The Feasibility of a Smart Contract "Kill-Switch"

***Abstract*—The advent of blockchain technology and its adoption across various sectors have raised critical discussions about the need for regulatory mechanisms that can ensure consumer protection, maintain financial stability, and address privacy concerns without compromising the foundational principles of decentralization and immutability inherent in Distributed Ledger Technology (DLT) platforms. We examine the existing mechanisms for smart contract termination across several major blockchain platforms, including Ethereum, BNB Smart Chain, Cardano, Solana, Hyperledger Fabric, Corda, and IOTA. We assess the compatibility of these mechanisms with the requirements of the EU Data Act, focusing on aspects such as consumer protection, error correction, and regulatory compliance. Our analysis reveals a diverse landscape of approaches, from immutable smart contracts with built-in termination conditions to upgradable smart contracts that allow for post-deployment modifications. We discuss the challenges and opportunities associated with implementing kill-switches, such as the balance between enabling regulatory compliance and preserving the decentralized ethos, the technical feasibility of such mechanisms, and the implications for security and trust in the ecosystem.**

***Index Terms*—Smart Contract Design, Technology Regulation, Governance Models, Standards Development**

I. Introduction

Distributed Ledger Technologies (DLTs) offer unprecedented opportunities for innovation, efficiency, and trust. At the heart of this transformation are smart contracts — autonomous, self-executing agreements embedded in code, which have the potential to redefine interactions within various sectors, from finance and healthcare to supply chain management and beyond. However, integrating these technologies into the fabric of societal systems raises complex regulatory, ethical, and operational challenges. Among these is the necessity to reconcile the inherently decentralized and immutable nature of DLTs with the evolving landscape of global regulations to ensure consumer protection, privacy, and the stability of financial systems.

The European Union's Data Act, specifically Article 30 [1], spotlights the urgent conversation around the regulation of smart contracts, proposing the concept of a "kill-switch" mechanism to address these concerns. This regulatory proposition seeks to empower authorities and possibly participants within DLT ecosystems to intervene directly in the operation of smart contracts — a concept that, at first glance, seems at odds with the principles of decentralization and immutability that define blockchain technology. This paper examines the feasibility of implementing kill-switches in smart contracts and the implications for the DLT ecosystem. We explore various pathways for developing smart contract standards that can accommodate regulatory expectations without compromising the unique advantages of DLTs.

II. Background

The smart contract "kill-switch" concept has garnered significant attention in academic literature and industry discussions. This attention stems from the increasing realization of the potential risks and challenges associated with deploying immutable and autonomous smart contracts, especially in critical financial, legal, and social applications. As Chen et al. [2] noted, many smart contracts deployed on blockchains have security, availability, performance, maintainability, and reusability problems. There are as many as 23,327 vulnerable contracts on the Ethereum platform alone, putting millions of dollars worth of cryptocurrencies owned by unsuspecting users in jeoprady [3].

The literature on kill-switches is diverse, with calls for implementation from regulatory bodies to domain-specific applications. Among these contributions, Liu et al. focused on strengthening chaincode smart contracts implemented on Hyperledger Fabric to handle unexpected situations [4]. This innovative approach placed all mutable smart contract program states in a sandbox environment in chain code, which could be unlocked through a novel voting algorithm leveraging computational social choice mechanisms. A similar social choice mechanism has been utilized in the dApp account recovery proposal by Zhu et al. [5], where one could recover any "lost" tokens after a voting round that was empirically shown to be resilient in the face of any Sybil attacks and adversarial collusion. Another noteworthy contribution by Mohsin et al. [6] presents an innovative approach to enhancing smart contract resilience by incorporating ontologies as a guiding framework for action in the event of anomalies or errors in deployed contracts. By integrating a structured set of concepts and relationships pertinent to the operational domain of the smart contract, the ontology serves as a decision-support mechanism, enabling the smart contract to adapt to unforeseen circumstances without requiring external intervention.

