute to DeFi pools. The result of accounting is brought on-chain by a signed message from the issuer that allows on-chain shareholders to redeem rights through a Rights Redemption Contract that performs signature verification, which can be customized to distribute arbitrary holding rights.

The proposed solution has low gas costs, costing at least 27% less gas than other solutions for tokenizing dividend-paying stocks presented in [35] and [18]. Furthermore, as opposed to the solutions in [35] and [18], our solution is not limited to dividend-paying stocks—it can represent stocks with arbitrary holding rights such as voting rights. Crucially, our solution allows inter-operability with general DeFi protocols including over 90% of liquidity pools, and it can upgrade its accounting procedure without incurring additional costs of on-chain smart contract updates.

The work in [21] focuses on stocks; here we expand the prior work by providing details for a more complex tokenization use case—namely, *coupon-paying bonds*. For bonds, there is another layer of complexity in the ownership accounting. Bondholders accumulate rights to the next coupon between coupon payments. Therefore, when there is a transaction of bonds between coupon payments, the buyer must make an *accrued interest* payment to the seller [7]. Accrued interest payment is based on the passage time since the last coupon payment. However, similar to stock tokenization, the pooling of assets in DeFi complicates the distribution of accrued interest payments considerably. Here we address this by proposing the use of an expanded version of our Off-Chain Accounting procedure to account for ownership of the bond, not only at a single block (as in the case of, for instance, dividend-paying stocks), but over the course of the entire period between coupon payments. This approach therefore expands the payment options of bonds, possibly allowing for better risk sharing, and it intuitively extends to other assets such as structured notes. Moreover, the approach can be applied, for instance, to situations allowing an issuer to pay dividends or grant voting rights based on holding horizons.

Finally, this paper discusses on-chain shareholder voting, alluded to in [21], in further detail, addressing the problem of fractional ownership and fractional voting which may occur on-chain and cause deviation from real-world shareholder voting setups.

The remainder of the paper is organized as follows: Section 2 provides a background of the relevant tools and concepts, Section 3 details the proposed solution, Section 4 describes a modification to the proposed solution to accommodate bond tokenization, Section 5 presents an evaluation of the proposed solution, Section 6 discusses the trust assumptions and use cases of the proposed solution, Section 7 provides an overview of the related works in the literature, blockchain standards community, and industry, and Section 8 outlines potential directions for future work.

2 BACKGROUND

This section presents the blockchain concepts relevant to the proposed solution, including the token standards and cryptographic algorithms used in the solution, and the DeFi protocols that the solution is designed to operate with.

2.1 Ethereum Request for Comment Token Standards

Ethereum Request for Comment (ERC) token standards provide standardized APIs for interacting with specified token types, such as ERC20 and ERC721. An ERC20 token is a fungible token smart contract that tracks the balances of its holders [33]. An ERC721 token is a Non-Fungible Token (NFT) smart contract, where each token has a unique tokenId that is mapped to its owner and other additional data [15].

2.2 Elliptic Curve Digital Signature Algorithm

Elliptic Curve Digital Signature Algorithm (ECDSA) is built upon elliptic curve cryptography, a form of public-key cryptography that leverages the algebraic structure of elliptic curves over finite fields [27]. It is the algorithm used to generate Ethereum public and private keys and to sign and verify Ethereum transactions.[4] OpenZeppelin provides an audited library for the safe implementation of ECDSA signatures on-chain.[5]

2.3 EIP-712

EIP-712 defines a procedure for hashing and signing typed structured data [9]. In EIP-712, "\x19\x01" is prepended to messages before they are signed, followed by a domain separator and the hash struct. The domain separator encodes the name and version of the signing domain (e.g. name and version of the application), the active chain ID, and the verifying contract's address. The hash struct encodes the structured message datatypes and data.

2.4 DeFi Protocols

The DeFi protocols of interest in this work are lending protocols and Automated Market Makers (AMMs), which have liquidity pools in which users deposit tokens.

2.4.1 Lending. Decentralized lending platforms replace lenders with liquidity providers, who provide liquidity to a lending pool by depositing tokens. They are issued ERC20 liquidity pool tokens representing their deposits, which they can burn to make withdrawals from the pool. A liquidity provider's ownership of tokens in the pool is proportional to the amount of liquidity they provide. For instance, on Aave,[6] Spark,[7] and Morpho [16], liquidity providers can withdraw the same amount of tokens as they deposited. Meanwhile, on Compound[8] and Fluid,[9] the amount is scaled by the platform's exchange rate.

2.4.2 AMMs. AMMs are decentralized exchanges that replace traditional order book pricing with liquidity pools and algorithmic pricing. Liquidity providers deposit tokens into an AMM's liquidity pool at a rate determined by the AMM's invariant function, which are then used by exchange users to swap against at a rate that adjusts dynamically based on the pool's token balances. The pool can be structured as a uniform liquidity pool or as a concentrated liquidity pool.

Uniform liquidity AMMs: In these AMMs, liquidity is distributed uniformly across the entire price range. Liquidity providers are issued ERC20 liquidity pool tokens to represent their positions, which they can burn to withdraw tokens from the pool. Their ownership is, again, proportional to the amount of liquidity they

[4]For more information, see https://ethereum.org/en/glossary/#ecdsa.
[5]The OpenZeppelin library documentation can be found at https://docs.openzeppelin.com/contracts/5.x/.
[6]Aave documentation can be found at https://docs.aave.com/developers/v/2.0/the-core-protocol/atokens and https://docs.aave.com/developers/tokens/atoken.
[7]Spark documentation can be found at https://devs.spark.fi/.
[8]Compound documentation can be found at https://compound.finance/docs/ctokens and https://docs.compound.finance/collateral-and-borrowing/.
[9]Fluid documentation can be found at https://docs.fluid.instadapp.io/.

provide [34]. Protocols such as Uniswap V2 [2], Sushiswap V2,[10] and PancakeSwap V2[11] use a Constant Product AMM (CPAMM), wherein the product of token balances in the pool is fixed. Other protocols like Balancer[12] use a constant geometric mean invariant with support for multiple tokens per pool, while Bancor[13] extends the CPAMM model with single-sided liquidity provision.

Concentrated liquidity AMMs (CLAMMs): CLAMMs allow liquidity providers to distribute their liquidity along a selected price interval [1]. The price interval and the liquidity distributed along it is called a liquidity position and is represented by an ERC721 NFT. CLAMMs maintain an invariant for each position, $L = f(x, y)$, where L is the liquidity amount, and x, y are the (virtual) token balances. The price is represented as a function of ticks, $p(i)$, where i is the tick, and is equal to $\frac{dy}{dx}$. Then, the invariant can be rewritten as $y = g(p, L)$ and $x = h(p, L)$ for some functions g and h. For instance, in Uniswap V3, Sushiswap V3, and Pancakeswap V3, the invariant is $L = \sqrt{xy}$, the price is $p = \frac{y}{x}$, the tick-to-price mapping is $p(i) = 1.0001^i$, and $x = \frac{L}{\sqrt{p}}$ and $y = L\sqrt{p}$.In general, Equations (1) and (2) show the ownership of x and y tokens in the pool for a position with L liquidity in the interval $(p(i_l), p(i_u))$, when the current price is $p(i_c)$.

$$\Delta x = \max\{h(\max\{p(i_c), p(i_l)\}, L) - h(p(i_u), L), 0\} \tag{1}$$

$$\Delta y = \max\{g(\min\{p(i_c), p(i_u)\}, L) - g(p(i_l), L), 0\} \tag{2}$$

2.5 Homemade Dividends

In traditional finance, Modigliani-Miller Dividend Irrelevance implies that, instead of being paid dividends, shareholders can, equivalently, create *homemade dividends* by selling portions of their shares for cash [6]. This concept can be used for the tokenization of dividend-paying stocks, wherein the tokenized stock is an ERC20 token pegged to a reserve of physical shares of the stock. When dividends are paid, the issuer uses the dividend to buy physical shares of the stock from a traditional exchange and adds these shares to the reserve. After the reserve is increased, each ERC20 token's pegged value increases to the new ratio of shares in reserve to tokens in circulation. Any on-chain price not reflecting this value increase creates an arbitrage opportunity where one can buy the token on-chain for less and redeem it off-chain for more. Thus, arbitrageurs will ensure price responsiveness in on-chain markets, allowing the token's value to increase monotonically with every dividend payment. As a result of the value increase and the divisibility of ERC20 tokens, shareholders wanting cash dividends may create homemade dividends on-chain.

For instance, let x be the number of tokens in circulation and s be the shares in the reserve, so that one token represents $\frac{s}{x}$ shares. A cash dividend of d per share (and thus $\frac{sd}{x}$ per token) is paid out, and the issuer receives sd in dividends and buys $\frac{sd}{p}$ shares, adding them to the reserve, where p is the off-chain share price after dividend payout. Then, the token represents $\frac{s}{x} + \frac{sd}{px}$ shares. The value of the token on-chain converges to $\frac{sp+sd}{x}$ due to arbitrage, so shareholders can sell $\frac{sd}{px}$ tokens on an exchange like Uniswap V3 to get $\frac{sd}{x}$ in cash (ignoring fees), which is the original cash dividend per token.

This solution is lightweight and inexpensive: dividend payments only require gas costs if shareholders choose to create homemade dividends. It also does not disrupt liquidity pools as the reserve increases are agnostic to any on-chain DeFi infrastructures. However, it can be expensive to swap small amounts on DeFi exchanges as the gas costs do not scale with transaction value. Furthermore, the solution cannot generalize to stocks that guarantee its holders other rights (such as voting), or assets that are not exchange-traded.

