Advances in Blockchain and Cryptocurrency

1

Sergey Y. Yurish, Editor

Advances in Blockchain and Cryptocurrency

Book Series, Volume 1

S.Yurish

Editor

Advances in Blockchain and Cryptocurrency

Book Series, Volume 1

IFSA International Frequency Sensor Association Publishing

S. Yurish, *Editor*

Advances in Blockchain and Cryptocurrency, Book Series, Vol. 1

Published by IFSA Publishing, S. L., 2026
E-mail (for print book orders and customer service enquires):
ifsa.books@sensorsportal.com

Visit our Home Page on https://sensorsportal.com/ifsa_publishing.html

ISBN: 978-84-09-90030-5
BN-20260805-XX
BIC: UNK; Thema: UNKD

Contents

Preface **9**
Contributors **11**

1. Smart-contracts Forensics: Unravelling Mysteries Through On-chain Logic Analysis **13**
1.1. Introduction 13
1.1.1. The Rise of Trustless Exploitation *13*
1.1.2. Categories of Smart Contract-based Financial Crime *14*
1.1.3. Real-world Case References in Smart Contract Financial Crime *21*
1.2. Investigative Setup 22
1.2.1. Using Full Ethereum Nodes vs. RPC Endpoints *23*
1.2.2. Snapshotting and Replaying Contract Interactions *25*
1.2.3. Parsing the EVM State *25*
1.3. Static and Dynamic Analysis Techniques 27
1.3.1. Static Analysis *28*
1.3.2. Dynamic Analysis *28*
1.4. Signature-based and Heuristic Detection 29
1.4.1. Identifying Common Scam Contract Patterns *30*
1.4.2. Automated Signature Libraries *32*
1.4.3. On-chain Honeypot and Obfuscation Techniques *32*
1.5. Cross-contract Interaction Tracing 34
1.5.1. Proxies and Upgradable Contracts (ERC-1967, EIP-1822) *34*
1.5.2. Multisig Wallets and DAO Interactions *35*
1.5.3. Bridge and Liquidity Pool Exploits *36*
1.6. Forensics of DeFi Protocol Exploits 37
1.6.1. Flash Loan Lifecycle Analysis *37*
1.6.2. Oracle Manipulation and Attack Surface Mapping *39*
1.6.3. LP Token Theft and Draining Behavior *40*
1.7. Tools and Automation 40
1.7.1. Forensic Tools Overview *41*
1.7.2. Custom Scripting with Web3.py, Brownie, and Hardhat *42*
1.7.3. Automated Pipelines for Flagging Malicious Behavior *43*
1.8. Compliance and Attribution 44
1.8.1. Linking Attacker Contracts to Wallets and Exchanges *45*
1.8.2. Tagging Known Attacker Addresses *45*
1.8.3. Cross-referencing with Off-chain Data *46*
1.9. Open Research Questions 48
1.9.1. Formal Verification for Post-Exploit Analysis *48*
1.9.2. Detecting Malicious Logic in zkApps and Account Abstraction (ERC-4337) *49*
1.9.3. Forensics of Layer 2 Contracts *50*

1.10. Conclusions 51
Acknowledgements 52
References 52

2. Asymmetric Risk Allocations in Cryptocurrency Portfolios: A VaR/ES-GARCH-Copula Approach 55
2.1. Introduction 55
2.2. Methodology 60
2.2.1. Var/ ES Term 60
2.2.2. Conditional Mean and Conditional Volatility 61
2.2.3. Copula Function and Multivariate Modelling 64
2.2.4. Marginal Risk Allocation: Euler and Kernel Methods 65
2.3. Data and Descriptive Statistics 70
2.4. Empirical Results, Interpretation, and Discussion 72
2.4.1. Univariate GARCH Models for Volatility 72
2.4.2. Marginal Distributions 74
2.4.3. Copula Function 76
2.4.4. Term Structure 78
2.4.5. Marginal Risk Contributions 79
2.4.6. Locally – Constant Kernel at Various Tail Risks 81
2.4.7. Locally – Linear Kernel at Various Tail Risks 87
2.5. Critical Discussion and Outlook 94
2.6. Conclusion 95
References 95

3. Anonymization with Controlled De-Anonymization in Private Blockchain 99
3.1. Introduction 99
3.2. Digital Signatures and Proof of Knowledge 103
3.2.1. Schnorr Digital Signature Scheme 107
3.2.2. Elliptic Curve Digital Signature Algorithm (ECDSA) 109
3.2.3. Non-Interactive Zero Knowledge Proof – NIZKP 113
3.3. Anonymization-De-Anonymization Based On Ring Signatures (RS) 115
3.3.1. Signature Creation in Monero 117
3.3.2. Signature Verification in Monero 118
3.3.3. Correctness 118
3.3.4. Anonymization 118
3.3.5. Ring Signature 119
3.3.6. Ring Signature Verification 119
3.3.7. Correctness 120
3.3.8. De-Anonymization 120
3.3.9. Efficiency 120
3.4. Anonymization-De-Anonymization Based on the Schnorr Signature Approach 121
3.4.1. Anonymization Based on Schnorr Signature 121
3.4.2. De-Anonymization Based on the Schnorr Signature 123

3.5. Anonymization-De-Anonymization Based Elliptic Curve Schnorr Signature Algorithm (ECSSA) 126
3.5.1. Anonymization Based on ECSSA *127*
3.5.2. De-Anonymization Based on ECSSS *129*
3.6. Example 130
3.7. Security Considerations 131
3.8. Conclusions 133
References 135

Preface

Blockchain and cryptocurrencies have developed from experimental technologies into a broad field of research and practice. They now bring together distributed systems, cryptography, programmable finance, data analysis, cybersecurity, regulation, and institutional design. Their expansion has created remarkable opportunities for trusted digital coordination without conventional intermediaries; it has also exposed difficult questions about security, privacy, accountability, market risk, and the evidential value of on-chain activity.

The open access book series *Advances in Blockchain and Cryptocurrency* is intended to provide a focused forum for such questions. Its purpose is not merely to describe new protocols or applications, but to examine how blockchain-based systems behave under operational, economic, and adversarial conditions. Volume 1 reflects this interdisciplinary perspective through three chapters that move from investigation of malicious smart-contract behavior, to quantitative measurement of cryptocurrency portfolio risk, and finally to cryptographic mechanisms that reconcile anonymity with controlled disclosure.

Chapter 1 develops a practical and technical account of smart-contract forensics. It classifies financial crime conducted through programmable contracts, including rug pulls, pump-and-dump tokens, honeypots, flash-loan exploits, oracle manipulation, and liquidity-pool attacks. The chapter then explains the investigative foundations required to reconstruct these events: access to full nodes or RPC services, historical state replay, EVM storage and call analysis, and complementary static and dynamic techniques. Particular attention is given to heuristic signatures, cross-contract tracing, proxy and upgrade patterns, bridges, multisignature wallets, automated forensic pipelines, and the relationship between on-chain evidence and off-chain attribution. The discussion concludes by identifying research challenges created by formal verification, zero-knowledge applications, account abstraction, and Layer 2 systems. In this way, the chapter treats forensics not only as post-incident diagnosis, but also as a means of strengthening accountability in programmable finance.

Chapter 2 addresses the economic dimension of blockchain-based assets. Using six widely traded cryptocurrencies, it examines tail risk across investment horizons and dependence among assets. The framework combines Value at Risk and Expected Shortfall with ARMA-realized-GARCH modelling, copulas, Monte Carlo simulation, and marginal risk allocation. By separating risk within an asset over time from risk arising through portfolio co-movement, the analysis shows why conventional volatility and correlation measures can obscure asymmetric contributions to extreme losses. It thus offers a more informative basis for portfolio construction in heavy-tailed, volatility-clustered markets.

Chapter 3 turns to privacy and proof of ownership. After reviewing signatures, proofs of knowledge, and Monero mechanisms, it presents user-controlled anonymization and voluntary de-anonymization for a UTxO-based private blockchain. Schnorr signatures, aggregated multisignatures, and non-interactive zero-knowledge proofs enable a user to prove control of selected addresses and funds without exposing unrelated history. The chapter compares four forms of anonymization, gives security arguments, and evaluates efficiency. Its principal contribution is selective disclosure without a mandatory tracing authority, while retaining legitimate ownership verification under recognized cryptographic assumptions.

Together, the chapters show that decentralization changes rather than removes risk and responsibility. Execution traces expose misuse; tail-sensitive models reveal risks hidden by averages; and cryptographic proofs let privacy and verifiability coexist. These methods invite dialogue across research, regulation, security, finance, and industry.

We hope this first volume will assist specialists and readers entering adjacent fields. I thank the authors for their contributions and revisions, and the reviewers and production team for the expertise that brought the book to publication. Their work has produced a volume that is analytical, practical, and attentive to technological change.

Sergey Y. Yurish

Editor
IFSA Publishing Barcelona, Spain

Contributors

Mamdouh Abdulaziz Saleh Al-Faryan

University of Portsmouth, Portsmouth, United Kingdom
E-mail: al-faryan@hotmail.com

Antanas Bendoraitis

Kaunas University of Technology, Scientific Group
of 'Cryptography and blockchain systems'
Studentu str. 50, Kaunas, Lithuania

Lina Dindienė

Kaunas University of Technology, Scientific Group
of 'Cryptography and blockchain systems'
Studentu str. 50, Kaunas, Lithuania

Vahidin Jeleskovic

Humboldt-Universität zu Berlin, Berlin, Germany
E-mail: vahidin.jeleskovic@hu-berlin.de

Kęstutis Lukšys

Kaunas University of Technology, Scientific Group
of 'Cryptography and blockchain systems'
Studentu str. 50, Kaunas, Lithuania

Mirko Meloni

Independent researcher, Italy
E-mail: mirkomeloni1@gmail.com

Aleksejus Michalkovič

Kaunas University of Technology, Scientific Group of
'Cryptography and blockchain systems'
Studentu str. 50, Kaunas, Lithuania

Otilia Maria Muntean

Faculty of Mathematics and Informatics,
Computer Science Department
West University of Timisoara, Romania
E-mail: otilia.muntean97@e-uvt.ro

Ciprian Pungilă

Faculty of Mathematics and Informatics,
Computer Science Department
West University of Timisoara, Romania
E-mail: ciprian.pungila@e-uvt.ro

Eligijus Sakalauskas

Kaunas University of Technology, Scientific Group
of 'Cryptography and blockchain systems'
Studentu str. 50, Kaunas, Lithuania
Tel.: + 370 698 78477
E-mail: eligijus.sakalauskas@ktu.lt

Zahid Irshad Younas

Berlin School of Business and Innovation, Berlin, Germany
E-mail: zahid1132_gcu@yahoo.com

Chapter 1

Smart-contracts Forensics: Unravelling Mysteries Through On-chain Logic Analysis

Otilia Maria Muntean and Ciprian Pungilă

1.1. Introduction

A smart contract is an autonomous program deployed on a blockchain that executes predefined instructions once specified conditions are satisfied. These contracts are intended to enforce agreements without reliance on traditional intermediaries such as banks, legal institutions, or centralized authorities. By virtue of their deployment on a blockchain, smart contracts exhibit three fundamental properties: immutability (they cannot be altered post-deployment), transparency (their code is publicly accessible for inspection), and trustlessness (they eliminate the need for trust in counterparties by ensuring that execution is governed solely by code).

1.1.1. The Rise of Trustless Exploitation

Smart contracts represent one of the most transformative innovations in blockchain technology. Designed to automate transactions and enforce conditions without intermediaries, they enable decentralized applications (dApps) to operate in a trustless environment. In decentralized finance (DeFi), smart contracts handle everything from token swaps and lending to yield farming and asset custody, often holding millions, or even billions of dollars in total value locked (TVL). Their deterministic logic,

Otilia Maria Muntean
Faculty of Mathematics and Informatics, Computer Science Department,
West University of Timisoara, Romania

transparency, and composability have been instrumental in accelerating the growth of decentralized ecosystems.

However, these qualities also introduce significant vulnerabilities. Once deployed, a smart contract's logic is both immutable and self-executing, which means that without thorough auditing and oversight, it can be exploited by malicious actors. Unlike traditional software platforms, which can patch bugs and reverse fraudulent transactions through centralized authorities, smart contracts operate without a safety net. Exploits, whether due to overlooked logic flaws or deliberate backdoors, often lead to irreversible losses. This new paradigm has given rise to what can be called trustless exploitation: a class of financial crime where the attack vector is embedded within the protocol's own logic. Attackers no longer need to break through firewalls or steal data. They just call the contract's functions as designed, but in unexpected combinations or sequences. This has shifted the battleground of financial crime from off-chain deception to on-chain manipulation, where tracing logic and execution paths becomes critical to understanding and attributing malicious behavior.

For forensic investigators, the implications are profound. Instead of relying solely on transactional metadata or exchange records, smart contract forensics requires dissecting contract bytecode, interpreting storage layouts, and reconstructing the execution flow of transactions. A clear understanding of how trustless systems can be manipulated without violating protocol rules serves as the foundation for the investigative techniques presented throughout this chapter.

1.1.2. Categories of Smart Contract-based Financial Crime

As DeFi evolves, malicious actors increasingly exploit smart contracts for fraud. Instead of targeting the blockchain itself, they abuse flaws in contract logic. Recognizing common exploit patterns is essential for forensic analysis, as it links behaviors to technical signatures.

Smart contract-related offenses generally fall into several main categories. The first involves deceptive contracts, such as Ponzi schemes or rug pulls, which may restrict token transfers after purchase. The second category includes logic-based exploits, such as reentrancy attacks or flaws in access control, which enable malicious actors to alter the intended behavior of a contract. The third involves money laundering, where individuals leverage decentralized exchanges (DEXs), mixers, or

custom contracts to conceal the source and movement of illicit funds within decentralized ecosystems.

1.1.2.1. Rug Pulls

A rug pull is a scam in the crypto and DeFi space where project developers suddenly withdraw all funds (usually from a liquidity pool) and disappear, leaving investors with worthless tokens. The name refers to "pulling the rug out" from unsuspecting users. These scams often involve flashy token launches with promises of high returns, fueled by social media hype. Once enough funds are collected and token prices rise, the developers drain the liquidity, crashing the token's value.

Rug pulls can be hard (malicious code blocks users or drains funds) or soft (developers sell their own tokens and abandon the project). In the absence of regulation and with irreversible transactions, victims of DeFi scams rarely recover their losses. This highlights the importance of thorough due diligence and smart contract audits.

Forensic Signatures and On-Chain Indicators of Rug Pulls

Rug pulls, though often carefully orchestrated, leave behind on-chain evidence that can be analyzed using blockchain forensic tools. Since all transactions and contract interactions are permanently recorded on public blockchains, investigators can trace fund flows and identify behavioral patterns linked to fraud. These on-chain signatures are crucial for detecting and analyzing rug pull schemes. A key indicator of a rug pull is the sudden withdrawal of liquidity from a DEX pool, usually by the project creator or a privileged wallet, causing the token's value to crash and leaving investors with worthless assets. The extracted funds are typically converted to stable tokens or routed through mixers and bridges to obscure the trail. The smart contract itself often holds forensic clues. Rug pull contracts may include hidden functions or backdoors enabling creators to freeze assets, block sales, or impose high fees. These are often concealed in complex or misleading code to evade detection. Repeat patterns across scams like reused wallets, contract templates, or malicious infrastructure can be clustered to identify coordinated schemes. Coupled with off-chain data (e.g., social media or domain names), these linkages support attribution efforts. Certain technical patterns signal malicious intent, such as owner-only *removeLiquidity()* or *emergencyWithdraw()* functions that enable fund extraction.

Real-world cases across DeFi, tokens, and NFTs illustrate how rug pulls are carried out through deception, technical manipulation and abuse of privileges. These examples reveal vulnerabilities that forensic analysis can help uncover while offering lessons for both investigators and users.

Squid Game Token (SQUID) [1]: One of the most widely publicized rug pulls, the Squid Game Token capitalized on the popularity of the Netflix show. Promoted as a "play-to-earn" game token, it quickly gained traction and skyrocketed in price. However, the smart contract included a mechanism that prevented investors from selling the token, effectively trapping liquidity. Once a large number of users had bought in, the developers drained the liquidity pool and vanished, resulting in a loss of over $3 million in investor funds.

The company presented a textbook example of a deceptive token-based rug pull. The smart contract contained restrictive logic that prevented users from selling their tokens, effectively locking them into the system. This functionality was not clearly disclosed in any public documentation or user interface, making it difficult for investors to detect the trap before purchasing. Forensic analysis of the *transferFrom()* logic revealed that it was selectively disabled for users, while remaining active for the contract owner.

Meerkat Finance [2]: Launched on Binance Smart Chain, it was presented as a DeFi yield farming platform. Within 24 hours of launch, developers used the proxy contract pattern to update the logic and insert a function that enabled them to withdraw over $30 million. Initially reported as a hack, blockchain analysis later pointed to an insider rug pull. The case revealed how proxy-based upgradability without proper governance can be exploited for fraud.

In this case, a more sophisticated rug pull was executed using a proxy contract pattern. The developers retained administrative control over the proxy and, shortly after launch, used the *upgradeTo()* function to silently swap out the original contract logic with a malicious implementation. This new version introduced a hidden withdrawal function that allowed them to extract over $30 million in user funds. On-chain forensic cues included *AdminChanged* and *OwnershipTransferred* events occurring just prior to the exploit. Furthermore, a low-level analysis method confirmed changes in the EIP-1967 storage slot, indicating that the implementation address had been updated: an advanced but detectable forensic signature of contract manipulation.

1.1.2.2. Pump-and-dump Tokens

Pump and dump tokens are a type of market manipulation scheme in the cryptocurrency space, where the value of a token is artificially inflated or "pumped" through coordinated hype or misleading information, only for the orchestrators to sell off ("dump") their holdings once the price peaks. This sudden sell-off causes the token's value to crash, leaving unsuspecting investors with significant losses. These schemes often rely on creating a sense of urgency encouraging retail traders to buy in rapidly while the price is climbing.

In the context of crypto, pump and dump schemes are commonly executed with newly launched or low-liquidity tokens that can be easily manipulated. Promoters may use social media, private Telegram groups, or even influencers to drive hype, sometimes falsely claiming that the token is associated with major partnerships or technological breakthroughs. Once enough investors buy in and the price spikes, the original holders sell off their large share of tokens, triggering a rapid price collapse.

While pump-and-dump schemes have their roots in traditional penny stock markets, their decentralized and unregulated nature in the crypto space makes them significantly more difficult to monitor and control. Unlike rug pulls, which involve malicious contract behavior, pump and dumps rely more on off-chain manipulation and market psychology, but both result in similar investor harm. Blockchain forensics can still help identify such schemes by analyzing wallet clusters, trade timing patterns, and sudden shifts in token holder distributions.

Forensic Signatures and On-Chain Indicators of Pump and Dump Tokens

Pump and dump schemes often rely on off-chain hype and social engineering but leave behind identifiable on-chain indicators that can be uncovered through forensic analysis. A typical sign is a concentrated token distribution, where a small number of wallets, usually controlled by insiders, hold the majority of the supply. These wallets remain inactive before the price surge and then sell aggressively after new buyers inflate the price, leading to a sharp decline in value.

Other indicators include clusters of buy and sell transactions occurring in quick succession, synchronized dumping patterns, and connections

between deployer wallets, promoters, and recipients through shared funding sources or repeated interactions. Many such tokens also lack genuine functionality, using basic or cloned contracts with hidden conditions that give insiders trading advantages.

By correlating these on-chain patterns with off-chain promotional efforts, forensic analysts can reconstruct the scheme, identify its orchestrators, and flag related addresses for further investigation.

UfoToken (UFO) [3] is often cited as an example of a suspected pump and dump scheme. The token gained rapid popularity following aggressive promotion on Telegram and social media, followed by a sharp crash in value. Observers noted that the token had a highly concentrated supply and lacked fundamental utility, such as a functioning website or whitepaper, raising concerns about its legitimacy.

KingDog Token (KINGDOG) was heavily promoted via Telegram and YouTube, gained traction within hours of launch, and subsequently experienced a liquidity drain shortly after peak trading volume. The contract was unverified on BscScan upon deployment, preventing auditors from reviewing its code. It only implemented standard BEP-20 functions, no staking, farming, or governance features, suggesting a lack of utility. After an initial price surge, the deployer had withdrawn most of the liquidity, causing a rapid collapse. Additionally, token scanning services flagged the contract as 'unverified,' a common red flag for cloned or repetitive scam token operations.

1.1.2.3. Fake Token Factories and Clone Contracts

Fake token factories and clone contracts are commonly used techniques in smart contract-based scams, particularly in pump and dump schemes and rug pulls. A *fake token factory* is a smart contract designed to rapidly generate multiple new tokens with identical or near-identical parameters, such as name, symbol, supply, and basic functionality. These factories allow scammers to deploy tokens at scale, often changing only superficial details to make each one appear unique. Because these tokens typically lack meaningful utility or audited code, they serve primarily as vehicles for fraud, enabling the repeated execution of scams under different names and branding.

Clone contracts, on the other hand, refer to smart contracts that are direct copies (often byte-for-byte) of existing contracts, sometimes with minor

modifications such as new owner addresses. Scammers use clones to mimic the appearance of legitimate projects, hoping to deceive users into thinking the clone is affiliated with a trusted protocol. In many cases, clones are used to impersonate popular DeFi platforms or tokens to attract unsuspecting investors. From a forensic standpoint, both fake token factories and clones are identifiable through code similarity analysis, repeated deployer addresses, and consistent patterns in contract creation, making them important indicators in detecting large-scale fraud networks.

Forensic Signatures and On-Chain Indicators of Fake Token Factories and Clone Contracts

From a forensic perspective, fake token factories and clone contracts exhibit distinctive on-chain indicators that can be used to detect and link fraudulent activity. One of the clearest signs is the repeated deployment of multiple tokens from the same factory address or deployer wallet. These tokens often share identical bytecode or exhibit high code similarity, with only minor changes such as token names, symbols, or supply values. Forensic tools can identify these patterns using code fingerprinting techniques and track how these tokens are distributed, traded, and ultimately dumped.

Additional indicators include the use of minimal or unaudited ERC-20 templates, the absence of utility functions, and repeated interaction with the same liquidity pools or router contracts. In many cases, tokens generated by these factories are distributed to multiple wallets controlled by the attacker, which then coordinate trading activity to simulate organic interest. When combined with transaction clustering, timing analysis, and reuse of infrastructure (such as proxy contracts or marketing domains), these forensic cues can help uncover broader networks of automated fraud built on fake token generation.

Binance Smart Chain (BSC) [4]: One illustrative example of a fake token factory scheme was uncovered on the Binance Smart Chain (BSC) in 2021, where a single deployer wallet launched dozens of nearly identical meme tokens within a short timeframe. Each token had a different name and logo, often referencing trending topics or pop culture, but shared the exact same contract logic and lacked any real utility. These tokens were promoted through Telegram channels and social media groups to lure retail investors. Forensic analysis revealed that all tokens

were deployed from the same contract factory and interacted with the same liquidity router. Wallet clustering further exposed that many of the initial trades and liquidity provisions were made by addresses linked to the deployer, enabling full control over the market. Once volume peaked, the attacker drained liquidity or dumped large amounts of tokens across multiple addresses.

Uniswap V2 [5]: Another high-profile case involved the cloning of Uniswap's V2 contracts, where scammers deployed a visually identical decentralized exchange using open-source Uniswap code. The frontend mimicked the original platform's interface, but the underlying contracts directed user deposits to attacker-controlled wallets. This clone contract attack was especially effective due to Uniswap's brand trust and open-source nature. Forensic cues included bytecode comparison showing near-identical logic, except for the router address and liquidity destination. Further analysis of DNS records and wallet linkages exposed that the scam frontend had been registered using the same metadata as previously flagged phishing sites. The attackers relied on impersonation rather than code exploitation, making forensic traceability (through infrastructure reuse and smart contract similarity) critical.

1.1.2.4. Flash Loan Exploits

Flash loan exploits are a type of attack in decentralized finance (DeFi) where an attacker takes out a large, uncollateralized loan from a smart contract-based lending protocol and uses it within the same transaction to manipulate markets, exploit vulnerabilities, or drain funds, before repaying the loan instantly. Since flash loans must be repaid within a single block, they pose no risk to the lender, but they give attackers temporary access to massive capital. Exploits often target price oracles, liquidity pools, or contract logic that can be manipulated under abnormal conditions created by the loan, resulting in significant financial loss for protocols and users.

Forensic Signatures and On-Chain Indicators of Flash Loan Exploits

Forensic examination of flash loan exploits identifies several distinctive on-chain characteristics that differentiate these incidents from standard blockchain transactions. A key indicator is the initiation of a large flash loan, commonly sourced from protocols such as Aave, dYdX, or Uniswap. This is typically followed by a complex sequence of contract

interactions executed within a single block. These interactions often involve significant price distortions on decentralized exchanges, abrupt shifts in collateral valuations, or the manipulation of on-chain oracle data. Attackers generally route funds through multiple smart contracts in rapid succession, exploiting protocol vulnerabilities before repaying the borrowed assets within the same atomic transaction. Additional forensic markers include elevated gas consumption, temporary surges in token transfer volume, and calls to functions such as *liquidate()*, *swap()* or *updatePrice()*. When analyzed in combination and aligned temporally, these indicators provide compelling evidence of flash loan-based exploitation.

bZx Lending Protocol Exploit [6]: In February 2020, the bZx protocol was the target of one of the earliest and most illustrative flash loan attacks. The exploiter borrowed 10000 ETH via a flash loan from dYdX and manipulated the price of a collateral asset through large-volume trades on low-liquidity decentralized exchanges. This induced a price distortion that misled the bZx oracle, enabling the attacker to over-borrow assets. The flash loan was repaid within the same transaction, resulting in a net profit of approximately $350000.

From a forensic perspective, the transaction exhibited clear indicators of a flash loan exploit: it began with a flash loan call, included multiple DEX interactions and concluded with calls to *borrow()* and *liquidate()*.

1.1.3. Real-world Case References in Smart Contract Financial Crime

While the mechanisms of smart contract-based financial crime are diverse, their impact becomes most evident through real-world attacks that have caused substantial losses and exposed vulnerabilities in DeFi protocols. High-profile incidents such as Harvest Finance, Uranium Finance, and Meerkat Finance offer practical insight into how different exploit techniques, ranging from flash loan manipulation to rug pulls, are executed in live environments. These cases not only demonstrate the evolving threat landscape but also highlight the forensic indicators and response challenges faced by investigators, auditors and developers alike.

Harvest Finance [7] experienced a flash loan exploit resulting in a loss of approximately $24 million. The attacker manipulated stablecoin

prices (USDC and USDT) on Curve Finance (used by Harvest as a pricing oracle) by inflating their value through flash loan-funded trades. This enabled over-withdrawal from Harvest's vaults. Forensic analysis showed high gas usage, cross-protocol interactions, and repayment within a single transaction. Although the attacker later returned $2.5 million, the incident significantly undermined trust in Harvest's oracle mechanism.

Uranium Finance was exploited due to a critical bug introduced during a contract upgrade on Binance Smart Chain. A miscalculation in the swap function allowed an attacker to drain over $50 million by exploiting rounding errors. The funds were bridged to Ethereum and laundered through privacy tools like Tornado Cash. The absence of a code audit and centralized control over deployment keys contributed to the protocol's vulnerability.

Meerkat Finance suffered a deceptive rug pull disguised as a flash loan exploit. Within 24 hours of launch, over $30 million was drained through a proxy contract upgrade executed by the deployer. Forensic evidence including *AdminChanged* events and storage slot analysis confirmed that the implementation logic had been altered to include a withdrawal function. The incident highlighted the inherent risks of upgradeable contracts lacking transparent governance.

These real-world cases showcase the intersection of technical flaws, trust assumptions, and economic manipulation in DeFi ecosystems. By dissecting these events through a forensic lens, investigators can better understand attack patterns and develop proactive defenses against smart contract-enabled financial crime.

The following Table 1.1 represents a classification of all the above-mentioned smart contract exploit types alongside their forensic signatures.

1.2. Investigative Setup

A comprehensive forensic investigation of smart contract-related financial crime necessitates direct engagement with the Ethereum Virtual Machine (EVM) and access to the blockchain's complete historical state. In contrast to traditional blockchain explorers, which offer limited visibility, forensic analysis requires fine-grained control

over the querying and interpretation of transaction data, storage states, and contract execution behavior. This section presents the essential technical infrastructure for such investigations, beginning with node configuration and extending to advanced methodologies such as storage slot parsing, internal call tracing, and self-destruction event tracking. Together, these tools and techniques constitute the foundation of a robust forensic workflow. The following illustration (Fig. 1.1) shows an end-to-end workflow for smart contract forensic investigations.

Table 1.1. Smart Contract Exploit Types & Forensic Signatures.

Exploit Type	Primary Behavior	Key On-Chain Indicators	Case Study Example
Rug Pull	Liquidity withdrawal post-hype	removeLiquidity, contract self-destruct	Meerkat Finance
Pump-and-Dump	Coordinated buys followed by insider dump	Wallet clustering, token distribution skew	UFO Token, KingDog
Flash Loan Exploit	Temporary capital for manipulation	Single-tx logic chain, price/oracle distortion	bZx, Harvest Finance
Oracle Manipulation	Fake prices exploited for gain	updatePrice, TWAP skew, low-liquidity pool	Harvest, Saddle Finance
Proxy Exploit	Malicious upgrade of contract logic	upgradeTo, storage slot changes	Uranium Finance

1.2.1. Using Full Ethereum Nodes vs. RPC Endpoints

Accessing accurate and complete blockchain data is a foundational requirement in any smart contract forensic investigation. Investigators typically retrieve data either through remote procedure call (RPC) endpoints, often provided by services like Infura, Alchemy, or Ankr or

by running their own full Ethereum node. Each approach has significant trade-offs in terms of visibility, control and trust.

RPC endpoints offer convenience and speed, allowing analysts to interact with Ethereum without maintaining the hardware or synchronization infrastructure. However, these endpoints often limit access to historical state, trace data, and internal execution details. Many hosted services restrict or disable archive data and deep tracing features to optimize performance and reduce cost. This can hinder investigations that require visibility into past contract storage, internal call trees, or pre-state balances. Additionally, relying on third-party RPC providers introduces a trust dependency that may be unacceptable in forensic contexts where data integrity and reproducibility are critical.

