BakUp: Automated, Flexible, and Capital-Efficient Insurance Protocol for Decentralized Finance

(Anonymous submission)

***Abstract*—This paper introduces BakUp, a smart contract insurance design for decentralized finance users to mitigate risks arising from platform vulnerabilities. While providing automated claim payout, BakUp utilizes a modular structure to harmonize three key features: the platform's resilience against vulnerabilities, the flexibility of underwritten policies, and capital efficiency. An immutable core module performs capital accounting while ensuring robustness against external vulnerabilities, a customizable oracle module enables the underwriting of novel policies, and an optional and peripheral yield module allows users to independently manage additional yield. The implementation incorporates binary conditional tokens that are tradable on automated market maker (AMM)-based exchanges. Finally, the paper examines specific liquidity provision strategies for the conditional tokens, demonstrating that a conservative strategy and parameterization can effectively reduce the divergence loss of liquidity providers by more than 47% compared to a naive strategy in the worst-case scenario.**

***Index Terms*—smart contract, decentralized finance, decentralized insurance, risk management, capital efficiency.**

I. Introduction

Programmable blockchains [1]–[5] have led to an era of decentralized finance, or DeFi, where centralized intermediaries are replaced by software code, also known as *smart contracts*, to provide financial services. This has spawned novel financial innovations such as automated execution, resulting in no market downtime, and permissionless access, increasing user inclusiveness and applications' interoperability compared to its traditional (TradFi) counterpart. This paradigm shift has led to increased utilization of DeFi systems in recent years, with several billion dollars exchanging hands daily [6].

Despite its success, DeFi is not free from risks, thus impeding its rapid adoption [7]–[9]. These risks can generally be classified as *technical* or *economic* [10]. Technical risks arise from exploiting the vulnerabilities of smart contracts, such as programming bugs and/or unintentional functional specification inconsistencies that can lead to irreversible loss of funds held by a DeFi protocol. The second type of risks, stem from economic inefficiencies within a protocol. Technical and economic exploitation, along with external factors, often lead to unexpected behavior in DeFi metrics. This includes fluctuations in token prices and liquidity in automated market-making (AMM) pools, as well as extreme borrowing rates in lending protocols[1]. Such deviations can rupture the functioning of dependent protocols [11] or result in losses for end users. For example, the liquid staking protocol Lido Finance [12] has a wrapped token for staked ETH (stETH), which is expected to maintain its price stability relative to ETH. However, if this characteristic fails due to the aforementioned reasons, stETH investors may incur unforeseen losses. This emphasizes the imperative need for robust *DeFi risk management tools* to safeguard users from such consequential risks.

One category of solutions to the above problem involves insurance, where the risk of users (policyholders) is transferred to underwriters. However, the very nature of DeFi demands an insurance platform that is (a) automated, ensuring instantaneous claim payout; (b) permissionless in access to achieve interoperability; (c) resilient against both technical and economic vulnerabilities, as it is the last resort for financial adversities; (d) flexible to underwrite a vast pool of policies; and (e) capital-efficient, *i.e.*, any locked capital should be capable of earning yield.

An insurance platform completely implemented using smart contracts achieves the first two properties, *viz.,* automation and permissionless access. However, the latter three merits, *viz.,* resilience, flexibility, and capital-efficiency *inherently* compromise the objectives of each other. For instance, a high level of resilience in contract design implies high immutability, which in turn hinders the underwriting flexibility derived from adaptability or upgradability. Simultaneously, the resilience of the smart contract platform may not align well with capital efficiency, as yield generation often involves interactions with third-party protocols that are susceptible to vulnerabilities.

This paper proposes BakUp, a design for a modular, flexible, and capital-efficient smart contract platform that uses binary conditional tokens to provide insurance policies. BakUp allows users to independently align between resilience, flexibility, and capital efficiency by comprising three distinct modules: a *core module* for basic capital accounting to ensure no defaults, an *oracle module* for underwriting policies, and a *yield module* on the periphery of the above two for capital management. The core module is immutable, while the oracle module can be created permissionlessly with a custom developer-defined logic, which can range from conservative to highly customizable. Simultaneously, the peripheral yield module operates independently from the main protocol, making it optional for users to engage in yield-generation activities. Interestingly, this optional feature doesn't pose any risk to the capital of users who choose not to participate, allowing users—both underwriters and policyholders—with varying risk preferences to interact simultaneously on the platform without imposing risks on others.

Here, we first present the design of the core module. Subsequently, we describe the oracle module and demonstrate a contract design capable of underwriting the de-pegging of stable assets using a combination of on-chain and off-chain price sources. Thereafter, we present the yield module's design that integrates the DeFi Risk Transfer protocol [13] for secure yield generation, allowing users to mitigate their technical risks by paying an additional premium. This allows users to choose between unsecured yield (high-risk/reward),

[1]https://finance.yahoo.com/news/defi-lenders-spooked-curve-exploit-193953614.html

secure yield (medium-risk/reward), or no yield at all (low-risk/reward). The insurance policies, implemented as a binary conditional ERC20 token, can be traded on both decentralized (DEX) and centralized exchanges (CEX). As such, this paper also presents a DEX marketplace by analyzing the divergence loss of liquidity providers in three broad strategies for the passive liquidity provision of these tokens. Empirical results show that conservative strategies and parameterization can help mitigate the loss of liquidity providers by over 47% compared to the naive strategy in the worst-case scenario.

This paper is organized as follows. Section II gives background knowledge, Section III presents the design and implementation of the various modules, Section IV gives a market analysis of the underlying tokens, including liquidity provision strategies, that are later evaluated in Section V. Section VI discusses related works, and Section VII concludes this work.

II. Background

A. DeFi Risk Transfer

Yield-generating protocols such as Aave [14] and Compound [15] are susceptible to risks arising from platform vulnerabilities. Thus, investing in these platforms carries a high probability of net positive yield and a low probability of net negative yield, the latter being attributed to platform vulnerabilities. DeFi Risk Transfer (DRT) [13] is a protocol inspired by collateral debt obligations (CDOs) [16] that allows investors interested in earning yield on an underlying asset C, such as USDC, to distribute the risk between risk-averse and risk-taking investors. The former can transfer the risk of these low-probability adverse events to the latter by paying a premium. DRT accomplishes this through the following steps.