[10]Sushiswap documentation can be found at https://docs.sushi.com/.
[11]PancakeSwap documentation can be found at https://developer.pancakeswap.finance/.
[12]Balancer documentation can be found at https://docs.balancer.fi/reference.
[13]Bancor documentation can be found at https://bancor-network.gitbook.io/v2.1/master and https://docs.bancor.network/.

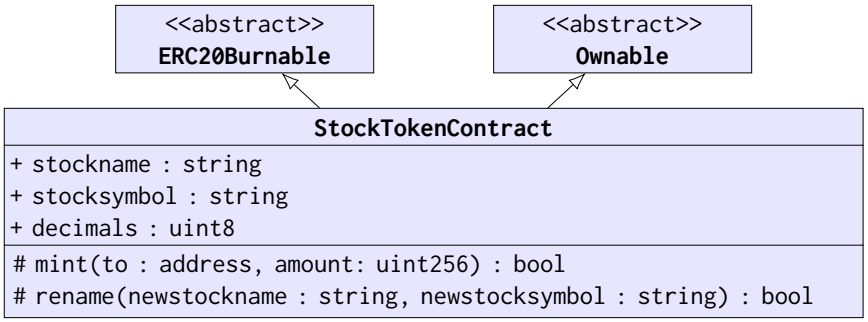

Fig. 1. StockTokenContract class conforming with ERC-20 standard

3 TOKENIZING STOCK SECURITIES

The proposed solution allows stocks to be represented and traded on the blockchain, tapping into the resources and liquidity available on-chain and enabling investors to take advantage of DeFi efficiencies and decentralization. Meanwhile, the solution does not sacrifice a stock's real-world properties such as the holding rights with which it is associated, nor does it compromise the tokenized stock's usability on-chain. The solution is designed to be low in gas cost, generalizable to different securities types, DeFi-compatible, and adaptable/flexible. This section presents the proposed solution in detail in its three parts: the Stock Token Contract, the Off-Chain Accounting procedure, and the Rights Redemption Contract.[14]

3.1 Stock Token Contract

The Stock Token Contract, deployed by the issuer, extends the OpenZeppelin ERC20 token smart contract, enabling compatibility with DeFi protocols described in Section 2.4. While standard ERC-20 tokens have just a name and symbol, this contract also includes a `stockname` and `stocksymbol`, which can be updated to allow for changes in the stock's name and symbol. The contract class is depicted in Figure 1.

3.2 Off-Chain Accounting

Accounting share ownership at a block (called the cutoff block) is done by querying the blockchain's events off-chain. First, each shareholder's wallet balance at the cutoff block is calculated by querying the Stock Token's logged `Transfer` events, which are emitted upon transfer of any ERC20-compatible token, up to the cutoff block. The wallet balance is the net of incoming and outgoing transfer amounts of that wallet address. Next, the shareholders' balances held in liquidity pools must be accounted for. These tokens may be held in lending pools or uniform liquidity AMMs (uniform liquidity pools), or CLAMMs. To do this, the issuer will determine the proportion of total liquidity that each shareholder can withdraw from the pool at the cutoff block. The issuer maintains a whitelist of valid liquidity pools in each category by keeping a list of addresses of these pools, and performs accounting for these pools.

3.2.1 Uniform Liquidity Pools. For each lending pool and uniform liquidity AMM, the issuer first calculates the pool's Stock Token balance at the cutoff block by querying the Stock Token's logged `Transfer` events and netting the transfers involving the pool. Then, for each shareholder, the issuer determines the shareholder's balance of the liquidity pool's token, this time querying the liquidity pool token's `Transfer` events. Finally, the amount of

[14]The contracts can be found at https://github.com/reinali07/tokenization.

Stock Tokens attributed to each shareholder is their proportion of liquidity pool tokens, multiplied by the pool's Stock Token balance. This procedure is summarized in Algorithm 1.

Algorithm 1 Procedure for calculating attributed Stock Token for a uniform liquidity pool.

$B \leftarrow$ pool's Stock Token balance
for each shareholder, i **do**
  $L_i \leftarrow i$'s liquidity pool token balance
end for
$L \leftarrow \sum_i L_i$
for each shareholder, i **do**
  $\text{result}_i \leftarrow \frac{L_i}{L} B$
end for

3.2.2 CLAMMs. Calculating shareholder pool proportions is more complicated in CLAMMs. In CLAMMs, each position may have a different rate of conversion from liquidity to tokens, depending on the position's tick interval. Thus, dealing with CLAMMs requires more exchange-specific information than the previous case, which was easily generalized. First, the issuer must include in its whitelist the address of the NFT position manager associated with each CLAMM, as well as the factory contract (wherein one can lookup pools by token pairs and specified fee). Next, we require knowledge of the pool's invariant functions, g, h in Equations (1) and (2). Finally, we require that shareholders report, before the cutoff block, their address, the liquidity pools they contribute to, and the corresponding `tokenId` of any concentrated liquidity positions they hold.

The issuer first confirms the ownership of each on-chain shareholder's reported NFTs by querying the NFT contract's `Transfer` events, which are emitted upon transfer of any ERC721-compatible NFT, and confirming that the latest transfer of the NFT prior to the cutoff block is to the shareholder. The issuer then calls the NFT contract's `positions()` method to determine the tick interval of the position. The `positions()` method also returns the `token0` and `token1` address, and the pool fees, which allows the issuer to call the pool factory contract's `getPool()` method to confirm the NFT corresponds to the reported pool. It also allows the issuer to determine whether the Stock Token is `token0` or `token1` in the pool. Note that calling the `positions()` method requires that the shareholder does not burn their NFT before the accounting is completed. This constraint does not affect the shareholder, as positions can be closed out without burning their associated NFT. Then, the issuer determines the NFT's liquidity by querying the NFT contract's `IncreaseLiquidity` and `DecreaseLiquidity` events for the `tokenId`, which are emitted upon liquidity position updates, and netting these updates.

Next, the issuer determines the pool's tick, i_c, at the cutoff block. To do this, it will query the Pool contract's `Swap` events to find the last swap prior to the cutoff block, which contains the tick after the swap. Then, for each NFT, the issuer uses Equation (1) or Equation (2) (if `token0` or `token1` is the Stock Token, respectively) to determine the number of Stock Tokens attributed to the on-chain shareholder. The procedure for accounting for one shareholder's concentrated liquidity position is summarized in Algorithm 2.

3.3 Rights Redemption Contract

On-chain shareholders are attributed the same rights as their off-chain counterparts, which they can redeem or exercise on-chain. Rights on-chain can be tokenized and thus represented by different token types. In this work, we consider three types of tokens: native tokens (e.g. Ether), ERC20 tokens (e.g. cryptotokens), and NFTs (e.g. DAO membership). A discussion on the use cases of rights represented by these token types is provided in Section 6.2.

Algorithm 2 Procedure for calculating attributed Stock Tokens for a on-chain shareholder's concentrated liquidity position.

Confirm position NFT ownership
token0, token1, fee, $i_l, i_u \leftarrow$ NFTContract.positions(nft)
Check pool == PoolFactory.getPool(token0,token1,fee)
$L \leftarrow$ position liquidity
$i_c \leftarrow$ tick before cutoff
if Stock Token is token0 **then**
    result $\leftarrow \max(h(\max(p(i_c), p(i_l)), L) - h(p(i_u), L), 0)$
else if Stock Token is token1 **then**
    result $\leftarrow \max(g(\min(p(i_c), p(i_u)), L) - g(p(i_l), L), 0)$
end if

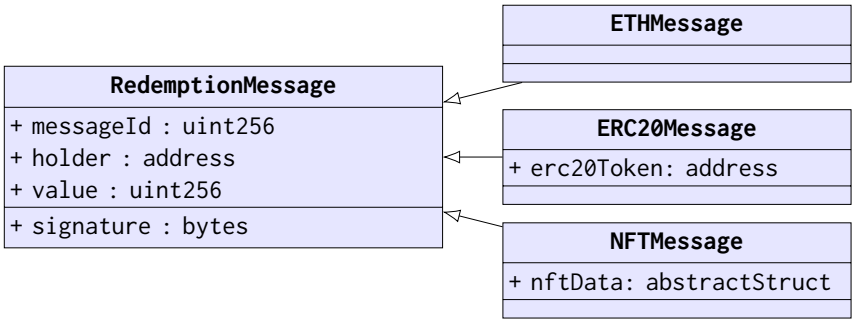

Fig. 2. Signed redemption message data structures for ETH, ERC20 tokens, and NFTs.

The Rights Redemption Contract allows on-chain shareholders to redeem the rights they are owed after their share ownership is accounted off-chain by the issuer (Section 3.2). The contract extends the OpenZeppelin EIP-712 contract and the data structure being signed depends on the type of token being distributed. The base Rights Redemption Contract allows shareholders to redeem tokens by providing a message with a valid ECDSA signature. The message is constructed and signed by the issuer after accounting, and distributed to on-chain shareholders off-chain on a public channel. The message data structure is shown in Figure 2, where the value is the amount of the tokenized rights the shareholder can claim. In the case of native tokens, no additional data is required. ERC20 token redemption requires messages to include the address of the ERC20 token in which the right is denominated. For NFT-denominated rights, the message can also include additional structured data.

