



**THÈSE DE DOCTORAT**
**SORBONNE UNIVERSITÉ**

**Spécialité: Informatique**

École doctorale Informatique, Télécommunications et Électronique (Paris)
ED130 — EDITE de Paris

Présentée par

**Lionel Beltrando**

Pour l'obtention du grade de docteur

Sujet de la thèse :

# Blockchain à faible empreinte énergétique adaptée au domaine financier

Composition du jury :

| | |
|---|---|
| **Présidente du Jury** | |
| Emmanuelle Anceaume | Directrice de Recherche – CNRS, IRISA, Rennes |
| **Rapporteurs** | |
| Pascal Felber | Professeur – Université de Neuchâtel, Suisse |
| Pascal Lafourcade | Professeur – Université Clermont Auvergne, Clermont-Ferrand |
| **Examinateurs** | |
| Silvia Bonomi | Associate Professor – SAPIENZA University, Rome |
| Marc Shapiro | Directeur de Recherche – INRIA, LIP6, Sorbonne Université |
| **Encadrement de la Thèse** | |
| José Alfaro | Encadrant, Directeur Technique – Finaxys |
| Yackolley Amoussou-Guenou | Encadrant, Maître de Conférences – U. Paris-Panthéon-Assas |
| Maria Potop-Butucaru | Directrice, Professeure – Sorbonne Université |

Thèse soutenue publiquement le *16 décembre 2025*

*À ma femme Eugénie,*

*mon fils Léandre,*

*ma fille Eurydice,*

*et au voyageur qui s'apprête à embarquer.*

*"Energy is the only universal currency."*

Vaclav Smil, *Energy and Civilization: A History* (2017)

# Remerciements

Dans tout voyage, plus que la destinée, ce que l'on retient surtout ce sont les compagnons de voyage et les personnes rencontrées sur le chemin. Cette thèse a été un long voyage, jalonné de découvertes, de doutes, d'efforts et de belles rencontres. À toutes celles et ceux qui ont croisé ma route et m'ont accompagné, je souhaite exprimer ici ma profonde reconnaissance.

Je tiens tout d'abord à adresser mes sincères remerciements aux membres du jury pour le temps qu'ils ont consacré à l'évaluation de ce travail. Je souhaite également remercier l'ensemble des enseignants et chercheurs qui ont contribué, directement ou indirectement, à mon parcours universitaire, ainsi que mes maîtres de stage chez Thales, Sébastien Keller et Romain Ferrari, et toute leur équipe, pour leur accompagnement attentif et les précieuses compétences qu'ils m'ont transmises. Ma gratitude va aussi à Jean-Christophe Rona et à l'équipe R&D d'Archos, avec qui j'ai eu le plaisir de plonger concrètement dans les détails techniques de la blockchain.

Je souhaite ensuite exprimer ma profonde gratitude à José Alfaro, qui a été à l'origine de ce projet. Il en a imaginé les contours et m'a donné la chance de l'explorer. Son intuition, son regard sur les enjeux du monde de l'IT dans la finance, ainsi que sa confiance et sa bienveillance, ont été des repères essentiels tout au long de ce voyage.

Ma gratitude va aussi à Maria Potop-Butucaru, ma directrice de thèse. Elle a été un véritable phare dans la mer parfois agitée qu'est la recherche. Grâce à sa rigueur et à son exigence bienveillante, elle m'a guidé pas à pas, m'aidant à déconstruire mes certitudes pour reconstruire une vision plus claire et plus solide. Elle a su me donner une direction lorsque le chemin semblait se perdre dans la brume, tout en me laissant la liberté d'explorer. Sa présence et ses conseils ont été essentiels à l'aboutissement de cette thèse.

Je tiens également à remercier chaleureusement Yackolley Amoussou-Guenou. Nos nombreuses sessions de travail ont été des moments clés, où mes idées ont pu être développées et challengées. Sa pédagogie et sa rigueur m'ont permis d'avancer avec clarté dans les moments les plus complexes. Il a joué un rôle déterminant dans l'élaboration de ce travail, et je lui suis profondément reconnaissant pour sa patience et son engagement.