1) *Tranche Creation:* DRT creates two tranches: a low-risk A-tranche with members holding an ERC20 token A and a high-risk B-tranche with members holding token B. A user depositing 1 unit of C mints 1 unit each of A and B.
2) *Investing:* Given k units of C as the total deposited capital, DRT equally invests this ($k/2$ units each) between two yield-generating platforms. In return, it receives yield-bearing tokens C_1 and C_2 respectively, serving as receipts. This capital is kept invested until a set deadline.
3) *Risk Splitting and Divesting:* After the deadline passes, DRT divests the capital by exchanging C_1 and C_2 at the first and second platform, respectively. Denoting k' as the total units of C redeemed and k_i as the interest accrued for each tranche, the capital is split between the two tranches based on one of the three scenarios:
   - $k' \geq k$: In the normal scenario, where DRT receives its entire capital plus additional yield, both A- and B-tranche receives $k'/2$ units each. This capital is then uniformly distributed among holders of A and B.
   - $k > k' > k/2$: In an adverse scenario where DRT receives only a partial fraction of the invested amount, A-tranche receives $k/2 + k_i$ units, while B-tranche receives $k' - (k/2 + k_i)$ units of C. Therefore, the risk is transferred from A- to B-tranche.
   - $k' \leq k/2$: In an extremely adverse scenario where DRT receives less than half of its initial investment, the entire capital, *i.e.,* k' units, is received solely by A-tranche.
4) *Pricing A and B:* External markets for A and B are established by creating a trading pair A/B on AMM-based DEXs. Since the expected pay-off of A is higher than B, the fair price of A in terms of B is greater than 1 and is decided by the market.
5) *End user:* A user willing to acquire either only A or B can first mint equal amounts of A and B and then exchange the other token for the desired token. For example, a user willing to invest 1 unit of C in A-tranche first mints 1 unit of A and B and then swap B in exchange for A at the AMM marketplace.

Since the architecture of DRT only transfers the risk related to yield-generating platforms, it misses out on the flexibility of underwritten policies.

B. Concentrated Liquidity Automated Market Makers

In the context of Concentrated Liquidity Automated Market Makers (CLAMM) [17], we consider a pair of tokens X and Y and measure the price of token X in terms of Y. When providing liquidity for this pair, a liquidity provider (LP) chooses a price interval $[p_0, p_1]$ with utility as explained in the next paragraph. This, along with the current price p, determines the ratio of the two tokens that the LP needs to deposit. In turn, the amount of tokens that the LP actually deposits (in accordance with the above ratio) determines the liquidity l provisioned to the interval.

The current price of the pool changes with each trade. The corresponding trading fees are distributed to all LPs in proportion to their liquidity in the interval containing the current price. If the price moves outside the chosen interval of an LP, their position consists of only one of the two tokens, and they stop earning fees. However, this resumes if the price returns to within their interval. Given a fixed amount of tokens that the LP wants to provision, the size of the price interval affects the proportion of fee earned. Depositing the tokens in a narrower interval increases LPs proportion of fees but leads to greater LP losses when price changes [18].

Considering a liquidity position in the interval $[p_0, p_1]$, let $r'_X(p)$ and $r'_Y(p)$ denote the *virtual reserves* of the position at a price of $p \in [p_0, p_1]$, *i.e.,* the amounts of tokens used to compute an incoming trade. Then $r'_X(p) r'_Y(p) = l^2$ and $r'_Y(p)/r'_X(p) = p$. This implies $r'_X(p) = l/\sqrt{p}$ and $r'_Y = l\sqrt{p}$. Therefore, the amount of tokens that the position actually consists of or the *real reserves* $r_X(p)$ and $r_Y(p)$ can be calculated as follows:

$$\begin{gathered} r_X(p) = r'_X(p) - r'_X(p_1) = l \cdot (\frac{1}{\sqrt{p}} - \frac{1}{\sqrt{p_1}}) \\ r_Y(p) = r_Y(p) - r'_Y(p_0) = l \cdot (\sqrt{p} - \sqrt{p_0}) \end{gathered} \tag{1}$$

Note that the above equations only hold for $p \in [p_0, p_1]$. Otherwise, we have $r_X(p) = r_X(p_0)$ and $r_Y(p) = 0$ for $p < p_0$, and $r_X(p) = 0$ and $r_Y(p) = r_Y(p_1)$ for $p > p_1$. Therefore, an LP entering a position in the above range with $p < p_0$ and exiting with $p > p_1$ is analogous to swapping $r_X(p_0)$ of X for $r_Y(p_1)$ of Y. The average execution price of this trade is therefore:

$$\frac{r_Y(p_1)}{r_X(p_0)} = \frac{l \cdot (\sqrt{p_1} - \sqrt{p_0})}{l \cdot (\frac{1}{\sqrt{p_0}} - \frac{1}{\sqrt{p_1}})} = \sqrt{p_0 p_1} \tag{2}$$

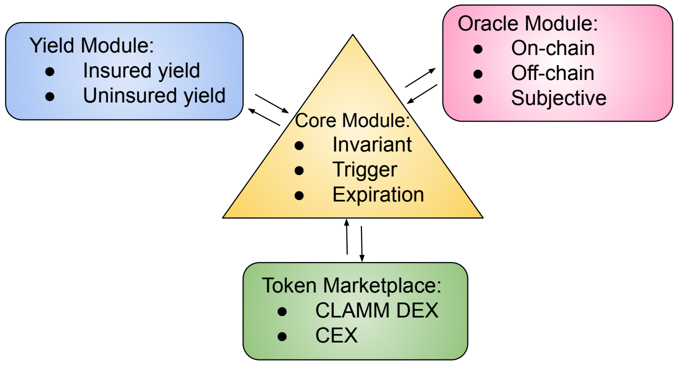


Fig. 1: An illustration of interactions within the BAKUP protocol.

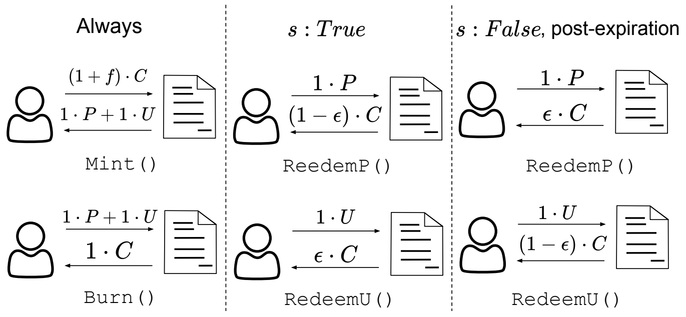


Fig. 2: Interaction between user and core module.

which is the geometric mean of the extreme price of the LP's position. We will use this result to evaluate liquidity provision strategies in Section V.

III. BAKUP PROTOCOL DESIGN

A. Protocol Overview

Consider an example where our protocol creates an insurance mechanism for the event of stETH de-pegging in which the price of stETH relative to ETH falls below 0.95. In its simplest version, it creates two tokens: a policy token P and an underwriting token U for a predetermined period, *e.g.,* 1 year. The payoffs of these tokens depend on the occurrence of the underlying event: if the price falls below 0.95 before the deadline, P pays \$1, while U pays \$0. Otherwise, after the deadline passes, P pays \$0, while U pays \$1. For P-holders, it creates a hedging mechanism for the de-pegging event during the specified period, whereas U-holders provide hedging in return for a premium.