The base contract is depicted in Figure 3. It keeps track of messages that have been redeemed by mapping unique message IDs to their redemption status to ensure that a message cannot be redeemed more than once. This replaces the single incrementing nonce typically used in ECDSA contracts as message IDs allow the issuer to issue indefinitely many messages at once that may be redeemed in any order. When a message is being redeemed, the contract first checks that the message's ID has not yet been redeemed. It then uses the message and the signature to recover the signer address and verifies that it belongs to the issuer. This recovery procedure also confirms the correct redeemer and signature domain. Upon successful signature verification, the contract transfers the recipient the tokens specified by the message. The implementation of the `redeem()` and `verify()` methods are depicted in Figure 4. The signature is passed into the `redeem()` method as its components `v`, `r`, and `s`. The ellipses represent the possibility of including other data in the signed message, as required.

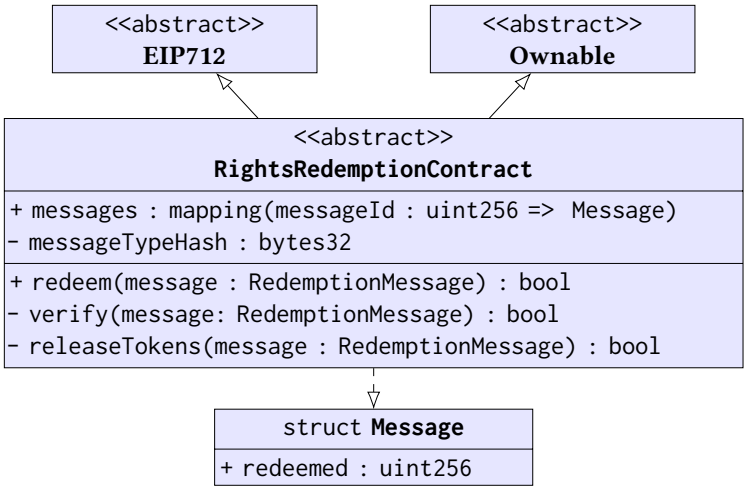

Fig. 3. Base RightsRedemptionContract class

```
function redeem(
    uint256 messageId,uint256 value,...,uint8 v, bytes32 r, bytes32 s
) external {
    verify(messageId,value,...,v,r,s);
    releaseTokens(msg.sender,value,...);
}
function verify(
    uint256 messageId,uint256 value,...,uint8 v, bytes32 r, bytes32 s
) internal {
    Message storage message = messages[messageId];
    require(message.redeemed == 0,"Already redeemed");
    message.redeemed = 1;

    bytes32 structHash =
    keccak256(abi.encode(messageTypeHash,messageId,msg.sender,value,...));
    bytes32 h = EIP712._hashTypedDataV4(structHash);

    address signer = ECDSA.recover(h,v,r,s);
    require(signer == owner,"Invalid Signature");
}
```

Fig. 4. RightsRedemptionContract's `redeem()` and `verify()` methods.

3.3.1 Native Tokens and ERC20 Token. When shareholder rights are distributed in the form of native tokens or ERC20 tokens, no additional functionality is required for the Rights Redemption Contract. However, for native tokens, the contract must include an empty `receive()` function to receive native tokens so that it can eventually distribute them. The `messageTypeHash` for these contracts are:

`keccak256("Redeem(uint256 messageId,address holder,uint256 value)")`

and,

`keccak256("Redeem(uint256 messageId,address holder,uint256 value,address erc20Token)")`

for native tokens and ERC20 tokens, respectively.

3.3.2 NFTs. When shareholder rights are distributed in the form of NFTs, the Rights Redemption Contract also extends the Openzeppelin ERC721 contract. Additionally, it may have data members that map NFTs to structured data. It may also implement additional methods for the usage of the NFT. An example of such is given in Section 6.2. The `messageTypeHash` for these contracts is:

```
keccak256("Redeem(uint256 messageId,address holderAddress,uint256 value,abstractStruct
nftData)")
```

where the `nftData` is optional NFT data of type `abstractStruct`.

4 TOKENIZING BONDS

A bond traditionally has two components: the principal (face value), F, and the coupons, measured by the coupon rate c of the face value, $C = c \cdot F$. A person who holds the bond between two coupon dates (the *coupon period*) accrues a claim on the latter coupon payment based on the fraction of time in that period that they hold the bond. This claim is referred to as accrued interest [7]. When the bond trades between coupon payments, the buyer has to pay the seller the price for the bond and pass on the accrued interest to the seller. Formally, when the length of the time between two coupon payments is T and the bond changes hands after $t < T$ time since the last coupon payment, the accrued interest is $\frac{t}{T}C$.

In a simple peer-to-peer sale, the buyer of a bond pays the purchase price and the accrued interest, $\frac{t}{T}C$, to the seller. However, if the bond is not traded over-the-counter, but rather with a liquidity pool, there may be issues, as existing liquidity pool contracts cannot, to the best of our knowledge, handle the accrued interest portion of the trade.

Our tokenization procedure provides an alternate solution that can accommodate DeFi trading. Namely, the blockchain keeps track of all bond trades, and the issuer allocates the respective claims once the coupon itself has been paid, based on each owner's holding horizons in the coupon period. This approach implies that all interim owners can claim their portion of the coupon (or, rather, their accrued interest) only when the coupon payment occurs, but the net payment is identical to that in the traditional approach. More broadly, this approach allows for what economists call the expansion of the contract space: in addition to handling the institutional arrangements for traditional bond trading, our process can also accommodate novel arrangements for asset payments. For instance, our process would allow firms to issue dividends or voting rights based on holding period horizons. In principle this allows our approach also to be applied to other assets than bonds and equities such as structured notes.

An additional advantage of this approach is that a potential default risk for a coupon payment is shared between the coupon claimants. In traditional bond sales, the buyer passes on the accrued interest prior to receiving the coupon payment. If the bond issuer defaults on the coupon payment, the seller is unaffected, but the buyer is not compensated for the amount it passed on to the seller. In our approach, it is the issuer that effectively pays out accrued interest, so any buyers and sellers bear the same risk. Furthermore, the process also allows for the aggregation of multiple payments when a bond changes hand back and forth multiple times, simplifying the bond sale process.

This section describes a modified version of the Off-Chain Accounting procedure detailed in Section 3.2 to accommodate this novel approach for coupon payments. In this section, we refer to the token, identical to the Stock Token described in Section 3.1, as the *Bond Token*. The ownership of Bond Tokens in a coupon period is calculated as the time-weighted average of the bondholder's balance of the bond token over the period, including the assets held both in the bondholder's wallet and in liquidity pools.

Before we outline the formal procedure, we first provide several illustrative examples.

Table 1. Average Wallet Balance Example

Time	Txn Amount	Time Δ	Balance
30 s	+100		100
100 s	Start of period	10 s	100
110 s	-40	50 s	60
160 s	+20	40 s	80
200 s	End of period		
	Average period balance:		72

Table 2. Average Uniform Liquidity Pool Balance Example

Time	LP Token Txn	Bond Token Txn	Time Δ	Pool Balance	Total LP Tokens	LP Balance
100 s	Start of period		30 s	100	100	10
130 s		-50 (Swap)	10 s	50	100	10
140 s	-5	-5 (LP)	20 s	45	95	5
160 s	+5	+5 (LP)	40 s	50	100	5
200 s	End of period					
				Average period balance:		4.97

4.1 Coupon Period Accounting Example

Table 1 shows a sample transaction history for an address transacting the Bond Token in a coupon period from time 100 seconds to 200 seconds. At each transaction, the balance is the running sum of transaction amounts. The final amount of Bond Tokens attributed to the wallet is the time-weighted average balance over the 100-second coupon period.

Table 2 shows a sample transaction history for a liquidity provider (LP) of a Bond Token uniform liquidity pool. For simplicity, the transactions in the pool occur as though liquidity pool tokens (LP tokens) are one-to-one with Bond Tokens, although this is often not the case with real liquidity pools. Three transactions occur here: at 130 s, a user swaps 50 Bond Tokens out of the pool; at 140 s, the LP withdraws 5 Bond Tokens from the pool, burning 5 LP tokens; and at 160 s, a different LP deposits 5 Bond Tokens to the pool, minting 5 LP tokens. The Bond Tokens attributed to the LP is the time-weighted average of the LP's share of the liquidity pool, calculated as the ratio of the *LP Balance* column to the *Total LP Tokens* column, times the pool's Bond Token balance, shown in the *Pool Balance* column.

Finally, Table 3 shows a sample transaction history for a liquidity position in a Bond Token CLAMM. Two transactions occur here: at 120 s, the LP increases its position liquidity by 30; and at 150 s, a swap occurs, updating the pool's tick to 1010. At each of these updates, either Equation (1) or Equation (2) (if `token0` or `token1` is the Bond Token, respectively) is applied, and the time-weighted average of the result is the amount of Bond Tokens attributed to the position. In this example, suppose `token0` is the Bond Token, $p(i) = 1.0001^i$, $h(p, L) = \frac{L}{\sqrt{p}}$, and the liquidity position's price interval is $(p(1000), p(1010)) = (1.1052, 1.1063)$. Then, Equation (1) becomes

$$\Delta x = \max\left\{\frac{L}{\max\{p(i_c), 1.1052\}} - \frac{L}{1.1063}, 0\right\} \tag{3}$$

Table 3. Average CLAMM Balance Example

Time	Liquidity Δ	Tick Update	Time Δ	Liquidity	Latest Tick i_c	$p(i_c)$	Δx (Eq (3))
100 s	Start of period		20 s	20	1000	1.1052	0.0180
120 s	+30		30 s	50	1000	1.1052	0.0450
150 s		1010	50 s	50	1010	1.1063	0
200 s	End of period						
				Average period balance:			0.0171

4.2 Coupon Period Accounting Procedure

We now specify the detailed accounting procedure, starting with wallet balances. To account for the bonds held in the bondholder's wallet, the issuer begins by querying all Bond Token `Transfer` events up to the end of the period. For each address, all transactions are netted up to the beginning of the period to determine the beginning-of-period wallet balance. Then, the wallet balance over the period is calculated as the running sum of transactions. The issuer also calculates the time difference between one transaction and the next for each address. Finally, the weighted average of the balances over the period is taken, with the time differences as weights. The procedure is summarized in Algorithm 3. Then, the issuer must also perform accounting for bond tokens held in liquidity pools, which is described next.