En dehors du cadre académique, j'ai trouvé un équilibre précieux dans la course à pied. Je n'oublie pas mes partenaires d'entraînement, avec qui j'ai relevé des défis parfois vertigineux. Ces moments partagés ont nourri ma détermination et m'ont permis de garder l'énergie nécessaire pour avancer, même dans les périodes les plus tourmentées.

Enfin, ma plus profonde reconnaissance va à mes proches, en particulier les grands-parents, oncles, tantes, parrains, marraines de mes enfants qui ont été d'un inestimable soutien durant cette période. Leur présence, leurs encouragements et leur bienveillance ont compté bien plus qu'ils ne l'imaginent. À ma mère, dont les encouragements, constants au fil du temps, ont été un vent porteur sur ma route, m'accompagnant dans chacun de mes projets et nourrissant mon envie d'aller toujours plus loin. À mon père, qui a lui aussi accompli il y a longtemps, ce voyage académique et qui a inspiré mes premiers pas dans ce monde de la recherche, avant de voguer vers d'autres rivages, laissant derrière lui une étoile qui guide encore ma traversée.

À Eugénie, ma femme, pour son soutien indéfectible et sa patience face à mes innombrables explications sur la blockchain et les généraux byzantins. Merci d'avoir été mon ancrage et mon souffle, m'offrant la sérénité nécessaire pour traverser les tempêtes et la force de poursuivre ce voyage jusqu'au bout. À Léandre et Eurydice, qui, par leur joie et leur énergie, m'ont apporté une source constante d'inspiration et d'apprentissage, m'aidant à grandir et à porter un regard nouveau sur bien des choses. Ils ont été mes véritables piliers.

À toutes et tous, merci d'avoir été, chacun à votre manière, le gouvernail, les vents favorables et les étoiles qui m'ont guidé, me permettant de mener à bien cette odyssée.

# Abstract

Blockchain technology, and more broadly distributed ledger systems, hold the promise of transforming financial infrastructures by enabling more transparent, auditable, and tamper-resistant systems. It is used to facilitate cross-border payments, bond issuance, and insurance processes. This technology has also fostered the emergence of decentralized finance (DeFi), which allows financial operations, such as lending and asset tokenization, to be conducted without relying on traditional financial intermediaries. However, beyond this ambition of disintermediation, the economic and technical reality reveals a more nuanced landscape. While the original promise of blockchain was to eliminate trusted third parties, in practice we are witnessing a transformation rather than a disappearance of their role. The landscape of trust has been reshaped around new intermediaries, such as cryptocurrency exchanges, digital asset custodians, and stablecoin issuers, which now play central roles within the ecosystem. The adoption of blockchain technology nonetheless remains constrained by energy costs, scalability limitations, and strict regulatory requirements. Achieving a sustainable balance between performance, security, and compliance therefore represents a key challenge for blockchains in financial applications. In this context, this thesis explores how the judicious integration of trusted components can enhance the efficiency and resilience of financial blockchains. It makes three main contributions.

First, we demonstrate that adding a trusted component to each node is insufficient to improve Byzantine Reliable Broadcast resilience: even with attestations preventing equivocation, the threshold remains at $n \geq 3t+1$ because malicious processes retain the selective omission strategy. This negative result establishes that naive integration of trusted hardware is ineffective and that a fundamental protocol redesign is necessary.

Second, it introduces new Byzantine Reliable Broadcast protocols that leverage a minimal trusted component, a Trusted Monotonic Counter at the initiator, to achieve optimal resilience with only $n \geq 2f + 1$ processes, while reducing message complexity by nearly half compared to classical approaches.