The actual payoffs of P and U are slightly different from the above description, however, the underlying motivation remains the same. The protocol design is modular with key components divided into the core, oracle, and yield modules, as illustrated in Figure 1. The subsequent subsections describe the modeling of financial adversities, followed by a detailed and formal description of each of these modules.

B. Binary Event

We define a *binary event* being the tuple (a, t, s) as follows:

- Assertion a: A binary event is defined by an immutable assertion a. For the previous example, the assertion is defined as the price of stETH relative to ETH, $p_{\text{stETH/ETH}}$, being less than 0.95 and represented as $p_{\text{stETH/ETH}} < 0.95$.
- Expiration period t: This denotes the time duration since the inception of a binary event during which it remains valid. In the previous example, this was 1 year.
- State s: A binary event can be in either of the two states: *True* or *False*. By default, an event is in the *False* state. If the underlying assertion holds at any instance before the event's expiration, the state shifts to *True* and becomes immutable. Moreover, the state does not change once the event expires.

Therefore, for an adverse effect modeled as a binary event, the occurrence of the $True$ state signifies an insurance claim.

Modelling complex adversities: A binary event only captures a discrete incident. However, in reality, adversities can have varying degrees of damage. To address this, our protocol requires an adversity to be modeled as a collection of $n \in \mathbb{N}$ binary events, where the i^{th} event is represented as (a_i, t, s_i), with all of them having a common expiration.

For instance, a model of continuous incident of stETH de-pegging consists of the following three assertions:

1) $a_1 : p_{\text{stETH/ETH}} < 0.99$.
2) $a_2 : p_{\text{stETH/ETH}} < 0.98$.
3) $a_3 : p_{\text{stETH/ETH}} < 0.95$.

In the above example, if only a_1 holds by the deadline, *i.e.,* $(s_1, s_2, s_3) = (True, False, False)$, it corresponds to a mild de-pegging incident; if both a_1 and a_2 hold, *i.e.,* $(s_1, s_2, s_3) = (True, True, False)$, it represents a moderate de-pegging incident; and lastly, if all the three assertions hold, *i.e.,* $(s_1, s_2, s_3) = (True, True, True)$, it represents an extreme de-pegging incident. Therefore, the final state values together signifies the degree of damage.

C. Core Module

For an adversity modeled as a collection of n binary events, the core module contract creates n pairs of ERC20 tokens P_i, U_i with $i \in \{1 \dots n\}$. Each core contract is characterized by a base currency C (*eg.,* ETH, USDC) in which claims are disbursed. The following module methods can be invoked for each binary event from the inception of the module to perpetuity. Here f represents a constant fee such that $f \in [0, 1]$.

- `Mint(i)`: A user depositing $k(1+f)$ units of C, where $k > 0$, receives k units each of P_i and U_i. Here, kf units of C get deposited as fees in the `feeTo` address specified by the core module's creator.
- `Burn(i)`: A user depositing k units of P_i and U_i receives k units of C.

These methods are depicted in the left part of Figure 2 and they enforce the following two invariants:

Invariant 1. $(1 + f) \cdot C \Rightarrow P_i + U_i$

Invariant 2. $P_i + U_i \Rightarrow C$

Next, we describe methods to redeem the above tokens in exchange for C based on the state of a binary event. At any point in time, $\forall i$ s.t. $s_i = True$, P_i and U_i can be redeemed for the following:

$$P_i : (1 - \epsilon) \cdot C, \quad U_i : \epsilon \cdot C \tag{3}$$

Here ϵ is a constant whose value is set by the module creator and is generally close to zero, *i.e.,* $0 < \epsilon \ll 1$. The reason for such a choice of ϵ is justified in Section V as it relates to the risks of the LPs on AMM. After the events expire, the module tokens can be redeemed for binary events with $False$ state as well, *i.e.,* $\forall i$ s.t. $s_i = False$, P_i and U_i can be redeemed for:

$$P_i : \epsilon \cdot C, \quad U_i : (1-\epsilon) \cdot C \tag{4}$$

Note that the above payoff values satisfy **Invariant** (2) since the sum of payoffs of $1 \cdot P_i$ and $1 \cdot U_i$ is always $1 \cdot C$, regardless of the event's state. The redemptions are implemented using the `RedeemP(i)` and `RedeemU(i)` methods, respectively, which are depicted in the middle and right parts of Figure 2.

Let p_{P_i}, p_{U_i} be the prices of P_i, U_i in terms of C. A policyholder purchasing $1 \cdot P_i$ from the market pays a one-time premium of p_{P_i} to obtain an insurance coverage of $(1-\epsilon) \cdot C$. This gives a coverage-to-premium ratio of the policy to be $(1-\epsilon)/p_{P_i}$.

On the other hand, a user underwrites a policy only if there is sufficient incentive for them. One such scenario is when the price of the policy token increases significantly. This is because **Invariant** (1),(2) imply:

$$1 \leq p_{P_i} + p_{U_i} \leq 1 + f \tag{5}$$

as one can always acquire $1 \cdot P_i + 1 \cdot U_i$ for $(1+f) \cdot C$ using `Mint()` and can always dispose the same for $1 \cdot C$ using `Burn()`. Therefore, if the price of the policy token increases by more than $f + \epsilon$, the following holds:

$$f + \epsilon + p_{U_i} < p_{P_i} + p_{U_i} \leq 1 + f \tag{6}$$

Hence, the price of the underwriting token becomes less than $1 - \epsilon$. This incentivizes underwriters to acquire U_i at a lower price and later redeem it for $(1-\epsilon) \cdot C$ if the assertion does not hold, with the difference representing the earned premium.

In summary, the immutability of the core module and an invariant-based redemption guarantee zero risk of default, thereby ensuring the robustness of the BAKUP protocol.

D. Oracle Module

The oracle module is an externally deployed contract with a custom-defined logic. This logic takes input values from external data sources and executes the logic of assertion for each event. A user willing to create a core contract on BAKUP specifies an oracle contract address to associate with it. This is because the core module consists of a `Trigger(i)` function for the i^{th} event that executes a callback function `OTrigger(i)` in the oracle contract. `OTrigger(i)` executes the logic of the i^{th} assertion and returns a boolean value back to `Trigger(i)` in the core contract. This is presented as a flowchart diagram in Figure 3.

The returned value $False$ is ignored; however, in the other case, the following occurs:

1) s_i transitions to *True* and becomes immutable.
2) The `RedeemP(i)`, and `RedeemU(i)` methods are set to exchange as per (3).

When the contract deadline passes, the following occurs:

1) The `Trigger()` method is rendered invalid.
2) s_i becomes immutable for all events.