III. Smart Contract Termination and Interruption

Article 30 of the Data Act [1] focuses specifically on requirements with respect to smart contracts used in a data spaces context. In essence, platform providers and individuals that deploy smart contracts for data sharing purposes need to ensure that the smart contract is robust against errors or malicious attacks; the smart contract is protected via rigorous access control mechanisms; the smart contract can be terminated or interrupted; and if terminated the smart contact data, logic, and code can be archived to facilitate auditing. In this paper, we focus specifically on controversial requirements in relation to termination and interruption and explore how said requirements could potentially be implemented.

A. Potential Applications

Beyond legislative interest, domain-specific applications may emerge around the utility of kill switches in various industries. In healthcare, for example, smart contracts managing sensitive patient data or automated drug delivery systems might incorporate kill switches to address privacy breaches or operational malfunctions. In supply chain management, kill switches could be used to halt operations in response to detected anomalies or breaches of contract. However, perhaps the most concrete examples of "kill-switch" implementation to date can be found in the domain of stablecoins and other financial instruments on blockchain platforms. Several projects have designed smart contracts with built-in mechanisms to freeze transactions, adjust operational parameters, or even terminate services in response to critical threats, such as market crashes or security breaches [7].

B. Effects on the Current Ecosystem

Integrating kill switches into smart contracts has broad implications for the current DLT ecosystem. These mechanisms, designed to offer intervention in unforeseen circumstances or malfunctions, bring about debates concerning decentralization, asset management, and security.

- **Concerns Regarding True Decentralization**: One of the foundational principles of blockchain technology is decentralization — the idea that any single entity does not control operations and governance. Introducing kill switches into smart contracts presents a paradox; while they can provide necessary safety nets for users, they also introduce a vector for centralized control. Critics argue that this undermines the very essence of decentralization [8]. However, it's essential to recognize that many DLTs already incorporate mechanisms for updates and upgrades (as noted in Section IV), some of which require centralized decision-making or a coordinated consensus among stakeholders.
- **Loss of Assets**: Activating a "kill-switch" could potentially lead to scenarios where users lose access to their assets temporarily or permanently. This risk is particularly acute in financial applications where smart contracts govern the custody and transfer of significant value. Therefore, any "kill-switch" implementation must include safeguards to prevent unintentional or unjustified wiping out of value. This could involve mechanisms for restoring operations and assets post-intervention, transparent and fair criteria for activation, and perhaps insurance mechanisms to cover losses in the worst-case scenarios.
- **Security Issues**: Implementing kill switches introduces specific security considerations, particularly regarding managing the keys or permissions required to activate or deactivate the switches. If not managed securely, these could become targets for malicious actors looking to disrupt operations or extract ransom. It's suggested that separate keys or permissions be used for the activation (pausing) and deactivation (unpausing) processes to minimize risks. Furthermore, these keys should be rotated or changed once used to prevent reuse attacks.

IV. Existing "Kill-Switch" Approaches

We outline approaches for smart contract termination that are already available in several prominent blockchains in Table I, and describe additional details below.

A. Ethereum

In Ethereum [9], smart contract termination and interruption are primarily handled through the built-in functionalities of the smart contracts themselves. Ethereum does not provide an external "kill-switch" or mechanism for forcibly terminating or interrupting smart contracts from outside the contract's code. Instead, the implementation of such features is left to the developers who write the smart contracts, typically managed through the following mechanisms:

- **Self-Destruct Function:** This function allows a contract to be "killed" or terminated, removing its code and storage from the blockchain [10]. When a contract is self-destructed, it sends the remaining Ether stored in it to a designated address. It removes the contract's code from the blockchain, making it inoperable. However, it's worth noting that the contract's code and past transactions are still part of the blockchain history and are immutable. The 'selfdestruct' function is typically used to remove no longer needed contracts or recover funds in an emergency. However, it must be explicitly included in the smart contract code. It can only be triggered by a function call within the contract, often restricted to the contract owner or other authorized entities. However, there is a recent proposal on removing this function [11].
- **Pause and Emergency Stop Patterns:** For interruption rather than complete termination, smart contracts can be designed with pause or emergency stop functionalities [12]. These patterns allow certain contract functions to be disabled temporarily without removing the contract from the blockchain. This can be useful when a bug is discovered, and the contract needs to be paused to prevent further damage while a fix is being developed. The pause pattern typically involves setting a boolean variable that controls the execution of sensitive functions. By changing this variable's state, the contract's critical operations can be enabled or disabled. The emergency stop pattern is more comprehensive, allowing for a phased approach to pausing and resuming contract functionalities, often with different levels of access control and conditions for triggering and reversing the pause state [13].
- **Upgradeable Contracts:** Another approach to managing smart contract behavior over time, including termination and interruption, is through upgradeable contracts [14]. This design pattern involves deploying a proxy contract that delegates calls to an implementation contract containing the logic. If the implementation needs to be changed, updated, or fixed, a new implementation contract can be deployed, and the proxy contract is updated to delegate

calls to the new contract. This approach allows bugs to be fixed and functionalities to be updated without terminating the contract. However, it may introduce complexity and potential security considerations.

B. BNB Smart Chain

Other popular public permissionless blockchains, such as BNB Smart Chain (BSC) [15], formerly known as Binance Smart Chain, is a blockchain platform that runs parallel with Binance Chain. It offers smart contract functionality and compatibility with Ethereum's existing infrastructure, such as the Ethereum Virtual Machine (EVM). This compatibility allows it to support Ethereum tools and DApps, making it a popular choice for developers looking to leverage the scalability and performance benefits of BSC while maintaining access to Ethereum's rich ecosystem. Handling smart contract termination and interruption in BNB Smart Chain is similar to Ethereum, primarily because of its EVM compatibility.

C. Cardano

Cardano [16] is a blockchain platform that employs a layered architecture. It separates the settlement layer, which handles transactions, from the computational layer, where smart contracts run. Cardano uses a unique proof-of-stake consensus algorithm called Ouroboros and supports smart contracts through its native programming language, Plutus. Plutus [17] is designed to enable the creation, execution, and management of smart contracts on the Cardano blockchain. Plutus contracts are written in Haskell, a functional programming language known for its high fault tolerance and security features. The use of Haskell influences how smart contracts, including their termination and interruption, are handled in Cardano.

- **Termination by Design:** In Cardano, the termination or interruption of a smart contract is primarily a matter of the contract's design. Because Plutus (and, by extension, Haskell) allows for creating highly deterministic and secure contracts, developers can incorporate specific conditions under which a contract may terminate or pause its operations. These conditions are encoded directly into the contract's logic and can be triggered by predefined events or states.
- **Stateful Smart Contracts:** Cardano's smart contracts can manage the state through the blockchain ledger, but how the state is handled is distinct from other platforms. Termination or modification of a contract could involve creating transactions that update or end the contract's state according to the logic defined in the contract itself, which ensures that the contract's behavior remains predictable and tamper-proof.
- **Off-chain Code:** Cardano also supports off-chain code execution through its application framework, which allows for complex interactions with on-chain smart contracts. Interruptions or terminations initiated by off-chain components can be designed to interact with the on-chain contracts, offering another layer of control for managing contract lifecycles. This off-chain logic can facilitate scenarios where user interaction or external data triggers the pause or stop conditions in the smart contract.
- **Governance and Updates:** Cardano's governance model can play a role in contracts that require the ability to evolve or might need to incorporate mechanisms for interruption or termination post-deployment. Through on-chain governance mechanisms, stakeholders can propose and vote on updates or changes to smart contracts, assuming the contract is designed to be upgradable, and the governance model supports such actions. This approach allows the community or stakeholders to have a say in the contract's lifecycle management.