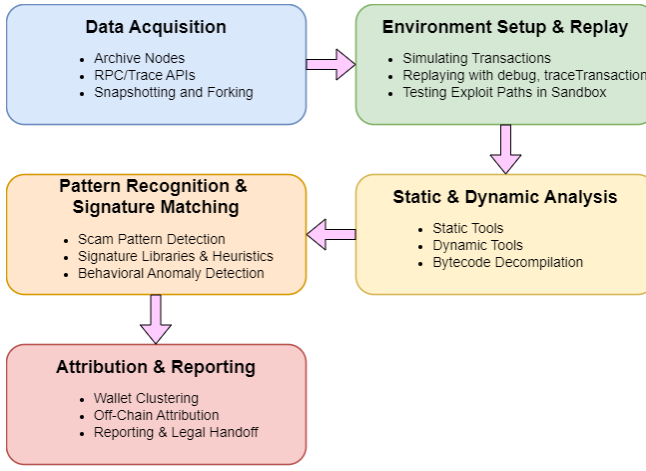

Fig. 1.1. End-to-end Workflow for Smart Contract Forensic Investigations.

By contrast, operating a full Ethereum node, especially in archive mode, gives investigators complete access to the blockchain's full history and state. This includes the ability to query any contract's storage at any past block height, trace the full execution of transactions, and replay contract interactions deterministically. While this approach requires substantial disk space, memory and synchronization time, it ensures data fidelity and allows for deeper analysis, such as reconstructing oracle values, verifying historical balances, and observing how contract logic evolves over time. For forensic purposes, particularly when handling complex exploits or litigation-sensitive cases, full nodes are often indispensable.

1.2.2. Snapshotting and Replaying Contract Interactions

An essential technique in smart contract forensics is the ability to reproduce past contract behavior by snapshotting the blockchain state at a specific block height and replaying transactions in a controlled environment. This allows investigators to validate how a particular interaction unfolded on-chain, detect anomalies, and verify claims about contract logic and fund movements, especially in cases involving oracle manipulation, reentrancy attacks or unexpected storage changes.

Forensic investigation of smart contract activity often requires access to the blockchain's historical state. By operating a full Ethereum node in archive mode, analysts can query contract state and behavior at any given block. Tools such as debug_traceTransaction, Nethermind's tracing API or the EthereumJS VM (paired with manual state snapshots) allow for detailed reconstruction of transaction execution paths, including internal calls, event logs, storage changes and gas usage at the opcode level. This enables precise differentiation between normal functionality and exploitative behavior.

Snapshotting is equally important for controlled testing. Using frameworks like Hardhat, Foundry or Ganache, analysts can fork the chain at a specific block to simulate historical conditions. This enables reproducible testing of scenarios such as flash loan exploits or oracle manipulation in an isolated environment, facilitating rigorous validation of attack hypotheses without interacting with the live network.

1.2.3. Parsing the EVM State

1.2.3.1. Storage Slots

In Ethereum, smart contracts persist data in a low-level structure known as storage slots - a fixed-size key-value store where each key corresponds to a 32-byte word. Understanding and parsing these storage slots is crucial in forensic investigations, especially when analyzing changes in contract state over time, such as token balances, ownership roles, liquidity positions or governance parameters. Since the EVM does not expose variable names or data structures on-chain, investigators must often reverse-engineer how data is stored using knowledge of Solidity's storage layout and compiler behavior.

Each state variable in a smart contract is assigned to a specific storage slot based on its declaration order. Simple variables are stored sequentially, while mappings and dynamic arrays use a *keccak256* hash-based scheme to determine their storage position. For example, a mapping like *mapping(address => uint256) balances* stores the value for each address at the slot calculated as *keccak256(key . slot),* where the *key* is the address and *slot* is the mapping's base position. Forensic tools such as eth_getStorageAt, Slither, Dedaub's Storage Layout Inspector, and Ethers.js with manual decoding allow investigators to read and decode raw storage values.

Parsing storage slots enables analysts to reconstruct contract states at specific block heights, detect unauthorized changes and correlate those with transaction activity. This is particularly important in cases involving rug pulls, unexpected minting or burning of tokens, manipulated voting power or compromised admin roles. By comparing storage snapshots before and after key transactions, forensic analysts can pinpoint the exact state transitions that enabled or resulted from malicious actions.

1.2.3.2. Internal Calls

Internal calls refer to function calls made by a smart contract to another contract or to itself during execution, without creating a separate transaction. These calls do not appear in the top-level transaction list on block explorers like Etherscan, making them harder to trace without specialized tooling. However, they often play a critical role in complex attack scenarios, particularly in reentrancy exploits, proxy upgrades, and contract interactions that manipulate state or control flow across multiple contracts.

To uncover internal call activity, forensic analysts rely on transaction tracing, which reveals the full execution context of a transaction, including nested calls, value transfers, function selectors and return data. Tools like Geth's *debug_traceTransaction*, Nethermind's *trace modules*, or external services such as Tenderly and OpenEthereum's *tracing RPC* allow analysts to build a call tree for each transaction. This enables reconstruction of how control and data flowed across multiple contracts in a single atomic operation.

Identifying internal calls is especially important in detecting reentrancy attacks, where a contract is called back before its state is updated, enabling exploits like draining funds or bypassing access control.

Similarly, internal delegate calls *(DELEGATECALL)* used in proxy patterns must be closely analyzed to determine where logic execution occurs and who has authority over upgrades or control functions. By tracing internal calls and their associated storage changes, investigators can expose hidden contract behaviors and link them directly to exploit outcomes.

1.2.3.3. Self-destruct Tracking

The SELFDESTRUCT opcode in the Ethereum Virtual Machine enables a smart contract to remove itself from the blockchain state and optionally transfer its remaining ether to a designated address. While used in legitimate scenarios such as upgradeable proxies or contract decommissioning, it can also serve malicious purposes. In forensic investigations, tracking SELFDESTRUCT is crucial for identifying fund obfuscation, erasure of malicious logic, or the deliberate termination of an exploit sequence. Once a contract self-destructs, its bytecode is removed from the active state, although historical transactions remain immutable. This limits the window for code analysis, making real-time monitoring essential. Investigators can detect such events by analyzing transaction traces for the SELFDESTRUCT opcode or by observing a change in the contract's code hash to `0x0`.

In practice, self-destruct tracking reveals patterns where attackers erase traces after exploiting victims. This includes rug pulls and token scams that destroy the contract after draining funds, or temporary helper contracts designed to self-destruct after executing a malicious function. Through opcode-level tracing and lifecycle analysis, analysts can reconstruct the role of self-destructs in broader exploit campaigns.

1.3. Static and Dynamic Analysis Techniques

Smart contract forensics require both static and dynamic approaches to fully understand a contract’s behavior, identify vulnerabilities, and reconstruct attack vectors. Static analysis focuses on examining the contract’s code or bytecode without executing it, allowing analysts to extract control flow, detect malicious logic and uncover hidden functions or backdoors. On the other hand, dynamic analysis involves simulating or observing contract execution in real or test environments to study runtime behavior, gas consumption, reentrancy paths and event

emissions. By combining both techniques, forensic investigators gain a comprehensive view of how a smart contract behaves both in theory and in practice, enabling more accurate detection of exploits and attribution of malicious activity.

1.3.1. Static Analysis

Static analysis involves inspecting the structure, logic and bytecode of a smart contract without executing it. This method is essential in forensic investigations when source code is unavailable or obfuscated, as it allows analysts to reverse-engineer contract behavior directly from the Ethereum Virtual Machine (EVM) bytecode. Tools such as Panoramix, evm-decompiler and Mythril can decompile low-level bytecode into human-readable pseudo-code, enabling analysts to reconstruct control flow and understand the contract's functional components.

One of the key goals in static analysis is control flow and logic extraction for understanding how data moves through the contract, how conditional branches are evaluated, and which functions are publicly accessible or restricted. This step is particularly important when tracing how funds are moved, how access is granted or denied and whether specific functions (such as minting, withdrawal or upgrades) can be abused. Static analysis can also reveal hard-coded backdoors, such as functions accessible only to a specific address or kill switches: code paths that allow the contract to be disabled or drained by the deployer.

Beyond manual inspection, static analysis tools can perform symbolic execution and vulnerability scanning to automatically flag known anti-patterns like reentrancy risks, unrestricted *delegatecall* usage, integer overflows or unsafe external calls. While static analysis alone cannot predict runtime behavior in all cases, it provides the foundational understanding needed to formulate hypotheses, verify attack feasibility and uncover latent risks.

1.3.2. Dynamic Analysis

Dynamic analysis involves observing or simulating the runtime behavior of smart contracts to understand how they function under real-world conditions. Unlike static analysis, which examines code structure, dynamic techniques test how the contract responds to specific inputs,

transaction sequences or blockchain states. This approach is especially useful for reproducing exploits, identifying abnormal behaviors, and tracing vulnerabilities that depend on context, such as reentrancy, gas manipulation or oracle dependencies. Tools like Tenderly, Foundry, Hardhat and Ganache allow investigators to fork the blockchain at a specific block height and simulate transactions in a sandboxed environment. This enables the safe recreation of real-world attack scenarios such as simulating a flash loan exploit or testing how a contract behaves when an oracle is manipulated. By replaying interactions with controlled variables, analysts can observe gas usage patterns, execution traces, event emissions, and storage mutations at each step, gaining granular insight into the mechanics of an exploit.

Dynamic analysis also facilitates monitoring behavioral patterns, such as abnormal token minting, unexpected self-destructs or suppressed event logs. By tracing transaction paths and decoding emitted events, investigators can detect inconsistencies or stealthy contract logic that might not be obvious from code inspection alone. Combined with transaction-level tracing tools and memory inspection, dynamic analysis serves as a powerful complement to static techniques enabling end-to-end reconstruction of attack chains and validation of exploit feasibility.

1.4. Signature-based and Heuristic Detection

While manual analysis remains critical in complex investigations, large-scale detection of fraudulent smart contracts increasingly relies on signature-based and heuristic methods. These techniques focus on identifying known malicious patterns: either through explicit bytecode signatures or through behavioral characteristics that are frequently reused across scam contracts. By recognizing common traps such as honeypots, non-transferable tokens and high-tax reflection tokens, analysts and automated tools can flag suspicious contracts early in their lifecycle. This section explores these patterns in detail and introduces open-source tools and detection frameworks, such as Slither, that incorporate static signature matching, logic checks and deobfuscation techniques to uncover smart contract-based scams at scale.

1.4.1. Identifying Common Scam Contract Patterns

1.4.1.1. Honeypots

Honeypots are deceptive smart contracts designed to lure victims into interacting with them (usually by buying tokens) under the false impression that they can profit, only to discover that they are unable to withdraw or resell the assets. These scams are particularly dangerous because they often appear functional during casual testing: token transfers may work, liquidity may be available and prices may rise in response to buys [8]. However, the contract includes hidden logic that selectively blocks or reverts selling actions, usually based on the caller's address or transaction context.

From a forensic standpoint, honeypots typically contain conditional transfer restrictions, such as logic that reverts *transfer()* or *transferFrom()* calls for all addresses except the deployer or whitelisted wallets. These conditions may be obfuscated using misleading variable names, logic branches, or arithmetic tricks that hide the true behavior. Some contracts simulate a working token by allowing users to buy and hold the asset, but any attempt to sell triggers a silent fail or forces a transaction revert with no clear error message. Others may route tokens to dead addresses or burn them to create a false impression of scarcity.

Detection of honeypots often requires dynamic interaction testing using simulated wallets or automated tools that attempt to buy and sell tokens under various conditions. On-chain detection techniques can also include scanning for asymmetric transfer success, presence of *require(tx.origin == owner)* or conditional *revert()* paths. Tools like HoneyBadger and Honeypot.is automate this analysis by running behavioral simulations and comparing the outcomes of buy vs. sell actions. These forensic patterns are essential for both investigators and user-protection tools to identify and classify honeypot scams before they spread.

1.4.1.2. No-transfer Tokens

No-transfer tokens are fraudulent smart contracts that permit users to acquire tokens but restrict or prevent outbound transfers, effectively locking the assets in user wallets. Unlike honeypots, which conceal malicious logic in sell functions, no-transfer tokens often impose explicit or concealed conditions within *transfer()* or *transferFrom()* functions

that block user exits. These tokens are frequently used in rug pulls and deceptive airdrops to simulate market activity while ensuring victims cannot liquidate their holdings.

From a forensic perspective, such tokens can be identified by analyzing non-standard transfer logic, including hardcoded conditions that revert transactions or redirect tokens. Common patterns include restrictive clauses such as *require(sender == owner)* or excessive transfer taxes that nullify the sender's balance. Static analysis tools like Slither and Mythril can flag these anomalies, while dynamic testing frameworks, such as Honeypot.is or custom scripts, simulate transfers to detect identity-based or conditional restrictions. Detecting these schemes is essential for identifying scams where the deception lies in immobilizing user assets rather than in how the token is marketed.

1.4.1.3. Reflection and Tax Traps

Reflection and tax tokens are a class of smart contracts that impose automatic transaction fees, often marketed as features that reward holders or fund development. While some legitimate projects use reflection mechanics (e.g., distributing fees among holders), tax traps exploit this mechanism to manipulate trading behavior or extract disproportionate fees from unsuspecting users. In scam implementations, these taxes may reach extremely high or even total values, effectively burning or redirecting 99–100% of the transferred amount to the contract owner or a hardcoded wallet [9]. From a forensic perspective, these tokens frequently override the standard ERC-20 *transfer()* and *transferFrom()* functions to insert fee logic. This may include logic that sends a portion of each transaction to the developer's wallet, redistributes it among holders or burns it. Malicious variants may include dynamic tax logic, where the fee changes based on block number, transaction origin or liquidity conditions. In some cases, taxes are initially set to a low value to attract users and are later increased through owner-controlled functions (*setTaxRate(), enableMaxTax()*), often without user visibility or governance controls.

Heuristically, detection of tax traps involves scanning for non-standard tokenomics, particularly contracts that include hidden fee-setting functions, obscure routing logic, or conditions tied to sender/recipient roles. Analysts may also look for red flags like high *balanceOf()* reductions post-transfer or a spike in token accumulation by a single

wallet after trading activity. Tools like Slither's detectors, RugDoc and custom simulation environments can help reveal these hidden mechanics by emulating trade flows and measuring the effective net transfer. Reflection and tax traps exploit the complexity of token logic and the lack of user-facing clarity, making their identification a key priority in smart contract fraud detection.

1.4.2. Automated Signature Libraries

Given the growing volume of malicious smart contracts deployed across blockchain networks, manual inspection alone is no longer sufficient for timely detection. This has led to the development of automated signature libraries - rule-based and behavior-based systems that scan smart contract code for known exploit patterns, vulnerabilities, and suspicious logic. These libraries are integral to modern blockchain forensic workflows, providing scalable and consistent methods for detecting scam tokens and malicious behaviors across thousands of contracts.

One of the most widely used tools in this category is Slither [10], a static analysis framework developed by Trail of Bits. Slither includes a suite of built-in detectors that scan Solidity source code for suspicious and dangerous patterns. It can also be extended with custom detectors, allowing analysts to flag patterns like honeypot conditions, suspicious tax logic, or backdoors that enable fund extraction. Other tools such as Mythril, Conkas, and Surya offer complementary capabilities for symbolic execution and contract structure visualization.

Signature-based systems typically rely on pattern matching within abstract syntax trees (ASTs), control flow graphs or EVM bytecode. In forensic settings, these tools are particularly valuable for bulk screening of newly deployed contracts, auditing scam factory outputs, or identifying cloned logic across unrelated projects. While signature libraries can produce false positives in edge cases, they remain a vital first line of defense, enabling investigators to triage high-risk contracts before committing to deeper manual or dynamic analysis.

1.4.3. On-chain Honeypot and Obfuscation Techniques

Sophisticated scam contracts often rely on obfuscation techniques to hide malicious behavior from both users and automated analysis tools. These techniques aim to make the contract appear benign or unreadable while

retaining full exploit functionality. In honeypots, for example, developers frequently mask restrictive logic through convoluted conditional statements, misleading function names or unused variables that divert attention from the actual trap mechanism. Unlike clear backdoors or tax logic, these traps are deliberately subtle and are triggered only in specific scenarios, such as when a non-whitelisted address attempts a token sale.

```
function transfer(address to, uint256 amount) public returns (bool){
    if (tx.origin == owner || to == liquidityPool) {
        balances[msg.sender] -= amount;
        balances[to] += amount;
        emit Transfer(msg.sender, to, amount);
        return true;
    } else {
        return false; //Appears as a failed transfer, no revert message
    }
}
```

The above example shows a token that allows transfers only to a specific liquidity pool or from the owner address. All other transfers silently fail without reverting, deceiving the user into thinking the token is broken or the transaction failed due to network issues. Some contracts go further by splitting logic across multiple contracts, using *delegatecall* or hiding functionality behind proxy layers or initialization routines that change behavior post-deployment. Obfuscation can also take the form of encrypted strings, unused logic branches, or conditionals based on block height or transaction gas, which only activate under carefully timed conditions.

Detecting such techniques requires deep bytecode analysis, comparison with known scam contract templates, and behavioral testing under varied execution environments. While automated tools can catch some red flags, heavily obfuscated honeypots often demand manual deconstruction or symbolic execution to reveal their intent. These techniques represent a growing challenge in the arms race between scammers and forensic investigators and underscore the importance of continuous tool improvement and threat intelligence sharing across the security community.

As smart contract-based scams continue to evolve in complexity and scale, signature-based and heuristic detection methods provide critical capabilities for early identification and risk assessment. By analyzing

recurring patterns such as honeypots, no-transfer tokens and tax traps, and by leveraging automated tools like Slither and Honeypot.is, investigators can systematically flag malicious contracts, often before they are widely adopted or exploited. While no detection method is foolproof, combining static signatures, behavioral testing, and knowledge of obfuscation strategies creates a powerful multilayered defense. In the broader context of blockchain forensics, these techniques serve not only to detect scams but also to illuminate the methods and infrastructure behind them, contributing to a safer and more transparent decentralized ecosystem.

1.5. Cross-contract Interaction Tracing

As DeFi protocols become more complex, many attacks involve interactions across multiple contracts, including proxies, governance systems, token bridges and liquidity pools. Understanding these exploits requires tracing the flow of data, control and assets across contract boundaries. This process, known as cross-contract interaction tracing, entails reconstructing transaction call trees, identifying delegated execution paths and tracking value movement through layered abstractions. This section examines how analysts perform such tracing, with emphasis on upgradeable proxies, DAO-controlled contracts and bridge or liquidity pool exploit chains, each posing distinct forensic challenges in attribution and analysis.

1.5.1. Proxies and Upgradable Contracts (ERC-1967, EIP-1822)

Modern smart contract systems often rely on proxy patterns to enable contract upgrades without redeploying or migrating state. While essential for maintainability and flexibility, these patterns introduce forensic complexity by separating storage from logic with one contract holding the data (the proxy) and another executing the code (the implementation). Two common standards for this design are ERC-1967, which stores the implementation address in a designated storage slot, and EIP-1822 (Universal Upgradeable Proxy Standard), which uses a standardized *delegatecall*-based structure.

In forensic investigations, proxy contracts require analysts to trace not only user-facing transactions, but also *delegatecall* chains that redirect

execution to underlying implementations. Since the actual code resides in a separate contract, investigators must resolve the implementation address, often stored in a specific slot like *0x360894A13BA1A3210667C828492DB98DCA3E2076CC3735A920A3CA505D382BBC* for ERC-1967 and analyze the logic hosted there. This indirection can obscure the true behavior of a transaction, especially in upgrade attacks where the implementation is replaced with malicious code using functions like *upgradeTo()* or *setImplementation().*

Proxies are frequently targeted in attacks due to their centralized upgradeability. A notable forensic cue is a sudden change in the implementation address, often followed by suspicious activity such as liquidity withdrawal, token minting or privilege escalation. Detecting such changes requires monitoring contract storage over time and parsing events like *Upgraded* or *AdminChanged.* When combined with transaction tracing and storage diffing, proxy analysis enables investigators to reconstruct how an attacker gained control, altered logic and executed a coordinated exploit across interconnected contracts.

1.5.2. Multisig Wallets and DAO Interactions

Multisignature (multisig) wallets and Decentralized Autonomous Organizations (DAOs) introduce collaborative governance mechanisms into smart contract ecosystems. They are designed to enhance security and decentralization by requiring multiple parties to authorize critical actions, such as contract upgrades, fund transfers or protocol parameter changes. However, from a forensic standpoint, tracing interactions involving multisigs and DAOs is complex, as they often act as intermediaries, introducing layers of indirection between user intent and contract execution.

A multisig wallet typically logs proposal, confirmation and execution steps as discrete on-chain events. For investigators, it is crucial to reconstruct the full authorization flow: who proposed the action, which addresses signed it and when execution occurred. These workflows are observable through emitted events like *SubmitTransaction, ConfirmTransaction,* and *ExecutionSuccess.* Forensic clues often lie in timing patterns (e.g., approvals clustered within minutes), unusual quorum behavior or use of the multisig to rapidly push through high-risk operations such as *upgradeTo()* or *transferOwnership().*

DAOs, especially those built on frameworks like OpenZeppelin Governor or Compound Governance, involve smart contract-based voting logic and time-delayed execution mechanisms. Investigators must trace proposal lifecycle stages from *propose(),* to *castVote()* to *queue()* and *execute()*, and correlate them with downstream effects. In exploit scenarios, attackers may accumulate voting power via flash loans or stealth token transfers, then pass proposals that alter control or siphon funds. An example is the 2022 Beanstalk DAO attack, where the attacker used a flash loan to gain a supermajority, vote in their own malicious proposal and execute it in a single transaction.

Effective cross-contract tracing in DAO and multisig environments involves event log correlation, transaction path reconstruction, and governance logic analysis. These interactions rarely happen in isolation, and successful forensic analysis requires understanding both the technical architecture and the social governance models they encode.

1.5.3. Bridge and Liquidity Pool Exploits

Token bridges and liquidity pools are fundamental components of DeFi infrastructure, enabling interoperability and asset flow across blockchains and decentralized exchanges. However, their complexity and high-value holdings make them prime targets for sophisticated cross-contract exploits. From a forensic standpoint, these attacks often involve multi-transaction sequences, cross-chain data manipulation, or desynchronization between smart contracts and off-chain logic, requiring detailed tracing of control flow, oracle updates and asset movement across multiple protocols.

Bridge and liquidity pool exploits often involve complex, multi-contract interactions that require deep forensic tracing. In bridge attacks, vulnerabilities typically stem from flaws in cross-chain message validation. For example, the 2022 Wormhole exploit exploited a missing signature check in the Solana-Ethereum bridge, allowing unauthorized minting of 120000 wrapped ETH [11]. Investigating such incidents involves analyzing Ethereum-side events, bridge contract logic and state synchronization. Key forensic indicators include minting without corresponding burns, validator bypasses, and state mismatches.

Liquidity pool exploits commonly rely on price manipulation, reserve imbalance or flawed fee logic. Attackers often use flash loans to distort asset prices within automated market makers (e.g., Uniswap, Curve),

triggering oracle errors or undercollateralized borrowing. The 2020 Harvest Finance exploit followed this pattern, using a flash loan to manipulate Curve's stablecoin prices and withdraw inflated amounts from Harvest vaults, all within a single transaction.

Effective forensic analysis in these cases requires multi-contract and cross-chain transaction tracing, event log correlation, and call stack reconstruction. Analysts must also understand liquidity pool dynamics and bridge state models. As DeFi grows increasingly modular and interconnected, mastering cross-contract tracing is essential for uncovering exploit paths and anticipating future attack vectors.

1.6. Forensics of DeFi Protocol Exploits

Decentralized finance (DeFi) protocols present unique forensic challenges due to their composability, permissionless access, and high capital exposure. Exploits in this domain often involve sophisticated chains of transactions and cross-contract manipulations executed within a single block. To effectively reconstruct these attacks, forensic analysts must examine not only the code and logic of affected contracts, but also the real-time dynamics of flash loans, oracle dependencies, and liquidity pool mechanics. This section breaks down the forensic process for analyzing major exploit types, including flash loan-based attacks, oracle manipulation, and liquidity pool token (LP) theft and draining patterns, providing structured methods and on-chain indicators to decode complex financial attacks.

1.6.1. Flash Loan Lifecycle Analysis

Flash loan-based exploits are among the most common and complex forms of DeFi attacks, as they allow adversaries to borrow massive sums of capital with no collateral, provided the loan is repaid within the same transaction. The forensic process of analyzing such attacks involves breaking down the entire lifecycle of the flash loan into distinct phases: loan acquisition, execution sequence and liquidity extraction.

The first phase, loan acquisition, begins with a call to a flash loan provider such as Aave, dYdX or Uniswap. This call typically initiates the transaction and includes parameters that define the loan amount, token type and callback logic. From a forensic perspective, identifying

the initial call and its lender is essential to understanding the capital flow and the attack's economic feasibility. Analysts can detect these by scanning for calls to *flashLoan(), flashLoanSimple()* or *swap()* with high token volumes and no preceding collateralization.

Next is the execution sequence, where the borrowed funds are used across a series of tightly orchestrated contract calls. This is often the most complex part of the exploit and may include price manipulation on DEXs, oracle spoofing, token minting or triggering of under-collateralized liquidation mechanisms. Forensic tools like *debug_traceTransaction* or Tenderly's execution visualizer are used to reconstruct the call stack, track internal calls and analyze gas usage. Indicators such as abnormal reserve shifts, token price distortions and synthetic asset mispricing are critical during this phase.

Finally, in the liquidity extraction stage, the attacker consolidates gains by converting the ill-gotten assets into ETH, stablecoins, or bridgeable tokens and repays the loan within the same transaction. Analysts can trace this by following the token flows through the final few contract interactions, often ending with a *repay()* or equivalent settlement function. Any remaining profit is then withdrawn to an EOA or routed through mixers or bridges. The atomic nature of the flash loan makes it imperative to analyze the entire transaction as a single unit, as partial views may obscure the exploit logic.

Effective forensic analysis of flash loan attacks requires mapping the complete lifecycle from capital sourcing to execution and exit and correlating it with behavioral signatures such as high transaction density, multiple protocol hops, and coordinated token transfers. These patterns not only help attribute the attack but also inform future detection and mitigation strategies. The following figure (Fig. 1.2) illustrates the four main phases of a Flash Loan Exploit – capital acquisition, price manipulation, contract abuse and profit exit.

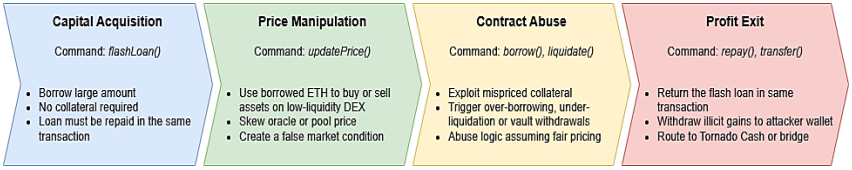

Fig. 1.2. Lifecycle of a Flash Loan Exploit: Phases and On-Chain Indicators.

1.6.2. Oracle Manipulation and Attack Surface Mapping

Many DeFi protocols rely on on-chain oracles to provide external data (usually asset prices) which are critical to maintaining fair trading, borrowing and liquidation processes. Oracle manipulation occurs when an attacker artificially skews price data to gain financial advantage, such as triggering under-collateralized loans, mispricing assets in a swap or influencing DAO votes. Forensic analysis of these exploits requires mapping how the oracle is sourced, how frequently it's updated, and how it influences downstream contract logic.

There are two main types of oracles used in DeFi: DEX-based oracles, such as those relying on Uniswap time-weighted average price (TWAP), and aggregated external price feeds, such as Chainlink. DEX-based oracles are particularly vulnerable to manipulation, as they can be skewed by high-volume trades or low-liquidity pools. In these cases, an attacker might use a flash loan to execute large swaps that temporarily inflate or deflate the token price, then exploit the manipulated price window before the oracle rebalances. This pattern was seen in the Harvest Finance and Saddle Finance exploits.

From a forensic standpoint, analysts must identify the oracle's data source, update frequency and integration points within the target contract. This involves tracing functions like *updatePrice(), getPrice(),* or calls to price aggregators. On-chain indicators include abrupt price shifts within a block, unusually large token swaps or changes to collateral requirements based on transient pricing. Mapping the oracle's attack surface also involves understanding fallback mechanisms, timelocks, and reliance on manipulable sources such as liquidity pool ratios or single-point external price updates.

By combining storage inspection, call tracing and event log analysis, investigators can uncover whether an oracle was exploited directly (e.g., price injection) or indirectly (e.g., manipulated liquidity ratios). Identifying such manipulation is critical not only for diagnosing the exploit but also for improving protocol resilience by integrating tamper-resistant oracles, adding delay mechanisms or enforcing multi-source validation before executing sensitive operations.

1.6.3. LP Token Theft and Draining Behavior

Liquidity provider (LP) tokens represent a user's share of assets deposited into a decentralized exchange (DEX) liquidity pool. Because these tokens often confer ownership rights over significant on-chain capital, they are a high-value target in many DeFi exploits. LP token theft typically occurs when an attacker finds a way to mint, seize or manipulate LP tokens, allowing them to drain reserves from DEX pools such as those used in Uniswap, SushiSwap or Curve. Forensic analysis of such exploits requires reconstructing how the attacker gained control over LP tokens and how the draining occurred, whether directly or by bypassing minting/burning safeguards.

In many cases, LP token theft involves exploiting poorly implemented access controls or manipulating the underlying reserves to gain disproportionate liquidity shares. For example, an attacker might inject a small amount of one token and a manipulated amount of another (using spoofed pricing or flash loans) to mint LP tokens at a skewed ratio. Once minted, these tokens can be burned to withdraw a far greater share of the pool than was deposited. Other attacks target functions that miscalculate pool balances or apply incorrect fees, allowing stealthy reserve extraction.

On-chain forensic indicators include unexpected LP token minting spikes, low-volume or asymmetrical liquidity additions and burn events that coincide with abnormal token outflows. Tracing these requires following calls to functions like *mint(), burn(), skim(),* or *sync()* and comparing them to reserve states before and after execution. Analysts may also identify repeatable patterns across cloned DEX contracts or scam tokens that auto-issue LP tokens without proper accounting.

By reconstructing the reserve changes, token flows, and contract events, forensic investigators can pinpoint how LP tokens were misused and what vulnerabilities were exploited. These patterns are critical not only for attributing the attack but also for helping protocol developers patch faulty logic in token math, liquidity management and permission models.

1.7. Tools and Automation

As smart contract ecosystems expand and threats become more sophisticated, manual investigation alone is no longer scalable. Effective

smart contract forensics now relies on a combination of specialized tools, scripting frameworks and automated detection pipelines. These tools not only accelerate analysis but also enhance accuracy and reproducibility by enabling programmatic inspection of contract behavior, storage changes and transactional patterns. This section highlights key forensic tools such as Slither, MythX, Echidna and Tenderly, along with development frameworks like Web3.py, Hardhat, and Brownie, which support the creation of custom analysis scripts. It also explores the growing role of automation, including real-time alerting based on on-chain events, contract upgrades, and governance transitions.

1.7.1. Forensic Tools Overview

A wide range of open-source and commercial tools are available to support smart contract forensic analysis, each tailored to specific stages of investigation such as vulnerability detection, execution tracing, or reverse engineering. Together, these tools form the backbone of modern investigative workflows, enabling analysts to inspect bytecode, simulate execution paths, and identify malicious patterns with precision and scale.