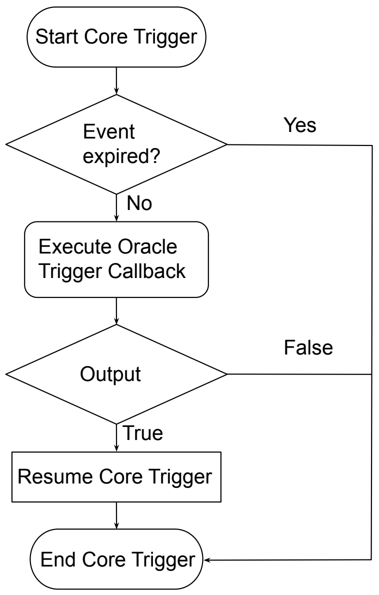


Fig. 3: Flow of Trigger execution using Oracle module.

3) The `RedeemP(i)` and `RedeemU(i)` functions become valid for events with a $False$ state, redeeming tokens based on Relation (4).

Custom developer-defined logic enables extensive functionalities for the oracle module. Simultaneously, a modular structure ensures that the accounting performed by the core module remains unaffected by the execution of the oracle module. This allows users to simultaneously reconcile platform resilience and flexibility of policies.

External information, such as prices and liquidity on AMM, can be sampled from on-chain or off-chain oracle sources. Given the potential vulnerability of oracle sources to manipulation [19], [20], a combination of sources with complementary strengths can be employed to mitigate such risks. We illustrate this with an example for the `OTrigger()` function below.

Illustrative Example: Listing 4 presents pseudocode for the `OTrigger()` function that detects a reduction in the price of stETH from ETH by more than 5%. The algorithm achieves this by sampling data from two sources: (a) Uniswap V3 pool for stETH/ETH, an on-chain oracle source, and (b) Chainlink [21] price feed for stETH/ETH, an off-chain oracle source (without loss of generality for the underlying price-feeding system). In the first, Uniswap provides the sum of the log of prices at each second since the inception of the pool until a given time. The algorithm samples these values at time 0 and 3600 seconds ago, and divides their difference by 3600. This computation yields the log of time-weighted geometric mean of the price over the last 60 minutes [17]. If both this value and the price from Chainlink are less than the threshold[2], `OTrigger()` returns $True$; otherwise, it returns $False$.

In the above example, it is in the interest of P-holders to call `Trigger()` when de-pegging occurs. However, this incurs a

[2]In Listing 4, the log is computed with the base 1.0001. Thus, `avgLogPrice` is compared with $log_{1.0001}(0.95) = -513$.

```
fn OTrigger() returns (bool outcome) {
    address pool = 0xfff;
    seconds = [0,3600];
    logAgg = IUniswapV3Pool(pool).observe(seconds);
    logAggDelta = logAgg[1] -logAgg[0];
    avgLogPrice = logAggDelta / 3600;
    price2 = ChainlinkFeed.latestRoundData()
    if (avgLogPrice < -513 && price2 < 0.95){
        return True;
    } else {
        return False;
    }}
```

Fig. 4: A code snippet demonstrating oracle contract to detect de-pegging of stETH/ETH pair.

gas fee to the caller, while the benefits of the call are shared among all P-holders, creating a first-mover's disadvantage. To address this issue, the caller gets compensated with the fee collected by the `feeTo` address. To detect uncommon events with insufficient or unreliable oracle sources, the oracle contract can delegate the assessment to decentralized dispute resolution platforms, such as Kleros [22]. This, coupled with the customizable nature of the oracle contract, broadens the range of events detectable by BAKUP.

E. Yield Module

The BAKUP protocol enables custom peripheral yield modules where P- and U-holders have the *option* to earn yield on their locked C. The key idea is to utilize the perpetual nature of the `Mint()` and `Burn()` methods of the core module, along with the trustless execution of smart contracts. Below, we present methods for a module design that delegates yield generation to a DRT protocol comprising A- and B-tranches (low- and high-risk respectively), and earning yield on C. Figure 5 depicts the sequence of the following methods.

1) `Deposit()`: P- and U-holders willing to earn yield within A-tranche of DRT deposit their tokens in the yield contract. Let there be k_P units of P and k_U units of U and let us define λ as $min(k_P, k_U)$.
2) `Invest()`: Since one can only burn equal quantities of the two tokens, the yield module burns λ units of P and U to obtain λ units of C and invests this capital in DRT's A-tranche. Observe that for maximum utilization of the tokens, the quantities of the two tokens needs to be equal.
3) `Divest()`: After the expiration of DRT, the module divests its position from A-tranche to obtain λ' units of C. Based on the success of divesting, λ' can be greater than, equal to, or less than λ as described in Section II-A.
4) `Distribute()`: The yield contract mints $\lambda'/(1+f)$ units of P and U and distributes them, along with the unburnt tokens, uniformly among depositors.

The above works analogously for the B-tranche of DRT.

Without loss of generality, let $k_P \geq k_U$ making $\lambda = k_U$ and let $k'_U = \lambda'/(1+f)$. Then, in the scenario of a successful divesting, $k'_U - k_U$ units of P and U are uniformly distributed as yield to their holders. Thus, the holder of $1 \cdot U$ receives a yield fraction of $(k'_U - k_U)/k_U$ while the holder of $1 \cdot P$ receives a yield fraction of $(k'_U - k_U)/k_P$, which is k_U/k_P times the yield earned per unit of U. Therefore, if there are more policyholders than underwriters, k_U/k_P becomes

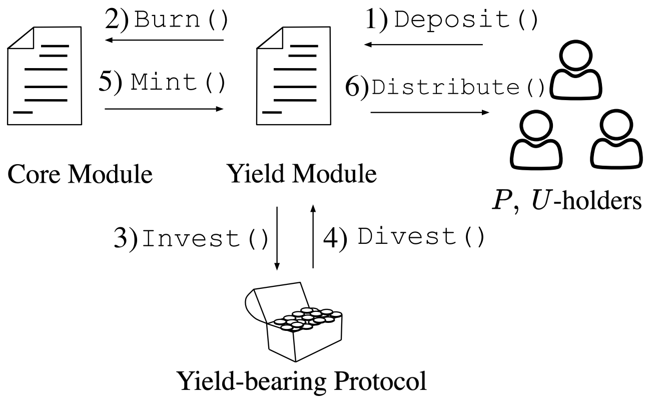


Fig. 5: Sequence of methods in the yield module with arrows indicating direction from caller to callee.

smaller, attracting fewer additional policyholders than underwriters until k_U/k_P returns to 1.

Optional yield earning and investing via DRT in the BAKUP protocol naturally creates multiple categories of user classification based on their risk appetite:

1) *High-risk / high-reward:* This encompasses underwriters investing in B-tranche. The risk for these users arises from the underlying insurance event, coupled with an increased divesting risk in the B-tranche of DRT. In return, they earn a premium for both underwriting insurance and investment in the B-tranche.
2) *Medium-high risk / medium-high reward:* This includes:
   - Underwriters investing in A-tranche: They bear risk from underwriting, with a lower risk of divesting.
   - Policyholders investing in B-tranche: Their risk related to adversity is hedged, but they face a higher risk of divesting.
3) *Medium-low risk / medium-low reward:* This includes:
   - Underwriters not investing at all: They bear risk solely from underwriting.
   - Policyholders investing in A-tranche: Their risk related to adversity is hedged, and they face a lower risk of divesting.
4) *Low-risk / Low-reward:* This includes policyholders not investing at all. While they are hedged against adversity with no additional risk, they miss out on yield returns.