D. Solana

Solana [18] is a high-performance blockchain platform designed to support scalable, decentralized applications and crypto-currencies. It uses a unique consensus mechanism called Proof of History (PoH), combined with the underlying Proof of Stake (PoS) consensus, to achieve high throughput and low latency. Unlike Ethereum and other blockchains, where smart contract termination and interruption mechanisms are more explicitly discussed and implemented, Solana's approach to smart contract management, including termination and interruption, is somewhat different due to its architecture and programming model. In Solana, smart contracts are referred to as "programs." These programs are written in Rust or C, compiled to Berkeley Packet Filter (BPF) bytecode, and deployed to the Solana blockchain. Once deployed, a program can be interacted with by sending transactions from Solana accounts. Once a program (smart contract) is deployed on the Solana blockchain, it is immutable — meaning it cannot be changed or deleted. This design choice emphasizes security and stability, ensuring that once a program is deployed, its logic cannot be altered, similar to the concept of immutability in Ethereum smart contracts. However, this also means there is no built-in "kill-switch" or termination mechanism for a Solana program once it is live on the network.

- **Upgradable Programs:** While individual programs are immutable, Solana introduces a mechanism for program upgradability through the use of a "Program Upgradeable Loader" [19]. This allows developers to deploy a new version of a program to replace the old one. The process involves deploying the new program version as a separate entity and then "switching" the program authority to point to the new program. This method does not terminate the old program but effectively redirects interactions to the new, upgraded program version.
- **State Management:** Termination or interruption of a program's operation in the traditional sense may not directly apply to Solana's model. However, programs can manage their state through accounts, which hold data. By modifying the state held in these accounts, a program can implement mechanisms to halt or modify its operations based on specific conditions, essentially allowing for a form of "interruption" of its functions.

E. Hyperledger Fabric

In Hyperledger Fabric [20], a permissioned blockchain platform designed for enterprise use, the handling of smart contract termination and interruption is notably different from how it is managed in public, permissionless blockchains like Ethereum. Hyperledger Fabric refers to smart contracts as "chain code." Hyperledger Fabric's approach to the smart contract (chain code) termination and interruption is characterized by its lifecycle management features, endorsement policies, and the control mechanisms provided by its permissioned network structure. Collectively, these features offer a structured and governed way to manage chaincode operations, including their update, interruption, and termination, in line with the needs and policies of the enterprise blockchain network.

- **Chaincode Lifecycle Management:** Hyperledger Fabric introduces sophisticated lifecycle management for chaincodes [21], allowing organizations to agree on chaincode parameters before deployment to the network. This lifecycle management process enables more granular control over the deployment, upgrade, and management of chain codes, including their termination and interruption. Rather than terminating a chain code, Hyperledger Fabric allows for upgrading the chain code to a new version. This process involves deploying a new version of the chain code on the network and performing an upgrade transaction. The upgrade can introduce new logic, fix issues, or modify the chain code's behavior. This process is controlled and requires a consensus from the participating organizations, ensuring that changes are agreed upon before implementation.
- **Chaincode Endorsement Policies:** Hyperledger Fabric employs endorsement policies [22] that define the rules under which a transaction (including those that might terminate or interrupt chaincode operations) is considered valid. These policies can require that transactions be endorsed by a specific number of peers from certain organizations within the network, offering a high level of control and security over chaincode execution, including any operations that could stop or alter the chaincode's function.
- **Private Data Collections:** Hyperledger Fabric supports private data collections [23], which allow a subset of the network to transact privately, maintaining confidentiality. If a chaincode associated with a private data collection is updated or removed, the data remains governed by the policies of the private data collection, ensuring that sensitive information is handled according to the network's requirements, even if the chaincode's operation is interrupted or terminated.
- **Administrative Operations:** Due to the permissioned nature of Hyperledger Fabric, network administrators have more control over the chain code, including their deployment, operation, and termination. This means that, if necessary, chain codes can be administratively stopped or removed by parties with the appropriate permissions, according to the governance model of the specific Hyperledger Fabric network.

F. Corda

In Corda [24], a platform designed for business applications with a focus on facilitating direct transactions with privacy and finality, smart contract termination and interruption are handled differently compared to platforms like Ethereum or Hyperledger Fabric. Corda's architecture and operational model offer unique mechanisms for managing the lifecycle of smart contracts, known as "Corda Contracts," within the platform. Corda's design emphasizes privacy and finality in transactions, influencing how contract termination and interruptions are perceived and managed. Transactions in Corda are only shared with parties directly involved, or who need to validate the transaction, and once a transaction is finalized, it is considered immutable and authoritative, aligning with business needs for certainty and finality in agreements.