Slither, developed by Trail of Bits, is a static analysis tool for Solidity that parses contract code into an abstract syntax tree (AST) and applies a suite of detectors to flag risky behaviors such as unprotected *delegatecall*, incorrect inheritance hierarchies, and reentrancy vulnerabilities. MythX (and its open-source counterpart Mythril) performs symbolic execution to explore different execution paths, detecting complex issues like transaction-order dependence, integer overflows and logic inconsistencies that may be exploited in practice.

For fuzzing and invariant testing, Echidna enables analysts to define security properties and simulate randomized input across a smart contract's state space, surfacing edge cases that traditional testing might miss. Tenderly offers a dynamic analysis and transaction tracing platform with an intuitive interface for visualizing call stacks, storage diffs, gas usage and emitted events. It is particularly effective for reconstructing flash loan attacks, reentrancy paths and multi-contract execution flows. Lastly, Ethersplay provides an EVM disassembler and bytecode visualizer for situations where source code is unavailable, allowing analysts to reverse engineer control flow from raw opcodes.

Each tool addresses a different layer of the EVM analysis stack (source, bytecode and runtime) and together they offer powerful coverage for both pre- and post-incident investigation. Mastery of these platforms significantly enhances an investigator's ability to detect exploits, validate vulnerabilities, and trace malicious contract logic.

1.7.2. Custom Scripting with Web3.py, Brownie, and Hardhat

While pre-built tools offer valuable detection capabilities, custom scripting is often essential for conducting deeper or more tailored forensic investigations. By leveraging development frameworks such as Web3.py, Brownie and Hardhat, analysts can programmatically interact with smart contracts, automate repetitive tasks and create custom pipelines to query historical state, analyze storage or simulate attack paths.

Web3.py [12], a Python library for interacting with Ethereum nodes via JSON-RPC, is widely used for forensic scripting due to its flexibility and ease of integration with data science and automation tools. It allows analysts to trace transactions, read and decode contract storage, filter event logs, and reconstruct block-by-block state transitions. For example, a Web3.py script can be used to iterate over historical blocks and detect changes to critical storage slots such as admin addresses or token balances.

Brownie [13] builds on Web3.py and adds a testing framework, project structure and deployment tools for smart contract analysis in Python. It supports forking main net chains, enabling forensic analysts to reproduce historical exploits and simulate attacker actions in a sandboxed environment. Brownie also makes it easy to write test cases for edge conditions, analyze failed transactions, and extract internal calls and reverts for deeper inspection.

Hardhat [14], a JavaScript/TypeScript-based development environment, is particularly effective for low-level analysis and scripting in Ethereum-compatible environments. It offers forking, tracing, and gas reporting capabilities and integrates with plugins like hardhat-tracer to expose internal call data and event emissions. For investigators with front-end or full-stack backgrounds, Hardhat provides a powerful alternative to Python-based workflows.

Together, these scripting frameworks empower forensic analysts to go beyond tool-based detection and build custom workflows tailored to specific contracts, exploits or behavioral patterns. Whether it's monitoring protocol upgrades, reproducing flash loan paths, or decoding proxy storage changes, scripting enables precision, automation and scalability in investigative work.

1.7.3. Automated Pipelines for Flagging Malicious Behavior

As the deployment of smart contracts accelerates, manual review is no longer sufficient to detect threats in real time. To address this, security researchers and protocol teams are increasingly building automated detection pipelines - systems that continuously monitor on-chain activity and flag suspicious behavior based on predefined rules, event patterns or code signatures. These pipelines are critical for early warning, protocol defense and forensic triage, especially when dealing with rug pulls, governance takeovers, contract upgrades or unexpected token behavior.

A common automation strategy involves setting event-based triggers that watch for high-risk contract activity. Examples include detecting *OwnershipTransferred* or *UpgradeTo* events, tracking abnormal token minting or burning or flagging flash loan usage above a certain threshold. These events can be captured using Ethereum node subscriptions or log filters and piped into alerting systems such as Slack bots, email notifiers or SIEM dashboards. More advanced pipelines may include real-time simulation layers, where suspicious contracts are automatically forked and tested using frameworks like Hardhat or Tenderly to assess behavior before public exposure.

Another useful feature in these pipelines is anomaly detection, powered by heuristics or machine learning models. These systems can identify sudden changes in transaction patterns, price oracles or governance activity across DeFi protocols. For instance, a DAO proposal receiving unusually fast votes from freshly funded wallets may trigger an alert, as could a sharp drop in liquidity within a single block.

By integrating contract scanners, transaction monitors, and custom alert systems, automated pipelines can proactively defend protocols and dramatically reduce time-to-detection. For forensic analysts, these systems also serve as a pre-filter, highlighting contracts and events that merit deeper investigation using manual or semi-automated techniques.

Table 1.2. Summary Table of Tools and Techniques.

Tool / Framework	Type	Used For	Example Use
Slither	Static Analysis	Detect patterns, access control flaws	Honeypot scan
Foundry / Hardhat	Dynamic analysis	Replay and simulate exploits	Flash Loan Simulation
Web3.py	Custom scripting	Read storage, automate checks	Check balances pre/post exploit
Tenderly	Runtime tracing	Visualize tx traces. gas usage	Cross-contract replay
Etherscan + Graph	Data aggregation	Address clustering, label enrichment	Trace funds to CEX

1.8. Compliance and Attribution

While technical forensics can reveal how a smart contract exploit occurred, compliance and attribution efforts aim to answer a more difficult question: “Who was responsible?”. This section explores the intersection of on-chain analysis and real-world identity tracing, focusing on how forensic investigators link malicious smart contracts to individual wallets, centralized exchange accounts and off-chain personas. Techniques include tagging known attacker addresses, tracking fund flows through mixers and bridges and cross-referencing on-chain activity with external data sources such as GitHub commits, ENS registrations, Telegram aliases or deployed contract metadata. These attribution methods are vital not only for law enforcement and compliance teams, but also for improving threat intelligence and preventing repeat offenders from operating anonymously across the DeFi ecosystem.

1.8.1. Linking Attacker Contracts to Wallets and Exchanges

One of the core goals of smart contract attribution is tracing the relationship between malicious contracts and the wallets that control or benefit from them. This linkage allows investigators to follow the money trail, uncover clusters of related addresses and ultimately identify points where an attacker interacts with regulated entities, such as centralized exchanges (CEXs) or fiat on/off-ramps.

The process typically begins by identifying the deployer address of a malicious contract and tracking its subsequent activity. Using tools like Etherscan, Arkham, Nansen, or custom graph-based analytics, investigators can map outgoing transfers, contract interactions and address clusters. Particular attention is paid to any transfers to or from known exchange wallets, mixer services (e.g., Tornado Cash) or bridge contracts, which may indicate attempts to obscure fund movement or exit to other chains. If the attacker later withdraws funds to a CEX deposit address, this may present a compliance opportunity: regulated exchanges often have KYC/AML procedures in place that can be leveraged by law enforcement with appropriate legal authority.

Forensic signatures that aid in this process include unusual gas patterns, repeated funding behavior across wallets, use of common tools or libraries in multiple malicious deployments and consistent fallback wallets receiving proceeds from various scams. Advanced investigations may also correlate contract creation timing with wallet funding events or transaction fee sponsorship patterns. These details help establish a linkage graph between on-chain entities and potential attribution targets, enabling investigators to bridge the gap between pseudonymity and identity.

1.8.2. Tagging Known Attacker Addresses

Tagging attacker addresses is a critical step in attribution workflows, enabling the blockchain security community to build a shared knowledge base of high-risk entities. These tags, whether applied manually, by automated heuristics or through community reporting, help investigators quickly identify repeat offenders, trace exploit proceeds and block known bad actors from interacting with compliant protocols or platforms [15].

Address tagging can be based on a variety of criteria, including proximity to known exploits, repeated contract deployments with similar bytecode, or behaviors like immediate liquidity withdrawal following token launches. Security platforms such as Etherscan, Chainalysis, TRM Labs, and Nansen maintain and publish curated address labels for scam deployers, mixer exit points, phishing sites and ransomware wallets. Analysts can also contribute to or consult open threat intelligence sources like ScamSniffer or Web3Sanctuary which often include wallet identifiers and exploit timelines.

Beyond public tagging, internal tagging systems within forensic platforms allow investigators to mark addresses as suspicious or linked to specific behaviors, even before formal attribution. This helps build address clusters using transaction graph analysis, detecting patterns like wallet reuse, shared relayer usage or interaction with the same honeypot factory contracts. By combining these tags with on-chain metadata such as function selectors, token pairings or deployment scripts, the analysts can proactively flag newly deployed contracts likely associated with known attacker infrastructure.

Tagging is not only retrospective - it also enables real-time threat detection. By subscribing to on-chain alerts tied to tagged addresses, investigators and protocol defenders can respond faster to new attacks, freeze funds where possible or warn users interacting with risky contracts. In this way, tagging serves both as a forensic asset and as a live operational defense mechanism.

1.8.3. Cross-referencing with Off-chain Data

While on-chain activity can reveal transaction patterns and wallet behaviors, meaningful attribution often requires connecting these patterns to off-chain identifiers such as usernames, code repositories, social profiles or infrastructure metadata. Cross-referencing smart contract activity with off-chain data sources enables investigators to uncover real-world identities or affiliations behind malicious operations, bridging the gap between blockchain pseudonymity and traditional compliance frameworks [16].

One common approach is analyzing GitHub commits and public repositories where smart contract developers may have published code. In several known cases, exploit contracts were found to be nearly identical to open-source code authored under identifiable GitHub

profiles. By comparing contract bytecode or function selectors with known repositories, investigators can sometimes establish a link between the deployed contract and its developer. This can be further strengthened by commit metadata (e.g., usernames, email addresses, timestamps) and behavioral traits across multiple repositories.

Another valuable signal is the use of Ethereum Name Service (ENS) names, which often appear in wallet metadata, contract ownership records or governance proposals. While some ENS names are generic, others include usernames, affiliations or references to social media profiles. Investigators can query ENS reverse records, domain ownership history and connected social handles to trace potential identities or teams [17]. Additionally, web hosting infrastructure, such as domains linked to scam projects or token launchpads, can be correlated with wallet addresses using WHOIS records, DNS lookups, or TLS certificate metadata.

Investigators may also analyze Telegram aliases, Discord handles, or email addresses exposed during phishing campaign takedowns, token presales or whitelist applications. Combining these external clues with transaction graph analysis enables the construction of a multi-layer attribution model that links on-chain behavior to a broader operational footprint. Cross-referencing off-chain data introduces challenges around privacy, legality, and attribution confidence, but when used responsibly and in conjunction with rigorous on-chain evidence, it significantly enhances the effectiveness of smart contract forensics and enables collaboration with regulators, exchanges, and law enforcement.

Attribution is the final and often most complex phase of smart contract forensics, requiring a careful blend of on-chain precision and off-chain intelligence. By tracing wallet interactions, tagging malicious addresses, and cross-referencing contract activity with external identifiers, investigators can move beyond technical diagnosis to establish accountability. These methods not only support compliance and legal action but also contribute to ecosystem-wide threat intelligence by exposing attacker infrastructure, funding patterns and behavioral signatures. As the line between decentralized anonymity and real-world responsibility continues to evolve, effective attribution strategies will remain essential to strengthening trust, transparency, and enforcement in the blockchain space.

1.9. Open Research Questions

As smart contract forensics matures, new architectural paradigms and cryptographic techniques are introducing both opportunities and challenges for investigators. While current forensic tooling is effective for standard EVM-based analysis, the rise of formal verification, zkApps, account abstraction, and Layer 2 rollups is reshaping the investigative landscape. These emerging technologies complicate visibility, execution modeling, and state reconstruction, prompting a need for new methods, standards, and research. This section outlines some of the most pressing open research questions in smart contract forensics, focusing on how to validate contract behavior post-exploit, detect malicious logic in privacy-enhanced or abstracted accounts, and analyze Layer 2 environments where traceability and access models differ significantly from Ethereum mainnet.

1.9.1. Formal Verification for Post-Exploit Analysis

Formal verification, which involves using mathematical proofs to guarantee that smart contracts function as intended, has traditionally been employed as a preventative measure during the development phase. However, its role in post-exploit forensics is gaining interest as investigators seek to reconstruct how an exploit occurred and whether a contract's behavior deviated from its intended logic. Unlike traditional debugging or symbolic execution, formal verification can offer provable guarantees or counterexamples for contract properties, which is especially valuable when source code is available, and the exploit path is non-obvious.

Post-exploit formal analysis typically involves re-establishing the intended invariants of the contract, such as balance preservation, access control restrictions, or correct payout logic, and determining whether these were violated under specific inputs or environmental conditions. Tools like Certora Prover, K Framework, Coq, and LEGOOS (Logic Engine for General On-chain Systems) are increasingly being explored to formalize such properties, particularly in financial protocols with complex accounting logic. When combined with transaction traces and state snapshots, formal methods can help confirm the exploit vector, identify whether it was enabled by subtle logic flaws, and provide a mathematically rigorous explanation of what went wrong.

Challenges remain, particularly in applying formal verification to contracts with opaque logic, dynamic upgradeability, or external dependencies such as price oracles. Moreover, verifying real-world contracts requires substantial expertise in formal specification languages and often cannot be fully automated. Still, as more projects adopt specification-driven development, forensic teams may increasingly use formal tools not only to prevent vulnerabilities but also to retroactively explain and prove exploit conditions, providing greater confidence in legal proceedings, audits, and community disclosures.

1.9.2. Detecting Malicious Logic in zkApps and Account Abstraction (ERC-4337)

The emergence of zkApps (zero-knowledge applications) and account abstraction via standards like ERC-4337 is reshaping how user interactions and application logic are encoded on-chain introducing major challenges for forensic visibility and exploit detection. Unlike traditional smart contracts, where logic and state changes are transparent and auditable, zkApps obscure internal computations through zero-knowledge proofs, and ERC-4337 introduces customizable user operations that can bypass standard transaction flows. This evolution enhances privacy and flexibility, but also opens the door for stealthy, non-transparent malicious logic.

In the context of zkApps, malicious behavior can be embedded within a circuit that executes privately, proving only that certain constraints were met without disclosing the input data or the execution path. This significantly hinders investigators from auditing control flow or state transitions through traditional bytecode analysis or transaction tracing. Forensic efforts in this setting may need to pivot toward analyzing proof verification logic, inspecting metadata and detecting behavioral anomalies, such as unexpected balance changes, state commitments, or interactions with external contracts following zero-knowledge executions.

ERC-4337 introduces another set of forensic challenges by decoupling the transaction model from traditional externally owned accounts (EOAs). Under this model, "user operations" are bundled and executed by bundlers and validated by custom smart contract logic defined by the user. Malicious actors can exploit this flexibility by implementing

invisible access controls, backdoors or conditionally malicious behavior that is only triggered by specific inputs or entry points. Investigators must now trace user operation flows, custom validation functions and paymaster logic, which may vary significantly across users and apps.

To address these challenges, new research is needed in proof metadata analysis, zk-circuit validation and automated reasoning over abstracted account models. Detecting malicious logic in zkApps and ERC-4337 wallets may ultimately rely less on static inspection and more on behavioral pattern analysis, simulation under varying inputs and collaborative intelligence gathering across privacy-preserving systems.

1.9.3. Forensics of Layer 2 Contracts

The growing adoption of Layer 2 (L2) solutions such as Arbitrum, Optimism and zkSync has introduced new scalability paradigms while also presenting fresh challenges for forensic investigations. These L2 platforms maintain Ethereum's trust model but shift execution to alternative environments like rollups or optimistic virtual machines. From a forensic perspective, this architectural change leads to delays in data availability, variations in transaction finality, and non-EVM-native execution traces, which complicate traditional Ethereum-based analysis methods.

In optimistic rollups such as Arbitrum and Optimism, transactions are processed off-chain and later submitted to Ethereum in the form of batched state roots or calldata. While this approach lowers gas fees, it obscures the execution of individual transactions unless a challenge or fraud proof is initiated. As a result, investigators must rely on L2-specific RPC endpoints, block explorers, and sequencer archives, which may not offer the same level of detail as mainnet tools like *debug_traceTransaction*. The centralized nature of sequencers also introduces trust assumptions that can impact the completeness and replayability of data.

Layer 2 forensics requires adapting conventional techniques like call tracing, storage inspection, and contract disassembly to accommodate compressed execution, delayed finality and protocol-specific transaction formats. Analysts may need to reconstruct rollup batches, decode compressed calldata or trace token bridges that obscure the flow of assets between L1 and L2. In incidents involving asset theft or malicious

contract deployments, investigations must span both layers to identify exploit paths that cross chain boundaries and involve varying settlement timelines.

As Layer 2 ecosystems continue to expand, the forensic community must prioritize the development of standardized L2 tracing tools, enhanced cross-layer event correlation, and real-time monitoring infrastructure. These advancements are critical to maintaining accountability, transparency and enforceability in the next generation of scalable decentralized applications.

1.10. Conclusions

As decentralized technologies continue to transform the global financial system, smart contracts have become both a powerful innovation and a potential avenue for abuse. This chapter has examined the forensic aspects of smart contract-based financial crime, covering everything from common scam tactics like rug pulls and pump-and-dump schemes to more complex exploits involving flash loans, oracle manipulation, and coordinated interactions across multiple contracts. Through case studies, tool-based strategies and in-depth technical analysis, we have demonstrated how forensic investigators can piece together intricate exploit chains, interpret contract behavior, and track malicious activity with growing accuracy.

The investigative process begins with a strong technical foundation, including access to nodes, historical state reconstruction, and low-level EVM analysis. Static and dynamic approaches work in tandem, offering insights into both the structure and execution of smart contracts. Signature recognition, heuristic models, and automated monitoring systems now enable detection across vast networks of contracts. Still, even the most advanced tools must ultimately connect with attribution and compliance efforts, where tracing wallet ownership, analyzing contract origins and linking on-chain activity to off-chain identities help uncover attacker infrastructure and prevent repeat offenses.

The cutting edge of this field brings forward several open research challenges. Areas like formal verification, zero-knowledge applications, account abstraction, and Layer 2 technologies call for new ways to observe, simulate, and extract forensic evidence. These innovations are

reshaping the tools and mindsets needed to ensure accountability in decentralized environments.

In the end, smart contract forensics is not only about identifying what went wrong, but also about rebuilding trust in systems that are designed to operate without central control. As DeFi and Web3 continue to expand, the need for transparent, reliable and adaptable forensic frameworks will only become more urgent. By equipping researchers, developers and compliance professionals with robust investigative methods, we take a vital step toward safeguarding the future of programmable finance.

Acknowledgements

This work has been partially supported by (1) the project *virtuaLedger* [18], (2) the project RoNaQCI, part of EuroQCI, DIGITAL-2021-QCI-01-DEPLOY-NATIONAL, 101091562, (3) the project "Romanian Hub for Artificial Intelligence - HRIA", Smart Growth, Digitization and Financial Instruments Program, 2021-2027, MySMIS no. 334906.

References

[1]. BBC News, Squid Game crypto token collapses: Scammers make off with millions, https://www.bbc.com/news/business-59129466

[2]. M. Sanyal, $31 million siphoned off Binance Smart Chain-based DeFi project Meerkat, Cryptonary, 2021, https://cryptonary.com/31-million-siphoned-off-binance-smart-chain-based-defi-project-meerkat/

[3]. CoinMarketCap, UFO Gaming price today, UFO to USD live, marketcap and chart, https://coinmarketcap.com/currencies/ufotoken/

[4]. F. Cernera, M. La Morgia, G. Sperlí, C. Picariello, et al., Token spammers, rug pulls, and sniperbots: An analysis of the ecosystem of tokens in Ethereum and the Binance Smart Chain (BNB), in *Proceedings of the International Conference on Computer Communication and Networks (ICCCN'23)*, 2023, pp. 1-10.

[5]. SlowMist, Analysis of the Uniswap phishing attack, Medium, 2022, https://slowmist.medium.com/analysis-of-the-uniswap-phishing-attack-3026bb49f65

[6]. K. Qin, L. Zhou, B. Livshits, A. Gervais, Attacking the DeFi ecosystem with flash loans for fun and profit, in *Proceedings of the Financial Cryptography and Data Security Conference (FC'21)*, 2021, pp. 3-32.

[7]. P. J. P. Guzman, Harvest Finance: $24M attack triggers $570M 'bank run' in latest DeFi exploit, CoinDesk, 2020, https://www.coindesk.com/

tech/2020/10/26/harvest-finance-24m-attack-triggers-570m-bank-run-in-latest-defi-exploit

[8]. C. F. Torres, M. Steichen, R. State, The art of the scam: Demystifying honeypots in Ethereum smart contracts, in *Proceedings of the 28th USENIX Security Symposium (USENIX Security'19)*, 2019, pp. 1591-1607.

[9]. OSL Academy, What are reflection tokens in crypto & how do they work?, 2022, https://osl.com/academy/what-are-reflection-tokens-in-crypto-how-do-they-work/

[10]. J. Feist, G. Grieco, A. Groce, Slither: A static analysis framework for smart contracts, in *Proceedings of the IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB'19)*, 2019, pp. 8-15.

[11]. TRM Labs, Solana Wormhole compromise: 120k wrapped ETH stolen, 2022, https://www.trmlabs.com/resources/blog/solana-wormhole-compromise-120k-stolen-eth

[12]. Ethereum, Web3.py: A Python library for interacting with Ethereum, https://github.com/ethereum/web3.py

[13]. Eth-Brownie, Brownie: A Python-based development framework for Ethereum smart contracts, https://eth-brownie.readthedocs.io/en/stable/

[14]. Nomic Foundation, Hardhat: Ethereum development environment for professionals, https://hardhat.org/

[15]. M. Wang, A. Chow, F. Liu, Cryptocurrency address clustering and labeling, *arXiv preprint*, 2020, arXiv:2003.13399.

[16]. Solidus Labs, Combining on-chain and off-chain analysis: A primer, https://www.soliduslabs.com/post/off-chain-and-on-chain-analysis

[17]. Z. West, ENS reverse records: What they are & how to create them, Medium, 2021, https://medium.com/geekculture/ens-reverse-records-what-they-are-how-to-create-them-9ff7907d0aed

[18]. VirtuaLedger, VirtuaLedger project homepage, https://virtualedger.com

Chapter 2

Asymmetric Risk Allocations in Cryptocurrency Portfolios: A VaR/ES-GARCH-Copula Approach

Vahidin Jeleskovic, Mirko Meloni, Zahid Irshad Younas and Mamdouh Abdulaziz Saleh Al-Faryan

2.1. Introduction

Considering cryptocurrency as a real asset is still a controversial topic [1]. However, in recent years the cryptocurrency market has observed a tremendous growth and quick rise in its market capitalization from $19 billion in February 2017 to $800 billion in December 2017. Due to this explosive growth, cryptocurrencies are considered to be a new investment avenue [2-5]. Economists believe this growth to be a bubble based on sentiments and price fluctuations [6]. A shift in these factors coupled with a change in market trends have subsequently resulted in a drastic drop in bitcoin prices [7, 8]. According to previous research, the lack of intrinsic value, an unregulated cryptocurrency market and major cryptocurrency holdings by market participants are significant volatility factors in cryptocurrencies [9].

This study aims to provide market participants with an evaluation of the daily risk of six widely-traded cryptocurrencies (Bitcoin, Binance, Ethereum, Litecoin, Ripple and Eos) based on the maximum possible short-term loss, i.e., 30 days. This kind of risk is also called "tail risk", i.e., asymmetrical distributions which are more probable to have extreme results. The traditional standard deviation, even if easily applied, suffers from a variety of limitations which are not in line with the main findings on the cryptocurrency market. In fact, [10-12] found this type

Vahidin Jeleskovic
Humboldt-Universität zu Berlin, Germany

of market to have the following characteristics: fat-tails and skewness, autocorrelation, long memory in volatility, volatility heteroskedasticity and clustering and leverage effect.

[1] All these features are not supposed to be correctly captured by standard deviation which is only a model for elliptical distributions.[2] Moreover, standard deviation expresses the variability resulting from both downside and upside movements of returns, thus it is not able to provide investors with a concrete measure of losses [13]. Considering the volatility in cryptocurrency prices, this analysis is conducted on a short-term framework, estimating the tail risk of daily volatility models optimized by the inclusion of high-frequency data (namely 1m, 5m, 15m, 30m and 1h). An intra-day perspective is justified by the fact that a higher frequency is rationally able to carry a higher grade of information which is more accurate in predicting the volatility dynamic, especially when it tends to change significantly and quickly [14]. Given these circumstances, a model which just relies on low frequency data will return a dynamic which is too slow with respect to the dynamic actually present in the market.

This study addresses the risk of loss from two complementary perspectives. First, it examines vertical risk, defined as the risk associated with a single asset across different investment time horizons. Second, it analyzes horizontal risk, referring to the risk arising from co-movements with other cryptocurrencies.

To capture vertical risk, we employ a novel EGARCH model combined with realized volatility, reflecting the widespread academic and practical relevance of GARCH-type models in financial risk modeling. For horizontal risk, we utilize a copula-based approach, leveraging the ability of copulas to model the joint distribution of multiple univariate marginal distributions, particularly under non-linear and asymmetric dependencies. Finally, we integrate both risk perspectives to illustrate their combined usefulness in the context of portfolio optimization under extreme value uncertainty. In doing so, we construct an equally weighted cryptocurrency portfolio, ensuring that each asset contributes equally in terms of return significance.

[1] Leverage effect denotes a different volatility response to different type of shocks.
[2] Distributions with symmetrical frameworks and linear relationships between variables.

A buy and hold strategy are applied to provide a more objective comparison while preventing managerial bias that could affect the portfolio risk-return profile.[1] The marginal risk contribution of each asset is computed by using the Value at Risk (VaR) and Expected Shortfall (ES). They measure how much each crypto currency affects the overall tail risk given the contribution of the other assets within the same portfolio. Both VaR and ES are computed with a Monte Carlo simulation which sets up the use of copula functions and univariate ARMA-real GARCH models [15, 16]. The contribution of this research is a clear understanding of all the risk conditions and diversification opportunities within the cryptocurrency market before entering it. This work is pertinent to banks, hedge funds and individual investors who want to understand the risk opportunities related to the cryptocurrency market in order to optimize/diversify their investments. Furthermore, the results of this study could be of use to entities forced to invest in risky assets in order to accumulate a certain amount of capital by a future date, e.g., pension funds. The before-mentioned investors might prefer to evaluate a risk measure that indicates the potential loss resulting from a long position rather than a measure which just expresses the variability around the expected performance, i.e., the standard deviation. Moreover, at the same time, they might want to know which cryptocurrency is the most suitable considering a defined time horizon. Marginal allocations can provide them with information about the level of risk from one asset compared to another. The proposed model can be used for simulation and risk management purposes, but also for asset allocation frameworks.

The previous literature deals with either a few assets or with interactions between cryptocurrencies and other asset classes assuming often basic models. For example, [17] analysed the co-movements between Bitcoin and Ethereum, modelling their dependence with a bivariate BEKK model. [18] and [19] investigate the relationship between Bitcoin and respectively three commodity indices and indices representative of all the other asset classes (e.g. stocks, bonds, real estate) with ARMA and DCC-GARCH models (1,1). [20] applied a DCC-GARCH methodology (1,1) to model the joint dynamic of several cryptocurrencies with stocks, bonds and commodity prices from 2014 to 2018. Considering the different approaches just mentioned, the inexistence of a common practice with respect to multivariate models can be seen even though [20]

[1] Once the starting weights are set, they are allowed to change over time depending on the new price level.

found a DCC model to outperform the basic BEKK. This is due to the fact that dynamic models are able to capture the potential change in correlations resulting from both negative and positive market shocks. As for the employment of copulas, [21] used vine-copulas to model the dependence between Bitcoin, Dash, Ether, Litecoin, Ripple and Stellar with data from 2015 to 2018. Furthermore, they also considered an AR (1) specification for the mean of each cryptocurrency and found that Archimedean copulas perform better than the Gaussian copula. Similar evidence (with a predominance of two specific types of Archimedean copulas) was found by [22] which used vine-copulas to study the co-movements among Bitcoin, Ether, Litecoin and Ripple from 2015 to 2018.

The approach proposed here, involves the usage of a given distribution for the GARCH computation and the subsequent fitting of alternative distributions on the residuals obtained. The distribution chosen for the first step is the Generlazed Error Distribution – GED [23]. This choice was due to the fact that, in this way, it was possible to model residuals given the presence of fat tails, which characterize the crypto-market as demonstrated by the already-mentioned [10, 11 and 12] papers. The GED distribution is then retained as a good proxy of the real distribution. Recent review about modeling of volatility, and thus the risk, of cryptocurrencies is given by [24]. The authors highlight that, despite numerous studies on volatility clustering, significant research gaps remain. Only few studies examine the structure of volatility clustering in terms whether it exhibits nonlinear and asymmetric behavior. [25] explore portfolio optimization involving cryptocurrencies by comparing traditional Markowitz models with GARCH-Copula and GARCH-Vine-Copula approaches. Their results suggest that mixed portfolios containing both cryptocurrencies and conventional assets can achieve higher Sharpe ratios and more stable performance. [26] employ a GARCH-EVT-Copula framework to analyze dependency structures within the cryptocurrency market and to quantify portfolio risk. Their findings reveal a strong positive dependence between Bitcoin and Ethereum, suggesting limited diversification benefits among major digital assets. [27] apply an eGARCH-EVT-Gumbel-Copula model to investigate the dependency between Bitcoin and the South African Rand. Their analysis focuses on quantifying the risk of a balanced two-asset portfolio and concludes that the two assets are largely independent, offering potential diversification benefits. [28] investigates dynamic spillover effects in the cryptocurrency market before and after the COVID-19 pandemic using an asymmetric breakpoint approach. The

study highlights changes in risk transmission patterns across digital assets in response to extreme market events.

In contrast, this chapter offers a novel contribution by integrating copula functions with the Marginal Expected Shortfall (MES) framework to isolate and quantify asset-level risk contributions under asymmetric tail dependencies, incorporating both vertical and horizontal risk perspectives. This approach not only provides theoretical insights into dependence structures, but also delivers a practical and granular tool for risk attribution, particularly valuable in volatile, non-Gaussian markets such as cryptocurrencies. It fills a methodological gap in the literature by shifting the focus from aggregate portfolio risk to its nonlinear, asymmetric marginal composition across distinct crypto assets.

Model selection, for both the GARCH specification (including lag structure) and the copula family, is based on a rolling-window estimation framework, implemented through walk-forward cross-validation. This approach reduces the risk of overfitting by relying on predictive consistency over time, rather than on static in-sample performance. Several window sizes were tested during model development, and the configuration demonstrating the most stable and robust behavior across cryptocurrencies was ultimately chosen. Since this chapter focuses on marginal risk attribution rather than forecast evaluation, the exact window length is not central to the core contribution. However, the rolling-window procedure ensures that the model remains adaptive to changing market conditions.