Algorithm 3 Procedure for calculating attributed Bond Token for wallet balances.

$\Delta \leftarrow$ total number of seconds in coupon period
$B \leftarrow$ blocks with Bond Token transfer events up to end of period
for each bondholder i **do**
  $B_i \leftarrow$ blocks in B with transactions involving i in period (sorted by timestamp)
  for (start of period and) each transaction b in B_i **do**
    $\delta \leftarrow$ seconds between b and next block in B_i or end of period
    $T_i \leftarrow i$'s balance after b
    $\text{result}_i \leftarrow \text{result}_i + T_i \cdot \frac{\delta}{\Delta}$
  end for
end for

4.2.1 Uniform Liquidity Pools. For each lending pool and uniform liquidity AMM, the issuer queries all relevant events up to the end of the period:

- Bond Token `Transfer` events involving the pool, indicating the pool's *Bond Token balance updates*, and
- The liquidity pool token's `Transfer` events involving any address (except for the `0x0` address and the pool itself), indicating *liquidity pool supply updates*.

At every block in the period with an update (as well as at the start of the period), the issuer calculates the bondholder's liquidity pool token balance at that block by maintaining a running sum of the liquidity supply updates involving that address. We divide this balance by the total liquidity supply across all liquidity providers at that block. Then, this ratio is multiplied by the pool's Bond Token balance at that block, which is the running sum of all Bond Token balance updates. Finally, the weighted average of these resulting values is computed over the period, using the time difference between consecutive update blocks as weights. This procedure is summarized in Algorithm 4.

Algorithm 4 Procedure for calculating attributed Bond Token for a uniform liquidity pool.

$\Delta \leftarrow$ total number of seconds in period
$B \leftarrow$ blocks with pool Bond Token balance updates or liquidity supply updates up to end of period
for (start of period and) each block b in B in period **do**
  $\delta \leftarrow$ seconds between b and next block in B or end of period
  $T \leftarrow$ pool Bond Token balance after b
  for each bondholder, i **do**
    $L_i \leftarrow i$'s liquidity pool token balance after b
  end for
  $L \leftarrow \sum_i L_i$
  for each bondholder, i **do**
    $\text{result}_i \leftarrow \text{result}_i + \frac{L_i}{L} \cdot T \cdot \frac{\delta}{\Delta}$
  end for
end for

4.2.2 CLAMMs. As in Section 3.2.2, the issuer must have knowledge of each CLAMM pool's NFT position manager, factory contract, and invariant functions. Bondholders are required to report the liquidity pools they contribute to and the corresponding `tokenId` of the positions they hold. For each position, the issuer uses the NFT contract's `positions()` method to determine the position's tick interval and the factory contract's `getPool()` method to confirm the associated liquidity pool. The issuer retrieves all relevant events for the `tokenId` up to the end of the period:

- `IncreaseLiquidity` and `DecreaseLiquidity` events from the NFT position manager contract,
- `Transfer` events from the NFT position manager contract indicating ownership changes of the position's NFT, and
- `Swap` events from the liquidity pool contract, indicating tick updates.

These events mark the updates to the bondholder's Bond Token holdings. At each block in the period with an update (as well at the start of the period), the issuer calculates the position's liquidity by maintaining a running sum of the `IncreaseLiquidity` and `DecreaseLiquidity` events' values. Additionally, the bondholder's ownership of the position's NFT is determined by tracking the net effect of the NFT's `Transfer` events up to that block. For blocks where the bondholder does not own the NFT, the position's liquidity is taken to be 0. The issuer also determines the pool's latest tick up to that block, using the `Swap` event data. Based on the position liquidity and tick value, the issuer applies Equation (1) or Equation (2) (if `token0` or `token1` is the Bond Token, respectively) to calculate the bondholder's holdings at that block. Finally, the weighted average of these holdings is computed over the period, using the time difference between consecutive update blocks as weights. This procedure is summarized in Algorithm 5.

5 EVALUATION

The performance of the proposed solution is evaluated against the four metrics/properties introduced at the beginning of Section 3: gas cost, the ability to generalise to stocks with arbitrary holding rights, the ability to operate seamlessly with general DeFi protocols without disruption, and the ability to adapt to arbitrary accounting methods. We also measure the time required to run the accounting procedure.

Algorithm 5 Procedure for calculating attributed Bond Token for a CLAMM position.

$\Delta \leftarrow$ total number of seconds in period
token0, token1, fee, $i_l, i_u \leftarrow$ NFTContract.positions(nft)
Check pool == PoolFactory.getPool(token0,token1,fee)
$B \leftarrow$ blocks with liquidity updates, nft transfers, or tick updates up to end of period (sorted by timestamp)
if Bond Token is token0 **then**
  $f(L, i_c)$
    $\leftarrow \max(h(\max(p(i_c), p(i_l)), L) - h(p(i_u), L), 0)$
else if Bond Token is token1 **then**
  $f(L, i_c)$
    $\leftarrow \max(g(\min(p(i_c), p(i_u)), L) - g(p(i_l), L), 0)$
end if
for (start of period and) each block b in B in period **do**
  $\delta \leftarrow$ seconds between b and next block in B or end of period
  $T \leftarrow 1$ if bondholder owns position nft after b,
      otherwise 0
  $L \leftarrow$ position's liquidity after b
  $i_c \leftarrow$ latest tick up to b
  result $\leftarrow$ result + $f(L, i_c) \cdot T \cdot \frac{\delta}{\Delta}$
end for

5.1 Gas cost

The gas cost of the proposed solution is realized in the on-chain components: the Stock/Bond Token Contract and the Rights Redemption Contract. The costs are the same for both the tokenized stock use case and the tokenized bond use case as the use cases differ only in the off-chain component. The gas costs are measured by deploying the contracts on a local Ethereum network. The Stock/Bond Token Contract is an extension of ERC20 and the gas costs associated with calling its methods are equal to the standard ownable and burnable ERC20 contract. Deployment of the Stock/Bond Token Contract costs 400k more gas as it implements one extra function and two extra data members.

Table 4 summarizes the gas cost for the Rights Redemption Contract's deployment, redeeming a message, and reverted redemption attempts. The order of operations is checking (and updating) whether a message has already been redeemed, verifying the message signature, and finally, releasing tokens. Based on this order, Table 4 indicates, roughly, that checking and updating whether a message has been redeemed costs 26–27k gas, verifying a signature costs 28k gas, and releasing tokens takes 9k, 18k, and 62k gas for ETH, ERC20, and NFTs respectively. In comparison, transferring ETH costs 21k gas, transferring standard ERC20 tokens costs 30k gas, and minting standard mintable ERC721 NFTs costs 57k.

5.2 Generalizing to arbitrary holding rights

The proposed solution is able to handle any holding rights that can be represented as ETH, ERC20 tokens, or NFTs. This includes rights such as dividend payments, coupon payments, and voting in shareholder meetings, which are discussed in Section 6.2. Moreover, it is possible to extend the Rights Redemption Contract to accommodate rights that come in other forms, as long as they can be represented as on-chain tokens that may be transferred or minted by the Rights Redemption Contract to the shareholder.

Table 4. Gas used for token redemption

Contract action	Token Type *ETH*	*ERC20*	*NFT*
Deployment	1,187k	1,168k	2,735k
Redeem	63k	73k	116k
Failed: already redeemed	26k	27k	26k
Failed: invalid signature	54k	55k	54k

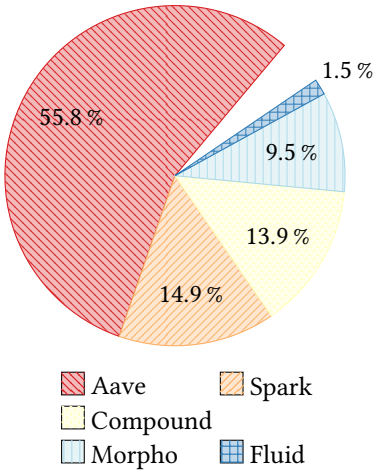

Fig. 5. Compatible lending protocols by TVL, 95.6% of total.

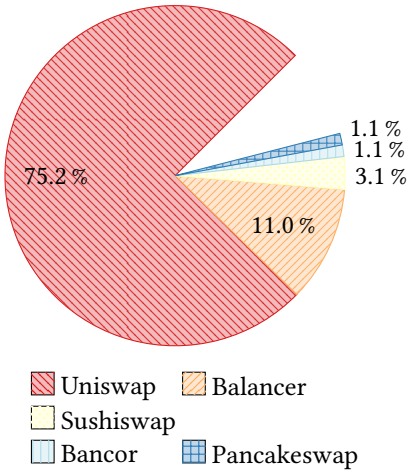

Fig. 6. Compatible AMMs by TVL, 91.5% of total.