Third, it introduces TenderTEE, a new consensus protocol extending Tendermint (recently renamed Ignite) through TEE attestations, thereby reducing the number of required validators and communication costs while maintaining both safety and liveness. TenderTEE not only decreases the number of necessary nodes (or, for a given number of nodes, increases Byzantine fault tolerance) but also enables governance schemes characteristic of financial systems, such as tripartite structures, which were previously infeasible under classical consensus protocols tolerating one-third of Byzantine nodes.

In addition to that, the thesis provides broader design guidelines for building efficient blockchains in the financial sector, highlighting the complementary role of trusted components alongside Proof-of-Stake mechanisms, sharding, and optimistic protocols.

# Résumé

La blockchain et les registres distribués, portent la promesse de transformer les infrastructures financières en offrant des systèmes plus transparents, auditables et résistants à la falsification. Ils sont utilisés pour faciliter les paiements transfrontaliers, l'émission d'obligations ou dans l'assurance. Cette technologie permet aussi l'émergence de la finance décentralisée qui permet de réaliser des opérations financières (emprunts, tokénisation d'actifs) sans passer par les acteurs financiers traditionnels. Cependant, au-delà de cette ambition de désintermédiation, la réalité économique et technique révèle un paysage plus nuancé. Si la promesse initiale de la blockchain était d'éliminer les tiers de confiance, la réalité montre plutôt une mutation de leur rôle qu'une véritable disparition. Le paysage de la confiance s'est recomposé autour de nouveaux intermédiaires, tels que les plateformes d'échange, les prestataires de conservation d'actifs numériques ou encore les émetteurs de stablecoin, qui assument désormais des fonctions centrales dans l'écosystème. L'adoption de la blockchain reste toutefois freinée par les coûts énergétiques, les limites de passage à l'échelle et des exigences réglementaires strictes. Trouver un équilibre entre performance, sécurité et durabilité constitue ainsi un défi central pour les blockchains à usage financier. Dans ce contexte, cette thèse s'attache à explorer comment l'intégration raisonnée de composants de confiance peut renforcer l'efficacité et la résilience des blockchains financières. Elle apporte trois contributions principales.

Premièrement, nous démontrons qu'ajouter un composant de confiance à chaque nœud ne suffit pas à améliorer la résilience du *Byzantine Reliable Broadcast* : même avec des attestations empêchant l'équivocation, le seuil reste à $n \geq 3t+1$ car les processus malveillants conservent la stratégie du silence sélectif. Ce résultat négatif établit qu'une intégration naïve de matériel de confiance est inefficace et qu'une refonte du protocole est nécessaire.

Deuxièmement, elle propose un nouveau protocole de diffusion fiable byzantine (Byzantine Reliable Broadcast) qui exploite un composant de confiance minimal, un compteur monotone fiable au niveau de l'initiateur, pour atteindre une résilience optimale avec seulement $n \geq 2f + 1$ processus, tout en réduisant la complexité de communication de près de moitié par rapport aux approches classiques.

Troisièmement, elle introduit *TenderTEE*, un nouveau protocole de consensus qui dérive de Tendermint (rebaptisé Ignite) à l'aide d'attestations TEE, réduisant le nombre de validateurs requis et les coûts de communication tout en maintenant sûreté et vivacité. TenderTEE diminue non seulement le nombre de noeuds nécessaires (ou pour un nombre de noeuds identiques assure une tolérance accrue aux agents byzantins) et la consommation de bande passante, mais permet également des schémas de gouvernance caractéristiques de la finance, tels que les structures tri-parties qui n'étaient pas réalisables avec un protocole de consensus classique tolérant un tiers de noeuds byzantins.

En plus de ces résultats, la thèse propose des lignes directrices plus générales pour concevoir des blockchains efficaces dans le secteur financier, en soulignant le rôle complémentaire des composants de confiance aux côtés de la preuve d'enjeu, du sharding ou des protocoles optimistes.