The decision to employ a GARCH-type model, specifically the EGARCH specification combined with realized volatility (RV), is driven by the need to model vertical risk, and so the time-dependent risk of a single asset across different investment horizons. EGARCH captures asymmetric reactions to positive and negative shocks, while RV incorporates high-frequency volatility dynamics, enhancing precision in capturing tail risks and clustering effects that are typical for cryptocurrencies.

To capture horizontal risk (i.e., the dependence structure across different cryptocurrencies) we employ copula functions. Copulas allow flexible, non-linear modeling of joint distributions and tail dependencies, making them especially useful in risk settings where assets exhibit asymmetric co-movement during stress periods.

The methodological innovation of this chapter lies in the combination of these two components: vertical risk captured through EGARCH–RV, and horizontal risk through copula-based dependence modeling. Additionally, the use of an equally weighted portfolio of six major cryptocurrencies facilitates transparent and symmetric marginal risk attribution. This design enables the derivation of marginal Expected Shortfall (MES) that reflects both time-scale sensitivity and asset-specific tail dependencies. To our knowledge, such a dual-layered modeling approach has not been previously applied in this form to multi-cryptocurrency portfolios, making this a novel contribution to both financial risk management and crypto asset modeling.

Due to the recent crises caused by the COVID-19 pandemic and the war in Ukraine, we have chosen to use data from before the outbreak of COVID-19. This approach allows us to avoid the impact of some disruptive events such as the Ethereum merge or the SEC lawsuit against Ripple. Consequently, our dataset is more representative of the normal state of economies.

The rest of the chapter is organized as follows. Section 2.2 outlines the methodology. Section 2.3 presents the data and descriptive statistics. Section 2.4 discusses the results and their interpretation. Section 2.5 provides a critical discussion and outlook for future research. Finally, Section 2.6 concludes the chapter.

2.2. Methodology

2.2.1. Var/ ES Term

The two asymmetric risk measures used for this analysis are Value at Risk (VaR) and Expected Shortfall (ES). In particular, VaR quantifies the maximum achievable loss given a certain level of confidence and a specified time horizon [29 ,30]. On the other hand, ES measures the expected return given that the return is more extreme than VaR. It is important to note that ES has a higher informative content since two distributions can have the same VaR, but a different probability to reach extremes quantiles.

To determine how cryptocurrencies affect the VaR/ES of a portfolio, it is necessary to build a dynamic model which accounts for how the two indices vary over different time horizons. This instrument is called

Term structure and can be built by crossing VaR or ES with the relative maturity.

For Term structure, a Monte Carlo simulation is chosen out of all the possible model due to the fact that simulation methods are generally highly accurate as they tend to smooth forecasts, increasing the reliability of long-term predictions.[1]

Returns are simulated using a location scale model:

$$R_{t+1} = \mu_{t+1} + \sigma_{t+1} z_{t+1} \quad (2.1)$$

Here, z_{t+1} is an identically independently distributed random variable and both μ_{t+1} and σ_{t+1} are conditional mean and conditional variance, respectively, which serve as the basis for modeling the return distribution. They are not directly observable so that their quantities must be estimated or forecasted based on historical data contained in the available information set to the given timepoint. Thus, they are considered as conditional expectations to the timepoint *t+1* based on the information set contained up to the time point *t*. Considering the heteroskedasticity phenomenon and volatility clustering over the time, it is not convenient to work with unconditional distributions (where moments remain fixed over time [30]. This model was chosen for two main reasons: 1) It is easy to compute; 2) It is particularly useful if combined with a univariate ARMA-GARCH model, which allows both the conditional volatility and the mean to be used as inputs; 3) It is possible to model the shape and the kurtosis of the return series by modifying the distribution of z.

2.2.2. Conditional Mean and Conditional Volatility

The Generalized Autoregressive Conditional Heteroskedasticity (GARCH) class of models became a standard framework for modelling the volatility of financial assets. [15] presents first ARCH(p) models which are generalized by [16] to the GARCH (p,q) allowing for greater

[1] The simplest model is the non-parametric one which assumes that historical returns are perfectively representative of the future ones. The semi-parametric approaches are aimed at estimating the parameters of a portion of the return distribution. Full-parametric models are based on the whole return distribution and make it possible to compute risk measures through closed formula. Simulation methods require the most computation but can lead to the most accurate results [30].

flexibility in capturing the dynamics of volatility. From a univariate point of view, it is necessary to create a model which is able to predict both the conditional mean and the conditional volatility for all the cryptocurrencies within the sample portfolio. While our framework will integrat VaR/ES with GARCH and copula models, the core methodological innovation lies in the use of copulas for decomposing asymmetric marginal risk contributions. GARCH is employed not as an end in itself, but as a reliable and well-established volatility estimator that supports the copula-based dependence structure. Its widespread use in financial econometrics makes it a transparent and replicable starting point for our framework. More sophisticated volatility models, such as FIGARCH or ARFIMA, are acknowledged but deliberately left for future work to avoid over-complicating the baseline structure in this first empirical application.

Hence, this study focuses on the ARMA-real GARCH model proposed by [31], that is not only able to predict daily volatility by exploiting intra-day information, but is also able to include the leverage effect:

$$\begin{aligned} R_t &= \mu_t + \sigma_t z_t, \qquad z_t \sim i.i.d(0,1) \\ \log(\sigma_t^2) &= \omega + \sum_{k=1}^{q} \alpha_k \log(r_{t-k}) + \sum_{k=1}^{p} \beta_k \log\left(\sigma_{t-k}^2\right) \\ \log(r_t) &= \xi + \delta \log(\sigma_t^2) + \tau(z_t) + u_t, \qquad u_t \sim N(0,\lambda) \\ \tau(z_t) &= \eta_1 z_t + \eta_2 (z_t^2 - 1) \end{aligned} \tag{2.2}$$

The parameters $\alpha_k, \beta_k, \delta, \eta_1$ and η_2 are estimated by means of a suitable information criterion (e.g., AIC or BIC). After identifying the optimal GARCH specification, the analysis proceeds with portfolio optimization employing the copula framework. We adopt this sequential procedure rather than jointly optimizing the GARCH and portfolio parameters, since the latter leads to a highly nonlinear and analytically opaque optimization problem, further complicated by nonlinearities in the data and numerical instability in high-dimensional settings.

As it can be noticed by the first line in Equation 2.2, a return in time "t" follows the location-scale model, while the conditional volatility is treated in logarithmic terms (second line). Here, the current state of innovation (or market shock) is not measured by the squared daily return (like in the standard GARCH), but by the log of r_{t-k}, where r_{t-k} is a measure of realized volatility. The realized volatility (which can be

denoted by "RV") of a generic day t, given regular-framed intra-day data with the frequency m, can easily be computed as shown in Equation 2.3:[1]

$$RV_t = \sum_{m=1}^{M} R_{m,t}^2 \,, \tag{2.3}$$

where $R_{m,t}$ is the return of the sub-period m in day t and M is the total number of sub-periods for that day.

For this research, 1m, 5 m, 15 m, 30 m and hourly data related to the period from 31 May 2018 to 30 July 2019 are used to create different series for the realized volatility, to be exploited in an anchored walk forward cross validation procedure. All the GARCH are estimated considering the GED distribution as starting assumption (since the distribution for standardized quantiles is fitted in a second step, it is not convenient to try several variants) and a maximum order of (5,5) for both the mean and volatility processes (to optimized during the process). GED is retained to be able to return less-biased GARCH residuals (on which the real fitting procedure is later carried out), modelling the parameter that describe fat-tails. The walk forward procedure is implemented generating 5 different series of training and validation sets (as shown in Table 2.1).

Table 2.1. Training and test sets for WF cross validation.

Training		**Validation**	
Start	**End**	**Start**	**End**
31/05/2018	30/01/2019	31/01/2019	02/03/2019
31/05/2018	01/03/2019	02/03/2019	01/04/2019
31/05/2018	31/03/2019	01/04/2019	01/05/2019
31/05/2018	30/04/2019	01/05/2019	31/05/2019
31/05/2018	30/05/2019	31/05/2019	30/06/2019

The conditional volatility forecasted through the models fitted on the training sets is compared with the one observable in the related validation set though the RMSE loss function:

[1] See [30] or [32].

$$RMSE = \sqrt{\frac{1}{n_{val}} \sum_{i=1}^{n_{val}} \left(vol - E(vol)\right)^2} \quad (2.4)$$

The walk forward process implies that the RMSE values resulted from each pair of training and validation set shown in the previous Table need to be averaged, so that the best model could be considered the one minimizing the mean of all the past value of the loss function. This model will be optimized considering the final training set (31/05/2018 – 30/06/2019) to generate the forecasts needed to answer the research question. This approach is aimed at reducing the overfitting risk (potentially resulting from an in-sample selection criteria).

2.2.3. Copula Function and Multivariate Modelling

On a multivariate point of view, it is necessary to estimate a dependence structure between the best-fitting cryptocurrency distributions. This research exploits the informative content of copulas[1]. The combination between different marginal univariate distributions and the multivariate copula makes it possible to obtain new distributions with respect to the traditional ones [33, 34]. Inputs of copulas are represented by probabilities, which are uniformly distributed, and not by quantiles, thus they are not required to be normally distributed even setting a particular copula (Simard & Rémillard, 2015).

[38] compares several types of copulas– Normal, t-Student, Gumbel, Clayton, Frank [39], Joe [40] – in order to assess which, one returns the best out-of-sample results (in terms of RMSE function between the predicted returns of each asset resulting from 10,000 simulations for each day included in the validation set and the effective ones.[2]

[1] Copula functions are cumulative probability functions which link any given set of marginal distributions to construct a joint dependence structure [30, 35-37].

[2] There are several types of copula families, but the ones considered for this research are (see Appendix A.2 for mathematical specifications): 1) Elliptical copulas and 2) Archimedean copulas. The first family is made up of Normal and t-Student copulas, which, being symmetrical, might not model in a proper way the tail of the multivariate distribution between assets. Moreover, elliptical distributions assume a linear dependence between two or more objects, so that the unknown parameters can be represented by Pearson's correlations. This means that it could be hard to reach a convergence in a high dimensional multivariate framework (in presence of a high

$$RMSE_c = \sqrt{\sum_{asset=1}^{N_{asset}} \sum_{i=1}^{n_{val}} \left(r_{ai} - E(r_{ai})\right)^2} \tag{2.5}$$

r_{ai} is the return of the a[th] asset within the portfolio of the j[th] day included in the validation set.

The procedure follows a walk forward approach, like the GARCH models. After having fixed all the pairs of training and validation sets (already shown in Table 2.1), the RMSE resulting from each of them is averaged to the ones coming from the others. The best copula is the one which minimizes the final mean.

Please note that all these models are estimated under the hypothesis of constant conditional correlations, i.e. assuming that the correlation between assets does not change over time. Copulas are then estimated just once (this assumption can be justified by the fact that this research deals with the very short term, but it was also due to limitations on the computational power exploitable). The models were computed through the R package *copula* [41].

2.2.4. Marginal Risk Allocation: Euler and Kernel Methods

The research question of this study is related to how much each cryptocurrency affects the overall risk of a loss. To provide investors with an answer, it is necessary to compute marginal risk allocations:[1]

$$Risk\ Mesaure_p = \sum_{i=1}^{N} Marginal\ Risk_i \tag{2.6}$$

number of assets in the portfolio of interest). Archimedean copulas, on the other hand, can not only treat the symmetry differently according to the type of function chosen, but also simplify the optimization as they depend on a unique parameter λ (independently from the number of assets), which measures the strength of the dependence between series (not necessarily linear). This property, however, could also make them inappropriate in case of very high dimensions as the quality of final approximation could not be sufficiently satisfactory [42].

[1] This is also called additivity rule [43].

In the literature, the main framework used for this purpose is the Euler allocation principle. It states that if the risk measure is a continuously differentiable function of the portfolio weights, the total risk can be decomposed as follows:[1]

$$Risk\ Mesaure(w) = \sum_{i=1}^{N} w_i \frac{\partial Risk\ Mesaure}{\partial w_i} \tag{2.7}$$

Here, $w_i \frac{\partial Risk\ Mesaure}{\partial w_i}$ represents the marginal risk contribution of the i^{th} asset. As can be seen, this approach relies on computing the partial derivatives of the risk measure with respect to each portfolio weight. In case of standard deviation, they are quite easy to compute, but, in case of VaR and ES (calculated through a discrete simulation approach), it is impossible to obtain reliable results. First of all, a discrete approach does not guarantee a continuous function, so that derivatives should be approximated by discrete differences. Discrete differences, in turn, do not guarantee the respect of the additivity rule. Moreover, results of simulated discrete calculations are too volatile, meaning that a re-simulation could change them significantly. Therefore, Eulerian marginal allocations cannot be used in combination with Monte Carlo simulations. Another approach could be taking the return (profit or loss) in correspondence to the portfolio VaR and ES as marginal allocations for each cryptocurrency:[2]

$$mVaR_i = w_i E(PL_i | PL_p = VaR_p) \tag{2.8}$$

$$mES_i = w_i E(PL_i | PL_p \leq VaR_p) \tag{2.9}$$

Considering an equally-weighted portfolio, this would guarantee the additivity rule for both the risk measures, but results would be still too volatile with respect to different simulations. In this sense, [45], as well as their extended kernel-based methodology presented in 2016, proposed approaches aimed at smoothing marginal risk estimates and improving their numerical stability. Kernel functions are non-parametric tools used to estimate a conditional expectation not knowing its functional form [47]. [45] suggests using the Nadaraya-Watson kernel regression

[1] See [44].
[2] This result can be obtained by extending the Euler approach [43].

(considering a locally constant conditional expectation)[1] and starting from the loss function shown in Equation 2.10. The final marginal VaR estimator is the value of the unknown conditional expectation (A) which minimized the loss function given here by the following equation 2.11:

$$Loss(A) = \sum_{j=1}^{N} \left(PL_i^j - A\right)^2 K\left(\frac{PL_p^j - VaR_p}{h}\right) \tag{2.10}$$

$$mVaR_i = \frac{\sum_{j=1}^{N} K\left(\frac{PL_p^j - VaR}{h}\right) PL_i^j}{\sum_{j=1}^{N} K\left(\frac{PL_p^j - VaR}{h}\right)} \tag{2.11}$$

N is the number of simulations, K(./h) is the kernel function, PL_p^j the absolute portfolio profit/loss related to the j-th simulation, $PL_p^j$the absolute ith cryptocurrency profit/loss related to jth simulation and h the smoothing parameter of the kernel chosen (equal to $2{,}275\sigma N^{-\left(\frac{1}{5}\right)}$). As it can be noticed, Equation 2.10 is a kernel-weighted average since PL_{i} has a weight equal to K, which is higher as the j-th simulation returns a profit profit/loss near the portfolio VaR. This kind of function has a smoothing power basing on the weighted average among different simulations, but it does not satisfy the additivity rule (depending on h). It is then necessary to apply a rescaling factor, equal to the portfolio Value at Risk as follows:

$$mVaR_i = VaR_p \frac{\sum_{j=1}^{N} K\left(\frac{PL_p^j - VaR}{h}\right) PL_i^j}{\sum_{j=1}^{N} K\left(\frac{PL_p^j - VaR}{h}\right)} \tag{2.12}$$

[1] Where *h* is a smoothing parameter A kernel with subscript *h* is called "scaled kernel" and is defined as Kh(x) = 1/h K(x/h). Please note that the choice of *h* can strongly influence the final estimation (the lower *h* is, the less smoothed the resulting kernel distribution might be).

In light of the above, marginal Expected Shortfalls can be computed as the weighted average of all the profit/loss values in correspondence of scenarios leading to a more extreme loss than the portfolio VaR:

$$mES_i(\alpha) = \sum_{j=1}^{k\sim VaR_p} \frac{w_j E(PL_i | PL_p = PL_p^j)}{1-\alpha} \tag{2.13}$$

Substituting $(PL_i|PL_p = PL_p^j)$ with the kernel-averaged $mVaR_i$ (considering at each *j* the j-th more extreme scenario as portfolio VaR), it is possible to obtain the following result:

$$mES_i(\alpha) = \sum_{j=1}^{k\sim VaR_p} \frac{w_j mVaR_i}{1-\alpha} \tag{2.14}$$

[45], in order to keep the rescaling factor from affecting the statistical estimations, proposed another approach, in which the conditional expectation is no longer considered locally constant, but locally linear with respect to the portfolio profit/loss function:

$$Loss(A,B) = \sum_{j=1}^{N} \left(PL_i^j - \left(A + BPL_p^j\right)\right)^2 K\left(\frac{PL_p^j - VaR_p}{h}\right) \tag{2.15}$$

Once computed A and B which minimize the loss function [47], marginal VaR can be computed as shown in Equation 2.16 and marginal ES as shown in Equation 2.14.

$$mVaR_i = A_i^* + B_i^* VaR_p \tag{2.16}$$

$$A_i^* = \sum_{j=1}^{N} \frac{K_j S_2 - K_j PL_p^j S_1}{S_0 S_2 - S_1^2} PL_i^j \tag{2.17}$$

$$B_i^* = \sum_{j=1}^{N} \frac{K_j PL_p^j S_0 - K_j S_1}{S_0 S_2 - S_1^2} PL_i^j \tag{2.18}$$

$$K_j = K\left(\frac{PL_p^j - VaR_p}{h}\right) \tag{2.19}$$

$$S_0 = \sum_{j=1}^{N} K_j \tag{2.20}$$

$$S_1 = \sum_{j=1}^{N} K_j PL_p^j \tag{2.21}$$

$$S_2 = \sum_{j=1}^{N} K_j \left(PL_p^j\right)^2 \tag{2.22}$$

Considering Equation 2.16 and that the sum of the profit/loss of all the cryptocurrencies for the j[th] simulation is equal to the j[th] portfolio profit/loss by definition, it can be easily verified that:

$$\begin{aligned} \sum_{i=1} A_i &= 0 \\ \sum_{i=1} B_i &= 1 \end{aligned} \tag{2.23}$$

Combining Equation 2.23 with Equation 2.16, it is possible to verify the additivity rule on VaR (and on ES as extension; Equation 2.24) without the usage of rescaling factors.

$$\sum_{i=1} mVaR_i = VaR_p \tag{2.24}$$

The two approaches proposed by the extended paper version of [45] in 2016 which considered the Gaussian kernel in their estimations, are used and compared in terms of variability of results:

$$K(x,h) = \frac{1}{\sqrt{2\pi}h} e^{-\frac{1}{2}\left(\frac{x}{h}\right)^2} \tag{2.25}$$

The gaussian kernel with a zero mean, designed to model the distribution of the difference between the portfolio P/L and its VaR, is particularly suitable as it is applied to estimate the weights to be assigned to each simulation. When the two measures are equal, the difference between them is equal to 0, in correspondence to which the gaussian curve reaches its maximum value. The weights tend to decrease when simulations are far from the portfolio VaR.

The above-illustrated methodology is structured as to generate 5 different (one per frame) contributions for each cryptocurrency and with respect to VaR and ES at both 99 % and 95 % level of confidence.

2.3. Data and Descriptive Statistics

The current study takes into considerations six cryptocurrencies (Bitcoin, Binance, Ethereum, Litecoin, Ripple and Eos) chosen among the most traded ones on the Binance platform considering the 24h volumes. For the analysis, 1m, 5 m, 15 m, 30 m, 1 h and 1 d data (price against Tether, USDT)[1] – from 31 May 2018 to 30 July 2019 were exploited in order to assess the incidence of the time frame on the final marginal risk allocations (retrieved through a *Binance* API). Data were first analysed to detect the presence of repetitions or missing values (and to be filled through an interpolation method), but they did not show irregularities. The presence and the relevance of extreme returns were instead studied considering several alternatives of fat-tailed distributions. Returns from 31 May 2018 to 30 June 2019 was chosen to represent the training set; July 2019 was chosen as test set, through which carry out the forward-looking backtesting and evaluate the forecasts goodness of fitting. This period of time is considered to be sufficiently representative of the short trend (avoiding taking into consideration the "bubble crash" of the early 2018) and consistent with this kind of analysis. Assuming a portfolio starting with an investment of 100€ in each individual cryptocurrency (Fig. 2.1), it can be noticed that Bitcoin, Ethereum, Litecoin, Ripple and Eos have a similar trend; Binance, on the other hand, seems to diverge showing a stronger increasing positive trend starting from February 2019.

[1] According to cryptocompare.com (last access: 02-08-2019), the trading volumes for all the pairs with Tether are significantly higher than the pairs with respect to fiat currencies (e.g. US Dollar). Moreover, the usage of stablecoins as base currency has already become a common practice for crypto-traders as their stability can help to hedge against volatile market phases.

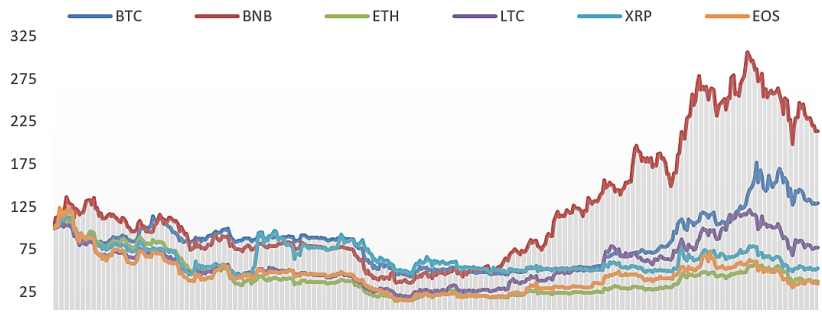

Fig. 2.1. Daily cryptocurrency prices (indexed in percentage terms), from 31 May 2018 to 30 June 2019.

To have a clearer vision, Table 2.2 shows the unconditional descriptive statistics of the training set for each cryptocurrency (31 May 2018 – 30 June 2019).

Table 2.2. The unconditional descriptive statistics for each cryptocurrency in the training period (31 May 2018 – 30 June 2019). The results reveal pronounced differences in distributional properties, particularly in terms of skewness, kurtosis, and overall variability. All assets deviate significantly from normality, as confirmed by the Jarque–Bera test [46]. This provides strong justification for employing copula-based risk measures, which capture dependence structures beyond what standard deviation alone can reflect.

Stats	**BTC**	**BNB**	**ETH**	**LTC**	**XRP**	**EOS**
min	-14.50%	-21.44%	-22.25%	-16.12%	-19.63%	-22.98%
1st q	-0.98%	-2.15%	-2.29%	-2.49%	-2.38%	-2.75%
median	0.22%	0.25%	-0.11%	-0.36%	-0.11%	-0.16%
mean	0.10%	0.24%	-0.16%	0.01%	-0.10%	-0.18%
3rd q	1.57%	2.94%	2.11%	2.79%	1.88%	2.42%
max	15.87%	17.77%	17.35%	25.75%	31.60%	21.90%
dev st	3.51%	4.94%	4.97%	5.10%	5.13%	5.96%
skew	-0.17	-0.21	-0.35	0.45	1.03	-0.04
kurt	6.55	4.83	5.59	6.04	9.94	5.46
J-B	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

It is possible to note how the return distributions tend to present different characteristics (despite the similarity shown in Fig. 2.1). This fact makes it possible to assume a different risk contribution in terms of potential losses. Going into more detail, all the cryptocurrencies (except Bitcoin

and Binance) have a negative median and mean (no Litecoin). In terms of variability, Eos, Ripple and Litecoin present a higher standard deviation. Bitcoin, on the other hand, seems to be the less risky asset in terms of erratic movement around the expected value, with the lowest standard deviation and the lowest variability range (max-min). As for the other measures, all the distributions are not symmetrical. Bitcoin, Binance, Ethereum and Eos show a negative skewness (in presence of which there is a longest tail for the negative returns). Litecoin and Ripple, contrary, are positively skewed (with a longer right tail for extreme quantiles). Given the skewness, and that each coin has a kurtosis higher than 3, the Jarque-Bera test (last row) returns an almost null p-value for all the assets. As consequence, none of cryptocurrencies can be considered normally distributed. This result justifies the usage of a copula-based simulated VaR and ES as measures of risk. In presence of shape and fat tails, indeed, standard deviation cannot be considered the only relevant risk factor.

2.4. Empirical Results, Interpretation, and Discussion

To assess the suitability of nonlinear modeling, we conducted Ljung–Box tests [47] and ARCH LM tests [15] on returns, squared returns, and realized volatility series for each cryptocurrency across various time frequencies. The results, summarized in Table 2.3, reveal statistically significant serial correlation and heteroskedasticity, especially at higher-frequency (intraday) data. In most cases, the null hypothesis of linearity can be confidently rejected. These findings confirm the presence of nonlinear dynamics in the time series and provide strong empirical motivation for employing GARCH-type models, as well as incorporating realized measures of volatility to capture short-term risk structures more effectively.

2.4.1. Univariate GARCH Models for Volatility

Table 2.4 contains the order for both the volatility and mean processes and final RMSE value as out-of-sample goodness of fitting measurements. Unfortunately, there is not a standard result in terms of which frame is the most suitable in terms of volatility forecasts: 1) 1 m data are the best solution for Ethereum, Litecoin, Ripple and Eos; 2) 1 h for Bitcoin and 3) 30 m for Binance. As for the worst results, on the contrary, 15 m data seem to be the less accurate time frequency

(followed by 5 m ones). Despite these evidences, the RMSE assumed by each frame for each cryptocurrency remains at a low level, testifying a general adaptability of realized GARCH models with respect to conditional volatility predictions. This result, in accordance to literature, testifies how a combination between ARMA and GARCH can improve the quality of forecasts, even if extended to higher frequencies perspectives.

Table 2.3. Tests for nonlinear dependencies in returns and volatility across cryptocurrencies. This table reports the p-values from the Ljung–Box test and the ARCH LM test applied to returns, squared returns, and realized volatility series across different frequencies. The results provide strong evidence of autocorrelation and conditional heteroskedasticity, particularly at intraday levels. On almost all intraday frequencies, the null hypothesis of linearity can be rejected, thereby justifying the use of nonlinear modeling approaches such as GARCH and realized volatility frameworks.

	Ljung-Box (returns)					
	BTC	**BNB**	**ETH**	**LTC**	**XRP**	**EOS**
1 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
5 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
15 m	0.00 %	0.00 %	0.03 %	0.01 %	0.00 %	0.81 %
30 m	0.01 %	0.00 %	0.00 %	0.19 %	0.00 %	0.00 %
1 h	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
1 d	3.58 %	7.35 %	2.77 %	7.36 %	8.23 %	3.66 %

	Ljung-Box (realized volatility)					
	BTC	**BNB**	**ETH**	**LTC**	**XRP**	**EOS**
1 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
5 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
15 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
30 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
1 h	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
1 d	3.24 %	4.45 %	7.04 %	9.49 %	0.08 %	6.40 %

	LM arch					
	BTC	**BNB**	**ETH**	**LTC**	**XRP**	**EOS**
1 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
5 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
15 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
30 m	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
1 h	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
1 d	5.06 %	8.94 %	3.81 %	9.82 %	0.12 %	3.30 %

Table 2.4. The ARMA and GARCH models with GED residuals were selected based on model performance and are reported here. The table presents optimal specifications of ARMA and GARCH models with the shape parameter of the Generalized Error Distribution (GED). The relatively low Root Mean Squared Error (RMSE) of the estimated returns is consistent with values commonly found in the empirical literature. Nonetheless, we retain the ARMA structure to explicitly model the expected return component, which is essential for portfolio optimization, and to reduce the risk of model misspecification.

BTC						**LTC**					
Orders	**1m**	**5m**	**15m**	**30m**	**1h**	**Orders**	**1m**	**5m**	**15m**	**30m**	**1h**
garch	(2;1)	(3;1)	(1;1)	(1;1)	(2;2)	garch	(1;1)	(3;1)	(3;1)	(3;1)	(3;1)
arma	(1;1)	(1;1)	(1;1)	(1;1)	(1;1)	arma	(0;0)	(1;1)	(0;0)	(0;0)	(0;0)
RMSE	0.24 %	0.27 %	0.36 %	0.22 %	0.21 %	RMSE	0.39 %	0.51 %	0.53 %	0.46 %	0.47 %
BNB						**XRP**					
Orders	**1m**	**5m**	**15m**	**30m**	**1h**	**Orders**	**1m**	**5m**	**15m**	**30m**	**1h**
garch	(3;1)	(2;1)	(1;1)	(3;1)	(3;1)	garch	(1;1)	(1;1)	(2;1)	(2;1)	(1;1)
arma	(0;1)	(0;1)	(1;1)	(1;1)	(0;1)	arma	(1;1)	(0;0)	(1;1)	(0;1)	(0;0)
RMSE	0.31 %	0.31 %	0.32 %	0.27 %	0.28 %	RMSE	0.39 %	0.51 %	0.53 %	0.46 %	0.47 %
ETH						**EOS**					
Orders	**1m**	**5m**	**15m**	**30m**	**1h**	**Orders**	**1m**	**5m**	**15m**	**30m**	**1h**
garch	(1;1)	(2;1)	(2;1)	(3;3)	(1;3)	garch	(1;1)	(2;1)	(2;1)	(2;1)	(2;1)
arma	(1;1)	(0;1)	(0;1)	(0;1)	(0;1)	arma	(1;1)	(0;1)	(1;1)	(0;1)	(0;1)
RMSE	0.30%	0.34 %	0.36 %	0.32 %	0.33 %	RMSE	0.39 %	0.51 %	0.53 %	0.46 %	0.47 %

2.4.2. Marginal Distributions

Table 2.5 reports the significance levels of both the Kolmogorov–Smirnov test and its Uniformity-based variant [48]. The results indicate that the GED and Skewed-GED distributions provide the best fit for all considered time frames. Indeed, they show the highest p-values: Skewed GED for Bitcoin and Eos; GED for Binance, Ethereum, Litecoin and Ripple. Even from a graphical point of view (QQ-plot),[1] these distributions seem to be the most similar to the empirical returns (although if in absence of an evident significant difference). Given their predominance, the above-mentioned probability functions were chosen as marginal distributions to be inserted in the multivariate copula.

[1] Please note that the plots are not shown given their huge numerosity (6 assets x 5 frames x 8 distributions = 240 plots).

Table 2.5. (Part I). Kolmogorov–Smirnov and Uniformity test results for copula-based simulations. This table reports the results of goodness-of-fit diagnostics applied to the simulated return distributions derived from the copula-based model. The Kolmogorov–Smirnov test evaluates the closeness between the empirical and simulated distributions, while the Uniformity test checks for uniformity of the transformed margins. The test statistics generally support an adequate fit of the copula model to the data, indicating that the simulated joint distribution captures the underlying dependence structure sufficiently well.