5.3 Operability with DeFi

The Stock/Bond Token Contract implements the ERC20 API, making it operable with DeFi protocols using ERC20 tokens. The accounting procedure is also flexible in its operability as the off-chain nature allows it to be upgraded to maintain compatibility with potential new DeFi liquidity pool logic. As it is, both the procedures in Section 3.2 and Section 4.2 are compatible with the top DeFi liquidity pools. The top five DeFi protocols on Ethereum by total volume locked (TVL) in the lending category and the AMM category are shown in Figures 5 and 6, respectively.[15] These protocols are compatible with the accounting procedure as they are lending pools, uniform liquidity AMMs, or CLAMMs. They make up 95.6% of lending pools and 91.5% of AMMs by TVL. These are not exhaustive in terms of compatibility—for instance, other protocols forked from Uniswap are also compatible due to the liquidity pool logic.

5.4 Adapting to arbitrary accounting methods

The off-chain nature of the two accounting procedures allow them to adapt to accommodate evolving regulations or decisions regarding the ownership accounting of tokenized stocks or bonds. For instance, if it is decided that on-chain shareholders/bondholders do not own any tokens they contribute to liquidity pools, the accounting

[15]From https://defillama.com/protocols/Lending/Ethereum and https://defillama.com/protocols/Dexes/Ethereum (accessed May 11, 2024).

Table 5. Accounting procedure running time

	Wallet balances	Uniswap V2	Uniswap V3
Number of users	380k[1]	3k[2]	1k[3]
Time per user	0.02 ms	2.67 ms	3 s[4]
Total time	6 s[5]	8 s[6]	3000 s

[1] From https://etherscan.io/token/0x1f9840a85d5af5bf1d1762f925bdaddc4201f984 (accessed May 11, 2024).
[2] From https://etherscan.io/token/0xd3d2E2692501A5c9Ca623199D38826e513033a17 (accessed May 11, 2024).
[3] From https://dune.com/queries/4710117.
[4] From https://dune.com/queries/3715137.
[5] From https://dune.com/queries/3713463.
[6] From https://dune.com/queries/3708748.

procedure can simply skip the accounting for liquidity pools. Furthermore, the modularity means that updates to the accounting procedure or redeployment of the Rights Redemption contract leave the Stock/Bond Token Contract itself unaffected.

5.5 Running time

The time required to perform accounting, taking the Uniswap token (UNI) as the stock token, is summarized in Table 5. The running time is measured as the time to retrieve the necessary event logs and perform the calculations to account for the wallet balances of the holders of UNI, the liquidity providers of a Uniswap V2 UNI/ETH pool, and the liquidity positions of a Uniswap V3 UNI/ETH pool. The log queries are done using Dune SQL's free tier. The results present a slight improvement in speed compared to [21] due to optimizations in the SQL queries. The work in [21] employed repetitive and nested `Select` queries, which fetch and process the same data multiple times. Meanwhile, the results here use `With` queries in order to reuse processed data from sub-queries, rather than repeating the sub-query. These queries serve only as demonstrations, as Dune SQL's free tier is not optimized for query performance. Using other APIs or storing and querying the data locally may result in better performance.

For dividend payments, the record date, the date at which the shareholders' holdings are recorded for payment, is typically several weeks prior to the actual payment [6]. For shareholder voting, the record date is on or before the day that notice of a meeting is issued, which typically occurs several weeks prior to the meeting [11, 17]. As the accounting for each pool takes less than an hour in the most complex case (Uniswap V3), there is more than enough time to complete accounting.

Furthermore, we summarize the running time for the modified, more complex accounting (detailed in Section 4.2) required to manage accrued interest payments for tokenized bonds in Table 6. These running times tend to be longer than the stock accounting case because it requires block-by-block data and calculations. Depending on the length of a coupon-period and the amount of activity in liquidity pools (such as trades, borrowing and lending operations), the running time can become significant. However, because coupon payments occur on scheduled dates, the issuer can perform accounting in parts. For instance, if a coupon is paid yearly, the issuer can perform accounting quarterly and save the results, aggregating them at the end of the year.

6 DISCUSSION AND USE CASES

This section presents a discussion on the trust assumptions related to off-chaining calculations and details how our solution is applied in four use cases: dividend payments, coupon payments, shareholder voting, and mergers/acquisitions.

Table 6. Accounting procedure running time for tokenized bonds in quarter-year coupon period 2024-01-01 to 2024-04-01

	Wallet balances	Uniswap V2	Uniswap V3
Number of users	380k	3k	1k
Time per user	0.02 ms	60 ms	4 s[1]
Total time	6 s[2]	3 min[3]	4000 s

[1] From https://dune.com/queries/4585810.
[2] From https://dune.com/queries/4598320.
[3] The query performs accounting for 1/8th of a year (2024-01-01 to 2024-02-15) in 1 minute. Dune SQL's free tier allows a maximum runtime of 2 minutes, but we estimate a 2-3 minute runtime for a quarter-year period. From https://dune.com/queries/4585561.

6.1 Trust Assumptions

Our proposed approach has trade-offs. The accounting process introduces centralization, but has lower cost because payout calculations are performed off-chain, and only calculation results are entered on-chain through a signed message. The centralization manifests itself in the token issuer's control over redemption, allowing them to refuse a shareholder their rights by refusing to provide a signed message. In an on-chain-only solution, depending on the implementation, the rights redemption process would typically be governed by a transparent smart contract over which the token issuer has less fine-grain control. However, even on-chain smart contracts are not necessarily fully decentralized. Smart contracts can include provisions for their owners to arbitrarily amend access parameters. For example, the stablecoins USDT and USDC allow their respective issuers—Tether for USDT and Circle for USDC—to blacklist specific blockchain addresses, blocking their access to the respective coins.[16] This means users must trust Tether and Circle to adhere to their policies of blacklisting addresses only in accordance with regulations.

That being said, in our proposed solution, the token issuer's actions are auditable. An individual shareholder can verify the issuer's accounting results by simply redoing the calculation. This is possible as the procedure for one's accounting requires only publicly available blockchain data and knowledge of one's own CLAMM positions. Similarly, third parties can also verify and audit the calculations (provided with shareholders' CLAMM positions) if required for regulatory purposes. Therefore, since off-chain calculations can be verified by other parties, any tampering or mistake by the issuer is evident.

Most importantly, in the case of securities tokenization, the issuer is already inherently a point of centralization with a host of formal legal requirements and regulatory obligations. For instance, the issuer has custody of the physical shares, is responsible for relaying holding rights from the security issuer to the on-chain shareholders, and administers the token smart contracts. In any tokenization solution, such an issuer will be regulated to ensure it performs its duties fairly and correctly and regulatory recourse must be available for shareholders. Hence, it is fair to assume that the issuer is trusted and there is a fair, centralized system off-chain that regulates and oversees the issuer, regardless of whether the solution is fully on-chain. Arguably, since issuer actions and ownership attribution are auditable, our solution signifcantly improves and expands upon traditional procedures, making frauds like the German Cum/Ex scandal impossible.

Finally, the results of off-chain accounting are uploaded faithfully to the blockchain. The Rights Redemption contract uses the issuer's signature to ensure that messages are not forged by the shareholder, are not reused, and are redeemed by the correct shareholder.

[16] Both issuers have exercised this authority; see https://cointelegraph.com/news/tether-freezes-27-million-usdt-sanctions-garantex-russia and https://www.theblock.co/post/162172/circle-freezes-usdc-funds-in-tornado-cashs-us-treasury-sanctioned-wallets.

6.2 Use Cases

6.2.1 Dividend Payments. For a dividend-paying stock, the issuer deploys, along with the Stock Token Contract, a dividend Rights Redemption Contract, in which the issuer will deposit the dividends as ETH or ERC20 tokens to be redeemed by on-chain shareholders.

(1) The company declares a future dividend payment.
(2) The issuer notifies on-chain shareholders of the accounting cutoff, e.g. the first block after the record date.
(3) On-chain shareholders report, off-chain, a list of pools and NFTs, as in Section 3.2, prior to the cutoff.
(4) After the cutoff, the issuer accounts for each on-chain shareholder using the procedure in Section 3.2, multiplying the result by the dividends per share.
(5) The issuer writes, signs, and sends (off-chain) to each shareholder a message with the amount of dividends to pay and the shareholder's address. If the dividend is paid in ERC20 tokens, the message also contains the address of that ERC20 token.
(6) When the issuer receives the dividends from the company, it deposits the dividends to the Rights Redemption Contract.
(7) The shareholder passes the message to the dividend Rights Redemption Contract to claim their dividends.

6.2.2 Coupon Payments. For a coupon-paying bond, the issuer deploys, along with the Bond Token Contract, a coupon Rights Redemption Contract, in which the issuer will deposit the coupons as ETH or ERC20 tokens to be redeemed by on-chain bondholders. This case is almost identical to the case of dividend-paying stocks, with the difference being the accounting procedure used (Section 4.2 in this case) and that bonds have regular coupon payments, meaning the issuer and bondholders can anticipate future accounting cutoffs. Bondholders, similar to shareholders, also report their pools and NFTs prior to the accounting cutoff which could, in this case, be set to the first block on the upcoming coupon payment's record date.

6.2.3 Shareholder Voting. For common stock, the issuer deploys, along with the Stock Token Contract, a voting Rights Redemption Contract, which mints voting NFTs to on-chain shareholders and acts as a general purpose voting contract.