# Contents


**1 Blockchain for Finance** **13**

1.1 Digital Transformation versus Blockchain Innovation . . . . . 14

1.2 Applications of Blockchain in the Financial Sector . . . . . 16

1.2.1 Case Study: So|bond Platform . . . . . 19

1.2.2 Insurance . . . . . 21

1.3 Decentralized Finance (DeFi) . . . . . 22

1.3.1 Technological Foundations of DeFi . . . . . 22

1.3.2 Loans and Alternative Financing . . . . . 23

1.3.3 Illustrative Examples . . . . . 24

1.3.4 Challenges and Confrontation with Traditional Finance . . . . . 24

1.3.5 DeFi in a Regulated Environment . . . . . 25

1.4 Cross-Cutting Structural Transformations . . . . . 25

1.4.1 Evolution of Intermediaries in Blockchain Ecosystems . . . . . 25

1.4.2 Risk Management and Regulatory Compliance . . . . . 27

1.4.3 Integration of External Data: The Role of Oracles . . . . . 28

1.4.4 Interoperability and Cross-Chain Mechanisms . . . . . 28

1.5 Challenges and Perspectives Specific to the Financial Sector . . . . . 29

1.5.1 Regulatory and Compliance Challenges . . . . . 29

1.5.2 Operational and Technical Challenges . . . . . 29

1.6 Research Question and Contributions . . . . . 30

**2 Overview of Blockchain Technology 32**

2.1 Agreement Mechanisms in Blockchains . . . . . 33

2.1.1 Proof-of-Work (PoW) . . . . . 34

2.1.2 Proof-of-Stake (PoS) . . . . . 35

2.1.3 Other agreement blockchain protocols . . . . . 35

2.2 Smart Contracts: A Powerful Tool with Inherent Limitations . . . . . 39

2.3 Challenges and trade-offs in the Blockchain domain: A quick overview . . 40

2.3.1 Oracle Mechanisms . . . . . 41

2.3.2 Cross-Chain Interoperability Protocols . . . . . 42

2.3.3 Scalability-Oriented Optimizations in Blockchain Architectures . . 44

**3 Cryptographic Foundations of Blockchains 48**

3.1 Asymmetric cryptography and Digital Signatures . . . . . 48

3.2 RSA algorithm . . . . . 49

3.3 Cryptographic hash functions . . . . . 51

3.4 Merkle Trees and Data Integrity . . . . . 52

3.5 Zero-Knowledge Proofs . . . . . 53

3.6 Cryptographic Challenges in Blockchain . . . . . 54

3.7 Understanding Key Blockchain Components: The Cases of Bitcoin and Ethereum . . . . . 54

3.7.1 Bitcoin: The First Practical Application of Blockchain Technology 55

3.7.2 The Ethereum Blockchain . . . . . 61

3.8 Conclusion . . . . . 67

**4 Blockchain System Model and Building Blocks 69**

4.1 System Model . . . . . 69

4.1.1 Notations . . . . . 70

4.1.2 Participants . . . . . 70

4.1.3 Communication Network . . . 72
4.1.4 Dynamicity of the System . . . 74
4.1.5 Failure Models . . . 74
4.1.6 Time Assumptions . . . 75
4.1.7 Communication Primitives . . . 76
4.1.8 Blockchain Consensus . . . 78
4.1.9 Blockchain Repeated Consensus . . . 79
4.1.10 FLP impossibility result . . . 80
4.2 Trusted Components for Enhancing Byzantine Fault Tolerance . . . 81
4.2.1 Trusted Hardware explained . . . 82
4.2.2 Trusted Hardware in Blockchain Systems . . . 84
4.2.3 Temporal Overhead Introduced by Trusted Hardware . . . 87
4.3 Trusted Monotonic Counter Object . . . 88
4.4 Conclusion . . . 89