Corda handles smart contract termination and interruption through its contract upgradeability features, contract constraints governing state evolution, and explicitly modeling termination logic within contract code. Its architecture supports the management of the contract lifecycle in a way that aligns with the platform's focus on direct, private, and final transactions among business entities.

- **Upgradability:** Corda provides a built-in mechanism for contract upgradability. This allows the network participants to evolve their contracts over time as business needs change or in response to discovering issues with the original contract. Upgrading a contract in Corda involves transitioning the states governed by an old version of the contract to a new version under the agreement of all relevant parties.
- **Contract Constraints:** Corda uses a concept called "contract constraints" to govern which contract codes can constrain the evolution of ledger states. These constraints ensure that once states are created under a specific contract, future transactions that consume and evolve these states are validated by the same contract code or an agreed-upon upgraded version, providing a form of governance over contract changes.
- **Explicit Termination and State Evolution:** While Corda does not have a specific "kill-switch" for contracts, contracts can be designed to include termination logic or conditions within their clauses. Since contracts in Corda govern the transition of states, a contract can explicitly define conditions under which a state is considered final or can no longer be evolved, effectively terminating the contract's applicability to that state. Additionally, business processes can be modeled to include explicit termination transactions that move states to a final, consumed status, where they cannot be used in future transactions.
- **Flow Framework:** Corda's Flow Framework [25], which facilitates the automation of transactions between nodes, can be used to manage the execution of contract termination or state evolution logic. Through flows, partici-

pants can coordinate complex processes, including those involving contract or state termination, under the rules defined by their Corda contracts.
- **Administrative Intervention:** In a permissioned network like Corda, network operators have administrative control over the network, including the ability to intervene in the operation of contracts and nodes in accordance with the network's governance policies. This includes managing membership and potentially coordinating contract upgrades or the resolution of disputes related to contract execution.

G. IOTA

IOTA [26] is a DLT designed primarily for the Internet of Things (IoT) environment, focusing on scalability, speed, and the elimination of transaction fees. Unlike traditional blockchain-based platforms like Ethereum or permissioned networks like Hyperledger Fabric, IOTA utilizes a unique data structure called the Tangle [27], which is a form of Directed Acyclic Graph (DAG). The Tangle allows for different mechanisms for smart contract execution, termination, and interruption compared to blockchain-based systems. Similar to other blockchain platforms, IOTA introduced smart contracts as part of its ecosystem to provide more complex and conditional transaction capabilities. These smart contracts allow developers to execute code on the IOTA network, enabling decentralized applications (dApps) and business logic implementation directly on IOTA.

The project has undergone significant updates and expansions to its technology stack, aiming to address various challenges and expand its use cases beyond the IoT. This includes enhancements to smart contract functionalities, interoperability features, and scalability solutions, which may influence how smart contract termination and interruption are handled in future iterations.

Given the unique nature of IOTA's Tangle, the approach to smart contract termination and interruption differs from other blockchain technologies in the following ways:

- **Asynchronous and Decentralized Execution:** The Tangle's structure facilitates asynchronous transaction confirmations, which impacts how smart contracts are executed and managed. Smart contracts in IOTA can operate independently of the main Tangle, allowing for greater flexibility and scalability. Termination or interruption mechanisms would, therefore, be designed within the smart contract's logic itself rather than relying on the underlying ledger's functionality.
- **Stateless Smart Contracts:** If IOTA's smart contracts are stateless (meaning they do not store their state on the ledger), termination might involve simply ceasing to call or interact with the contract. However, if a state is managed externally or through a layer built on top of IOTA, termination procedures would need to consider how to handle and preserve this state.
- **Contract Updatability:** Similar to other platforms, smart contracts on IOTA may include features for updatability or versioning, allowing them to be modified or terminated based on predefined conditions or through consensus among stakeholders. This approach would require explicit design within the smart contract framework and careful consideration of security and integrity.
- **External Triggers:** Smart contracts in IOTA can be designed to respond to external inputs or triggers, which could include termination signals. Before ceasing operation, these signals would initiate predefined termination logic within the contract, such as redistributing assets or finalizing states.
- **Permissioned Interventions:** In scenarios where IOTA is used within a permissioned environment or consortium, network administrators might have the authority to enforce policies regarding the lifecycle of smart contracts, including their termination or suspension, according to the governance model adopted by the participants.