		Norm		**sNorm**		**C-F**		**t-S**	
		K-S	**U**	**K-S**	**U**	**K-S**	**U**	**K-S**	**U**
BTC	1 m	0.36	0.00	0.36	0.00	0.59	0.00	40.90	9.60
	5 m	0.28	0.00	0.36	0.00	5.22	0.21	31.61	10.01
	15 m	0.36	0.00	0.47	0.00	9.07	0.38	46.11	20.35
	30 m	0.28	0.00	0.36	0.00	7.58	0.09	51.63	11.78
	1 h	0.28	0.00	0.28	0.00	12.78	1.20	46.11	7.61
BNB	1 m	15.06	0.03	15.06	0.01	90.30	52.64	90.30	35.73
	5 m	9.07	0.01	10.79	0.01	90.30	53.23	80.79	32.71
	15 m	9.07	0.00	10.79	0.00	69.34	34.20	80.79	19.99
	30 m	10.79	0.00	12.78	0.00	75.21	31.25	85.88	12.01
	1 h	9.07	0.01	12.78	0.01	80.79	25.83	75.21	30.30
ETH	1 m	1.20	0.00	0.59	0.00	5.22	4.66	80.79	13.52
	5 m	0.59	0.00	0.28	0.00	4.29	0.68	57.41	17.30
	15 m	0.76	0.00	0.36	0.00	5.22	2.39	63.36	14.89
	30 m	0.47	0.00	0.22	0.00	3.52	2.11	51.63	2.00
	1 h	0.47	0.00	0.22	0.00	4.29	1.44	51.63	2.57
LTC	1 m	5.22	0.02	2.33	0.15	0.02	0.00	63.36	32.22
	5 m	1.51	0.00	0.96	0.00	0.07	0.00	93.89	48.49
	15 m	3.52	0.00	1.51	0.01	0.00	0.00	46.11	54.43
	30 m	3.52	0.00	1.51	0.02	0.00	0.00	51.63	55.04
	1 h	4.29	0.00	1.88	0.04	0.00	0.00	57.41	49.08
XRP	1 m	1.20	0.00	0.96	0.00	0.47	0.00	75.21	55.64
	5 m	1.20	0.00	1.51	0.00	2.87	0.00	75.21	67.01
	15 m	0.96	0.00	0.76	0.00	2.87	0.83	63.36	62.86
	30 m	0.76	0.00	0.76	0.00	5.22	1.72	57.41	61.06
	1 h	3.52	0.00	4.29	0.00	23.88	1.40	72.55	22.98
EOS	1 m	0.47	0.00	0.59	0.00	93.89	75.47	57.41	1.95
	5 m	0.96	0.00	0.96	0.00	99.34	77.06	51.63	5.46
	15 m	0.36	0.00	0.47	0.00	75.21	48.49	15.06	2.51
	30 m	0.36	0.00	0.47	0.00	75.21	63.46	23.88	2.05
	1 h	0.47	0.00	0.47	0.00	63.36	15.18	17.65	0.44

Table 2.5. (Part II).

		st-S 1		st-S 2		GED		sGED	
		K-S	**U**	**K-S**	**U**	**K-S**	**U**	**K-S**	**U**
BTC	1 m	90.30	47.32	57.41	1.77	98.33	63.46	98.33	66.42
	5 m	80.79	34.20	51.63	0.01	98.33	96.63	98.33	97.39
	15 m	90.30	16.67	63.36	0.05	99.80	91.75	99.80	91.75
	30 m	93.89	16.36	63.36	7.94	99.34	91.44	99.34	91.44
	1 h	93.89	5.34	63.36	0.05	99.34	79.11	99.34	79.11
BNB	1 m	85.88	19.64	80.79	0.16	100.00	82.96	99.96	69.33
	5 m	80.79	24.58	75.21	32.22	98.33	78.60	98.33	74.93
	15 m	85.88	6.98	75.21	10.43	96.55	84.74	96.55	84.74
	30 m	85.88	4.15	80.79	6.39	90.30	71.61	90.30	65.83
	1 h	80.79	24.58	75.21	26.69	90.30	97.25	90.30	96.29
ETH	1 m	85.88	9.21	93.89	17.30	99.80	89.80	96.55	76.53
	5 m	63.36	14.33	85.88	10.86	96.55	97.52	93.89	95.72
	15 m	69.34	12.01	80.79	11.78	96.55	96.46	93.89	98.01
	30 m	51.63	5.46	75.21	0.01	90.30	97.90	85.88	92.90
	1 h	51.63	1.95	75.21	9.80	90.30	96.63	85.88	95.53
LTC	1 m	75.21	44.44	57.41	74.93	85.88	56.84	63.36	42.18
	5 m	75.21	32.71	69.34	61.06	99.34	85.17	98.33	89.10
	15 m	75.21	67.01	46.11	60.46	85.88	68.17	69.34	58.65
	30 m	69.34	52.04	51.63	71.05	90.30	36.25	69.34	34.70
	1 h	75.21	30.30	51.63	2.30	90.30	47.90	69.34	31.73
XRP	1 m	93.89	83.41	75.21	56.24	96.55	86.82	90.30	76.00
	5 m	69.34	61.05	76.33	0.73	78.11	64.44	70.90	62.74
	15 m	90.30	47.90	69.34	44.44	85.88	95.11	85.88	94.43
	30 m	85.88	78.09	63.36	49.67	96.55	80.59	90.30	74.39
	1 h	69.34	41.62	68.33	0.84	75.21	41.07	63.36	40.20
EOS	1 m	75.21	3.96	69.34	1.33	99.80	95.53	99.80	95.53
	5 m	63.36	4.88	63.36	3.87	97.73	98.01	98.33	98.61
	15 m	36.06	3.52	46.11	0.01	63.36	19.29	63.36	19.29
	30 m	36.06	0.15	40.90	0.09	80.79	74.93	85.88	77.06
	1 h	27.55	0.07	31.61	0.42	69.34	27.57	69.34	30.78

2.4.3. Copula Function

Before analyzing risk metrics such as Value at Risk and Expected Shortfall, it is important to evaluate the performance of the copula-based simulations across different time frames. The RMSE resulting from the out-of-sample forecasts are summarized in Table 2.6, where results tend to differ considering different time frames: 1) Gumbel copula for 1m and 15 m data; 2) Joe copula for 5 m; 3) Clayton copula for 30 m and

4) Frank copula for 1 h.[1] RMSE is calculated for each frequency, measuring the distance between the simulated and empirical return distributions. Although the differences in RMSE are not large, a clear trend emerges: lower-frequency data (e.g., 30-minute and 1-hour) generally result in more stable model fits. This confirms that high-frequency data, while rich in information, may introduce additional noise and estimation challenges—an important consideration when choosing the appropriate modeling granularity.

Archimedean copulas are usually preferred to the Elliptical ones, testifying a higher accuracy for models which do not assume a priori linear dependencies among the assets within the same portfolio. Similar results, as already mentioned in the Introduction, have been already found in literature. Another important evidence is represented by the quality of 5m data. All the RMSE are indeed higher than the values obtained from the other frames. This could be related to the accuracy obtained for all the GARCH models (higher than the one resulting from a 15m frame, but still lower the other frequencies), capable to influence the multivariate forecasts. 1m data, contrary, have the lowest RMSE, showing a greater predictive power. However, while the RMSE for GARCH has been computed considering real out-of-sample realized volatilities, here it is computed considering the expected return for each day (mean of all the simulations per period).

Table 2.6. RMSE of copula-based simulations across data frequencies. This table presents the RMSE values for each data frequency used in the copula-based simulation of portfolio returns. The RMSE is computed by comparing simulated and empirical return distributions. Although differences across frequencies are relatively small, the results indicate that lower-frequency data (e.g., 30-minute or 1-hour) tend to yield slightly more stable fits, while higher-frequency data exhibit greater sensitivity to noise and microstructure effects.

Copula	Frame				
	1 m	**5 m**	**15 m**	**30 m**	**1 h**
Normal	5.2031 %	5.2622 %	5.2237 %	5.2219 %	5.2157 %
t-Student	5.2051 %	5.2609 %	5.2209 %	5.2191 %	5.2246 %
Gumbel	5.1980 %	5.2572 %	5.2063 %	5.2058 %	5.2295 %
Clayton	5.2103 %	5.2594 %	5.2175 %	5.2158 %	5.2291 %
Frank	5.2075 %	5.2655 %	5.2216 %	5.2162 %	5.2142 %
Joe	5.2045 %	5.2559 %	5.2226 %	5.2159 %	5.2193 %

[1] We use the R-package provided by [47] to obtain these results.

2.4.4. Term Structure

After having optimized the copula models, 10,000 simulations have been carried out for each frame for the month of July 2019, in order to forecast daily multi-period Value at Risk and Expected Shortfall for both the 99 % (Fig. 2.2) and 95 % (Fig. 2.3) levels of confidence.[1] As can be seen, 5 m data is generally the less defensive frame, returning lower asymmetrical risks for almost all the time horizons (indicating a reduced daily risk of losses for all the investment maturities). As for the other frequencies, 15 m, 30 m and 1 h data tend to assume similar values (especially for VaR), while 1m converges with them from below, but still maintains a lower level. This finding testifies how important choosing the right framework could be.

Anyway, even if 1m data are the ones which best fit the prediction of the real returns from a multivariate point of view, they cannot be directly considered as the most reliable for asymmetrical risk prediction purposes. The reason is linked to the fact that the tails of a distribution could act differently from the mean according to the moments. The same evidence can be found in Table 2.7, which summarizes the risk measures week by week.

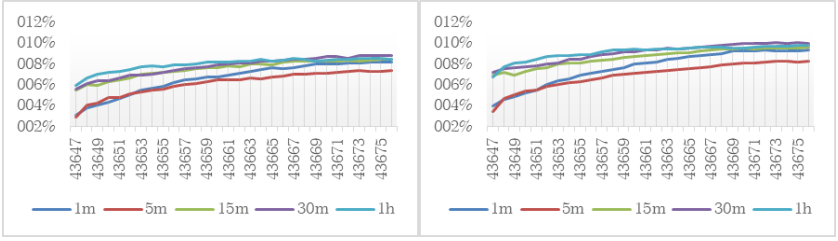

Fig. 2.2. VaR and Marginal Expected Shortfall (ES) for 99% Term Structure: estimates for each cryptocurrency across different data frequencies (1 m, 5 m, 15 m, 30 m, 1 h, 1 d) were done based on 10,000 copula-based Monte Carlo simulations. The figure illustrates how tail risk contributions vary with time aggregation. MES values tend to be higher at higher frequencies, especially for more volatile assets, emphasizing the importance of time-scale sensitivity in marginal risk assessment. Results shown are representative of the simulation outcomes.

[1] Please note that VaR was back-tested using frequency tests. None of them rejected the model proposed by this article.

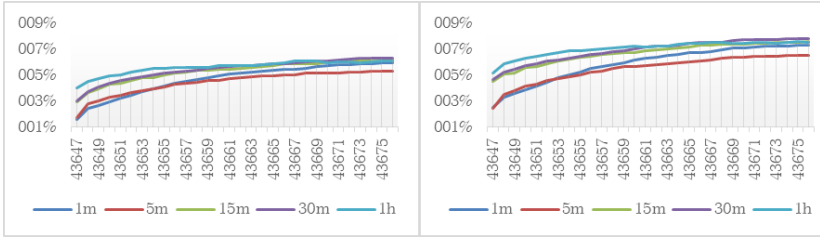

Fig. 2.3. VaR and Marginal Expected Shortfall (ES) estimates at the 95% confidence level for each cryptocurrency across different data frequencies (1 m to 1 d), based on 10,000 copula-based Monte Carlo simulations. This figure mirrors the structure of Fig. 2.2, but applies a lower tail threshold, thereby illustrating how marginal risk contributions respond to less extreme quantiles.

Table 2.7. Term structures of Value at Risk (VaR) and Expected Shortfall (ES) across investment horizons (in %, by weeks). This table presents the evolution of simulated VaR and ES values over increasing time horizons, ranging from one day to four weeks. The results illustrate the nonlinearity of tail risk accumulation: while risk levels increase with longer holding periods, the rate of increase diminishes. This concave structure reflects the subadditive nature of risk under fat-tailed distributions and highlights how extending the investment horizon may reduce marginal exposure to extreme losses.

	VaR 99 %					**ES 99 %**				
Time	**1m**	**5m**	**15m**	**30m**	**1h**	**1m**	**5m**	**15m**	**30m**	**1h**
01/07	3.10	2.90	5.48	5.59	5.90	3.98	3.47	6.95	7.16	6.77
07/07	5.51	5.35	7.05	6.89	7.77	6.43	6.03	7.96	8.10	8.81
14/07	6.74	6.44	7.60	7.87	8.18	7.96	7.07	8.70	9.17	9.40
21/07	7.60	6.98	8.24	8.48	8.53	8.89	7.74	9.36	9.70	9.58
30/07	8.22	7.33	8.48	8.80	8.47	9.34	8.25	9.63	10.01	9.79
	VaR 95 %					**ES 95 %**				
Time	**1m**	**5m**	**15m**	**30m**	**1h**	**1m**	**5m**	**15m**	**30m**	**1h**
01/07	1.54	1.72	2.90	3.01	4.01	2.50	2.44	4.50	4.62	5.15
07/07	3.72	3.80	4.76	4.86	5.38	4.81	4.75	6.05	6.14	6.76
14/07	4.90	4.58	5.46	5.58	5.70	6.15	5.66	6.76	6.99	7.22
21/07	5.48	5.00	5.86	5.97	6.12	6.83	6.17	7.29	7.54	7.55
30/07	5.93	5.32	6.14	6.30	6.06	7.31	6.55	7.57	7.80	7.55

2.4.5. Marginal Risk Contributions

Before showing the marginal allocations, it is necessary to have an understanding of what happens to the expected weights (computed as the mean of all the simulations per day) for each cryptocurrency at each

future investment maturity. As already explained in Sect. 2.6, a risk contribution depends directly on the weight of the related asset. Assuming a *ceteris paribus* framework for all the other parameters, an increase in the weight would lead to an increase in the risk contribution. Results are shown in Fig. 2.4, where it is possible to note that all the weights tend to be stable over time for each frequency. This finding allows us to proceed with the analysis, making it possible to compare risk allocations.

Fig. 2.4. Average portfolio weights for each cryptocurrency across different data frequencies (1m to 1d), computed as the daily mean across 10,000 Monte Carlo simulations. The figure provides a preliminary view of how each asset is weighted in the simulated portfolios prior to risk attribution. Notably, the estimated weights remain relatively stable over time at each frequency, suggesting that short-term fluctuations in individual asset characteristics do not significantly distort the overall allocation structure. This weight stability is crucial, as marginal risk contributions are directly linked to these weights. Thus, the consistency observed in this figure offers a solid foundation for the comparative analysis of marginal risk allocation across time horizons.

2.4.6. Locally – Constant Kernel at Various Tail Risks

Fig. 2.5 and Fig. 2.6 presents how different frames tend to modify the marginal contributions of each cryptocurrency on the tail risk measures. Focusing first on the evolution of the contributions over time for each frame, it is possible to note how they tend to assume a more stable shape as the time horizon increases for both VaR and ES and for each tail risk level (99 % and 95 %).[1]

Looking at the longest time horizon (1 month) in Table 2.8 and Table 2.9,[2] even if all the frames return different contributions, they tend to respect the same ordinality for the riskiest cryptocurrencies. On a scale from the least to the riskiest one, cryptocurrencies generally follow this order (that is more stable in case of a 99% level of confidence): 1) BTC; 2) BNB; 3) XRP; 4) ETH; 5) EOS, 6) LTC. Looking at the shortest time horizon (overnight perspective), the ordinality seems to be completely different and more variable with respect to the different frequencies (and also with respect to different risk measures). However, the main difference between shorter and longer horizons is the level of concentration. It is also possible to note how the range between the least and the riskiest cryptocurrency tend to decrease after the overnight scenario. It means that time smooths the risk associated to each cryptocurrency, providing a portfolio with a risk time diversification. This is important evidence considering the standardized framework related to a trend in the weights, and it justifies the importance of studying the risk composition within a portfolio (being in line with the expectations): an asset might influence the portfolio return by a certain percentage, but affect the overall risk in a more severe way.

Another interesting finding is highlighted in the overnight scenario in Table 2.9 (95 % level of confidence), 1 m, 15 m and 30 m data present some negative percentages both in terms of Value at Risk and Expected Shortfall. A negative contribution leads to a strong diversification effect between the related asset and the combination of the others as its inclusion in the portfolio is certainly reducing the overall tail risk.[3]

[1] Respectively related to a level of confidence of 99 % and 95 %.
[2] Respectively related to a level of confidence of 99 % and 95 %.
[3] These diversification possibilities are not shown in tail risk measures at a 99 % level of confidence. This fact is certainly due to the evidence that an error component of 1 % returns highly extreme negative returns, so that, even in presence of weak positive correlations, it would not be possible to have a positive return for one asset when all the others suffers losses.

Fig. 2.5. Marginal tail risk allocations at the 99% confidence level using the Constant Kernel method. This figure illustrates the time-evolving marginal contributions of each cryptocurrency to total portfolio tail risk, estimated at the 99% confidence level using the Constant Kernel smoothing approach. The results highlight strong asymmetries and short-term risk concentration in specific assets, especially under extreme market conditions. This visualization complements the tabular results by showing how marginal risks shift across time horizons.

Fig. 2.6. Marginal tail risk allocations for each cryptocurrency at the 95 % confidence level, computed using the Constant Kernel estimator. The figure illustrates how individual assets contribute to total portfolio tail risk under moderate stress conditions. The results show frequency-sensitive variations in marginal risk contributions, highlighting that some cryptocurrencies exert greater influence on downside risk at specific time horizons. This intermediate quantile level provides additional insight beyond extreme-value cases such as the 99 % threshold.

Fig. 2.7. Marginal tail risk allocations for each cryptocurrency at the 99 % confidence level, calculated using the Linear Kernel estimator. This figure captures the contribution of each asset to extreme downside risk in the portfolio. The use of the Linear Kernel highlights sharp tail dependencies that become more prominent under severe market stress, offering a complementary perspective to the Constant Kernel approach at lower quantile levels.

Fig. 2.8. Marginal tail risk allocations for each cryptocurrency at the 95% confidence level, estimated using the Linear Kernel method. This figure captures relative risk contributions under moderate stress conditions, allowing comparison with more extreme tail risk results (as shown in Fig. 2.7). While overall ranking patterns persist, marginal effects are less pronounced, highlighting the smoother risk distribution at lower quantiles.

After examining the asymmetric risk marginal allocation of each cryptocurrency, it is possible to split the six cryptocurrencies into two

groups: the leastless risky and the riskiest ones. The former is made up of Ripple, Binance (which have a minimum risk for overnight frameworks) and Bitcoin (which seem to be the safest currency in terms of potential losses, especially for longer short-term horizons); the latter, on the other hand, is composed of Eos, Ethereum and Litecoin (which presents the highest risk for almost all the scenarios).

Table 2.8. Marginal tail risk allocations at the 99% confidence level using the Constant Kernel method. This table shows the marginal contributions of each cryptocurrency to total portfolio tail risk, computed at the 99% confidence level based on copula simulations using a Constant Kernel estimator. The results provide a granular view of risk attribution under extreme loss scenarios. The concentration of tail risk in certain assets, especially at shorter horizons, highlights asymmetries in the dependence structure and the need for tailored risk mitigation strategies.

		VaR 99 %						ES 99 %					
	Time	**BTC %**	**BNB %**	**ETH %**	**LTC %**	**XRP %**	**EOS %**	**BTC %**	**BNB %**	**ETH %**	**LTC %**	**XRP %**	**EOS %**
1d	**1m**	13.00	0.03	34.77	29.10	2.78	20.31	15.04	0.99	35.34	25.55	3.94	19.15
	5m	10.53	4.63	31.59	22.07	11.35	19.83	10.16	4.39	32.00	22.76	9.87	20.82
	15m	17.52	9.32	25.63	28.97	0.26	18.31	18.11	9.81	24.84	27.04	0.82	19.38
	30m	20.61	20.67	17.03	20.73	0.45	20.52	21.61	18.81	16.83	20.05	1.33	21.37
	1h	18.00	14.00	20.92	22.29	5.82	18.97	18.26	13.49	21.19	20.40	5.29	21.37
7d	**1m**	10.99	9.16	24.01	24.19	13.70	17.94	11.75	9.31	24.08	23.08	13.54	18.24
	5m	11.11	13.49	20.69	24.00	12.69	18.02	11.34	13.28	20.88	24.07	12.14	18.29
	15m	14.27	12.88	19.19	24.54	10.93	18.19	14.45	12.70	19.30	24.48	10.79	18.27
	30m	14.82	17.02	18.24	21.92	10.37	17.62	14.68	17.63	17.79	21.62	10.17	18.10
	1h	15.06	16.07	18.97	20.90	10.43	18.58	15.04	16.19	19.14	20.75	10.24	18.65
14d	**1m**	9.81	12.35	20.74	22.35	15.92	18.84	10.57	12.70	20.68	21.56	16.19	18.30
	5m	10.26	14.64	19.98	23.00	14.49	17.63	10.16	14.89	20.12	22.96	14.32	17.55
	15m	12.42	13.75	18.18	23.48	14.56	17.60	12.56	14.09	18.14	23.16	14.06	18.00
	30m	12.66	16.35	17.33	21.21	14.88	17.57	12.71	16.68	17.46	21.00	14.56	17.60
	1h	13.70	15.77	17.49	21.39	13.17	18.49	13.81	16.22	17.40	21.36	13.11	18.10
21d	**1m**	9.81	13.86	18.86	21.80	17.18	18.49	10.11	14.31	18.92	21.12	17.15	18.38
	5m	10.16	15.11	19.28	21.85	15.29	18.30	10.39	15.18	19.60	21.32	15.29	18.22
	15m	11.27	14.58	17.64	22.24	16.26	18.00	11.54	14.87	17.88	21.41	16.01	18.28
	30m	11.72	16.37	16.86	21.07	15.80	18.18	12.20	16.36	17.06	20.47	15.63	18.28
	1h	12.51	15.88	17.47	21.33	14.73	18.09	12.78	15.87	17.29	21.19	14.78	18.08
30d	**1m**	9.50	14.44	18.63	20.85	17.62	18.96	9.95	14.65	18.60	20.08	17.23	19.49
	5m	9.83	14.93	18.39	22.07	15.75	19.03	9.86	15.26	18.64	22.01	15.48	18.73
	15m	10.87	14.41	17.48	21.50	16.86	18.89	11.23	14.62	17.48	21.03	16.59	19.06
	30m	10.92	16.38	16.64	20.80	17.19	18.07	11.32	16.42	16.75	20.43	17.03	18.05
	1h	11.84	15.97	17.42	21.07	15.30	18.39	12.16	16.64	16.92	20.40	15.64	18.23

Table 2.9. Marginal tail risk allocations at the 95 % confidence level using the Constant Kernel method. This table displays the marginal risk contributions of each cryptocurrency to portfolio losses under the 95 % confidence level, estimated via copula simulations with a Constant Kernel. Compared to the 99 % level, the relative contributions are less concentrated, indicating smoother tail behavior and lower extreme sensitivity. The results highlight how the confidence level affects perceived diversification and asset-specific risk weights.

		VaR 95 %						ES 95 %					
	Time	**BTC %**	**BNB %**	**ETH %**	**LTC %**	**XRP %**	**EOS %**	**BTC %**	**BNB %**	**ETH %**	**LTC %**	**XRP %**	**EOS %**
1d	**1m**	9.37	-7.40	36.39	44.63	-2.21	19.21	12.48	-1.83	35.08	32.72	1.86	19.68
	5m	9.84	1.69	32.07	19.36	18.05	18.99	10.30	3.74	31.59	21.39	13.18	19.80
	15m	17.81	8.73	22.91	37.91	-3.95	16.60	17.94	9.47	24.37	31.01	-0.84	18.05
	30m	18.50	19.57	16.49	29.22	-2.92	19.14	20.20	19.93	16.70	23.24	-0.31	20.23
	1h	16.69	14.01	19.44	25.07	7.40	17.38	17.58	13.94	20.38	22.93	6.34	18.83
7d	**1m**	9.89	8.29	23.62	26.42	14.24	17.53	10.76	8.91	23.86	24.80	13.86	17.83
	5m	10.80	12.87	20.96	24.05	13.52	17.79	11.06	13.21	20.83	23.99	12.93	17.98
	15m	13.49	12.57	19.05	26.33	11.29	17.28	13.97	12.74	19.21	25.23	11.05	17.81
	30m	14.20	16.80	17.84	23.26	10.91	16.98	14.61	17.11	17.99	22.30	10.50	17.50
	1h	14.62	15.11	18.98	22.09	11.32	17.87	14.90	15.66	19.04	21.32	10.77	18.31
14d	**1m**	9.10	11.52	20.39	24.03	16.71	18.24	9.75	12.14	20.60	22.75	16.23	18.54
	5m	9.90	13.60	19.92	23.22	15.10	18.26	10.14	14.27	19.96	23.02	14.72	17.90
	15m	11.53	12.94	18.17	24.94	15.48	16.93	12.10	13.52	18.20	23.96	14.82	17.40
	30m	11.70	15.87	17.01	22.90	15.27	17.26	12.31	16.25	17.19	21.84	14.94	17.47
	1h	12.52	15.15	17.96	22.44	13.99	17.95	13.26	15.61	17.72	21.70	13.46	18.26
21d	**1m**	8.86	12.86	19.04	23.16	17.80	18.27	9.54	13.62	18.95	22.13	17.37	18.39
	5m	9.47	14.07	18.81	23.46	15.42	18.76	9.93	14.71	19.14	22.42	15.33	18.47
	15m	10.62	13.50	17.61	24.21	16.80	17.26	11.07	14.22	17.67	22.88	16.37	17.79
	30m	10.70	15.56	16.54	22.79	16.92	17.49	11.45	16.11	16.79	21.56	16.15	17.93
	1h	11.56	15.43	17.71	22.35	15.14	17.81	12.14	15.72	17.59	21.67	14.86	18.02
30d	**1m**	8.55	13.14	18.13	22.58	18.80	18.80	9.22	13.99	18.40	21.38	18.06	18.95
	5m	8.98	14.36	18.39	22.99	15.99	19.29	9.53	14.77	18.42	22.37	15.82	19.09
	15m	9.97	13.94	17.20	23.35	17.83	17.73	10.59	14.30	17.36	22.10	17.17	18.48
	30m	9.95	15.54	16.39	22.37	17.99	17.76	10.61	16.09	16.56	21.33	17.43	17.98
	1h	11.09	15.04	17.86	22.56	15.69	17.76	11.64	15.73	17.54	21.47	15.47	18.14

2.4.7. Locally – Linear Kernel at Various Tail Risks

The contributions related to tail risk measures at a 99 % and 95 % level of confidence present a higher variability in comparison to the ones

obtained with a locally constant methodology (as shown in Fig. 2.7 and Fig. 2.8).[1]

Fig. 2.7. Marginal tail risk allocations for each cryptocurrency at the 99 % confidence level, calculated using the Linear Kernel estimator. This figure captures the contribution of each asset to extreme downside risk in the portfolio. The use of the Linear Kernel highlights sharp tail dependencies that become more prominent under severe market stress, offering a complementary perspective to the Constant Kernel approach at lower quantile levels.

[1] Respectively related to a level of confidence of 99% and 95 %.

Fig. 2.8. Marginal tail risk allocations for each cryptocurrency at the 95 % confidence level, estimated using the Linear Kernel method. This figure captures relative risk contributions under moderate stress conditions, allowing comparison with more extreme tail risk results (as shown in Fig. 2.7). While overall ranking patterns persist, marginal effects are less pronounced, highlighting the smoother risk distribution at lower quantiles.

Indeed, it is possible to note how their trend is not smoothed over different time horizons, leading to a non-clear ordinality among the less risky and riskier cryptocurrencies and to a difficultly interpretable time diversification effect. After having a look at Table 2.10 and Table 2.11,[1] it becomes evident that the range between the riskiest asset and the safest one does not tend to reduce in the mean for all the frames as the investment maturity increases (like in the constant kernel framework). For example, Bitcoin appears to be the least risky investment for a great number of scenarios of a 99% level of confidence, but this result is not reliable.

The results for a 95 % level of confidence, however, seems to be more stable. Indeed, results for all the frames from a certain investment maturity follow almost the same ordinality in terms of risk. As for the riskiest asset, Litecoin has the highest percentage of contribution for all the scenarios summarized in Table 2.11. Bitcoin, on the other hand, is not always the least risky investment. This generally happens for frames with data that is higher than 1m and 5m data. This fact makes it difficult to rank the currencies with respect to their contribution to the overall risk. What is interesting is that 1m data still provides some strong diversification effects by returning negative contributions for the overnight scenario (Binance) and the 14d and 21d horizons (Ripple).

Such a model has to be carefully evaluated before its application as it tends to be more reactive to different intra-day frameworks, but it could be relevant to note how Bitcoin, Ripple and Binance are in general the coins which are chosen as safer investments. As for the riskier alternatives, Ethereum, Eos and Litecoin show, in the mean, generally higher contributions for both the tail measures.

The results indicate that, from a univariate perspective, intraday data more accurately reflect the actual volatility dynamics in cryptocurrency markets. Among the models tested, the ARMA–realized GARCH model with Generalized Error Distribution (GED) residuals proved to be the most effective for capturing both volatility clustering and the leverage effect, while leveraging the informational richness of high-frequency returns. The use of a fat-tailed distribution over the normal distribution was justified by the empirical presence of extreme return values, which are a well-documented characteristic of crypto-assets.

[1] Respectively related to a level of confidence of 99 % and 95 %.

Table 2.10. Marginal tail risk allocations at the 99 % confidence level using the Linear Kernel method. This table presents the marginal contributions of each cryptocurrency to total portfolio tail risk at the 99 % confidence level, estimated using a Linear Kernel approach in the copula simulation. Compared to the Constant Kernel, the Linear Kernel introduces smoother weight adjustments across assets and time frames. The results reveal how the choice of kernel affects the distribution of extreme risk and may alter portfolio-level risk management decisions.