**5 Byzantine Reliable Broadcast with Trusted Monotonic Counters 90**
5.1 System model and Problem Definition . . . 91
5.2 Bracha's Byzantine Reliable Broadcast with Trusted Components . . . 91
5.3 Byzantine Reliable Broadcast with optimal TMC-Object . . . 98
5.4 Impact of Trusted Components on Byzantine Reliable Broadcast Complexity 102
5.4.1 Message complexity . . . 102
5.4.2 Temporal overhead: Access to TEE . . . 104
5.5 Conclusion . . . 104

**6 TenderTee Consensus 105**
6.1 Tendermint . . . 106
6.1.1 Tendermint Consensus . . . 106
6.1.2 Tendermint Repeated Consensus . . . 110

6.1.3 Tendermint limitations . . . 112
6.2 TenderTee . . . 112
6.2.1 TenderTee Consensus . . . 113
6.2.2 Correctness of TenderTee . . . 117
6.2.3 TenderTee Repeated Consensus . . . 122
6.2.4 Correctness of TenderTee Repeated Consensus . . . 124
6.3 Protocol comparison: TenderTee vs. Tendermint . . . 127
6.3.1 Message Complexity . . . 127
6.3.2 Temporal overhead: Access to TEE . . . 129
6.4 TenderTee for Financial-Grade Blockchains . . . 129
6.5 Conclusion . . . 131

**7 Synthesis and Conclusion 133**

7.1 Synthesis of Research Findings . . . 133
7.2 Contributions to the Field . . . 134
7.3 Limitations . . . 135
7.4 Future Reasearch Directions . . . 136
7.5 Broader Implications . . . 137
7.6 Final Remarks . . . 138

# Chapter 1

# Blockchain for Finance

« *Banking is necessary; banks are not.* »
Bill Gates (1994)

Blockchain is a decentralized and immutable ledger that securely records transactions across a distributed network of computers. Each block in the chain contains a set of transactions that are cryptographically linked to the previous block, ensuring data integrity and security while eliminating the need for a central authority. This decentralized structure fosters trust and transparency, which are critical in applications where multiple parties interact without a mutually trusted intermediary.

The emergence of blockchain technology, initially embodied by the Bitcoin [1] protocol in 2009, has profoundly challenged established paradigms in finance by introducing decentralized mechanisms for trust, transparency, and secure transaction validation [2]. Beyond cryptocurrencies, the integration of advanced cryptographic primitives and smart contracts has expanded blockchain's applicability to a diverse range of financial scenarios, including asset management, securities settlement, risk management, and decentralized financial services [3]. The current chapter focuses specifically on the application of blockchain in the financial sector, a domain characterized by stringent regulatory requirements, complex stakeholder interactions, and significant operational challenges. Financial institutions, faced with pressure to innovate and reduce operational costs, increasingly consider blockchain-based solutions as avenues to enhance efficiency, transparency, and resilience in critical processes. However, the path toward widespread adoption remains intricate, with substantial technological, regulatory, and operational hurdles yet to be overcome.

We begin by discussing the distinction between benefits genuinely attributable to blockchain's unique properties and those resulting primarily from general digitization trends. This analysis is essential to prevent the misallocation of resources and to ensure realistic expectations of blockchain's transformative potential in financial services.

Then, we examine concrete blockchain applications within finance, analyzing their specific technological architectures, use cases, benefits, and inherent limitations. These ap-

plications include cross-border payments, capital market infrastructures, digital asset issuance, insurance, lending, and alternative financing. Real-world examples, such as Société Générale's blockchain bond issuance and the so|bond platform, will be detailed to illustrate practical implementations, highlighting the tension between innovation and regulation.

The chapter also examines Decentralized Finance (DeFi), analyzing its underlying technological foundations and illustrating its impact through notable protocol examples [4]. DeFi represents a particularly radical innovation, leveraging blockchain's decentralization capabilities to challenge conventional financial intermediation and governance. However, the section will also critically evaluate DeFi's integration challenges within traditional regulatory frameworks, as well as its operational and security concerns [5].