V. Conclusion

This paper explored implementing a "kill-switch" mechanism in smart contracts within the framework of DLTs, specifically focusing on its feasibility and implications in light of the European Union's Data Act legislation [1]. The paper contributes to the ongoing debate on the regulation of blockchain technology, providing insights into how current DLT platforms can adapt to meet legislative requirements without stifling innovation and be accessible, understandable, and controllable by non-technical users. Adopting kill switches in smart contracts within the DLT ecosystem demands careful consideration of their impacts on decentralization, asset security, and the broader trust in blockchain technologies. By addressing these concerns thoughtfully, it's possible to design systems that retain the benefits of decentralization while providing mechanisms to protect users and the integrity of the network. This involves a delicate balance between control and freedom, requiring ongoing dialogue and innovation within the community to navigate these complex issues effectively.

Future studies could explore the design, implementation, and effectiveness of decentralized governance models specifically tailored for managing "kill-switch" mechanisms in smart contracts. Investigating automated mechanisms within smart contracts that dynamically adjust to changing regulatory requirements without manual intervention could be a significant area of future work, particularly for already deployed smart contracts. It may be necessary to include protocol updates through hard forks or the governance models of various blockchain projects that allow for changes to be made to operational parameters. A deeper analysis of how "kill-switch" mechanisms affect the security, trust, and overall perception of blockchain networks among users and stakeholders should also be carried out. This could involve empirical studies on user trust before and after the implementation of kill switches and security vulnerability assessments related to their deployment. Finally, with the increasing diversity of blockchain platforms, future work could focus on developing cross-chain solutions

Blockchain	Kill-Switch Approach	Additional Considerations	EU Data Act Support
Ethereum	Self-destruct function; Pause and emergency stop patterns	Contracts can include mechanisms for termination or temporary interruption.	Possible with custom implementations
BNB Smart Chain	Similar to Ethereum: Self-destruct function; Upgradeable contracts	Inherits Ethereum's EVM compatibility, allowing for similar mechanisms.	Possible with custom implementations
Cardano	Design-specific conditions within smart contracts	Uses Plutus and Haskell for deterministic and secure contract development with built-in conditions for termination.	Potentially, with design-specific conditions
Solana	Upgradable programs through Program Upgradeable Loader	No direct kill-switch; emphasizes program immutability with upgradability for version control.	Potentially, through upgradable programs
Hyperledger Fabric	Chaincode lifecycle management; Administrative control	Permissioned network allows for more centralized control over chaincode operations.	Yes, through administrative control
Corda	Contract upgrade and explicit termination conditions	Supports explicit termination and upgrading of contracts through network consensus.	Yes, through explicit contract conditions
IOTA	Decentralized control; State management in smart contracts	Unique Tangle architecture affects implementation; focuses on state management for control.	Potentially, with decentralized control mechanisms

TABLE I
COMPARISON OF "KILL-SWITCH" APPROACHES IN VARIOUS BLOCKCHAIN IMPLEMENTATIONS

and interoperability standards that facilitate regulatory compliance across different DLTs. Future research is likely to delve deeper into these discussions, proposing frameworks, models, and real-world trials that balance the autonomy of smart contracts with the safety, security, and compliance requirements of the broader ecosystem.

In summary, the discourse around smart contract kill switches is multifaceted, reflecting a cross-section of academic, legislative, and industry perspectives. The challenge lies in designing "kill-switch" mechanisms that align with the ethos of decentralization as much as possible, perhaps through decentralized governance models or community consensus mechanisms.