		VaR 99 %						ES 99 %					
	Time	**BTC %**	**BNB %**	**ETH %**	**LTC %**	**XRP %**	**EOS %**	**BTC %**	**BNB %**	**ETH %**	**LTC %**	**XRP %**	**EOS %**
1d	**1m**	13.78	0.76	35.25	26.83	3.09	20.29	20.01	1.25	34.46	22.38	1.53	20.37
	5m	10.92	9.62	25.63	22.76	12.82	18.25	12.60	8.51	22.01	21.87	13.57	21.44
	15m	9.88	12.78	20.61	21.68	16.35	18.69	11.79	10.86	20.32	19.30	19.28	18.46
	30m	11.06	14.30	18.56	21.51	16.93	17.66	12.00	9.97	20.83	21.58	17.30	18.31
	1h	10.92	13.52	19.16	20.38	16.58	19.44	8.72	7.48	19.13	23.25	26.44	14.98
7d	**1m**	10.38	5.01	32.59	22.99	8.70	20.32	16.41	2.98	31.30	17.98	8.13	23.20
	5m	11.44	14.55	20.15	24.36	11.75	17.76	11.01	16.69	19.78	21.50	12.10	18.93
	15m	10.12	15.52	21.10	23.35	13.82	16.10	10.47	16.01	21.76	18.17	15.78	17.81
	30m	10.71	15.23	20.27	20.96	15.42	17.41	10.90	16.46	22.30	17.13	13.59	19.62
	1h	10.03	15.42	19.22	21.40	15.04	18.89	8.90	18.80	15.00	18.38	15.17	23.75
14d	**1m**	17.16	9.25	26.04	28.12	0.79	18.65	15.45	11.92	26.36	28.04	2.56	15.66
	5m	14.64	13.27	18.21	23.61	11.24	19.04	13.10	12.56	19.51	24.73	11.11	19.00
	15m	12.64	14.57	17.73	22.83	14.22	18.01	11.43	13.85	19.75	21.99	12.90	20.09
	30m	12.26	15.50	16.97	22.13	16.04	17.09	11.45	15.41	19.06	18.87	15.76	19.46
	1h	11.59	13.66	18.45	21.04	15.99	19.28	14.33	10.61	15.20	25.36	15.67	18.83
21d	**1m**	20.94	20.07	17.29	19.81	0.84	21.05	15.32	20.01	23.37	22.10	0.43	18.78
	5m	14.62	17.18	18.27	21.71	10.34	17.89	13.03	20.12	18.75	22.60	8.39	17.11
	15m	12.57	15.67	18.55	21.26	14.21	17.74	13.30	18.14	18.52	19.20	13.84	17.00
	30m	11.78	16.23	17.90	19.33	16.08	18.68	13.01	16.08	18.52	19.54	15.06	17.79
	1h	11.13	17.23	17.79	19.88	16.59	17.38	11.25	21.69	16.07	14.43	20.11	16.46
30d	**1m**	18.44	13.77	21.53	20.76	5.06	20.44	13.64	14.00	21.39	18.12	5.90	26.95
	5m	15.18	16.68	18.55	20.65	10.02	18.92	13.85	15.94	20.39	22.48	12.20	15.14
	15m	14.18	16.03	17.14	21.42	13.05	18.17	13.40	15.77	18.86	19.74	16.68	15.54
	30m	12.99	15.80	17.64	20.30	15.16	18.11	12.00	15.64	17.54	20.35	14.83	19.64
	1h	11.50	15.83	16.49	21.64	15.38	19.16	11.27	22.50	10.55	26.34	8.99	20.36

Table 2.11. Marginal tail risk allocations at the 95 % confidence level using the Linear Kernel method. This table shows the marginal risk contributions of each cryptocurrency at the 95% confidence level, based on simulations using the Linear Kernel estimator. Relative to the 99 % level, the results display more balanced contributions across assets, reflecting the reduced influence of extreme tail events. The use of a Linear Kernel provides a smoother and more gradual weighting structure, which may be preferable for risk attribution under moderate stress scenarios.

		VaR 95 %						ES 95 %					
	Time	BTC %	BNB %	ETH %	LTC %	XRP %	EOS %	BTC %	BNB %	ETH %	LTC %	XRP %	EOS %
1d	1m	11.08	-3.89	34.88	37.42	0.67	19.83	15.92	0.81	30.23	33.97	1.96	17.11
	5m	10.13	9.02	24.30	25.39	13.65	17.52	12.14	10.16	20.25	24.04	16.84	16.56
	15m	9.26	12.31	20.56	22.35	16.56	18.97	11.24	13.74	20.13	21.12	16.83	16.93
	30m	9.99	14.28	18.74	21.73	15.65	19.61	10.93	14.90	18.92	20.99	17.51	16.75
	1h	9.52	13.75	18.59	21.32	19.11	17.71	13.13	16.94	18.26	22.13	16.45	13.09
7d	1m	11.04	5.00	30.22	21.75	12.49	19.51	8.60	6.75	28.18	21.10	10.82	24.55
	5m	11.77	12.84	20.53	23.93	13.27	17.66	10.11	14.10	20.66	22.23	13.98	18.91
	15m	10.70	13.93	19.61	22.45	15.03	18.29	9.77	15.03	19.53	22.61	15.56	17.50
	30m	9.38	13.86	19.07	23.10	15.01	19.58	9.67	15.54	19.07	22.01	15.53	18.18
	1h	9.39	15.01	17.67	22.84	15.71	19.38	10.73	14.24	18.23	22.41	15.08	19.31
14d	1m	17.96	9.64	23.57	33.54	-1.88	17.17	18.77	8.53	23.44	32.81	1.14	15.31
	5m	13.68	13.17	19.57	25.43	11.22	16.94	14.25	12.45	21.90	24.33	10.06	17.00
	15m	12.13	13.24	18.79	23.45	15.43	16.97	12.11	14.29	18.74	24.39	13.72	16.74
	30m	10.98	13.86	18.00	23.23	16.34	17.59	10.68	15.61	18.79	23.02	15.80	16.10
	1h	10.72	14.11	17.43	22.27	17.37	18.10	13.31	12.03	14.32	20.48	18.34	21.52
21d	1m	19.13	20.32	16.56	25.67	-1.27	19.58	19.12	17.86	17.92	23.21	0.17	21.72
	5m	14.99	16.92	18.14	21.95	10.33	17.68	14.93	16.74	18.64	22.78	9.37	17.55
	15m	12.31	16.38	16.94	22.07	14.57	17.73	12.61	17.77	16.93	21.27	14.12	17.29
	30m	11.08	16.57	16.74	21.37	16.24	18.01	10.75	16.80	17.66	20.75	15.61	18.43
	1h	10.05	16.79	16.32	21.60	17.52	17.71	11.08	16.88	15.28	20.80	16.66	19.31
30d	1m	17.37	14.24	19.97	23.87	6.81	17.75	16.93	14.75	21.80	21.08	6.26	19.17
	5m	15.14	14.95	19.40	21.40	11.16	17.94	15.10	15.80	18.87	21.08	10.65	18.50
	15m	12.44	15.54	18.33	21.83	13.45	18.41	13.34	16.44	19.12	20.88	12.86	17.36
	30m	11.25	15.20	18.24	22.06	14.06	19.19	13.09	16.78	18.41	20.75	13.15	17.83
	1h	12.38	15.53	17.80	21.41	14.85	18.03	12.24	17.60	17.72	19.19	14.15	19.10

The incorporation of realized volatility (RV) measures into GARCH-type models emphasized the strong dependence of volatility estimation on data frequency. The results confirmed that tail risk contributions are highly sensitive to the choice of time frame, underlining the relevance of frequency-aware modeling for risk budgeting and allocation. While the model does not fully satisfy all possible evaluation metrics, it provides new insights into the structure of tail risk within cryptocurrency portfolios. One key expectation, namely, that an asset's contribution to portfolio risk may diverge significantly from its return contribution, was confirmed across all configurations.

That said, the variability across time horizons and the differences observed between the two kernel estimators (Constant and Linear) prevent a precise quantification of risk incidence in percentage terms. This lack of ordinal consistency limits definitive rankings across frequencies, though some robust patterns do emerge.

In terms of relative performance, the findings suggest a stable risk hierarchy among the cryptocurrencies analyzed. Over short investment horizons (up to 30 days), Bitcoin consistently emerged as a relatively low-risk asset in terms of tail loss exposure, contrary to its reputation following previous market bubbles. Binance Coin and Ripple also demonstrated strong performance in the very short term (overnight to one week), with marginal risk contributions often falling below 1 %, depending on the frequency. This opens up potential for short-term risk diversification.

Conversely, Litecoin exhibited a concentration of risk, particularly at higher frequencies, thus increasing total portfolio risk. In the absence of a risk-focused optimization strategy, a naive approach of equalizing marginal loss across assets might be advisable. However, as the investment horizon lengthens, this effect naturally emerges, with marginal contributions tending to converge toward more uniform levels, resulting in overall portfolio risk reduction.

These dynamics are further corroborated by the observed term structures of VaR and ES, whose slopes indicate sublinear growth in tail risk with time. This non-linearity implies that extending the investment horizon leads to disproportionately lower incremental tail risk, a finding that could be highly relevant for the design of tactical and strategic trading frameworks.

Finally, although the lack of consistent out-of-sample goodness-of-fit across different frequencies prevents recommending a single optimal data granularity, a 30-day horizon appears to strike a practical balance between model stability and diversification. For conservative traders and risk-averse strategies, particularly those employing stop-loss mechanisms, Ethereum and Litecoin should be treated cautiously in the short term, while Bitcoin and Binance, with their favorable unconditional return statistics and relatively low tail risk, may offer more robust risk–return profiles.

2.5. Critical Discussion and Outlook

While the proposed methodology provides a novel and structured approach to decomposing tail risk in an asymmetric and nonlinear manner across cryptocurrency portfolios over different time horizons, several potential extensions remain open for future research.

One promising direction concerns the use of long-memory volatility models, such as FIGARCH and ARFIMA, which could account for persistent autocorrelation in volatility processes beyond the scope of standard GARCH-type models. These frameworks may enhance the modeling of long-term risk persistence and memory effects in asset returns.

However, it is important to critically reflect on the practical limitations of such extensions, particularly in the context of high-frequency trading environments. Long-memory models often require a longer learning phase and exhibit slower adaptation to extreme events as well as to structural shifts and regime changes. In settings where rapid response to incoming information and execution of short-horizon strategies is essential, this delayed adjustment may pose a significant disadvantage. Thus, the added theoretical richness of long-memory processes must be weighed against their real-time usability and computational burden.

A second line of potential extension lies in the incorporation of dynamic copula models, which allow the dependence structure among assets to evolve over time. While conceptually attractive, dynamic copulas are highly nonlinear and involve complex estimation procedures, often demanding considerable computational resources and sensitivity to parameter calibration. In addition, the lack of consensus on model

selection and interpretability further complicates their implementation in practical portfolio risk management.

Beyond methodological innovations, future research should also explore how structural breaks and distinct market regimes (such as those caused by the COVID-19 pandemic, the war in Ukraine, the Ethereum Merge, or major political and policy shifts following elections in economically dominant countries) affect marginal risk contributions and dependence structures. Understanding how such disruptions alter model behavior and tail risk composition is vital, particularly when combined with advanced modeling approaches. These investigations could help determine whether added complexity meaningfully improves predictive accuracy or whether the current framework offers a more robust and practical solution under real-world uncertainty.

2.6. Conclusion

Overall, the findings demonstrate that marginal risk attribution based on asymmetric and tail-sensitive models uncovers dependence structures that remain invisible in return-based assessments or conventional correlation measures. The empirical evidence shows that assets often labelled as “safe” or “risky” purely on the basis of return volatility can exhibit markedly different patterns in their contribution to tail risk. This highlights the importance of a decomposition-based perspective in portfolio construction—particularly in highly volatile environments such as cryptocurrencies.

Moreover, the proposed methodology strikes a practical balance between model sophistication and real-world applicability, making it suitable for both academic research and applied portfolio risk management. While future work may extend this framework through dynamic or long-memory specifications, its current form already offers a solid foundation for analysing nonlinear dependencies and time-scale-specific risk patterns.

References

[1]. Corbet, S., Lucey, B., Urquhart, A., & Yarovaya, L. (2018). Cryptocurrencies as a Financial Asset: A Systematic Analysis. International Review of Financial Analysis, 62, 182–199.

[2]. G. Aggarwal, V. Patel, G. Varshney, K. Oostman, Understanding the social factors affecting the cryptocurrency market, arXiv preprint, arXiv:1901.06245, 2019.
[3]. T. Ankenbrand, D. Bieri, Assessment of cryptocurrencies as an asset class by their characteristics, Investment Management & Financial Innovations, Vol. 15, No. 3, 2018, pp. 169–180.
[4]. E. Demir, G. Gozgor, C. K. M. Lau, S. A. Vigne, Does economic policy uncertainty predict Bitcoin returns? an empirical investigation, Finance Research Letters, Vol. 26, 2018, pp. 145–149.
[5]. L. Catania, S. Grassi, F. Ravazzolo, Forecasting cryptocurrencies under model and parameter instability, International Journal of Forecasting, Vol. 35, No. 2, 2019, pp. 485–501.
[6]. P. Krugman, Bubble, bubble, fraud and trouble, The New York Times, 29 January 2018. (Available at: https://www.nytimes.com/2018/01/29/opinion/bitcoin-bubble-fraud.html)
[7]. Popken, B. (2018). Bitcoin falls below $6,000 as cryptocurrency slump deepens. NBC News. https://www.nbcnews.com/tech/tech-news/bitcoin-falls-below-6-000-cryptocurrency-slump-deepens-n848821 (nbcnews.com in Bing)
[8]. P. Makadiya, Cryptocurrency market down 54% in Q1 2018, losses top $500 billion, CryptoGlobe, 2018. (Available at: https://www.cryptoglobe.com)
[9]. A. Iinuma, Why is the cryptocurrency market so volatile: expert take, CoinTelegraph, 2017. (Available at: https://cointelegraph.com/news/why-is-the-cryptocurrency-market-so-volatile-expert-take)
[10]. J. Alvarez-Ramirez, E. Rodriguez, C. Ibarra-Valdez, Long-range correlations and asymmetry in the Bitcoin market, Physica A: Statistical Mechanics and its Applications, Vol. 492, 2018, pp. 948–955.
[11]. A. F. Bariviera, M. J. Basgall, W. Hasperué, M. Naiouf, Some stylized facts of the Bitcoin market, Physica A: Statistical Mechanics and its Applications, Vol. 484, 2017, pp. 82–90.
[12]. A. Phillip, J. S. Chan, S. Peiris, A new look at cryptocurrencies, Economics Letters, Vol. 163, 2018, pp. 6–9.
[13]. I. Basile, P. Ferrari, Asset Management and Institutional Investors, Springer International Publishing, 2016.
[14] Summinga-Sonagadu, R., & Narsoo, J., Risk Model Validation: An Intraday VaR and ES Approach Using the Multiplicative Component GARCH. Risks, 2029, 7(1), 10.
[15]. R. F. Engle, Autoregressive conditional heteroscedasticity with estimates of the variance of UK inflation, Econometrica, Vol. 50, No. 4, 1982.
[16]. T. Bollerslev, Generalized autoregressive conditional heteroskedasticity, Journal of Econometrics, Vol. 31, No. 3, 1986, pp. 307–327.
[17]. P. Katsiampa, Volatility co-movement between Bitcoin and Ether, Finance Research Letters, Vol. 24, 2018, pp. 24–29.

[18]. E. Bouri, N. Jalkh, P. Molnár, D. Roubaud, Bitcoin for energy commodities before and after the December 2013 crash: diversifier, hedge or safe haven?, Applied Economics, Vol. 49, No. 50, 2017, pp. 5063–5073.
[19]. E. Bouri, P. Molnár, G. Azzi, D. Roubaud, L. I. Hagfors, On the hedge and safe haven properties of Bitcoin: is it really more than a diversifier?, Finance Research Letters, Vol. 20, 2017, pp. 192–198.
[20]. N. Aslanidis, A. F. Bariviera, O. Martínez-Ibañez, An analysis of cryptocurrencies conditional cross correlations, Finance Research Letters, Vol. 31, 2019, pp. 130–137.
[21]. G. Boako, A. K. Tiwari, D. Roubaud, Vine copula-based dependence and portfolio value-at-risk analysis of the cryptocurrency market, International Economics, 2019, pp. 39–53.
[22] Saha, S., Goudarzi, H., & Gupta, R., Modelling the co-movement of cryptocurrencies: A high-dimensional vine copula approach. Physica A: Statistical Mechanics and its Applications, 512, 2018, pp. 405–421.
[23]. D. B. Nelson, Conditional heteroskedasticity in asset returns: a new approach, Econometrica, Vol. 59, No. 2, 1991, pp. 347–370.
[24]. M. Shaker Ahmed, A. A. El-Masry, A. I. Al-Maghyereh, S. Kumar, Cryptocurrency volatility: a review, synthesis, and research agenda, Research in International Business and Finance, 2024.
[25]. V. Jeleskovic, C. Latini, Z. Younas, M. Al-Faryan, Cryptocurrency portfolio optimization: utilizing a GARCH-copula model within the Markowitz framework, Journal of Corporate Accounting & Finance, Vol. 35, 2024, pp. 139–155.
[26]. P. Bruhn, E. Dietmar, Assessing the risk characteristics of the cryptocurrency market: a GARCH-EVT-Copula approach, Journal of Risk and Financial Management, Vol. 15, 2022, Article 346.
[27]. T. Ndlovu, D. Chikobvu, The GARCH-EVT-Copula approach to investigating dependence and quantifying risk in a portfolio of Bitcoin and the South African Rand, Journal of Risk and Financial Management, Vol. 17, No. 11, 2024, Article 504.
[28]. W. Lan, A dynamic spillover effect investigation on cryptocurrency market before and after pandemic, arXiv preprint, arXiv:2412.19983, 2024.
[29] Sironi, A., & Resti, A Risk Management and Shareholders' Value in Banking: From Risk Measurement Models to Capital Allocation Policies. Chichester: *Wiley*, 2007.
[30]. P. Christoffersen, Elements of Financial Risk Management, Elsevier Science, 2012.
[31]. P. R. Hansen, Z. Huang, H. H. Shek, Realized GARCH: a joint model for returns and realized measures of volatility, Journal of Applied Econometrics, Vol. 27, No. 6, 2012, pp. 877–906.
[32]. T. G. Andersen, T. Bollerslev, F. X. Diebold, P. Labys, Modeling and forecasting realized volatility, Econometrica, Vol. 71, No. 2, 2003, pp. 579–625.

[33]. J. A. Lopez, Methods for evaluating value-at-risk estimates, Economic Review, No. 2, 1999, pp. 3–17.
[34]. A. L. Delatte, C. Lopez, Commodity and equity markets: some stylized facts from a copula approach, Journal of Banking & Finance, Vol. 37, No. 12, 2013, pp. 5346–5356.
[35]. A. J. Patton, Copula-based models for financial time series, in Handbook of Financial Time Series, Springer, Berlin, Heidelberg, 2009, pp. 767–785.
[36]. P. Jaworski, F. Durante, W. K. Härdle, T. Rychlik (eds.), Copula Theory and its Applications, Springer, 2010.
[37]. F. Durante, C. Sempi, Principles of Copula Theory, Chapman and Hall/CRC, 2015.
[38] Embrechts, P., McNeil, A., Straumann, D., Correlation and Dependence in Risk Management. ETH Zürich, 1999, Preprint No. 1999-10.
[39] Frank, M. J. On the simultaneous associativity of F(x, y) and x + y – F(x, y). Aequationes Mathematicae, 19, 1979, 194–226.
[40]. H. Joe, Families of m-variate distributions with given margins and m(m–1)/2 bivariate dependence parameters, Lecture Notes-Monograph Series, Vol. 28, 1996, pp. 120–141.
[41]. M. Hofert, I. Kojadinovic, M. Maechler, Y. Yan, J. G. Neslehová, R Package "copula", CRAN, 2019.
[42]. R. B. Nelsen, An Introduction to Copulas, Springer, 2007.
[43] Tasche, D. (2007). Euler Allocation: Theory and Practice. Working paper.
[44]. M. D. Braga, Risk-Based Approaches to Asset Allocation: Concepts and Practical Applications, Springer, 2015.
[45]. E. Epperlein, A. Smillie, Portfolio risk analysis – cracking VaR with kernels, Risk Magazine, Vol. 19, No. 8, 2006 (and extended version as working paper from 2016), pp. 70–75.
[46] Jarque, C. M., & Bera, A. K., Efficient tests for normality, homoscedasticity and serial independence of regression residuals. Economics Letters, 6(3), 1980, pp. 255–259.
[47]. G. E. Box, G. M. Jenkins, G. C. Reinsel, G. M. Ljung, Time Series Analysis: Forecasting and Control, John Wiley & Sons, 2015.
[48]. A. N. Kolmogorov, Sulla determinazione empirica delle leggi di probabilità, Giornale degli Istituti Italiani degli Attuari, Vol. 4, 1933, pp. 1–11 (in Italian).

3.

Anonymization with Controlled De-Anonymization in Private Blockchain

Eligijus Sakalauskas, Antanas Bendoraitis,
Aleksejus Michalkovič, Lina Dindienė and Kęstutis Lukšys

3.1. Introduction

Anonymization is a relevant topic in private blockchain, alongside other confidentiality requirements. But besides anonymization, the opposite action, named as de-anonymization, also takes place.

For example, in the Bitcoin blockchain [3] the anonymization is achieved by replacing the user's identity with a Bitcoin address computed by hashing the user's public key with the H-function. Due to the H-function one-wayness, it is infeasible to determine the user's public key from the address.

Nevertheless, such an anonymization can be potentially de-anonymized using a special technique usually used for transactions analysis of outsiders to gain hidden information [4]. Such kinds of de-anonymization we name as attacked de-anonymization. Therefore, the anonymization based on a single user address is widely adopted as pseudo-anonymization.

The greater level of anonymization is achieved using Ring Signatures (RS). Then the signature is created by one true signer, and by including a selected number of users, creating a so-called ring. The obtained signature is on behalf of all ring members without any true signer communication with other ring members. The probability of guessing the actual signer is equal to $1/N_R$, where N_R is the number of ring members.

Eligijus Sakalauskas
Kaunas University of Technology, Kaunas, Lithuania

Therefore, the attacked de-anonymization becomes N_R times more difficult.

In the Monero blockchain, Ring Signatures (RS) based on Elliptic Curve Cryptography (ECC) are used in [5-7] to realize anonymization. On the background of ECC, the Elliptic Curve Digital Signature Algorithm (ECDSA) was created and standardized [8]. Nevertheless, such anonymity can be interpreted as partial anonymity on the Network, since the actual transaction signer is inevitably among the other ring signers who are members of the ring created by that signer. A shortcoming of Monero RS-based anonymity is that it should be performed during transaction signing, which requires additional computational resources. To increase this kind of anonymity, it is required to increase the number of ring members. At the same time, computational resources are increasing.

Therefore, there is a need to develop a more anonymous transaction system that requires fewer computational resources.

However, there are also situations in which the opposite process, such as de-anonymization, is required for business processes. For example, an income from anonymous transactions should be redirected to the Investment Company (IC) by revealing the identity of the cryptocurrency owner.

In contrast to the attacked de-anonymisation, we call this kind of anonymization controlled de-anonymization. Further, we deal only with this kind of de-anonymization and omit the word *controlled*.

In the case of Monero RS, de-anonymization does not require additional computational resources because all funds are held in the transaction creator's account.

The other approach to anonymization is the stealth addresses used in the Monero blockchain. Stealth addresses are a privacy-enhancing feature in the cryptocurrency space that allows users to send and receive funds without exposing their main wallet addresses. It is based on unique address generation. Each transaction is associated with a unique one-time address, ensuring that the recipient's identity remains private.

This approach requires two private-public key pairs for every user. One private key is named a *view key,* and the other is a *spend key*. To make

receive address anonymous, the Diffie-Hellman type non-interactive session key agreement protocol (KAP) is used. Therefore, this type of anonymity we can name as session-anonymity since it is realized during the cryptocurrency transfer session. It requires more interaction with the cryptocurrency sender and receiver and a knowledge of two receiver's public keys. The shortcoming of this approach is that to realize anonymization for transaction with more than one output, Monero uses the output index. So, this increases the computation resources.

Secure de-anonymization in Monero of this kind for a certain third party can be provided by presenting the *view keys*. In the case of multiple transaction outputs, the multiple view keys must be presented.

We do not consider session-anonymization and de-anonymization since we concentrate on methods that do not require additional interaction between the sender and receiver.

In contrast to traditional anonymous signature schemes, our work does not aim to replace or improve the anonymity guarantees provided by group signatures or ring signatures. Instead, it introduces a controlled proof-of-ownership and voluntary deanonymization workflow that enables users to selectively demonstrate ownership of chosen pseudonymous addresses and their associated funds without revealing unrelated addresses or transaction history. Traditional group signatures rely on a centralized opening authority capable of revoking signer anonymity, creating an inherent trade-off between privacy and traceability [9-11]. Linkable and traceable ring signatures extend conventional ring signatures by introducing mechanisms for signature linkability and traceability that prevent malicious actions, such as double signing, while preserving the anonymity of honest users during normal operation [11-13]. Similarly, practical mechanisms such as Monero's view-key disclosure, proof-of-reserve protocols, and selective-disclosure credentials represent alternative approaches to controlled information disclosure.

However, these approaches are generally not designed to provide fine-grained, owner-controlled proofs of ownership for selected blockchain addresses and their corresponding balances. Unlike group signatures and traceable ring signatures, which balance anonymity with authority or protocol-controlled traceability, our approach introduces no tracing authority or tracing mechanism. Instead, it enables users themselves to voluntarily establish and disclose the linkage among

selected pseudonymous blockchain addresses solely for verification purposes. Consequently, the proposed Schnorr-based multi-address construction employing aggregated multisignatures and non-interactive zero-knowledge proofs (NIZKPs) complements existing anonymous signature schemes by supporting a fine-grained, user-controlled proof-of-ownership workflow that they are not specifically designed to provide.

In this chapter, we present an alternative approach to achieve a more natural anonymity based on the natural ability to create as many accounts and addresses for every user as he/she wish, using Schnorr signature to sign every anonymous (or not anonymous) transaction. We name such anonymity as multi-anonymity. This work substantially extends the methods and results presented in our previous publication [1].

The de-anonymization is computationally effective and is based on the Schnorr aggregated-signature together with the additional Schnorr signature required to sign the Schnorr aggregated-signature.

To summarize the anonymization methods, we can name the following:

1. Pseudo-anonymization;
2. Ring-anonymization;
3. Session-anonymization;
4. Multi-anonymization.

The usage of these terms will be clear from the context.

We consider a blockchain system based on the Unspent Transactions Output (UTxO) paradigm [14]. According to this paradigm, the transaction Tx consists of inputs defining incomes of cryptocurrencies (CC) to the receiver address Addr and outputs defining CC expenses. One example of UTxO is the Bitcoin blockchain, where the sum of transaction incomes (In) must be equal to the sum of transaction expenses (Ex) or the sum of transaction inputs must be equal to the sum of transaction outputs. This guarantees the trustworthiness of transactions on the Net. We can see in Fig. 3.1 that Alice created a transaction with 4 incomes and 3 expenses. UTxO mechanism has valuable properties:

1. It guarantees the integrity of transactions, ensuring a balance between the incomes and expenses;

2. It implements the so-called divisibility of cryptocurrency, ensuring that the change from unspent transactions is returned to the sender.

In an example presented in Fig. 3.1, the UTxO mechanism provides a balance between incomes and expenses in 100 BTC:

$$In_1 + In_2 + In_3 + In_4 = Ex_1 + Ex_2 + Ex_3 = 100 \text{ BTC} \quad (3.1)$$

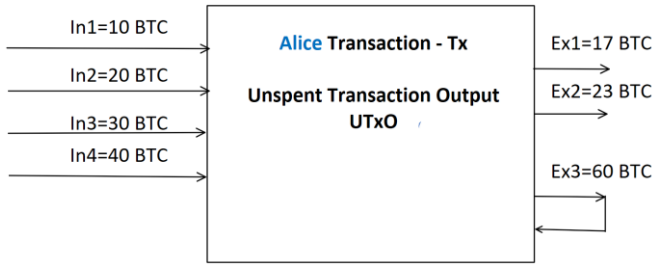

Fig. 3.1. Unspent Transactions Output (UTxO) mechanism example.

In this case, the change is 60 BTC and this sum Alice transfers to herself.

In Section 3.2, the general foundation of digital signatures, Schnorr digital signatures, Elliptic Curve Digital Signature Algorithm (ECDSA), and Non-Interactive Zero Knowledge Proof (NIZKP) are outlined. An anonymization-de-anonymization method based on Monero Ring Signatures (RS) is presented in Section 3.3. In Section 3.4, we present anonymization-de-anonymization methods based on Schnorr signature and Schnorr aggregated-multisignature. An anonymization-de-anonymization method based on the Elliptic Curve Schnorr Signature Algorithm (ECSSA) is presented in Section 3.5. In Section 3.6, the example is presented. Section 3.7 contains security considerations, and conclusions are presented in Section 3.8.

The list of abbreviations is presented in Table 3.1 below.

3.2. Digital Signatures and Proof of Knowledge

Digital signatures, like many other classical cryptographic methods, are defined in either multiplicative or additive (in the case of Elliptic Curve) groups. The cryptographic conjectured One-Way Function (OWF) is

introduced in these groups by defining a Discrete Exponent Function (DEF).

Table 3.1. The list of abbreviations.

Abbreviation	**Meaning**
Addr	Blockchain account address
CC	Crypto Currency
DEF	Discrete Exponent Function
DLF	Discrete Logarithm Function
DLP	Discrete Logarithm Problem
ECC	Elliptic Curve Cryptography
EC	Elliptic Curve
ECSSA	Elliptic Curve Schnorr Signature Algorithm
ECG	An elliptic curve group consisting of EC points with a special addition operation
ECDSA	Elliptic Curve Digital Signature Algorithm
Ex	Expenses
FIPS	Federal Information Processing Standards
IC	Investment Company
In	Incomes
N_{EC}	Number of EC points
NIZKP	Non-Interactive Zero Knowledge Proof
NIST	National Institute of Standards and Technology
PKC	Public Key Cryptography
PP	Public Parameters
PrK	Private Key
PuK	Public Key
RS	Ring Signature
Tx	Transaction
UTxO	Unspent Transactions Output
Notation	**Algebraic structures**
$\mathbb{F}_p$	Finite field of characteristic p: $\{0,1,2,3,\dots,p-1\}$ with operations mod p
G	Generator or base point of ECG
$\mathbb{G}_q$	Prime order subgroup of group $\mathbb{Z}_p^*$ where q is a prime divider of $(p-1)$
g	Generator in $\mathbb{Z}_p^*$ or in G_q
p	Prime number defining characteristic of F_p
q	Prime divisor of $(p-1)$: $p = 2q + 1$
$\mathbb{Z}_p^*$	Multiplicative group of integers: $\{1,2,3,\dots,p-1\}$ with operation modulo p.
$\mathbb{Z}_{p-1}$	Ring of integers: $\{0,1,2,3,\dots,p-2\}$ with operations mod $(p-1)$

A lot of cryptographic methods are based on the multiplicative group $\mathbb{Z}_p^*$ (Table 3.2) or on a prime-order subgroup $\mathbb{G}_q$ in $\mathbb{Z}_p^*$, where $p = kq + 1$ for some integer *k*.

Table 3.2. DEF parameters in $\mathbb{Z}_p^*$.