We finally discuss structural transformations arising from blockchain adoption in finance, particularly the evolving role of intermediaries. While blockchain was initially envisioned as an intermediary-free technology, practical deployment has paradoxically fostered new types of intermediaries, such as digital asset custodians, centralized exchanges, and compliance providers. Understanding these developments is crucial for evaluating blockchain's real impact and assessing its integration within current financial infrastructures.

Regulatory compliance and risk management constitute additional central themes of this chapter. The immutable and transparent nature of blockchain transactions offers both opportunities and challenges concerning anti-money laundering (AML), know-your-customer (KYC) procedures, and audit processes. We examine how financial institutions navigate these constraints through permissioned blockchain solutions, cryptographic innovations such as zero-knowledge proofs, and hybrid regulatory approaches [6].

## 1.1 Digital Transformation versus Blockchain Innovation

An essential analytical challenge in evaluating blockchain-based financial projects lies in disentangling the respective contributions of blockchain technology and broader digital transformation efforts. Many initiatives presented as blockchain-driven are, in fact, part of wider modernization strategies aimed at digitizing legacy systems, streamlining processes, and enhancing user experience. In such contexts, observed improvements in efficiency, cost reduction, or service accessibility may result primarily from the transition from analog or paper-based workflows to digital platforms, regardless of whether blockchain is involved [6].

This raises a critical methodological question: are the perceived gains truly attributable to blockchain's core features (immutability, decentralization, cryptographic trust, and distributed consensus) or would similar benefits have emerged through conventional IT modernization approaches such as centralized databases, web APIs, or cloud computing [7]?

This analytical distinction also echoes the classical separation between incremental and

radical innovation in technological diffusion theory, where blockchain is often positioned, perhaps prematurely, as a paradigm-shifting infrastructure [8].

For example, a digital identity verification platform integrated into a bank's onboarding process may claim innovation by using blockchain. Yet, if the identity attributes are stored off-chain and the verification logic remains centralized, the added value of blockchain becomes marginal [5]. In some cases, blockchain has been adopted primarily for marketing or signaling purposes, without solving any core technical limitations. A widely cited example is the renaming of Long Island Iced Tea Corp to Long Blockchain Corp in 2017, which led to a sharp, though short-lived, surge in its stock price despite no substantial blockchain-related activity [9]. Likewise, when a bank adopts smart contracts to automate loan approvals but retains full centralized control over data inputs and decision parameters, the efficiency gains may stem more from process automation than from any inherent blockchain-related decentralization.

Conversely, there are cases where blockchain genuinely reshapes process architecture and risk models. The use of distributed ledgers in bond issuance or interbank settlement can eliminate reconciliation steps between entities, reduce counterparty risk, and ensure real-time auditability (benefits unlikely to be achieved as effectively through traditional digitization techniques alone). One notable example is Société Générale's issuance of €100 million in covered bonds as security tokens on the Ethereum blockchain, which enabled near-instant settlement and programmable compliance management [10].

For researchers, analysts, and decision-makers, distinguishing between these effects is not merely academic; it is central to assessing the relevance, sustainability, and scalability of blockchain adoption in finance. Overestimating blockchain's impact risks misdirecting resources toward technically unnecessary or operationally burdensome solutions. Underestimating it, on the other hand, may cause institutions to miss opportunities where blockchain offers unique advantages.

A rigorous evaluation must therefore consider:

- the technical necessity of blockchain in the architecture;
- whether decentralization (or disintermediation) is effectively implemented;
- the degree of transparency and auditability offered by the ledger;
- and the extent to which trust is embedded in code rather than institutions.

By clarifying this distinction, financial institutions and technology providers can focus their innovation strategies where blockchain delivers a comparative advantage—rather than deploying it as a default label for digital progress. This approach enables a more efficient allocation of resources and contributes to more realistic expectations around the role of blockchain in financial transformation.