REFERENCES

[1] European Parliament and Council of the European Union, "Regulation (EU) 2023/2854 of the European Parliament and of the Council of 5 October 2023 on harmonised rules on fair access to and use of data (Data Act)." https://eur-lex.europa.eu/eli/reg/2023/2854/oj, 2023. Accessed: Mar 04, 2024.

[2] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defining smart contract defects on ethereum," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 327–345, 2020.

[3] D. Perez and B. Livshits, "Smart contract vulnerabilities: Vulnerable does not imply exploited," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1325–1341, 2021.

[4] S. Liu, F. Mohsin, L. Xia, and O. Seneviratne, "Strengthening Smart Contracts To Handle Unexpected Situations," in *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, pp. 182–187, IEEE, 2019.

[5] Y. Zhu, L. Xia, and O. Seneviratne, "A Proposal for Account Recovery in Decentralized Applications," in *2019 IEEE International Conference on Blockchain (Blockchain)*, pp. 148–155, IEEE, 2019.

[6] F. Mohsin, X. Zhao, Z. Hong, G. de Mel, L. Xia, and O. Seneviratne, "Ontology aided smart contract execution for unexpected situations," in *BlockSW/CKG@ ISWC*, 2019.

[7] D. Li, D. Han, T.-H. Weng, Z. Zheng, H. Li, and K.-C. Li, "On stablecoin: Ecosystem, architecture, mechanism and applicability as payment method," *Computer Standards & Interfaces*, vol. 87, p. 103747, 2024.

[8] F. Rodrigues, "Blockchain devs expect complications from EU smart contract kill switch," Nov 2023. Accessed: Mar 4, 2024.

[9] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, vol. 1, pp. 22–23, 2013.

[10] J. Chen, X. Xia, D. Lo, and J. Grundy, "Why do smart contracts self-destruct? investigating the selfdestruct function on ethereum," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–37, 2021.

[11] V. Buterin, "A note on selfdestruct." https://hackmd.io/@vbuterin/selfdestruct, 2024.

[12] "Pausing smart contracts." https://ethereum-blockchain-developer.com/022-pausing-destroying-smart-contracts/03-pausing-smart-contracts/, 2024.

[13] Fravoll, "Emergency stop." https://fravoll.github.io/solidity-patterns/emergency_stop.html, 2021.

[14] M. Salehi, J. Clark, and M. Mannan, "Not so immutable: Upgradeability of smart contracts on ethereum," *arXiv preprint arXiv:2206.00716*, 2022.

[15] Binance, "Bnb chain whitepaper." https://github.com/bnb-chain/whitepaper/blob/master/WHITEPAPER.md, 2022.

[16] C. Hoskinson, "Why we are building Cardano? A subjective approach." https://whitepaper.io/document/581/cardano-whitepaper, 2017.

[17] Cardano, "Plutus." https://developers.cardano.org/docs/smart-contracts/plutus, 2023.

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

[19] Solana, "Module solana program bpf loader upgradeable." https://docs.rs/solana-program/latest/solana_program/bpf_loader_upgradeable/index.html, 2023.

[20] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, pp. 1–15, 2018.

[21] H. Fabric, "Fabric chaincode lifecycle." https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode_lifecycle.html, 2024.

[22] H. Fabric, "Fabric Endorsement Policies." https://hyperledger-fabric.readthedocs.io/en/release-2.2/endorsement-policies.html, 2024.

[23] H. Fabric, "Fabric Private Data Collections." https://hyperledger-fabric.readthedocs.io/en/latest/private-data/private-data.html, 2024.

[24] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: an introduction," *R3 CEV, August*, vol. 1, no. 15, p. 14, 2016.

[25] Corda, "Flows." https://docs.r3.com/en/platform/corda/4.9/enterprise/cordapps/api-flows.html, 2024.

[26] O. Saa, A. Cullen, and L. Vigneri, "Iota 2.0 incentives and tokenomics whitepaper," 2023.

[27] S. Popov, "The tangle," *White paper*, vol. 1, no. 3, p. 30, 2018.