(1) The company announces a shareholder's meeting.
(2) The issuer notifies on-chain shareholders of the accounting cutoff, e.g. the first block after the record date.
(3) On-chain shareholders report a list of pools and NFTs (Section 3.2) prior to the cutoff.
(4) After the cutoff, the issuer performs accounting using the procedure in Section 3.2.
(5) The issuer writes, signs, and sends (off-chain) to each shareholder a message with the number of votes, the shareholder's address, and a vote ID specifying the vote the shareholder may partake in.
(6) The shareholder passes the message to the voting Rights Redemption Contract and is issued an NFT containing the vote ID and the number of votes, which they can cast to the same contract (Figure 7), emitting an event with the vote ID, choice, and number of votes cast.
(7) After the vote deadline, the issuer queries the event logs to tally the vote. This tally can be verified by any party, including the Company holding the vote.

Votes can be conducted either on- or off-chain. For example, a public vote may be conducted on-chain for transparency. Figure 7 depicts a simple smart contract method for vote casting, under the assumption that voting is public and conducted on-chain.

In some cases, though, it may be necessary to have the voting be conducted anonymously, which would require more complex smart contracts. Alternatively, for privacy preservation, it is possible to use anonymized off-chain voting methods (e.g. secret ballots often used by national elections). To register voters off-chain, have voters present the signed message from step (5) and cryptographically prove ownership of the keys corresponding to the address in the message to the off-chain voting administrator. In either case, the Off-Chain Accounting

procedure is used to determine the on-chain shareholder balances and positions. The issuer's signed messages containing the accounting results can then be used for on-chain votes with a Rights Redemption Contract, or used for off-chain votes to register voters as described above.

Another layer of complexity for shareholder voting arises in the translation of on-chain shareholders' votes to the real world, for which some extra work may be required. This is because tokens on-chain, being ERC20 compatible, may be configured to have the additional property of being divisible—shareholders can own fractions of shares on-chain. However, physical shares often cannot be divided. This means that the expectation in the real world is that one share can cast a single vote, which conflicts with the on-chain ability to cast fractional votes for each fraction of a tokenized share. As such, when votes from on-chain shareholders are aggregated to be cast off-chain, there may be the case where one share's vote is split among different choices. In the tokenization of a real world asset, the framework must be able to resolve this in order to translate the will of the on-chain shareholders to the real world vote.

A practical approach is to aggregate the on-chain votes and round down (*i.e.,* truncate) any leftover fractional votes. In all cases, votes must be rounded down to avoid conflict and misrepresentation. For example, if votes were rounded normally, the following unwanted scenarios may occur:

(1) Shareholder 1, with 0.5 shares, votes for option A, while shareholder 2, with 0.5 shares, votes for option B. In this case, rounding both votes to 1 exceeds the number of votes allowed between the two shareholders.
(2) Shareholder 1, with 0.4 shares, votes for option A, while shareholder 2, with 0.6 shares, votes for option B. Here, rounding the votes to 0 and 1 means that shareholder 1's vote is misrepresented as option B, *i.e.,* shareholder 1 is effectively forced to vote for option B.

Hence, while rounding down means that some votes are discarded, it ensures that the total number of on-chain votes does not exceed the number of votes allowed between the on-chain shareholders and that shareholders are not misrepresented due to having smaller fractional ownership. Furthermore, as rounding down occurs after aggregation, at worst, only one vote per option is discarded, preserving the integrity of the voting outcome while accommodating the limitations of translation to real-world systems.

Moreover, although discarding fractional voting rights may appear heavy-handed, such an approach is not without precedent: when two firms merge, a so-called "hold-out" who seeks to block a deal may be "frozen out": the firm performs a reverse split of the outstanding shares (e.g. 10 shares become 1) until the hold-out's shares are fractional, at which point this party loses their ability to block the merger.

The voting aggregation problem may not arise when a stock is issued natively on-chain or when the vote is conducted entirely on-chain, as fractional votes can be allowed without the need to match physical share certificates to votes. In some cases, real-world voting rules may also permit fractional votes. Even blockchain voting, however, has a maximum decimal precision. The smallest representable unit effectively becomes the smallest voting unit, making the question one of unit scale rather than indivisibility. The accounting results must be truncated accordingly, rounding down for the same reasons described above.

6.2.4 Mergers and Acquisitions. When a merger occurs, there are several possibilities: shareholders are paid out for their shares, shareholders exchange their shares of the old stock with shares of the new stock, or a mix of the two [5]. If shareholders exchange their shares, the issuer's physical holdings will be swapped. The issuer will then invoke the `rename()` function of the Stock Token Contract, updating the `stockname` and `stocksymbol` in Figure 1. However, this information may not be updated on blockchain explorers, such as Etherscan. To avoid confusion, the Stock Token Contract's `name` and `symbol` can be initially set to a unique identifier instead of the stock's actual name and symbol. If, instead, shareholders are paid out, on-chain shareholders send their tokens to a burning contract, which transfers them ETH or ERC20 tokens for every Stock Token it burns. In the case of a mixture of swapping and payout, the Stock Token can be renamed and the dividend payment process takes place to execute the payout.

```
function castVote(uint256 tokenId, uint256 value, uint8 choice) public {
    VotingNFT storage vote = tokenData[tokenId];
    require(owners[tokenId] == msg.sender, "Unauthorized voter");
    require(value <= vote.value,"Not enough votes");

    vote.value = vote.value - value;
    emit Voted(vote.voteId, choice, value);
}
```

Fig. 7. Implementation of simple vote casting for shareholder voting.

7 RELATED WORK

The tokenization of securities has been explored both in the literature, the blockchain community, and in the financial industry. The literature to date focuses on regulatory compliance and dividend distribution, but provides inadequate consideration for the feasibility of dividend distribution in terms of gas costs, the accounting of tokens held in DeFi protocol liquidity pools, and the potential for other types of holding rights. The blockchain community engages in standardization, compliance and identity integration, and expanding asset classes. While these efforts touch on holding rights, they are inadequate in the face of DeFi. The commercial projects focus on regulatory compliance and cost efficiency, but have the same shortcomings as the works in the literature. Furthermore, these commercial projects are opaque and provide little detail on technical implementation or empirical results.

7.1 Literature Review

The authors of [18] propose a blockchain-based solution for tokenizing revenue-generating real estate. This work uses a token to represent shares of a special purpose vehicle setup to own real estate property. Regulatory compliance and user validation are embedded in the token smart contract, although the details are not provided. The distribution of dividends to the token holders is executed by a smart contract that loops through every token holder, calculates their dividend as their balance multiplied by the dividend per token, and adds the dividend amount to the token holder's balance. Crucially, this loop scales linearly with the number of token holders, causing the dividend transaction to easily exceed the Ethereum block gas limit and fail. The authors leave the implementation of other functionalities such as voting or other holding rights to future work.

The work in [35] describe an asset tokenization system for dividend-paying assets. This work implements regulatory compliance, focusing on the Swiss jurisdiction, by requiring token holders to have undergone KYC certification by an approved KYC certificate provider, and enabling whitelisting and blacklisting. This work employs a user-initiated dividend distribution mechanism, wherein a user invokes a smart contract method that calculates its dividend payout and transfers the tokens to the user accordingly, mitigating the block gas limit issue. The solution is designed for dividend payments, and while it could be modified for other holding rights, this is not discussed in the paper and the lack of implementation details (code) makes it difficult to speculate. The authors estimate dividend redemption to cost 100k gas. The key innovation in [35] is a dividend payout based on users' holding durations, eliminating the sharp price drop observed ex-dividend. It does this by implementing balance accumulators in its smart contract, which are updated on each token transfer. This effectively performs the coupon bond accounting for wallet balances described in Section 4.2. However, our proposed solution's gas cost is agnostic to the complexity of the Off-chain Accounting procedure used and thus remains cheaper for coupon bonds. A simplified version of the solution from [35] without holding-time based calculations may reduce its costs, but without the author's code, this cannot be verified.

Table 7. Comparison of solutions

	Automatic dividends	Homemade dividends	Zhitomirskiy et al.	Our solution
Arbitrary holding rights	✗	✗	✗	✓
DeFi operability	✗	✓	✗	✓
Adaptable accounting	✗	✗	✗	✓
Gas cost per shareholder	✗[1]	127k[2]	100k	73k

[1] Automatic dividend distribution for 380k holders costs >100B gas, which exceeds the Ethereum block gas limit of 30M (see https://ethereum.org/en/developers/docs/gas/), making the distribution transaction fail.
[2] Average gas used for Uniswap V3's exact input single swap method from https://dune.com/queries/3694112.

7.1.1 Comparison. For context, we provide a comparison of our proposed solution with the work in [18], the solution described in Section 2.5, and the work in [35], each of which are dividend-paying stock tokenization solutions. Specifically, the solutions are qualitatively assessed in terms of their ability to generalize to arbitrary holding rights, operatibility with DeFi liquidity pools, and adaptability to arbitrary accounting methods, which are the criteria explored in Section 5 that our proposed solution is designed around. The gas costs per token holder for receiving dividend payments is also measured based on 380k holders, the number of holders of UNI. The results are summarized in Table 7, which shows that no other solution performs adequately against the metrics, despite being more costly than our proposed solution, with Zhitomirskiy et al's solution being the cheapest at 27k more gas per shareholder compared to our solution.

The solutions in both [18] and [35] are designed for dividend payments, and leave other types of entitlements such as voting to future work. Furthermore, they only account for shareholders' wallet balances, and not liquidity pool contributions. The authors of [35] acknowledge that distributing dividends to liquidity providers requires additional non-trivial logic. Moreover, both of these solutions employ smart contract-embedded, non-modular dividend mechanisms, making these solutions difficult to adapt to new accounting rules.