## 1.2 Applications of Blockchain in the Financial Sector

Initially recognized for its role in secure, peer-to-peer transfers of digital assets (exemplified by Bitcoin), blockchain technology soon evolved beyond the simple exchange of cryptocurrencies. The integration of smart contracts (see subsection 3.7.2) into this transactional core has enabled the tokenization of a wide range of assets, from traditional financial instruments to entirely novel digital tokens. In doing so, blockchain paves the way for innovative financial models that leverage its decentralized and transparent nature while retaining the fundamental benefits of a trusted and efficient transaction layer.

Building on blockchain technology, the emergence of decentralized finance (DeFi) has reshaped traditional services such as lending, borrowing, and trading by removing centralized intermediaries. This new paradigm potentially broadens accessibility, enhances transparency, and fosters trust by design. DeFi protocols, operating primarily on public blockchains, enable automated and programmable financial transactions; however, their fully decentralized nature introduces new regulatory and security challenges. Issues such as compliance with Anti-Money Laundering (AML) and Know Your Customer (KYC) regulations, as well as the management of smart-contract vulnerabilities, remain critical areas that must be addressed before DeFi solutions can integrate seamlessly into established financial frameworks. Consequently, despite these promising innovations, large-scale implementations of blockchain and DeFi in traditional financial systems remain limited. Regulatory constraints in the financial domain are substantial, and blockchain solutions must effectively navigate complex compliance requirements to achieve widespread adoption. Nevertheless, blockchain's distributed structure remains particularly valuable in scenarios involving multiple stakeholders, such as interbank collaborations or complex supply chain financing, where a tamper-evident, shared ledger could streamline processes and reduce friction. Thus, while blockchain and DeFi technologies continue to hold significant potential, transforming secure asset transfers into more collaborative and inclusive financial ecosystems will require overcoming substantial regulatory, operational, and security hurdles.

In the following sections, we will analyze several concrete application areas in which blockchain technology provides specific innovations or solutions. We will examine, in particular, the impact of blockchain on cross-border payments, capital markets, and digital securities, illustrating these through concrete examples such as the so|bond platform. Additionally, we will investigate insurance, as well as loans and alternative financing.

### Payments and Fund Transfers

Traditional cross-border transactions often rely on the SWIFT network and a complex chain of *correspondent banking* relationships, requiring banks to maintain *Nostro/Vostro* accounts for liquidity and currency exchanges. These arrangements, while robust and globally recognized, introduce multiple intermediaries that increase fees, prolong settlement times, and create potential points of failure or delay.

By contrast, blockchain-based payment systems can *settle transactions nearly instantly* on a shared ledger, reducing the need for third-party intermediaries at every step. This

can substantially lower costs and simplify reconciliation, especially for cross-border remittances. *Stablecoins* (i.e. digital assets pegged to fiat currencies) further mitigate volatility, making them suitable for everyday payments. However, implementing blockchain solutions at scale involves practical trade-offs. Financial institutions accustomed to SWIFT and established compliance frameworks must ensure that any new system aligns with stringent oversight measures (e.g., verifying transacting parties and preventing illicit flows). Moreover, on- and off-ramp liquidity for digital assets remains a key challenge.

While blockchain reduces reliance on some intermediaries, it does not eliminate them entirely. Exchanges, custodial services, and payment gateways often step in as new forms of facilitators, potentially reintroducing points of centralization. User education and robust security practices, such as managing private keys, are also critical to widespread adoption. Despite these considerations, a growing number of banks and fintech firms are piloting blockchain payment networks, frequently in *permissioned* environments that allow a known set of participants to collaborate more transparently and with tighter governance. As technology and regulations mature, blockchain-based payments may offer a compelling alternative to legacy systems, particularly in cross-border scenarios where speed, cost, and accessibility are paramount.