Since yield-bearing smart contract protocols are susceptible to vulnerabilities, the peripheral design of the yield module ensures that it doesn't disrupt the functionality of the other two modules. Moreover, the above methods can be adapted, allowing room for custom designs and enabling users to earn yield on their preferred platform.

In summary, instead of making decisions on the user's behalf, the above design grants them the flexibility to make individualized risk management decisions, *i.e.,* divesting early or substituting the underlying yield-bearing platform if possible, without impacting others. While having flexible policies, users therefore trade-off between platform risk exposure and capital efficiency.

IV. TOKEN MARKETPLACE

For the proper functioning of BAKUP, it is essential to have an efficient marketplace where users can buy or sell policy and underwriting tokens in exchange for a common currency. Therefore, studying the incentive mechanism for various users to participate in the marketplace is crucial. This includes examining the risk-adjusted returns for the liquidity providers. In this paper, we focus on studying LP strategies for the trading pair P_i/U_i on AMMs, although other pairs such as P_i/C or U_i/C can also exist. This is because the former pair is sufficient for buying or selling P_i and U_i in exchange for C (as explained next). Another advantage is that the liquidity provision for this trading pair does not require currency tokens.

Let p_i denote the spot price of P_i relative to U_i. A user purchasing a small quantity δ of P_i (such that slippage on AMM is negligible) can perform the following ordered actions in a single transaction:

- Mint $\frac{\delta p_i}{1+p_i}$ units of P_i and U_i by depositing $\frac{\delta p_i}{1+p_i}(1+f)\cdot C$.
- Swap $\frac{\delta p_i}{1+p_i}\cdot U_i$ for $\frac{\delta}{1+p_i}\cdot P_i$ on AMM.

The user now holds $\delta \cdot P_i$. Similar can be done when the user wants to sell $\delta \cdot P_i$:

- Swap $\frac{\delta}{p_i+1}\cdot P_i$ for $\frac{\delta p_i}{1+p_i}\cdot U_i$ on AMM.
- Burn equal P_i and U_i to receive $\frac{\delta p_i}{1+p_i}\cdot C$.

The procedure for buying and selling underwriting tokens is analogous to the above. As a result, the effective buying and selling prices of P_i in terms of C are $\frac{p_i}{1+p_i}(1+f)$ and $\frac{p_i}{1+p_i}$, respectively and for U_i, they are $\frac{1}{1+p_i}(1+f)$ and $\frac{1}{1+p_i}$, respectively. Therefore, the market's risk assessment results in different values of policy and underwriting tokens, and it gets expressed as a single price p_i. The following describes various strategies for liquidity providers on concentrated liquidity AMMs, which we then evaluate in the subsequent section.

LP strategy: Assuming $f \approx 0$ and negligible risk-free interest rate, the value of p_i satisfies:

$$\frac{\epsilon}{1-\epsilon} \leq p_i \leq \frac{1-\epsilon}{\epsilon} \tag{7}$$

This is because if $p_i < \frac{\epsilon}{1-\epsilon}$, then the price of P_i relative to C satisfies:

$$p_{P_i} = \frac{p_i}{1+p_i} = (1 - \frac{1}{1+p_i}) < \epsilon \tag{8}$$

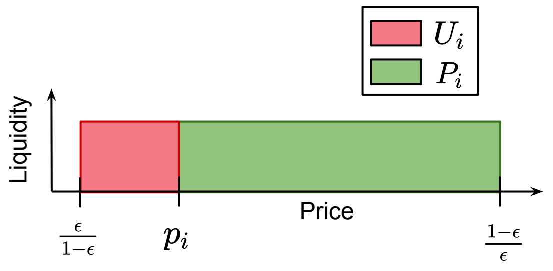
Given that P_i can be redeemed for at least $\epsilon \cdot C$ at the contract expiration, irrespective of the final state, a trader can purchase them at a price p_{P_i} and subsequently redeem them at a higher price of ϵ at contract expiration, thus creating an arbitrage opportunity.

Similarly, if $p_i > \frac{1-\epsilon}{\epsilon}$, then the price of U_i relative to C satisfies:

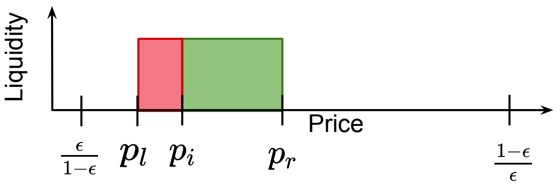
$$p_{U_i} = (\frac{1}{1+p_i}) < \epsilon \tag{9}$$

As U_i can also be redeemed for at least $\epsilon \cdot C$ upon contract expiration, another arbitrage opportunity arises. A trader can purchase them at a lower price of p_{U_i} and subsequently redeem them for at least $\epsilon \cdot C$ at the contract expiration. Consequently, Inequality (7) holds.

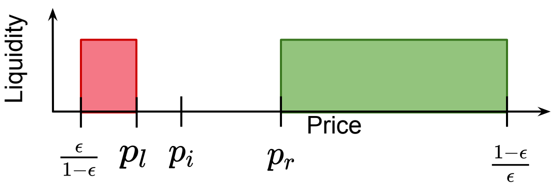
Considering a concentrated liquidity pool for the pair P_i/U_i, the LPs thus need to deposit policy tokens in the price interval $[p_i, \frac{1-\epsilon}{\epsilon}]$ (prices above the spot price) and underwriting tokens in the price interval $[\frac{\epsilon}{1-\epsilon}, p_i]$ (prices below the spot price) with some distribution. Suppose an LP mints 1 unit each of two tokens, and provisions them as liquidity with p_i being the entry spot price. We consider three limiting passive strategies for liquidity provision:



(a) Uniform LP strategy



(b) p_i concentrated LP strategy



(c) Edge concentrated LP strategy

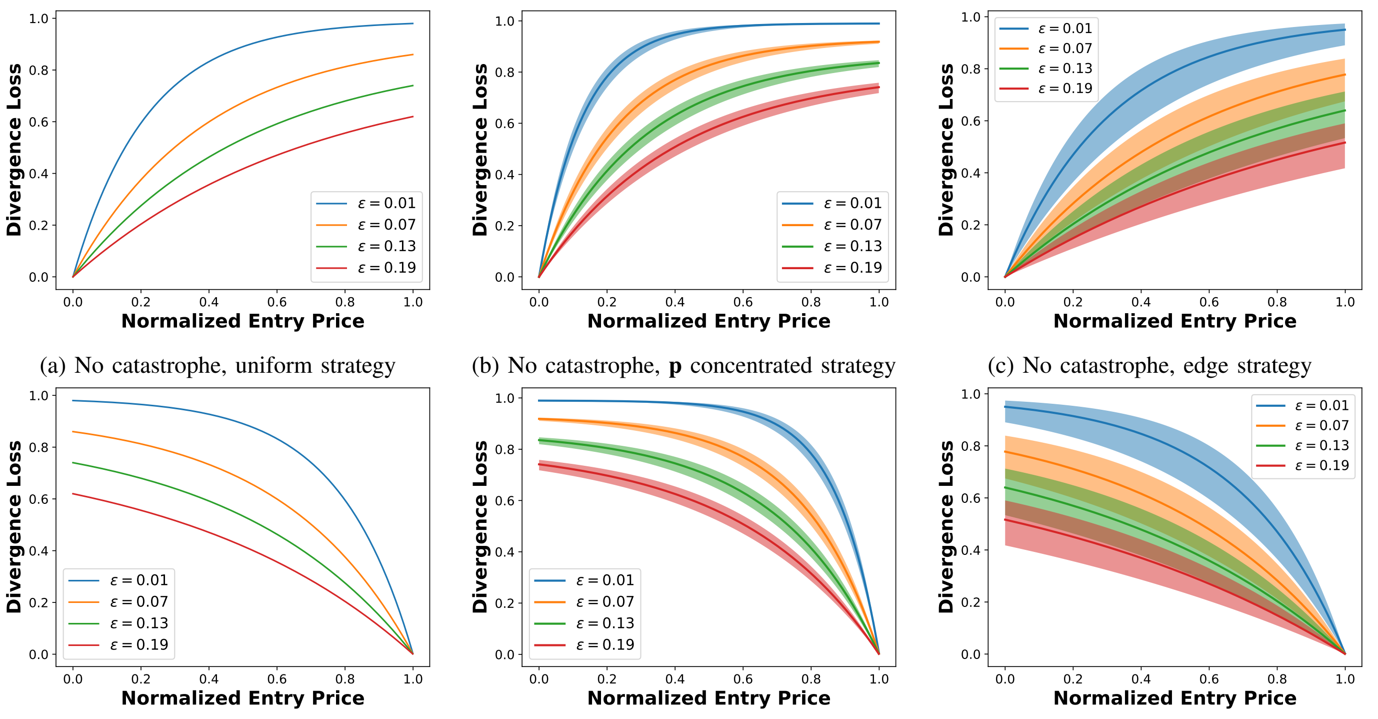
Fig. 6: Liquidity provision under the three categories.

1) **Uniform:** This strategy consists of depositing $1 \cdot P_i$ in the price interval $[p_i, \frac{1-\epsilon}{\epsilon}]$, and $1 \cdot U_i$ in the price interval $[\frac{\epsilon}{1-\epsilon}, p_i]$ with the two intervals having a constant liquidity. This is depicted in Figure 6a.
2) **Concentrated around p_i:** This strategy involves depositing, with a constant liquidity, $1 \cdot P_i$ in $[p_i, p_r]$, and $1 \cdot U_i$ in $[p_l, p_i]$ for some p_l, p_r. Such a strategy is expected to earn a higher trading fee than the uniform strategy since liquidity is concentrated around the spot price. This is depicted in Figure 6b.
3) **Concentrated at the edges:** This strategy consists of depositing, with a constant liquidity, $1 \cdot P_i$ in $[p_r, \frac{1-\epsilon}{\epsilon}]$, and $1 \cdot U_i$ in $[\frac{\epsilon}{1-\epsilon}, p_l]$ for some p_l, p_r. Such a strategy is expected to earn lower trading fees than the uniform strategy since liquidity is concentrated away from the spot price. This is depicted in Figure 6c.

The next Section evaluates the divergence loss of the LPs under the above strategies and examines ways to mitigate risk.

V. Divergence Loss Evaluation

Divergence Loss is defined as the opportunity cost for an LP to provide token reserves as liquidity compared to just holding them. In this section, we assess the divergence loss experienced by liquidity providers across the three strategies: uniform, concentrated around the entry price, and concentrated



(a) No catastrophe, uniform strategy (b) No catastrophe, **p** concentrated strategy (c) No catastrophe, edge strategy

(d) Catastrophe, uniform strategy (e) Catastrophe, **p** concentrated strategy (f) Catastrophe, edge strategy

Fig. 7: Comparison of divergence loss among three liquidity provision strategies (uniform, concentrated around the entry price, concentrated at the interval edges) in scenarios without and with catastrophes.

at the price interval edges (Figure 6). Our evaluation addresses the following two questions: (1) Which parameters significantly impact the divergence loss of liquidity providers and to what extent? (2) What are some optimal techniques to mitigate the risk of divergence loss for liquidity providers?

We adhere to an evaluation methodology where we assume that an LP mints 1 unit of P_i and U_i and creates a liquidity position with an entry price $p_i = \mathbf{p}$. If there is no liquidity provision, the value of these tokens at the conclusion of the contract is always 1 unit of C (enforced by Invariant (2)). To evaluate the divergence loss, we calculate the total value of the tokens held by the LP at the conclusion of the contract and represent it as a fraction of $1 \cdot C$. We use four equally-spaced values for ϵ : $\{0.01, 0.07, 0.13, 0.19\}$ and do not consider any further higher values since $\epsilon \ll 1$. For plotting, we take the natural logarithm of the x-axis, for the price interval $[\frac{\epsilon}{1-\epsilon}, \frac{1-\epsilon}{\epsilon}]$, and then min-max normalize it to the range $[0, 1]$. Here is a summary of divergence loss in the three strategies:

1) Uniform: If there is no catastrophe, *i.e.,* the underlying assertion does not hold, $p_i = \frac{\epsilon}{1-\epsilon}$ at event expiration and the LP ends up with only P_i. Their $1 \cdot U_i$ is exchanged for an average price of $\sqrt{\frac{\epsilon \mathbf{p}}{1-\epsilon}}$, which is the geometric mean of the extreme prices $\{\frac{\epsilon}{1-\epsilon}, \mathbf{p}\}$ as derived in Section II-B. Therefore, the LP ends up with $1+\sqrt{\frac{1-\epsilon}{\epsilon \mathbf{p}}}$ units of P_i each worth $\epsilon \cdot C$, thus a total value of $\epsilon + \sqrt{\frac{\epsilon(1-\epsilon)}{\mathbf{p}}}$ units of C.
When there is a catastrophe, the LP ends up with only U_i. As above, their $1 \cdot P_i$ is exchanged for an average price of $\sqrt{\frac{(1-\epsilon)\mathbf{p}}{\epsilon}}$, giving them a total of $1 + \sqrt{\frac{(1-\epsilon)\mathbf{p}}{\epsilon}}$ units of U_i. Since U_i is worth $\epsilon \cdot C$, they receive a value of $\epsilon + \sqrt{\mathbf{p}\epsilon(1-\epsilon)}$ units of C.
The above two cases are shown in Figure 7a & 7d, respectively. We can observe that when the normalized entry price is closer to $0(1)$, the divergence loss is higher for the catastrophic(non-catastrophic) scenario. Also, for a given $\mathbf{p}$, the divergence loss is lower at higher values of ϵ. The worst loss (at $\mathbf{p}$=$\frac{\epsilon}{1-\epsilon}$ or $\frac{1-\epsilon}{\epsilon}$) can be reduced from 0.98 to 0.62 by varying ϵ from 0.01 to 0.19.