The solution described in Section 2.5 accumulates reinvested yield towards the redemption value of the stock token, while not requiring any action to be taken on-chain. As a result, it is fully DeFi compatible. However, it is only suitable for dividends or interest and lacks flexibility. Furthermore, if token holders wish to receive dividends as cash (e.g. stablecoins), they must bear the cost of swapping on an exchange like Uniswap V3, which costs 127k gas on average.

Although our proposed solution meets the criteria while achieving a lower gas cost, it is important to consider that it does this by off-chaining important calculations. A drawback of this is its vulnerability to key theft. If the issuer's private key is stolen, messages can be forged, allowing malicious users to redeem rights to which they are not entitled. The off-chain solution can also be slower, as the issuer needs to perform the calculations for all users, whereas in on-chain solutions, shareholders only need to wait for their own redemption transaction to execute. This may not be a problem as long as the off-chain calculations are completed before rights are actually distributed. Furthermore, the implications of off-chaining on centralization and trust are explored in Section 6.1.

7.2 Token Standards

ERC3643 is a standard for permissioned tokens that integrates on-chain identity management, programmable transfer rules, and compliance enforcement at the smart contract level [19]. It leverages a customizable and modular architecture for its compliance enforcement to allow flexible and jurisdiction-specific logic. The standard

uses ONCHAINID[17] to allow users to manage their identities and claims about their identities, which can be part of the criteria set out by the applicable compliance rules—for example, as part of the jurisdictional KYC requirements—for permissioned transfers and ownership.

On the other hand, the ERC1400 security token standard takes a different approach by off-chaining its identity and compliance enforcement [13]. It requires an authorized entity to validate transactions off-chain, before providing a signed certification to allow the transaction to execute on-chain. This off-chain-to-signed-message approach is not unlike the proposed Rights Redemption mechanism in our solution and ensures compliance without bloating the smart contract with complex on-chain logic, while introducing a level of centralization.

CMTAT is another smart contract framework with functionality for transaction validation without on-chain identity integration [10]. It is tailored to the Swiss regulatory context but is customizable and upgradeable to other jurisdictions. It employs a modular framework to implement functionalities, where specific implementations can choose which modules to include, as well as customize functionalities for its regulatory and functional requirements. For example, a dividend-paying stock can implement the CMTAT snapshot module to keep track of user balances at the record date and distribute dividends proportionally to the snapshot balances. However, this can be expensive due to the duplication of on-chain data, requires further implementation details for the dividend distribution, and remains incompatible with DeFi liquidity pools,

Unlike ERC3643, ERC1400 and CMTAT also allow issuers to attach documents such as offering documents or notices to the token contract, which can be helpful for transparency. All three standards extend the ERC20 interface to support forced transfers, pauses, freezes, and account recovery mechanisms. These standards and protocols are designed to address the legal requirements of securities tokens, and do not consider or implement functionality such as the distribution of holding rights.

ERC2222 [28], a draft Funds Distribution Token (FDT) standard, addresses the distribution of future cash flows, such as dividends, coupons, or royalties, and is often used in conjunction with other security token standards. It extends ERC20 by a mechanism wherein users withdraw their revenue claims asynchronously, avoiding gas limit issues associated with automatic distributions. The standard requires that implementations guarantee users are able to withdraw past claims to cash flows, even after transferring away their FDT. This makes the ERC2222 standard somewhat DeFi-compatible, as it does not require users to withdraw their liquidity pool deposits in order to claim their revenue. However, unlike our proposed framework, a user has no claim on the revenue generated by their tokens after they have contributed them to a liquidity pool.

Other implementations of interest-bearing assets include Ondo Finance's OUSG and rOUSG.[18] OUSG is an accumulating token where interest is reinvested into the underlying asset, increasing the redemption value of the token over time, as in Section 2.5. rOUSG is a rebasing token, where the token maintains a constant redemption value while yield is paid out by applying a factor to each user's balance. These approaches are lightweight but have the same limitations as Section 2.5.

7.3 Commercial Projects

Several projects for the tokenization of securities or other real-world assets helmed by major banks, securities trading firms, and regulatory bodies are in the works. A collaboration named Project Guardian between policy-makers UK's FCA, Switzerland's FINMA, Japan's FSA, and Singapore's MAS and commercial groups including JP Morgan, Citibank, and HSBC, aims to develop and pilot the policies and technologies necessary for secure, interoperable, and innovative asset tokenization [25]. The project's focus thus far has been on identity and trust management, regulatory innovation, payment automation, and portfolio management [22]. It wrapped up its first pilot program for the trading of tokenized bonds and foreign exchange against permissioned liquidity pools in

[17]ONCHAINID is a decentralized identity system. Its documentation can be found at https://docs.onchainid.com/.
[18]Ondo's OUSG token documentation can be found at https://docs.ondo.finance/qualified-access-products/ousg.

2022 [26]. Swiss securities firm Taurus provides a platform for the tokenization of assets, and was approved by the FINMA to offer tokenized securities of unlisted Swiss firms to retail investors in January 2024 [3]. Private Swiss bank Cité Gestion, along with a handful of other firms used the Taurus platform to tokenize its own shares to be available to professional investors in 2023 [29]. In 2022, Hamilton Lane announced it would be tokenizing three of its funds using Securitize, another firm offering asset tokenizaton services [12]. In 2022, UBS launched a digital bond that is tradeable on the blockchain, as well as in traditional exchanges [32]. The BIS, along with the Swiss National Bank and the World Bank announced in 2024 Project Promissa, a pilot initiative for the tokenization of promissory notes [8]. HSBC, partnering with Goldman Sachs and the EIB, has launched in 2023 a digital bond that uses the blockchain as a record of ownership [20]. These projects are limited in scope, as they focus on specific types of assets, such as bonds and notes. Moreover, while they aim for efficiency gains through blockchain's accessibility and automation, they do not consider the potential arising from inter-operability with DeFi, as they either require limitations on compatible DeFi protocols or do not address DeFi compatibility at all.

8 CHALLENGES AND FUTURE DIRECTIONS

While our solution provides a flexible framework for tokenization, several challenges remain, motivating directions for further work on this topic. First, although the gas cost of our solution is lower than the alternatives studied in Section 7.1, further gas optimizations in the Rights Redemption Contract should be performed. Currently, reading and writing the message redemption status are expensive operations that could be improved by, for instance, using different data types, structures, or signature schemes. Furthermore, while we have shown a simple voting scheme using NFTs, some use cases may require greater levels of voter privacy, requiring more complex smart contracts and cryptographic guarantees. Other types of rights not discussed in this paper may also need specific peripheral utilities.

Second, some challenges, already identified in Section 7.1.1, arise due to the off-chain nature of the accounting procedure. The redemption process can be vulnerable to issuer key theft and recovery mechanisms should be implemented. Also, the accounting procedure's runtime should be improved, perhaps by optimizing the event log query methods. For dividend paying stocks, the speed is adequate, but as it is, if a use case requires quick distribution, it may be necessary to consider the trade-offs of using a different solution that is more costly or less flexible.

Third, the Off-Chain Accounting and Rights Redemption mechanism mitigates fraudulent reporting (assuming the issuer's keys are not compromised) in the context of DeFi liquidity pools. While the Off-Chain Accounting procedure relies on shareholder reporting for the CLAMM portion, it handles false reports of invalid NFTs and incorrect pool or user addresses. The CLAMM accounting algorithm checks that the shareholder's address owns the reported NFT,[19] that the NFT corresponds to the reported pool address, and that the pool is a whitelisted CLAMM for the Stock/Bond Token. Additionally, any entitlements are ultimately given to the reported address (so long as it owns the NFT reported), so providing a false address will not give the user any benefit. However, the accounting procedure may not be able to deal with exotic smart contracts. New algorithms can be made to handle such smart contracts, but this needs to be developed on a case-by-base basis. The off-chain channel used for communications between shareholders and the issuer also needs be reliable and have mechanisms in place to prevent spam, which can hinder the issuer's ability to perform accounting for honest CLAMM liquidity providers.

Finally, our proposed solution does not include full regulatory compliance integration. Instead, the system can be structured to incentivize *voluntary* compliance participation by requiring shareholders to undergo off-chain know-your-customer (KYC) verification in order to redeem holding rights, on-board, or off-board. For assets like coupon bonds, receiving coupons can be a strong incentive for users to undergo KYC. Meanwhile, the regular token operations such as transfers are unaffected, as in typical crypto-tokens. While this aligns with blockchain's

[19] Or in the case of coupons, it checks for how long the address owned the NFT.

values of incentivization over regulation, it leaves operability with real-world financial systems as an open question. As the focus on crypto-assets grows, so does the call for compliant-by-design systems with integrated regulatory compliance and KYC [14]. However, integrated compliance layers can be added as a modular extension to our solution. Our solution's Off-Chain Accounting and Rights Redemption are not directly integrated into the Stock Token Contract on-chain and only require a token with an ERC20-compatible interface. Security token standards with compliance integration, which are described in Section 7.2, tend to be extensions of ERC20. This makes those standards integrable with our solution by simply replacing our Stock Token Contract with the security token standard suite. Hence, while regulatory compliance is critical for adoption, it is beyond the scope of this work, which focuses on the mechanics for DeFi-enabled tokenization.