Multiplicative Group $\mathbb{Z}_p^*$
$\mathbb{Z}_p^* = \{1, 2, 3, \dots, p-1\}$, when p is a large prime of order 2^{2048}.
Operation: multiplication mod *p*
Neutral element is 1.
Generator g: $\mathbb{Z}_p^* = \{g^i, \mathrm{i} = 0,1,2,3, \dots, p-2\}$ A criterion to find a small $g \ll p$ when $p = 2q + 1$, where q is prime, is $g^q \neq 1 mod\ p$. For larger generators, we can calculate $g' = g^\alpha$, where $gcd(\alpha, p-1) = 1$.
DEF: $dexp_g(x) = g^x \bmod p$, where $x \in \mathbb{Z}_{p-1}$.

Definition. The DEF base a generator g in $\mathbb{Z}_p^*$ is a one-to-one mapping $dexp_g(x)$: $\mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$, which assigns an element $h \in \mathbb{Z}_p^*$ such that $g^x \equiv h\ mod\ p$ to any $x \in \mathbb{Z}_{p-1}$.

DEF can be computed effectively by applying the squaring algorithm for any value of the prime *p* [15].

Due to Little Fermat's theorem, the exponent $x \in \mathbb{Z}_{p-1}$, and hence all the operations in the exponent are performed modulo $p - 1$.

Since $dexp_g(x)$ is a one-to-one mapping, we can define its inverse as follows:

Definition. The Discrete Logarithm Function (DLF) based on a generator g in $\mathbb{Z}_p^*$, is a one-to-one mapping $dlog_g(h)$: $\mathbb{Z}_p^* \to \mathbb{Z}_{p-1}$ which assigns an element $x \in \mathbb{Z}_{p-1}$ such that $g^x \equiv h(\mathrm{mod} p)$. to any $h \in \mathbb{Z}_p^*$. We denote it as $x = dlog_g(h)$.

Notably, $dexp_g(x)$ can be calculated efficiently. In fact, due to the squaring algorithm, the complexity is $\mathcal{O}(log\, x)$ arithmetic operations, i.e., proportional to the size of x in bits, since any exponent x can be expressed as a bitstring. However, there are no non-quantum algorithms to efficiently calculate the inverse mapping $dlog_g(h)$ in the general case. The time complexity of the best-known non-quantum algorithms is $\mathcal{O}(\sqrt{p})$, which is sub-exponential in the bit size of the prime p. For this reason, DEF is an example of the so-called one-way functions (OWF): easy to compute, yet hard to reverse. Hence, the security of cryptographic DEF-based algorithms relies on the complexity of the Discrete Logarithm Problem (DLP). To put it simply, the problem is to find $x = dlog_g(h)$ in $\mathbb{Z}_{p-1}$, given a generator g and an element $h \in \mathbb{Z}_p^*$.

Definition. The DLP is computationally hard, if there is no efficient polynomial-time algorithm which the adversary could use to gain a statistically significant advantage in finding $x = dlog_g(h)$ as compared to random guessing.

Definition. The computational discrete logarithm assumption (DLA) states that the DLP is computationally hard in $\mathbb{Z}_p^*$.

The computational DLA is the basis of the security of cryptographic DEF-based algorithms.

To improve security, usually DEF is defined in the subgroup $\mathbb{G}_q$, where all elements are generators of order q. In this case, every non-identity element g is the generator of $\mathbb{G}_q$, and exponents x are in the ring $\mathbb{Z}_q = \{0,1,2,\dots,q\}$. This means that all operations in exponents are performed modulo q.

In asymmetric cryptography, every party has a unique key pair consisting of a private key PrK and a public key PuK. For example, in Fig. 3.2, Alice has a key pair denoted by $PrK_A = x$ and $PuK_A = a$.

To sign any message using any asymmetric signature method, the H-function denoted by $H()$ is used to compute the h-value on this message. Typically, this value is signed to avoid existential forgeries.

Signature σ is computed using the signer's Alice private key is denoted by $PrK_A = x$ on h-value and consists of two components $(r, s) = \sigma$. To avoid chosen message attacks, any asymmetric signature algorithm

must contain randomness, hence making signature schemes probabilistic [16].

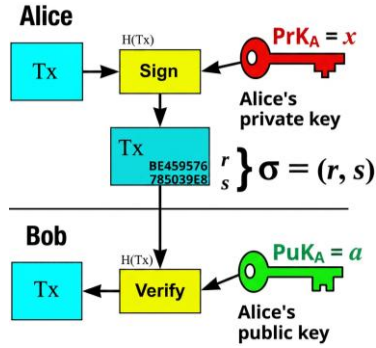

Fig. 3.2. Symbolical signature generation and verification.

This is done by introducing a random session key $k \leftarrow rand(\mathbb{S})$ in the signing algorithm $Sign(h, x, k)$, where $\mathbb{S}$ is a domain of all possible keys. This session key must be generated anew every time a message is signed and discarded afterwards. Failing to do so poses a critical threat to the signature scheme as the signer's key is immediately exposed.

The verification of the signature on h-value h is performed by using the signer's public key $PuK_A = a$ and the signature σ. Verification algorithm $Ver()$ is defined in the following way: $Ver(\sigma, h, PuK_A) = \{\text{True}, \text{False}\}$, which satisfies the property $Ver(Sign(h, x), h, a) = \text{True}$, where (x, a) is a valid key pair and $Sign(h, x)$ is a signature on h. In other words, the verification algorithm outputs True for every valid signature, and it outputs False for every forged signature.

In Fig. 3.2 symbolical signature generation and verification., Alice is a signer and signs a hashed transaction Tx denoted by $H(Tx)$. Tx together with a signature σ is sent to the verifier Bob, who uses Alice's public key $PuK_A = a$ for verification.

3.2.1. Schnorr Digital Signature Scheme

Schnorr digital signature is based on the DEF defined in G_q. In the Schnorr signature scheme, public parameters (PP) consist of the following triplet

$$PP = (p, q, g) \tag{3.2}$$

As mentioned above, to improve the security level of the Schnorr signature, it is defined over the Sylow subgroup $\mathbb{G}_q \in \mathbb{Z}_p^*$, where q is prime. Then all non-identity elements in G_q are its generators. To ensure security against cryptanalysis by classical computers, public parameters p and q are chosen sufficiently large, i.e., p has a 2048-bit length, and q having an order of about 1800-bit length. A Sophie Germain prime q is a reasonable choice since a prime $p = 2q + 1$ is easy to compute and is as small as it can be.

Let $g \in \mathbb{G}_q$ be a generator in $\mathbb{G}_q$, then the order of the generator g is q. This means that the operations of exponents of elements in $\mathbb{G}_q$, are computed modulo q, and exponents are in the set $\mathbb{Z}_{p-1}$. The algorithm $KeyGen(PP)$ uses the public parameters to generate a valid key pair (PrK, PuK) as follows:

$$\text{PrK} = x \leftarrow rand(\mathbb{Z}_q), \tag{3.3}$$

$$\text{PuK} = g^x \, mod \, p = a \tag{3.4}$$

Let the message be a transaction Tx to be signed by Alice and sent to Bob. The Schnorr signature algorithm $Sign(h, x)$ uses an H-function $H(x)$ to create a signature on Tx as follows:

$$k \leftarrow rand(\mathbb{Z}_q), \tag{3.5}$$

$$r = g^k mod p, \tag{3.6}$$

$$h = H(Tx \parallel r), \tag{3.7}$$

$$s = (k + x \cdot h) mod (p - 1) \tag{3.8}$$

The algorithm $Sign(h, x, k)$ outputs a signature (r, s), i.e. $Sign(h, x, k) = (r, s)$, where its components are defined by 6, 8 respectively.

Notably, the Schnorr signature algorithm $Sign(h, x, k)$ integrates a secure hash function within the calculation of the output. This is crucial, since it helps to avoid existential forgeries as opposed to the classical ElGamal e-signature scheme, where, despite the randomness component k, the hash function must be additionally integrated in the second

component s of signature. This ensures the strongest notion of security – unforgeability under an adaptive chosen message attack.

The verification algorithm works, since referencing 6, 8, the following identities are valid:

The verification algorithm computes the following parameters:

$$v_1 = g^s mod p, \tag{3.9}$$

$$v_2 = r \cdot a^h mod p, \tag{3.10}$$

$$Ver(\sigma, h, a) = \text{True if } v_1 = v_2 \tag{3.11}$$

The verification algorithm works, since referencing 6 and 8, the following identities are valid:

$$g^s mod p = g^{(k+x\cdot h) mod q} mod p = \\ = g^{k mod q} \cdot g^{x\cdot h mod q} mod p = r \cdot a^h mod p \tag{3.12}$$

As we see, the verifications of the Schnorr digital signature require two exponentiation operations.

3.2.2. Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA was invented by D. Johnson and A. Menezes in 1999 [17].

NIST has standardized elliptic curve cryptography for digital signature algorithms in FIPS 186 [8].

In cryptography, EC is defined over the finite field F_p and consists of N points with coordinates (x, y). The set of these points constitutes an additive group named the Elliptic Curve Group (ECG). The operation in this group is the special addition of EC points, which we temporarily denote by $\oplus$. This group is an analogue of $\mathbb{Z}_p^*$where the conjectured OWF based on DEF is defined in additive form, i.e., exponentiation is replaced by multiplication by a scalar. From this analogy, the term EC DLP was introduced when considering the security of ECC algorithms. Analogously to DLP in $\mathbb{Z}_p^*$, we can define a notion of EC computational DLA.

Traditionally, the sum of two integers $a$and b is denoted by $a + b$and their multiplication by ab. To point out the difference of arithmetic operations between the integers, EC points, and mixed operations, defined below, we temporarily denote the multiplication of two integers by $a \cdot b$.

EC is defined over F_p by a certain equation relating the coordinates y to coordinate x. In ECDSA, this equation is defined by parameters $a, b \in \mathbb{F}_p$ as follows:

$$y^2 = x^3 + a{\cdot}x + b \bmod p \tag{3.13}$$

Typically, EC points are denoted by capital letters and the elements in F_p are denoted in lowercase letters. Then, for example, the multiplication of EC point G by integer z in F_p, we temporarily denote by *. It means that point $G$is added to itself z-times, and as a result, we obtain a new point A.. It is an analogy of DEF in ECG.

$$A = z * G = \underbrace{G \oplus G \ldots \oplus G}_{z \text{ times}} \tag{3.14}$$

The list of operations is presented in Table 3.3 below for the first acquaintance with EC arithmetic and ECDSA.

Table 3.3. Arithmetic operations in EC.

Operation	Comment
+	Binary addition operation of integers
·	Binary multiplication operation of integers
$\oplus$	Binary addition operation of two points in EC
*	Multiplication operation of EC point by an integer

Analogously to $dexp_g(x)$, point A in (3.14) is computed effectively by doubling the addition operations in ECG when the number of EC points N_{EC} is large [6]. This number is defined by the EC type and the characteristic of the finite field F_p. In standardized ECC methods, p *is a* 256-bit prime.

The summary of ECG and multiplication of EC points by integers is presented in Table 3.4.

Table 3.4. Operations in Elliptic Curve Group (ECG).

Elliptic Curve Group (ECG)
Number of EC points N with coordinates (x, y) is an order of ECG.
Addition operation $\oplus$ of points in ECG: let points $P(x_P, y_P)$ and $Q(x_Q, y_Q)$ are in EC with coordinates (x_P, y_P), and (x_Q, y_Q), then $P \oplus Q = T$ with coordinates $(x_T, y_{T)}$ in EC.
Neutral element is group zero 0 at infinity (∞) of $[XOY]$ plane.
DEF in ECG is a multiplication of EC point $P$by scalar z.
Generator–Base Point G: $ECG = \{i * G;\ i = 1,2, \dots, N\}, N * G = = 0$ and $q * G \neq 0$ if $q < N$.

ECDSA in Bitcoin uses the standard EC denoted by secp256k1 [18].

Then, the Public Parameters (PP) of ECDSA are the following:

$$PP = \begin{Bmatrix} \text{secp256k1};\ \text{BasePoint-Generator } G; \\ \text{prime } p;\ \text{parameters } a, b \in F_p \end{Bmatrix} \tag{3.15}$$

Referencing Fig. 3.2, Alice uses the $KeyGen(PP)$ algorithm to generate a valid key pair as follows:

$$PrK_A = z \leftarrow rand(F_p), \tag{3.16}$$

$$PuK_A = A = z * G \tag{3.17}$$

Let a, b be any scalars in F_p and P be any point of EC. Then, in ECC, the following identities hold for operations with scalars $a, b \in F_p$ and points $P, Q \in EC$:

$$(a + b) * P = a * P \oplus b * P, \tag{3.18}$$

$$(h \cdot z) * A = h * (z * G), \tag{3.19}$$

$$a * (P \oplus Q) = a * P \oplus a * Q \tag{3.20}$$

Signature based on ECDSA [19] has some generalized similarities with the ElGamal signature scheme in the sense that it uses an additive EC cyclic group instead of a multiplicative group in the ElGamal cryptosystem.

Analogously to the Schnorr signature, ECDSA-based signatures on the transaction Tx are created by computing h-value h on Tx and by generating a random parameter k to provide a probabilistic signature. The signature algorithm $Sign(h, x, k)$ uses the H-function SHA256 and is realized in the following way:

$$h = H(M) = SHA256(Tx), \tag{3.21}$$

$$k \leftarrow rand(F_p), \tag{3.22}$$

$$R = k * G = (x_R, y_R); r = x_R mod p, \tag{3.23}$$

$$s = (h + z \cdot r) \cdot k^{-1} mod p \tag{3.24}$$

The output is $\sigma = (r, s)$, where its components $r$and s are defined by (3.23) and (3.24), respectively.

The ECDSA signature verification algorithm consists of the following steps.

1. Calculate $u_1 = h \cdot s^{-1} mod p$ and $u_2 = r \cdot s^{-1} mod p$;
2. Calculate the curve point $V = u_1 * G \oplus u_2 * A = (x_V, y_V)$;
3. $Ver(\sigma, h, A) = True$ if $R = V$; $r = x_V = x_R mod p$.

To prove the correctness of the verification algorithm, the identities (3.18)-(3.20) are used. By step 3 of ECDSA verification, we have:

$$R = u_1 * G \oplus u_2 * A \tag{3.25}$$

From the definition of the Public Key $A = z * G$, we have:

$$R = u_1 * G \oplus (u_2 \cdot z) * G \tag{3.26}$$

Because EC point multiplication distributes over scalar addition, we have:

$$R = (u_1 + u_2 \cdot z) * G \tag{3.27}$$

Expanding the definition of $u_1$and u_2 from the verification steps we have:

$$R = (h \cdot s^{-1} + r \cdot s^{-1} \cdot z) * G \tag{3.28}$$

Collecting the common term s^{-1}, we have:

$$R = [(h + r \cdot z) \cdot s^{-1}] * G \quad (3.29)$$

Expanding the definition of s from signature creation, we have:

$$R = [(h + r \cdot z) \cdot (h + r \cdot z)^{-1} \cdot k] * G = k * G \quad (3.30)$$

Since the inverse of an inverse is the original element, and the product of an element's inverse and the element is the identity, we are left with

$$R = k * G = (x_R, y_R); r = x_R$$

The construction based on ECDSA provides unconditional anonymity in the sense that even an infinitely powerful adversary with access to an unbounded number of chosen-message signatures produced by the same ring member cannot gain any statistically significant advantage to identify the signer as compared to random guessing, and cannot link additional signatures to the same signer.

3.2.3. Non-Interactive Zero Knowledge Proof – NIZKP

Unlike traditional interactive zero-knowledge proofs, NIZKs eliminate the need for multiple rounds of interaction, making them more efficient and scalable. NIZKP consist of two actors, Prover and Verifier, and allows Prover to prove knowledge about some secret by presenting the statement to the Verifier related to this secret without revealing any information about this secret (Fig. 3.3).

In Public Key Cryptography (PKC), every legit user has a valid key pair denoted by (PrK, PuK), and all the public keys are available to other users via open communication channels. As mentioned above, in each valid key pair, PrK and PuK are linked by a conjectured cryptographic One-Way Function (OWF). If any public key is linked with the user's identity, then NIZKP can be used to prove that this user knows his/her private key without revealing it by presenting his/her public key to the Prover. It is a principle of the identification protocol. In many cryptographic protocols, especially in identification protocols, **PrK** is named as a **witness** and **PuK** as a statement for **PrK**.

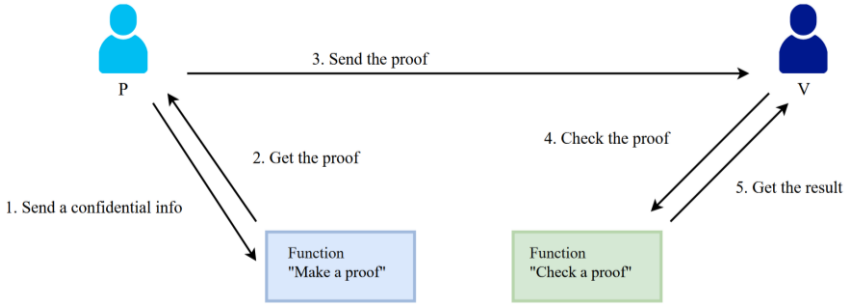

Fig. 3.3. The general idea of NIZKP.

More generally, NIZKP are the cryptographic protocols that allow a prover to demonstrate the truth of a statement to a verifier without revealing any additional information beyond the validity of the statement.

Because one should be able to generate a proof of some statement *only* when in possession of certain secret information connected to the statement, the verifier, even after having become convinced of the statement's truth by means of a zero-knowledge proof, should nonetheless remain unable to prove the statement to further third parties.

The key advantage of non-interactive zero-knowledge proofs is that they can be used in situations where there is no possibility of interaction between the prover and verifier, such as in online transactions where the two parties are not able to communicate in real time. This makes non-interactive zero-knowledge proofs particularly useful in decentralized systems like blockchains, where transactions are verified by a network of nodes, and there is no central authority to oversee the verification process.

In the case of the identification protocol, Alice wants to prove to Bob that she knows the discrete logarithm of a given value in a given group. As it was mentioned above, this discrete logarithm is the inverse function of DEF, and the problem of finding it is a DLP. The consideration of DEF was presented above in the set $\mathbb{Z}_p$ and in the set of EC points.

According to the assumption that DEF is a conjectured OWF, the secure NIZKP can be constructed. The idea of an NIZKP is similar to that of the signature scheme, in that the three-round interactive identification

protocol is transformed into a single-round non-interactive proof using the Fiat-Shamir heuristic [20].

A zero-knowledge proof of some statement must satisfy three properties:

1. **Completeness**: if the statement is true, then an honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover;
2. **Soundness**: if the statement is false, then no cheating prover can convince an honest verifier that it is true, except with some small probability;
3. **Zero-knowledge**: if the statement is true, then no verifier learns anything other than the fact that the statement is true. In other words, just knowing the statement (not the secret) is sufficient to imagine a scenario showing that the prover knows the secret. This is formalized by showing that every verifier has some *simulator* that, given only the statement to be proved (and no access to the prover), can produce a transcript that "looks like" an interaction between an honest prover and the verifier in question.

3.3. Anonymization-De-Anonymization Based on Ring Signatures (RS)

From now on, we will switch to professional notations for ECC, i.e., we denote the addition of two points $P$and Q by the usual notation $P + Q$, and the multiplication of a point P by an integer z by zP. Therefore, we use the same notation for operations in ECG and finite fields. The interpretation of the operation is clear from the context.

This anonymization method is used in the Monero blockchain, ensuring the anonymity of the creator of a transaction. RS is a type of digital signature that can be performed by any member of a group of users, each of whom has a public and private key pair [21]. Thus, a message signed using RS is verified by someone within a specific group. One of the security properties of ring signatures is that it is computationally infeasible to determine which group member's keys were used to create the signature. In RS, there is no way to revoke the anonymity of an individual signature, and any group of users can be used as a signing group without additional configuration.

Ring signatures are signer-ambiguous. In a ring signature scheme, there are no prearranged groups of users, there are no procedures for setting, changing, or deleting groups, there is no way to distribute specialized keys, and there is no way to revoke the anonymity of the actual signer (unless he decides to expose himself). Therefore, ring signatures are decentralized. They can be formed using existing public keys without requiring a central authority to initialize or manage the group.

The only assumption is that each member is already associated with the public key of some standard signature scheme.

To produce a ring signature, the actual signer declares an arbitrary set of possible signers that includes himself and computes the signature entirely by himself using only his private key and the others' public keys.

In particular, other possible signers may have chosen their private keys solely to conduct e-commerce over the Internet. They may be completely unaware that a stranger uses their public key to produce a ring signature on a message they have never seen and would not wish to sign. Due to this reason, the unforgeability of an RS is an important property: it is computationally infeasible for anyone outside the designated group (or an insider without a valid private key) to construct a valid signature. The signature proves that someone with access to one of the group's private keys authorized the message, but it prevents impersonation.

A set of possible signers is named a ring. The ring member who produces the actual signature is the signer, and each of the other ring members is a non-signer.

A ring signature scheme is set-up free: The signer does not need the knowledge, consent, or assistance of the other ring members to put them in the ring - all he needs is knowledge of their regular public keys. The size of the signature depends on the number of ring members.

Verification must satisfy the usual soundness and completeness properties. It is required that the signatures be signer-ambiguous in the sense that the verifier should be unable to determine the identity of the actual signer in a ring of size r with probability greater than $\frac{1}{r}$.

In Monero, RS is implemented using elliptic curve cryptography [22]. These curves are defined over a finite field $\mathbb{F}_p$. All the usual operations

like addition, multiplication, division, and so on are performed modulo p.

From now on, we turn to professional notations for ECC. For example, we denote the addition of two EC points P and Q by the common operation $P + Q$, and we denote the point P multiplication by an integer z as zP. Integers a and b addition is denoted by $a + b$ and we denote their multiplication by ab. In this section, we will follow the notations used in the Monero blockchain.

As it was mentioned above, elliptic curves are defined as the set of points (x, y) in $\mathbb{F}_p \times \mathbb{F}_p$satisfying a Weierstraß equation:

$$y^2 = x^3 + ax + b, \tag{3.31}$$

where $a, b, x, y \in \mathbb{F}_p$.

However, the cryptocurrency Monero uses a special curve known to offer improved security over other commonly used NIST curves, as well as excellent performance of cryptographic primitives. The curve used belongs to the category of so-called Twisted Edwards curves, which are commonly used [23].

In EC, the special points addition is defined, creating an abelian group of these points. A generator G of EC is a point on the curve such that for every other point P in EC there exists k such that $P = kG$.

Public key cryptography algorithms can be devised in an analogous way to modular arithmetic.

Let k be a randomly selected number satisfying $1 < k < N_{EC}$, where N_{EC} is a number or *EC* points. Number k is a *private session key*. Then the corresponding *public session key* $K = kG$, where *K* is an *EC* point.

Typically, a cryptographic signature is performed on a cryptographic hash of a message.

3.3.1. Signature Creation in Monero

Assume that Alice has the private/public key pair (k, K). To sign an arbitrary message M univocally, she could execute the following steps [6]:

1. Calculate a hash of the message using a cryptographically secure hash function, $h = H(M)$;
2. Generate a random integer r such that $1 < r < N$ and compute $P = (x, y) = rG$. If $x = 0$ generate another random integer;
3. Calculate $s = r^{-1}(h + xk)(mod\ N)$. If $s = 0$, then go to the previous step and repeat;
4. The signature $\sigma = (x, s)$.

3.3.2. Signature Verification in Monero

Any third party can verify the signature by calculating

$$u_1 = s^{-1}h, \tag{3.32}$$

$$u_2 = s^{-1}r, \tag{3.33}$$

$$Q = u_1G + u_2K \tag{3.34}$$

3.3.3. Correctness

The correctness of the scheme can be derived from the fact that

$$Q = u_1G + u_2K = s^{-1}hG + s^{-1}rkG = s^{-1}(h + xk)G \tag{3.35}$$

Since $s = r^{-1}(h + xk)$, it follows that $r = s^{-1}(h + xk)$, whereby it is proved that

$$Q = rG \tag{3.36}$$

3.3.4. Anonymization

In the Monero blockchain, the anonymization is performed using ring signatures (RS) presented in [22].

The simplified version of this scheme is presented below, assuming that we have the same number of keys for any value of the first index i. Assume that we have a set of public keys $\{K_{i,j}\}$ for $i \in \{1, 2, \dots, n\}$ and $j \in \{1, 2, \dots, m\}$. Furthermore, we also assume that for each *i,* there is an

index π_i such that the signer knows the private key k_{i,π_i} corresponding to K_{i,π_i}.

In what follows, we will use *M* for the hash of the message concatenated with keys $K_{i,j}$.

3.3.5. Ring Signature

1. For each $i = 1, \dots, n$:
   (a) generate a random value $\alpha_i \in_R \mathbb{Z}_q$;
   (b) set $c_{i,\pi_i} = H_n(M, \alpha_i G, i, \pi_i)$;
   (c) for $j = \pi_i + 1, \dots, m-1$ generate random numbers $r_{i,j} \in_R \mathbb{Z}_q$ and compute

$$c_{i,j+1} = H_n\big(M, r_{i,j}G - c_{i,j}K_{i,j}, i, j\big); \quad (3.37)$$

2. For $i = 1, \dots, n$ generate random numbers $r_{i,m} \in_R \mathbb{Z}_q$ and compute

$$c_1 = H_n\big(r_{1,m}G - c_{1,m}K_{1,m}, \dots, r_{n,m}G - c_{n,m}K_{i,m}\big); \quad (3.38)$$

3. For $i = 1, \dots, n$:
   (a) for $j = 1, \dots, \pi_i - 1$ generate random numbers $r_{i,j} \in_R \mathbb{Z}_q$ and compute

$$c_{i,j+1} = H_n\big(M, r_{i,j}G - c_{i,j}K_{i,j}, i, j\big) \quad (3.39)$$

   Here, we interpret references to $c_{i,1}$ as c_1, see the previous step;
   (b) set $r_{i,\pi_i} = \alpha_i + k_{i,\pi_i}c_{i,\pi_i}$

   The ring signature is

$$\sigma_{RS} = \big(c_1, r_{1,1}, r_{1,2}, \dots, r_{1,m}, \dots, r_{n,m}\big) \quad (3.40)$$

3.3.6. Ring Signature Verification

The verification of a given signature is performed as follows:

1. For each $i = 1, \dots, n$ and $j = 1, \dots, m$ compute:

$$R'_{i,j+1} = r_{i,j}G - c'_{(i,j)}K_{i,j}, \quad (3.41)$$

$$c'_{i,j+1} = H_n\big(M, R'_{i,j+1}, i, j\big) \quad (3.42)$$

Interpret any $c'_{i,1}$ as c_1;

2. Compute $c'_i = H\left(R'_{1,m}, \dots, R'_{n,m}\right)$.

The signature will be valid if $c'_1 = c_1$.

3.3.7. Correctness

1. For $j \neq \pi_i$ and for all i, we can readily see that $c'_{i,j+1} = c_{i,j+1}$;
2. When $j = \pi_i$, for all i

$$R'_{i,j+1} = r_{i,j}G - c'_{i,j}K_{i,j} =$$
$$= (\alpha_i + ki, \pi_i c'_i, \pi_i)\, G - c'_i, \pi_i K_i, \pi_i \qquad (3.43)$$

In other words, $c'_{i,\pi_i} = H_n(M, \alpha_i G, i, \pi_i) = c_{i,\pi+1}$.

Therefore, we can conclude that the verification step identifies correctly valid signatures.

3.3.8. De-Anonymization

De-anonymization is performed trivially. Alice simply transfers a required sum from her account to IC and signs her transaction with her private key without using any other public keys required for RS creation.

3.3.9. Efficiency

Most of the proposed algorithms have an asymptotic output size $O(n)$, i.e., the size of the resulting signature increases linearly with the size of the input (number of public keys) corresponding to the number of chosen ring members. The computational cost is estimated by identifying the operations that require significant computational resources. Such an operation in ECC is a multiplication of an EC point by the number, i.e., rG, where r is a number in the field $\mathbb{F}_p$, and G is an EC generator. This operation requires a significant computational resource and, in some sense, is analogous to the exponentiation operation used in the classical ElGamal cryptosystem. The number of EC point multiplications by scalars in $\mathbb{F}_p$ required for RS implementation is denoted by N_{ECO}. According to the construction presented in Section 3.3.5, this number depends on the number *n* of ring members and is equal to

$$N_{RS} = 4n^3 \tag{3.44}$$

As we see, to provide more anonymity, a greater number of ring members is needed, and cubically more of EC point multiplication operations of EC points are required.

The other drawback is that these operations must be carried out during the transaction execution, i.e., online.

3.4. Anonymization-De-Anonymization Based on the Schnorr Signature Approach

This approach is based on the Schnorr signature [24] for anonymization and Schnorr aggregated-multisignature [25] for de-anonymization. As usual, every user has a private and public key pair, which we denote by (PrK, PuK) for signature creation and verification, respectively. Assume that Alice generated $PrK = x$ and $PuK = a$, that are linked with the Alice identity, e.g., she has a certificate for her *Puk* = a. Every user of blockchain can generate as many (PrK, PuK) pairs as required for anonymization.

In many signatures schemes, the standard cryptographic collision-resistant H-function function named SHA256 is used [26].

Let M be a message to be signed. Then the h-value for this message, which we denote by h, is symbolically expressed in the following way

$$h = H(M) \tag{3.45}$$

Assume that i-th blockchain account address is computed traditionally, i.e., using the H-function. Symbolically, we denote it by

$$Addr_i = H(\mathrm{Puk}_i) \tag{3.46}$$

Then, according to the material presented below, Alice should be able to prove to anyone that this account belongs to her.

3.4.1. Anonymization Based on Schnorr Signature

To ensure anonymity, Alice generated *N* valid key pairs (PrK_1, PuK_1), $(PrK_2, PuK_2), \dots, (PrK_N, PuK_N)$. Then she needs to create *N* accounts

and addresses. This simple procedure conceals the true identity of the account owner.

Let Alice wish to be anonymous on the Net, except to her business partners, by generating at random N pairs of public and private keys $\{PrK_i, PuK_i\}$, $i = 1, 2, 3, \ldots, N$, to create N anonymous accounts.

First, she generates a set of $N-1$ private keys $PrK_{n-1} = x_{n-1}, n = 2, 3, \ldots, N$.

The last private key x_N is computed by the expression

$$x_N = x - x_1 - x_2 - \ldots - x_N - 1 \, mod \, q \qquad (3.47)$$

Hence, evidently, we have:

$$PrK = x = x_1 + x_2 + + \ldots + x_N \, mod \, q \qquad (3.48)$$

Then, using (3.3), (3.4) and (3.46) a set of public keys $\{PuK_i\}$ and addresses $\{Addr_i\}$, $i = 1,2,3, \ldots, N$, are computed.