2) Concentrated around $\mathbf{p}$: In the case with no catastrophe, the final price $\frac{\epsilon}{1-\epsilon}$ is less than p_l. Therefore, the LP's $1 \cdot U_i$ is exchanged for an average price of $\sqrt{p_l \mathbf{p}}$, which is the geometric mean of the extreme prices $\{p_l, \mathbf{p}\}$. As before, this gives a total value of $\epsilon + \frac{\epsilon}{\sqrt{p_l \mathbf{p}}}$ units of C. If there is a catastrophe, the LP's $1 \cdot P_i$ is exchanged for an average price of $\sqrt{\mathbf{p} p_r}$. This gives a total value of $\epsilon + \epsilon\sqrt{\mathbf{p} p_r}$ units of C.
These two cases are shown in Figure 7b & 7e. For a given $\mathbf{p}$ and ϵ, we choose a range of values for the widths $\mathbf{p} - p_l$, and $p_r - \mathbf{p}$. This range represents $10-50\%$ of $\mathbf{p}$ for $\mathbf{p} - p_l$, and $10-50\%$ of $1-\mathbf{p}$ for $p_r - \mathbf{p}$ and is represented by the shaded region in the figure. The median value of the widths, given by $\mathbf{p} - p_l = 0.3\mathbf{p}$, and $p_r - \mathbf{p} = 0.3(1-\mathbf{p})$, is represented by the solid line. For a given $\mathbf{p}$ and ϵ, the divergence loss in this strategy is higher compared to the uniform strategy, particularly between 1-19.4% at the interval edges ($\mathbf{p}$=$\frac{\epsilon}{1-\epsilon}$ or $\frac{1-\epsilon}{\epsilon}$) for different values of ϵ, and median values for p_l and p_r. This is the cost

LPs need to pay to achieve capital concentration (hence a higher fee revenue) in this strategy. However, the LPs can reduce their loss by choosing p_l, p_r farther from $\mathbf{p}$. The divergence loss, as in the previous case, reduces for higher values of ϵ with the worst loss reducing from 0.99 to 0.74 by choosing ϵ from 0.01 to 0.19.

3) Concentrated at the edges: As before, in the case with no catastrophe, the LP's $1 \cdot U_i$ is exchanged for an average price of $\sqrt{\frac{\epsilon p_l}{1-\epsilon}}$. This gives a total value of $\epsilon + \sqrt{\frac{\epsilon(1-\epsilon)}{p_l}}$ units of C. If there is a catastrophe, the LP's $1 \cdot P_i$ is exchanged for an average price of $\sqrt{\frac{p_r(1-\epsilon)}{\epsilon}}$. This gives a total value of $\epsilon + \sqrt{p_r \epsilon (1-\epsilon)}$ units of C. These cases are shown in Figure 7c & 7f respectively. For a given $\mathbf{p}$ and ϵ, we chose a range of values for the widths $\mathbf{p} - p_l$, and $p_r - \mathbf{p}$ as before. One can observe that the divergence loss in this strategy is the least of the three strategies. Compared to the uniform strategy, it is lower by 3.1-16.1% at the interval edges ($\mathbf{p}$=$\frac{\epsilon}{1-\epsilon}$ or $\frac{1-\epsilon}{\epsilon}$), and median values for p_l and p_r. The loss increases when p_l, p_r are farther from the edges. The lower divergence loss comes from a lower expected fee revenue since the position remains inactive when the price is outside its range. Lastly, the divergence loss reduces further with higher values of ϵ, with the worst loss reducing from 0.95 to 0.51 by choosing ϵ from 0.01 to 0.19. This reduction is more than 47% when compared to uniform LP strategy with $\epsilon = 0.01$.

Of the two tokens (policy and underwriting) that an LP initially holds, they always end up with the token of lesser value. If ϵ is set to 0, the LP ends with tokens worth zero and incurs a divergence loss of 1. As this is unfavourable to LPs, it underscores the necessity of enforcing $\epsilon > 0$.

The three LP strategies present a *trade-off* between divergence loss and increased fee revenue, resulting from enhanced capital concentration near the entry price. Additionally, an entry price closer to 0 entails lower risk in a non-catastrophic outcome but higher risk in a catastrophic outcome, while the opposite is true for an entry price closer to 1. Opting for a larger value of ϵ is another way to mitigate divergence loss risk, bringing it to as low as 0.51 in the worst-case. However, since the policy tokens pays at least $\epsilon \cdot C$ regardless of the final state, one consequence of larger ϵ is a higher token price relative to C. This results in a lower coverage-to-premium ratio ($(1-\epsilon)/p_{P_i}$), even for low-probability adversities, and thus, one must consider this trade-off.

To summarize, BAKUP gives its users–policyholders, underwriters, and liquidity providers–the tools to individually manage their risks and rewards. This is achieved through a modular approach. The invariant-based and immutable design ensures the robustness of the core module, the customizability of the oracle module enforces the flexibility of the insured policy, and most significantly, the optional and peripheral yield module empowers users to trade-off capital efficiency with external risk without compromising the security of the other users.

VI. RELATED WORK

Prediction Markets: These markets involve payoff contracts contingent on future events. Specifically, a contract is termed *winner-take-all* if priced at p units of C, its payoff is $1 \cdot C$ only if a specific event occurs. The price of such contracts reflect the market's expectation of the probability of the contingent event occurring, assuming risk neutrality [23]. For the case where $\epsilon = 0$, the payoff of the BAKUP's policy token aligns with that of a winner-take-all contract contingent on the underlying assertion of the binary event.

Decentralized Insurance: There have been several designs proposed for smart contract-based decentralized insurance for DeFi. Nexus Mutual [24] and inSure [25] are the largest of them and are inspired by the design of mutual insurance where platform users have a stake in the commercial success of the protocol [26]. Unlike the work presented here, they do not support permissionless policy listing or optional yield generation. Rather, a fraction of the pooled capital is invested in yield-generation protocols, transferring risk to all members. These decisions and others, like claim assessment, are done via voting using governance tokens. However, in the case of Nexus Mutual, the majority of NXM tokens are held by a handful of addresses.[3]

Other protocols, including ours, are based on Peer-to-Peer decentralized insurance [27] where individuals pool their insurance premiums and use these funds to mitigate individual damages. Some of them including Risk Harbour [28], Opium Protocol [29], cozy.finance [30], and Etherisc [31] facilitate automated underwriting done via smart contract while others like Unslashed Finance [32], Nsure [33], and Cover [34] have a voting-based claim assessment. Moreover, some of them including [29], [30] allow permissionless listing. In contrast to BAKUP, none of the above protocols give their users the optional ability to manage yield on their capital. Instead, either the protocol offers no yield, or the yield management is performed by governance or delegated to a third party [35]. Some designs, including Risk Harbour V1 [36], propose using receipt tokens from yield platforms (*e.g.,* cTokens on Compound) as disbursement currency. Such an approach permanently links the risk of the yield platform with the protocol's underwriting module. On the other hand, BAKUP disjoints the core, oracle, and yield modules and allows users to manage yield on their staked capital while having a permissionless policy creation.