9 CONCLUSION

This paper presents a stock tokenization solution that is gas-efficient, generalizes to arbitrary holding rights, operates with DeFi without friction, and can be adapted to arbitrary accounting methods or decisions. We find that these requirements are not adequately addressed by existing solutions. Our solution comprises three components: a Stock Token Contract, the Off-Chain Accounting procedure, and a Rights Redemption Contract. The Stock Token Contract is operable with DeFi by design and the Off-Chain Accounting works for lending pools, uniform liquidity AMMs, and CLAMMs as it is designed around their liquidity provision and pool ownership logic. Specifically, the accounting procedure is able to account for at least 90% of lending pools and AMMs, by TVL. We demonstrate that our solution can accommodate stocks with any rights that can be represented by ETH, ERC20 tokens, or NFTs, and give explicit use cases: dividend payments, voting rights, and mergers/acquisitions. Furthermore, we demonstrate a modification of the proposed tokenization solution for a coupon-paying bonds use case, which automates the calculation and payment of accrued interest in bond sales. This modification also demonstrates the flexibility of the proposed solution with respect to the accounting procedure, as modifications can be made in accordance with the real-world security's contract.

REFERENCES

[1] Hayden Adams, Noah Zinsmeister, River Keefer, Moody Salem, and Dan Robinson. 2021. *Uniswap v3 Core.* Technical Report. Uniswap Labs. https://uniswap.org/whitepaper-v3.pdf

[2] Hayden Adams, Noah Zinsmeister, and Dan Robinson. 2020. *Uniswap v2 Core.* Technical Report. Uniswap Labs. https://uniswap.org/whitepaper.pdf

[3] Ian Allison. 2024. Crypto Custody Specialist Taurus Brings Tokenized Securities to Retail Customers in Switzerland. *CoinDesk* (23 1 2024). https://www.coindesk.com/business/2024/01/23/crypto-custody-specialist-taurus-brings-tokenized-securities-to-retail-customers-in-switzerland/

[4] Jonathan B. Berk and Peter M. DeMarzo. 2019. The Corporation and Financial Markets. In *Corporate Finance* (5th global edition ed.). Pearson, UK, Chapter 1, 32–55.

[5] Jonathan B. Berk and Peter M. DeMarzo. 2019. Mergers and Acquisitions. In *Corporate Finance* (5th global edition ed.). Pearson, UK, Chapter 28, 1000–1030.

[6] Jonathan B. Berk and Peter M. DeMarzo. 2019. Payout Policy. In *Corporate Finance* (5th global edition ed.). Pearson, UK, Chapter 17, 635–675.

[7] Jonathan B. Berk and Peter M. DeMarzo. 2019. Valuing Bonds. In *Corporate Finance* (5th global edition ed.). Pearson, UK, Chapter 6, 207–243.

[8] BIS Innovation Hub Swiss Centre, Swiss National Bank, and World Bank. 2024. *BIS Innovation Hub, Swiss National Bank and World Bank launch Project Promissa to test tokenisation of financial instruments.* https://www.bis.org/about/bisih/topics/fmis/promissa.htm

[9] Remco Bloemen, Leonid Logvinov, and Jacob Evans. 2017. *EIP-712: Typed structured data hashing and signing.* https://eips.ethereum.org/EIPS/eip-712

[10] cmtat 2024. *CMTAT: Functional specifications for the Swiss law compliant tokenization of securities.* Technical Report. Capital Markets and Technology Association. https://github.com/CMTA/CMTAT

[11] Companies Act 2006 2006. Companies Act 2006: Notice of meetings. In *UK Public General Acts.* UK, Chapter 3, 144–146. https://www.legislation.gov.uk/ukpga/2006/46/part/13/chapter/3/crossheading/notice-of-meetings/enacted

[12] Jamie Crawley. 2022. Investment Manager Hamilton Lane to Tokenize 3 Funds Through Securitize. *CoinDesk* (5 10 2022). https://www.coindesk.com/business/2022/10/05/investment-manager-hamilton-lane-to-tokenize-3-funds-through-securitize/

[13] Adam Dossa, Pablo Ruiz, Fabian Vogelsteller, and Stephane Gosselin. 2018. ERC-1400: Security Token Standard. https://github.com/ethereum/EIPs/issues/1411

[14] Darrell Duffie, Odunayo Olowookere, and Andreas Veneris. 2025. The Stablecoin Balancing Act. *Finance & Development* 62, 3 (Sep 2025), 32–36.

[15] William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs. 2018. *ERC-721: Non-Fungible Token Standard* (no. 721 ed.). https://eips.ethereum.org/EIPS/eip-721

[16] Paul Frambot, Mathis Gontier Delaunay, Vincent Danos, Adrien Husson, and Katia Babbar. 2022. *Morpho Optimizer: Optimizing Decentralized Liquidity Protocols.* White Paper. Morpho Labs. https://whitepaper.morpho.org/

[17] Government of Canada Innovation, Science and Economic Development. 2023. *Share structure and shareholders.* https://ised-isde.canada.ca/site/corporations-canada/en/business-corporations/share-structure-and-shareholders

[18] Ashutosh Gupta, Jash Rathod, Dhiren Patel, Jay Bothra, Sanket Shanbhag, and Tanmay Bhalerao. 2020. Tokenization of Real Estate Using Blockchain Technology. In *Applied Cryptography and Network Security Workshops*. Springer International Publishing, Rome, Italy, 77–90. https://doi.org/10.1007/978-3-030-61638-0_5

[19] Joachim Lebrun, Tony Malghem, Kevin Thizy, Luc Falempin, and Adam Boudjemaa. 2021. ERC-3643: T-REX - Token for Regulated EXchanges. https://eips.ethereum.org/EIPS/eip-3643

[20] Ledger Insights. 2023. *Hong Kong confirms first $100m tokenized green bond.* https://www.ledgerinsights.com/hong-kong-tokenized-green-bond/

[21] Reina Ke Xin Li, Srisht Fateh Singh, Andreas Park, and Andreas Veneris. 2024. On Tokenizing Securities in Contemporary Decentralized Finance Ecosystems. In *2024 6th Conf. Blockchain Res. & Appl. for Innov. Netw. and Services (BRAINS)*. IEEE, Berlin, Germany, 1–9. https://doi.org/10.1109/BRAINS63024.2024.10732268

[22] Tyrone Lobban, Dennis Cristallo, Nikhil Sharma, Keerthi Moudgal, Joe Leung, Marlee Burr, Kirsten Jones, Stephanie Lok, Christine Moy, Jason Singer, and Vivan Sze. 2023. *The Future of Wealth Management: Ultra-efficient portfolios of traditional and alternative investments powered by tokenization.* Technical Report. J.P. Morgan and Apollo. https://www.jpmorgan.com/kinexys/documents/portfolio-management-powered-by-tokenization.pdf

[23] Katya Malinova and Andreas Park. 2023. Learning from DEFI: Would automated market makers improve equity trading? https://doi.org/10.2139/ssrn.4531670

[24] Katya Malinova and Andreas Park. 2025. Tokenized Stocks for Trading and Capital Raising. *Research Policy* (2025). https://doi.org/10.2139/ssrn.4365241 forthcoming.

[25] Monetary Authority of Singapore (MAS). 2022. *Project Guardian: Exploring Asset Tokenization.* https://www.mas.gov.sg/schemes-and-initiatives/project-guardian

[26] Monetary Authority of Singapore (MAS). 2023. *Enabling Open & Interoperable Networks.* https://www.mas.gov.sg/news/media-releases/2023/mas-proposes-framework-for-digital-asset-networks

[27] Svetlin Nakov. 2018. *Digital Signatures.* SoftUni (Software University). https://cryptobook.nakov.com

[28] Johannes Pfeffer, Roger Wu, Johannes Escherich, and Tom Lam. 2019. ERC-2222 Funds Distribution Standard. https://github.com/ethereum/EIPs/issues/2222

[29] Asa Sanon-Jules. 2023. Swiss Bank Cité Gestion Becomes First Private Bank to Tokenize Its Own Shares. *CoinDesk* (24 1 2023). https://www.coindesk.com/business/2023/01/24/swiss-bank-cite-gestion-becomes-first-private-bank-to-tokenize-its-own-shares/

[30] SEC's Office of Investor Education and Advocacy. 2012. *Investor Bulletin: American Depositary Receipts.* https://www.sec.gov/investor/alerts/adr-bulletin.pdf

[31] Alicia Sidik, Amit Agarwal, Artem Korenyuk, Barnaby Nelson, Chris Cox, Emma Johnson, Harsha Jethnani, Ingrid Collazo, Jolene Han Berg, Marcello Topa, and et al. 2025. *Securities Services Evolution 2025.* Technical Report. Citi. https://www.citibank.com/icg/docs/securities-services/Citi-Securities-Services-Evolution-2025.pdf

[32] UBS AG. 2022. *UBS AG launches the world's first ever digital bond that is publicly traded and settled on both blockchain-based and traditional exchanges.* https://www.ubs.com/global/en/media/display-page-ndp/en-20221103-digital-bond.html

[33] Fabian Vogelsteller and Vitalik Buterin. 2015. *ERC-20: Token Standard.* https://eips.ethereum.org/EIPS/eip-20

[34] Zhang Yi, Xiaohong Chen, and Daejun Park. 2018. *Formal Specification of Constant Product (x y = k) Market Maker Model and Implementation.* Technical Report. Uniswap Labs. https://github.com/runtimeverification/verified-smart-contracts/blob/uniswap/uniswap/x-y-k.pdf

[35] Efim Zhitomirskiy, Stefan Schmid, and Martin Walther. 2023. Tokenizing assets with dividend payouts—a legally compliant and flexible design. *Digital Finance* 5, 3 (Dec. 2023), 563–580. https://doi.org/10.1007/s42521-023-00094-w