The set of addresses and public keys is known to the nodes of the Net. But the Net does not know that these sets belong to Alice. To ensure anonymity, Alice uses her anonymous addresses to conduct transactions between her partners. All transactions are signed using the Schnorr signature scheme.

To be self-contained, we present the Schnorr signature scheme to sign the transaction $T\mathrm{x}_i$

$$j_i \leftarrow \mathrm{rand}(Z_q), \qquad (3.49)$$

$$r_i = g^{j_i} mod \, p, \qquad (3.50)$$

$$h_i = H(Txi||r_i), \qquad (3.51)$$

$$s_i = j_i + x_i \cdot h_i \, mod \, q \qquad (3.52)$$

Alice's signature on Txi is $\sigma_i = (r_i, s_i)$.

The verification of the signature σ_i is performed with Alice $\mathrm{PuK}_i = a_i$ by verifying the following identity

$$g^{s_i} \bmod p = r_i \cdot (a_i)^{h_i} \bmod p \quad (3.53)$$

All nodes in the net can verify the validity of the transaction signature by using (3.53) and acquiring the public key a_i. This verification is secure under the DL assumption in the random oracle model.

For more clarity, let us assume that Alice created two anonymous account addresses $\{Addr_1, Addr_2\}$, corresponding to ($PrK_1 = x_1$, $PuK_1 = a_1$) and ($PrK_2 = x_2$, $PuK_2 = a_2$) respectively.

Let Alice decide to invest some income in her addresses $\{Addr_1, Addr_2\}$ to the Investment Company (IC). Then she creates two transactions $Tx1, T\text{x}2$, where investment sums are outputs-expenses of these transactions. These transactions are signed by Schnorr signature using Alice's private keys $PrK_1 = x_1$ and $PrK_2 = x_2$ generated by her in advance. Using (3.49) – (3.52), two signatures are computed:

$$\sigma_1 = (r_1, s_1), \sigma_2 = (r_2, s_2) \quad (3.54)$$

IC, together with the received sums in $T\text{x}1$ and $T\text{x}2$, verifies signatures on these transactions using (3.53). But IC does not know that these transactions are sent from Alice's anonymous accounts. IC can also believe that it receives funds from the different investors.

3.4.2. De-Anonymization Based on the Schnorr Signature

Following (3.47) and (3.48), when $N = 2$, the following equations hold

$$x = x_1 + x_2 \bmod q, \quad (3.55)$$

$$g^x = g^{x_1 + x_2} = a_1 \cdot a_2 = a \bmod p \quad (3.56)$$

Then, having two signatures (σ_1, σ_2) Alice computes the multiplication of these signatures in a special way, yielding a Schnorr aggregated-multisignature. This special multiplication operation, we denote by $*$. As a result, the Schnorr aggregated-multisignature is computed in the following way:

$$\sigma_{12} = (\sigma_1 * \sigma_2) = (r_1, s_1) * (r_2, s_2) =$$
$$= (r_1 \cdot r_2 \bmod p, s_1 + s_2 \bmod q) = (r_{12}, s_{12}) \quad (3.57)$$

The Schnorr aggregated-multisignature is secure under the DL assumption in a random oracle model. In other words, any polynomial-time adversary cannot produce a forgery with non-negligible advantage based on the challenger's responses to hash and signing queries. This property is known as Existential Unforgeability under Chosen-Message Attack (EUF-CMA) [27].

However, Schnorr's aggregated-multisignature can be created by anyone on the Net, and, therefore, it itself cannot be a proof of Alice's possession of transactions Tx1 and Tx2. To prove that she is a creator of $Tx1$ and $Tx2$, Alice is using Schnorr-based NIZKP to the verifier IC.

To start with, Alice generates at random a number $j \leftarrow rand(\mathbb{Z}_q)$ and computes

$$r = g^j \, mod \, p \tag{3.58}$$

Using a collision-resistant H-function, Alice computes the following h-value playing the role of the challenge in Schnorr identification:

$$h = H(Tx1, Tx2, \sigma_1, \sigma_2, \sigma_{12}, a_1, a_2, a || \, r) \tag{3.59}$$

She now calculates

$$s = j + x \cdot h \tag{3.60}$$

Then NIZKP of the prover data we express as follows:

$$\eta = (r, s) \tag{3.61}$$

The prover Alice sends the following data to the verifier IC:

$$\{Tx1, Tx2, \sigma_1, \sigma_2, \sigma_{12}, a_1, a_2, a, \eta\} \rightarrow IC \tag{3.62}$$

IC, after receiving this data, performs the following verifications.

1. Are transactions $\{Tx1, Tx2\}$ on the account of IC;
2. Verifies signatures (σ_1, σ_2) on these transactions using (3.53);
3. IC Schnorr aggregated-multisignature σ_{12} by checking the validity of the following identity

$$g^{s_{12}} mod \, p = r_{12} \cdot (a_1)^{h_1} \cdot (a_2)^{h_2} mod \, p; \tag{3.63}$$

4. IC verifies if

$$a_1 \cdot a_2 \; = \; a \; mod \; p; \tag{3.64}$$

5. Then, IC verifies if NIZKP is correct using $\eta \; = \; (r, s)$ and checking the identity

$$g^s = \; r \cdot (a)^h mod \; p \tag{3.65}$$

The last equation can be rewritten in the form

$$g^{s_1+s_2} \; mod \; p \; = \; r_1 \cdot r_2 \cdot (a_1 \cdot a_2)^h \tag{3.66}$$

If (3.64)-(3.66) hold, then Alice proved to IC that she knows her $PrK \; = \; x$ satisfying (3.55) without revealing its value. Consequently, Alice proved that transactions Tx1 and Tx2 belong to her and that she is an investor. This scheme is summarized in Fig. 3.4.

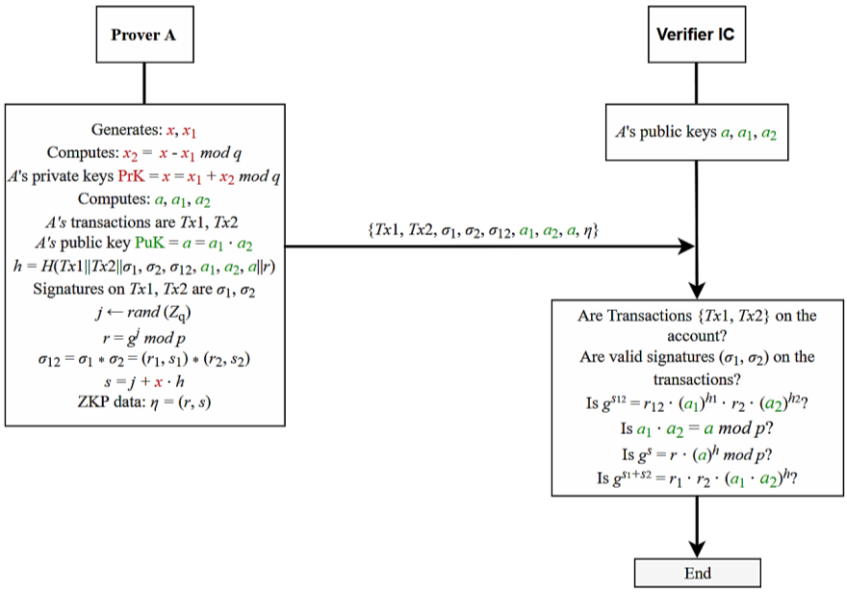

Fig. 3.4. De-anonymization scheme in our proposal.

Schnorr-based NIZKP is standardized by the RFC 8235 document for both finite fields and elliptic curves [28].

It is secure under the DL assumption in the random oracle model.

3.5. Anonymization-De-Anonymization Based Elliptic Curve Schnorr Signature Algorithm (ECSSA)

The Elliptic Curve (EC) digital signature realization based on classical Schnorr signature realization we name as Elliptic Curve Schnorr Signature Algorithm (ECSSA). This signature is presented in [29].

ECSSA Schnorr signature scheme was added to Bitcoin in 2021. This scheme allows signatures aggregation and NIZKP.

In this section, we present an anonymisation-de-anonymisation system based on ECSSA.

We are considering standardized EC denoted by secp256k1 adopted in Bitcoin and other cryptocurrencies. Public Parameters (PP) are the same as in (3.15). Private key PrK and public key PuK are expressed in (3.16) – (3.17). Let the transaction be signed is Tx.

ECSSA is constructed using the following steps according to the standard notations, and hence ignoring the special notations introduced in Table 3.3.

ECSSA Signature Creation

The ECSSA signature is created in the following way.

1. The random integer i in $\mathbb{F}_p$is generated

$$i = \text{rand}(\mathbb{F}_p); \tag{3.67}$$

2. Using a generator, compute EC point R and the first integer component r of signature

$$R = iG = (x_R, y_R); r = x_R; \tag{3.68}$$

3. Using H-function SHA256, the following h-value is computed

$$h = SHA256(Tx \parallel r); \tag{3.69}$$

4. The second component of the signature is the integer s computed by the equation

$$s = i + hz \ mod \ p \tag{3.70}$$

Symbolically ECSSA signature σ is expressed by the following notation

$$SignEC(i, z, h) = \sigma = (r, s) \quad (3.71)$$

ECSSA Signature Verification

The signature is valid if the following verification equation is valid:

$$sG = R + hA \quad (3.72)$$

It is seen that to verify the signature, the recovery of EC point $R = (x_R, y_R)$from its coordinate $x_R = r$ is required, i.e., it is needed to find a coordinate y_R from the EC equation (3.13). To solve this problem, the coordinate $(y_R)^2$ is computed. Then, the extraction of a square root modulo p from $(y_R)^2$is performed. As a result, two symmetric points corresponding to $R^+$and $R^-$are found. The square root extraction mod p is feasible if p is prime. Then (3.72) validity is verified for these two points $R^+$and R^-. One of these two points satisfies (3.72). We do not fall into the details of how to optimize this verification. To avoid the square root computation the replacement of r by R in (3.72) can be applied.

Correctness

Correctness follows from the identities (3.18)-(3.20) rewritten without special operation notations in Table 3.3.

$$sG = (i + hz)G = iG + (hz)G = R + h(zG) = R + hA \quad (3.73)$$

Analogously to the classical Schnorr signature, the security of ECSSA relies on the difficulty of solving the problem of finding z in (3.14) when A and G are given or to find i in (3.69) when R and G are given. This problem, following the analogy introduced in Section 3.2.2, is named the EC discrete logarithm problem (DLP).

3.5.1. Anonymization Based on ECSSA

Analogously to Section 3.4.1, suppose Alice has a private and public key pair $PrK = z$ and $PuK = A$. To realize anonymity, Alice intends to create two anonymous account addresses $\{Addr_1, Addr_2\}$ according to the procedure formally expressed in (3.46). Then Alice created two pairs

of private and public keys ($PrK_1 = z_1, PuK_1 = A_1$) and ($PrK_2 = z_2$, $PuK_2 = A_2$) respectively.

$$z \leftarrow \text{rand}(\mathbb{F}_p), \tag{3.74}$$

$$z_1 \leftarrow \text{rand}(\mathbb{F}_p), \tag{3.75}$$

$$z_2 = z - z_1 mod\ p, \tag{3.76}$$

$$z = z_1 + z_2 mod\ p, \tag{3.77}$$

$$A_1 = z_1 G, \tag{3.78}$$

$$A_2 = z_2 G \tag{3.79}$$

Then, referencing (3.18), public keys A_1, A_2 satisfy the following relation:

$$A_1 + A_2 = z_1 G + z_2 G = (z_1 + z_2)G = zG = A \tag{3.80}$$

Analogously to Section 3.4, after receiving incomes in $\{Addr_1, Addr_2\}$ Alice would like to invest some income in her addresses $\{Addr_1, Addr_2\}$ to the Investment Company (IC). Then she creates two transactions $Tx1$, $Tx2$, where investment sums are outputs (expenses) of these transactions. These transactions are signed by ECSSA private keys $PrK_1 = z_1$ and $PrK_2 = z_2$. Using (3.67)-(3.71) two signatures σ_1 and σ_2 are computed for transactions $Tx1$ and $Tx2$:

$$SignEC(z_1, h_1) = \sigma_1 = (R_1, s_1), \tag{3.81}$$

$$SignEC(z_2, h_2) = \sigma_2 = (R_2, s_2) \tag{3.82}$$

Notice that for every signature, the different random numbers i_1 and $i_2$are generated according to (3.67), and these signatures are placed on different h-values $h_1$and h_2 corresponding to two transactions $Tx1$ and $Tx2$.

Signatures σ_1, σ_2 are sent to IC. But IC has no understanding that $Tx1, Tx2$ are of the same person, namely Alice. IC, together with the received sums in $Tx1$and $Tx2$, verifies signatures on these transactions using (3.72).

3.5.2. De-Anonymization Based on ECSSS

After Alice sent signatures σ_1, σ_2 to IC, she must prove to IC that she is a holder of the sent funds in $Tx1, Tx2$. The proof is realized according to the methodology presented in Section 3.4.2 and is named as de-anonymization. This proof is based on the ECSSA analogy of Schnorr aggregated-multisignature and the knowledge of private key z satisfying without revealing information about z, i.e., using NIZKP.

Schnorr aggregated-multisignature σ_{12} of $T\text{x}1, T\text{x}2$ is created using the special multiplication operation $\circledast$ of signatures σ_1, σ_2 similar to

$$\sigma_{12} = \sigma_1 \circledast \sigma_2 = (R_1, s_1) \circledast (R_2, s_2) = \\ = \big((R_1 + R_2), (s_1 + s_2 mod\ p)\big) = (R_{12}, s_{12}), \quad (3.83)$$

$$R_{12} = (i_1 + i_2 mod\ p)G, \quad (3.84)$$

$$s_{12} = i_1 + h_1 z_1 + i_2 + h_2 z_2 mod\ p \quad (3.85)$$

Verification of σ_{12} passes if the following identity holds

$$s_{12}G = R_{12} + h_1 A_1 + h_2 A_2 \quad (3.86)$$

NIZKP of knowledge z is realized in the following steps:

1. Following (3.67), the random integer k in $\mathbb{F}_p$ is generated

$$k = \text{rand}\big(\mathbb{F}_p\big); \quad (3.87)$$

2. Using a generator G, the EC point R is computed as the first component of the signature

$$R_k = kG; \quad (3.88)$$

3. The following h-value is computed

$$h = SHA256(Tx1, Tx2, \sigma_1, \sigma_2, \sigma_{12}, A_1, A_2, A || R_{12}); \quad (3.89)$$

4. The second component s of the signature is according to (3.70)

$$s = k + hzmodp \quad (3.90)$$

Symbolically, the NIZKP is denoted by

$$\eta_{EC} = (R_k, s) \quad (3.91)$$

The information about $Tx1, Tx2$, public keys A_1, A_2, A, and signatures $\sigma_1, \sigma_2, \sigma$ are sent to IC. IC performs the following steps of verification:

1. Verification of $Tx1$ and Tx2 data;
2. Computation of Schnorr aggregated-multisignature σ_{12} according to (3.83)-(3.84);
3. Computation of h according to (3.45);
4. Verification of the following identity

$$sG = R_k + hA \tag{3.92}$$

If verification is passed, then IC is convinced that transactions $Tx1, Tx2$ are created by Alice, and IC accounts for the transferred sums as Alice's investment.

3.6. Example

Let us consider the case when Alice has an account address $Addr$ created according to (3.46). Then, Alice created two account addresses $\{Addr_1, Addr_2\}$ with corresponding private and public keys to ensure anonymity, as it is depicted in Fig. 3.5. Cryptocurrencies we denominate by bitcoins (BTC).

According to the Unspent Transactions Output (UTxO) paradigm, let Tx1 be realised in $Addr$, with two cryptocurrency incomes $m_1 = 20\,BTC$ and $m_2 = 30\,BTC$. Hence, the total sum of incomes is $m_{12} = 50\,BTC$. Let the expenses be $m_3 = 10\,BTC$ and $m_4 = 40\,BTC$ and in total is $m_{34} = 50\,BTC$. The amount m_3 is transferred to Emily (E), whereas the amount m_4, representing expenses, is transferred by Alice to her anonymous address $Addr_1$.

Let Alice receive two sums $m_6 = 10\,BTC$ and $m_7 = 20\,BTC$ in the other anonymous address $Addr_2$ in transaction Tx2. She transfers all the received sum $m_8 = 30\,BTC$ to IC by this transaction.

To prove ownership of m_5 and m_8, a combination of Schnorr signatures, Schnorr aggregated-multisignatures, and NIZKPs is used to perform the anonymization and de-anonymization processes, as presented above.

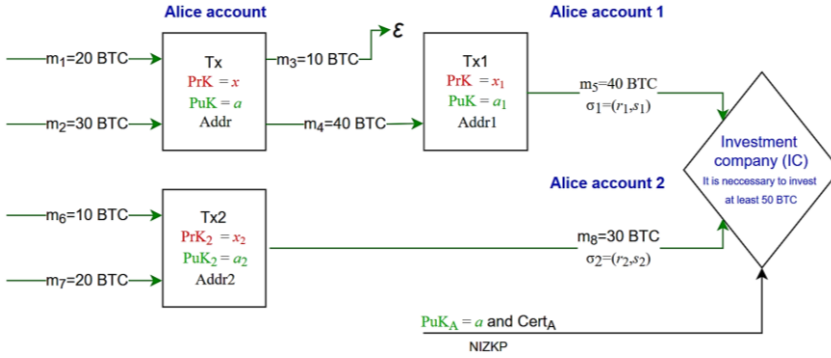

Fig. 3.5. Anonymization-De-anonymization scheme in a Private Blockchain.

3.7. Security Considerations

The security of anonymization is based on the security of the Schnorr signature scheme.

The security of the Schnorr signature relies on the following two pillars:

- The discrete logarithm (DL) assumption: given a generator g and the public key $a = g^x mod\, p$, no efficient adversary can gain an advantage in obtaining the private key x;
- Fiat-Shamir and random oracle assumptions: the Schnorr signature is obtained from the Fiat-Shamir transform of the Schnorr sigma identification protocol, and uses a collision-resistant H-function, which acts as a random oracle to ensure protection against existential forgeries.

As usual, we understand the adversary's advantage as a probability to improve the random guess of a discrete logarithm value, i.e. $AdvDL(g,a,p) = \left|\Pr(\hat{x} = x) - \frac{1}{q}\right|$.

When we say that a certain cryptographic primitive is secure under the DL assumption, we mean that the advantage $AdvDL(g,a,p)$ is negligible, i.e., for any fixed $d > 0$, the advantage $AdvDL(g,a,p)$ tends to zero faster than n^d tends to infinity, where n is the size of q in bits, after a certain n_0, i.e. $\lim_{n\to+\infty}\left(AdvDL(g,a,p)\cdot n^d\right) = 0$.

An effective adversary that forges Schnorr signatures can also efficiently compute discrete logarithms in large groups. In other words, any adversary who can efficiently forge a Schnorr signature must have a non-negligible advantage $AdvDL(g, a, p)$. Therefore, this adversary can also be used to solve for x in (3.45). According to Shor's results [30], such adversaries are possible on quantum computers. Hence, existing cryptographic methods should be replaced by post-quantum cryptographic (PQC) methods in the near future [31].

Therefore, the security of the Schnorr signature scheme against cryptanalytic attacks implemented by classical computers is based on the difficulty of the discrete logarithm problem (DLP) and on the H-function modelled as a random oracle [32]. In real life, random oracles do not exist, and the cryptographic primitives constructed in this model cannot be used directly. Therefore, in practice, the requirement to model H-function as a random oracle is replaced with collision-resistant H-functions.

The security of de-anonymization relies on the security of the Schnorr aggregated-multisignature scheme and Schnorr-based NIZKP of knowledge of Alice $PrK = x$ satisfying (3.48), (3.55)-(3.56).

However, in a multi-user environment, the rogue-key attack can be arranged [27]. In this case, the possible scenario is the following. Anyone on the Net can compute a rogue key and create a Schnorr aggregated-multisignature without using the signatures of other signers. But this attack is prevented, according to the results presented in [27]. Referring to [27], the rogue key attack is naturally prevented in our case, since in the verification equation (3.63), the public keys a_1 and a_2 are raised by different exponents h_1 and h_2, respectively. Moreover, in the alternative scenario, it is senseless for Alice to act against herself.

The security of Schnorr-based NIZKP based on (3.61) also relies on the discrete logarithm (DL) assumption, the Fiat-Shamir heuristics, and the random oracle model. This means that an efficient adversary who can impersonate a legitimate user must possess the corresponding secret key and is therefore able to efficiently solve the discrete logarithm problem (DLP) in large groups. As mentioned above, the group $\mathbb{Z}_p^*$ and the subgroup $\mathbb{G}_q$ are large. Moreover, the same adversary can also efficiently recover x in (3.48) if they gain access to all but one of the secret values x_i. However, the semiring $\mathbb{Z}_q$ is an additive group with the addition operation modulo q. Therefore, it preserves the uniform distribution of

the sum, given that all the summands are chosen independently and uniformly at random. This means that the probability of guessing the PrK in (3.48) is independent of the number of created anonymous accounts N, used for de-anonymization and is negligible under the DL assumption. Obviously, the same holds for (3.61) as well.

Also, the identity of a suspicious user can be restored using the special soundness property of the Schnorr sigma identification protocol in [20]: given two accepting conversations for the statement a_i the IC can compute the value of the witness x_i corresponding to it. Therefore, the IC can identify a dishonest user [20]. This property is preserved in an NIKZP adaptation.

Moreover, due to the soundness property of the Schnorr e-signature, the verifier accepts a false conversation with a negligible probability. Any adversary that can make the verifier accept a false conversation with non-negligible probability can also be used to solve DLP.

Finally, the direct attack is to recover the private key x in $\mathbb{Z}_q$ and to perform NIZKP in (3.61). However, due to the DL assumption, the adversary's advantage $AdvDL(g, a, p)$ is negligible if q is greater than 2^{1024}.

3.8. Conclusions

The proposed anonymization-de-anonymization system is based on a unified cryptosystem, usually named the ElGamal cryptosystem, and therefore uses the general public parameters.

The alternative method is based on ECC ring signatures, requiring more ECC "exponentiation" operations to sign transactions since Alice must sign transactions for all ring members. In this chapter, we are using a unified approach based on the integration of Schnorr signature, Schnorr aggregated-multisignature and Schnorr-based Non-Interactive Zero Knowledge Proof (NIZKP) to create anonymization, de-anonymization system in a private blockchain under the recognized security assumptions.

The proposed solution provides much more anonymity as compared with the anonymity provided in Ring Signatures (RS) based on the Elliptic Curve Cryptography (ECC) method used in Monero blockchain.

We estimate the effectiveness of realization between RS and our approach by comparing the number N_{ECO} in (3.44) of EC point multiplication operations to the number of exponentiation operations used in our proposal, based on Schnorr methods, which we denote by N_{EO}.

In our proposal, the effectiveness of anonymization is significantly higher than in the case of RS. In RS schemes, the signer must account for the fact that the number of operations N_{ECO} grows cubically with the number of users n in the ring, as shown in (3.44).

In contrast, in our construction, the anonymization is linearly dependent on the number of created anonymity addresses. For a single anonymous address, only two exponentiation operations are required. Moreover, anonymization can be performed offline at any time.

For Schnorr aggregated-multisignature, no exponentiation operations are required.

For Schnorr-based NIZKP, only two exponentiation operations are required.

The proposed system achieves significantly higher anonymity than RS-based schemes. Moreover, it provides a superior voluntary proof-of-ownership mechanism that offers better flexibility and privacy control than group signatures (decentralized user-initiated linkage instead of mandatory centralized tracing), traceable ring signatures (no need for large rings or online coordination), Monero view-key disclosure (selective and targeted rather than full exposure), proof-of-reserve systems, and selective-disclosure credentials (native on-chain integration via standard Schnorr primitives). In the latter case, when the ring consists of n members, the probability of guessing the transaction creator is $\frac{1}{n}$. In our proposal, this probability is equal to $\frac{1}{NoU}$, where NoU stands for the number of users.

This chapter is an extension of the B2C' 2025 conference report and proceedings of [1].

References

[1]. E. Sakalauskas, A. Kilčiauskas, A. Bendoraitis, A. Michalkovič, et al., Anonymization-deanonymization system in private blockchain, in *Proceedings of the 4th Blockchain and Cryptocurrency Conference (B2C)*, 2025, pp. 6-9.

[2]. E. Sakalauskas, A. Bendoraitis, S. R. Arshid, A. Kilčiauskas, et al., Private blockchain anonymization-deanonymization system preserving anonymity for the net, *Blockchain and Cryptocurrency*, Vol. 4, Issue 1, 2026, pp. 50-58.

[3]. S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2008, https://assets.pubpub.org/d8wct41f/31611263538139.pdf

[4]. Q. ShenTu, J. Yu, Research on anonymization and de-anonymization in the Bitcoin system, *arXiv*, 2015, arXiv:1510.07782.

[5]. D. J. Bernstein, T. Lange, Faster addition and doubling on elliptic curves, *Lecture Notes in Computer Science*, Vol. 4833, 2007, pp. 29-50.

[6]. D. Hankerson, A. Menezes, S. Vanstone, Guide to Elliptic Curve Cryptography, *Springer*, 2004.

[7]. G. Maxwell, A. Poelstra, Borromean ring signatures, 2015, https://github.com/Blockstream/borromean_paper/raw/master/borromean_draft_0.01_34241bb.pdf

[8]. Digital Signature Standard (DSS), 10.6028/NIST.FIPS.186-5, *National Institute of Standards and Technology*, 2023.

[9]. M. Bellare, D. Micciancio, B. Warinschi, Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions, *Lecture Notes in Computer Science*, Vol. 2656, 2003, pp. 614-629.

[10]. D. Boneh, X. Boyen, H. Shacham, Short group signatures, *Lecture Notes in Computer Science*, Vol. 3152, 2004, pp. 41-55.

[11]. M. N. S. Perera, T. Nakamura, M. Hashimoto, H. Yokoyama, et al., A survey on group signatures and ring signatures: Traceability vs. anonymity, *Cryptography*, Vol. 6, Issue 1, 2022, 3.

[12]. J. Liu, V. Wei, D. Wong, Linkable spontaneous anonymous group signature for ad hoc groups, *Lecture Notes in Computer Science*, Vol. 3108, 2004, pp. 325-335.

[13]. E. Fujisaki, K. Suzuki, Traceable ring signature, *Lecture Notes in Computer Science*, Vol. 4450, 2007, pp. 181-200.

[14]. M. M. T. Chakravarty, J. Chapman, K. MacKenzie, O. Melkonian, et al., The extended UTXO model, in Financial Cryptography and Data Security: FC 2020 International Workshops, *Springer*, Cham, 2020, pp. 525-539.

[15]. D. M. Gordon, A survey of fast exponentiation methods, *Journal of Algorithms*, Vol. 27, Issue 1, 1998, pp. 129-146.

[16]. S. Goldwasser, S. Micali, R. L. Rivest, A digital signature scheme secure against adaptive chosen-message attacks, *SIAM Journal on Computing*, Vol. 17, Issue 2, 1988, pp. 281-308.

[17]. D. Johnson, A. Menezes, The elliptic curve digital signature algorithm (ECDSA), Technical Report CORR 99-34, *University of Waterloo*, Canada, 1999.

[18]. SEC 2: Recommended elliptic curve domain parameters, *Standards for Efficient Cryptography Group (SECG)*, 2010.

[19]. D. Johnson, A. Menezes, S. Vanstone, The elliptic curve digital signature algorithm (ECDSA), *International Journal of Information Security*, Vol. 1, Issue 1, 2001, pp. 36-63.

[20]. D. Boneh, V. Shoup, A Graduate Course in Applied Cryptography, 2023, http://toc.cryptobook.us/

[21]. R. L. Rivest, A. Shamir, Y. Tauman, How to leak a secret, *Lecture Notes in Computer Science*, Vol. 2248, 2001, pp. 552-565.
[22]. S. Noether, Ring signature confidential transactions for Monero, Report 2015/1098, *IACR Cryptology ePrint Archive*, 2015.
[23]. S. Josefsson, I. Liusvaara, Edwards-curve digital signature algorithm (EdDSA), Request for Comments 8032, 10.17487/RFC8032, *Internet Engineering Task Force (IETF)*, 2017.
[24]. C. P. Schnorr, Efficient signature generation by smart cards, *Journal of Cryptology*, Vol. 4, Issue 3, 1991, pp. 161-174.
[25]. S. Micali, K. Ohta, L. Reyzin, Accountable-subgroup multisignatures, in *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS)*, 2001, pp. 245-254.
[26]. Secure Hash Standard (SHS), 10.6028/NIST.FIPS.180-4, *National Institute of Standards and Technology*, 2015.
[27]. M. Bellare, G. Neven, Multi-signatures in the plain public-key model and a general forking lemma, in *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, 2006, pp. 390-399.
[28]. F. Hao, Schnorr non-interactive zero-knowledge proof, 10.17487/RFC8235, *Internet Engineering Task Force (IETF)*, 2017.
[29]. A. Jain, E. S. Pilli, SoK: Digital signatures and Taproot transactions in Bitcoin, *Lecture Notes in Computer Science*, Vol. 14424, 2023, pp. 351-371.
[30]. P. W. Shor, Algorithms for quantum computation: Discrete logarithms and factoring, in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, 1994, pp. 124-134.
[31]. Post-quantum cryptography: A new security paradigm for the post-quantum era, *Penta Security Inc*., 2025, https://www.pentasecurity.com/blog/security-issue-post-quantum-cryptography
[32]. J. Katz, Y. Lindell, Introduction to Modern Cryptography (2nd Ed.), *Chapman and Hall/CRC*, 2015.

IFSA

Advances in Blockchain and Cryptocurrency 1

Sergey Y. Yurish, Editor

Written by 11 authors from five countries - Germany, Italy, Lithuania, Romania, and the United Kingdom - *Advances in Blockchain and Cryptocurrency, Volume 1* brings together diverse academic and professional expertise. This international perspective makes it especially valuable to researchers, postgraduate students, blockchain developers, cybersecurity and forensic specialists, financial analysts, risk managers, and industry professionals.

Readers will gain a clearer understanding of how blockchain technologies behave under real-world technical, economic, and security pressures. Combining rigorous methods with practical relevance, this volume provides useful analytical frameworks, identifies emerging research directions, and equips readers to evaluate blockchain and cryptocurrency systems with greater confidence.

Across three focused chapters, the book examines smart-contract forensics and on-chain evidence, asymmetric risk allocation in cryptocurrency portfolios, and cryptographic approaches that combine privacy with controlled de-anonymization and proof of ownership.

By connecting cybersecurity, financial modelling, and privacy-preserving cryptography, this volume offers a timely guide to some of the most important challenges shaping the future of blockchain and digital assets.

ISBN 978-84-09-90030-5

9788409900305