VII. CONCLUSION

A robust risk management system for the emerging area of DeFi requires a primitive platform with minimal external dependencies. This is successfully achieved in the BAKUP protocol presented in this paper through a modular approach. At the same time, simulations suggest various mitigation strategies to minimize the risk for liquidity providers, contributing to the establishment of an efficient AMM marketplace. Possible future work in this direction includes *(a)* extending the conditional tokens from binary to n-ary with n states to reduce the number of ERC20 tokens per adversity; *(b)* enabling the creation of a single underwriting and multiple corresponding policy tokens to model a collection of adversities as a single event. Such an approach is expected to reduce the total capital requirement from underwriters, thus improving capital efficiency.

[3]https://dune.com/queries/1445980/2529493

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, https://bitcoin.org/bitcoin.pdf.

[2] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2014, https://ethereum.org/en/whitepaper/.

[3] T. Rocket, "Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies," *Available [online].[Accessed: 4-12-2018]*, 2018.

[4] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0. 8.13," *Whitepaper*, 2018.

[5] A. Skidanov and I. Polosukhin, "Nightshade: Near Protocol Sharding Design," 2019, https://near.org/downloads/Nightshade.pdf.

[6] H. Arslanian, "Decentralised finance (defi)," in *The Book of Crypto: The Complete Guide to Understanding Bitcoin, Cryptocurrencies and Digital Assets*. Springer, 2022, pp. 291–313.

[7] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings 6*. Springer, 2017, pp. 164–186.

[8] L. Gudgeon, D. Perez, D. Harz, B. Livshits, and A. Gervais, "The decentralized financial crisis," in *2020 crypto valley conference on blockchain technology (CVCBT)*. IEEE, 2020, pp. 1–15.

[9] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais, "Sok: Decentralized finance (defi) attacks," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2444–2461.

[10] S. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. Knottenbelt, "Sok: Decentralized finance (defi)," in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, 2022, pp. 30–46.

[11] F. Bekemeier, "Deceptive assurance? a conceptual view on systemic risk in decentralized finance (defi)," in *Proceedings of the 2021 4th International Conference on Blockchain Technology and Applications*, 2021, pp. 76–87.

[12] "Lido: Ethereum liquid staking," 2020, https://lido.fi/static/Lido:Ethereum-Liquid-Staking.pdf.

[13] M. Nadler, F. Bekemeier, and F. Schär, "Defi risk transfer: Towards a fully decentralized insurance protocol," in *2023 IEEE international conference on blockchain and cryptocurrency (ICBC)*. IEEE, 2023, pp. 1–9.

[14] "Aave v3 technical paper," 2022, https://github.com/aave/aave-v3-core/blob/master/techpaper/.

[15] R. Leshner and G. Hayes, "Compound: The money market protocol," *White Paper*, 2019.

[16] D. Duffie and N. Garleanu, "Risk and valuation of collateralized debt obligations," *Financial analysts journal*, vol. 57, no. 1, pp. 41–59, 2001.

[17] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, "Uniswap v3 core," *Tech. rep., Uniswap, Tech. Rep.*, 2021.

[18] S. Loesch, N. Hindman, M. B. Richardson, and N. Welch, "Impermanent loss in uniswap v3," *arXiv preprint arXiv:2111.09192*, 2021.

[19] G. Angeris and T. Chitra, "Improved price oracles: Constant function market makers," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 80–91.

[20] B. Liu, P. Szalachowski, and J. Zhou, "A first look into defi oracles," in *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 2021, pp. 39–48.

[21] L. Breidenbach, C. Cachin, B. Chan, A. Coventry, S. Ellis, A. Juels, F. Koushanfar, A. Miller, B. Magauran, D. Moroz *et al.*, "Chainlink 2.0: Next steps in the evolution of decentralized oracle networks," *Chainlink Labs*, vol. 1, pp. 1–136, 2021.

[22] W. G. Clement Lesaege and F. Ast, "Kleros:a decentralized decision protocol," 2021, https://kleros.io/yellowpaper.pdf.

[23] J. Wolfers and E. Zitzewitz, "Prediction markets," *Journal of economic perspectives*, vol. 18, no. 2, pp. 107–126, 2004.

[24] H. Karp and R. Melbardis, "Nexus mutual whitepaper: A peer-to-peer discretionary mutual on the ethereum blockchain," 2017, https://nexusmutual.io/assets/docs/nmx_white_paperv2_3.pdf.

[25] inSureDAO, "insure ecosystem: Decentralized insurance platform," https://insuretoken.net/whitepaper.html.

[26] P. Albrecht and M. Huggenberger, "The fundamental theorem of mutual insurance," *Insurance: Mathematics and Economics*, vol. 75, pp. 180–188, 2017.

[27] R. Feng, M. Liu, and N. Zhang, "A unified theory of decentralized insurance," *Available at SSRN 4013729*, 2022.

[28] M. Resnick, R. Ben-Har, D. Patel, and A. Bipin, "Risk harbor v2," 2022, https://github.com/Risk-Harbor/RiskHarbor-Whitepaper/blob/main/Risk%20Harbor%20Core%20V2%20Whitepaper.pdf.

[29] O. team, "Opium protocol whitepaper," 2020, https://github.com/OpiumProtocol/opium-contracts/blob/master/docs/opium_whitepaper.pdf.

[30] Cozy.Finance, "Cozy finance developer docs," 2020, https://docs.cozy.finance/.

[31] Etherisc, "Whitepaper," 2022, https://docs.etherisc.com/learn/whitepaper-en.

[32] Unslashed.Finance, "Insurance for decentralized finance," 2021, https://documentation.unslashed.finance/.

[33] Nsure.Network, "Open insurance platform for open finance," 2020, https://nsure.network/Nsure_WP_0.7.pdf.

[34] "Cover protocol," https://github.com/CoverProtocol.

[35] "Enzyme finance," https://docs.enzyme.finance/.

[36] R. Ben-Har, D. Patel, A. Su, and M. Resnick, "Risk harbor v1," https://github.com/Risk-Harbor/RiskHarbor-Whitepaper/blob/8b41ca8f89ce116db725f405aeacddc18b9bd6b2/Risk%20Harbor%20V1%20Whitepaper.pdf.