**Illustrative Example: RippleNet and On-Demand Liquidity (ODL)** RippleNet [11], developed by the fintech company Ripple, provides a concrete example of how blockchain technology can enhance cross-border payments. Unlike traditional systems such as SWIFT, which depend on correspondent banking relationships and require banks to maintain pre-funded Nostro/Vostro accounts, RippleNet leverages a blockchain-based ledger combined with its native digital asset, XRP, to enable instant and direct currency transfers.

Through RippleNet's On-Demand Liquidity (ODL) service, institutions convert funds from the sender's currency into XRP on a digital exchange. XRP tokens are then transferred almost instantaneously over the blockchain network to an exchange in the recipient's country, where they are immediately converted into the destination currency. This mechanism removes the need for pre-funded accounts, significantly reducing liquidity costs, settlement times (from days to seconds), and fees associated with intermediary banks.

The blockchain aspect of RippleNet, specifically the use of a shared, decentralized ledger, enables real-time transaction visibility and transparency, allowing all parties involved to track payments accurately at every stage. Moreover, the cryptographic verification inherent to blockchain ensures transaction immutability and reduces reconciliation efforts and potential fraud, something difficult to replicate fully within traditional SWIFT-based settlement networks.

Thus, RippleNet exemplifies how blockchain technology fundamentally alters the operational framework of cross-border payments, delivering efficiency gains that the traditional SWIFT infrastructure, with its reliance on multiple intermediary parties and manual reconciliation processes, inherently cannot achieve.

## Capital Markets and Digital Securities

Capital markets rely on a multi-layered ecosystem of *clearinghouses*, *custodians*, and *central securities depositories* to finalize trades and manage asset custody. Although these intermediaries bring reliability and compliance, they also add cost and administrative overhead. Blockchain technology introduces the possibility of *tokenizing* a wide array of financial assets, such as equities, bonds, real estate, and more, by recording ownership and transfers on a shared ledger.

A key benefit of blockchain-based capital markets is *near-real-time settlement*, which can lower the risk of trade failures, reduce counterparty exposure, and simplify back-office operations. Additionally, smart contracts can automate certain compliance rules, such as restricting ownership to accredited investors or enforcing holding periods for private offerings. This level of built-in regulation stands in contrast to traditional markets, where manual processes and siloed databases frequently create operational delays.

### Illustrative Example: Société Générale's Digital Bond Issuance on Ethereum

A significant real-world example of blockchain technology integration into capital markets is Société Générale's issuance of covered bonds on the Ethereum blockchain. In 2019, through its subsidiary Société Générale SFH, the French banking group issued €100 million of covered bonds as security tokens directly registered on the Ethereum blockchain. This transaction demonstrated blockchain's ability to streamline the issuance, settlement, and management of traditional financial securities.

The blockchain-based issuance facilitated near-instantaneous settlement, bypassing the multi-day settlement cycles typical of traditional bond markets. Tokenization allowed Société Générale to digitally represent bond ownership on a decentralized ledger, providing real-time visibility and reducing reconciliation efforts between parties involved. Additionally, smart contracts embedded within the blockchain network automated compliance enforcement, including tracking ownership and restricting token transfers to verified participants.

This concrete implementation showcases blockchain's practical value in reducing administrative overhead, accelerating settlement cycles, and enhancing transparency compared to conventional financial instruments. However, the issuance also highlighted key regulatory and operational considerations, such as the need for clear legal recognition of digital securities and the integration with existing market infrastructures.

Despite these advantages, industry-wide adoption faces technological and regulatory hurdles. Many large institutions require robust *custodial solutions* for digital assets, complete with insurance against hacks or private key loss. Interoperability between different blockchain platforms also remains an open question; if a security token lives on one network, trading it across another without complex bridging solutions can limit market liquidity. Implementing blockchain in capital markets also demands significant infrastructural investment, ranging from core ledger technology to staff retraining for blockchain-oriented workflows. Yet the push toward more efficient, transparent markets continues, driven by the promise of reduced costs and easier access for global investors. As both technology and regulatory clarity advance, tokenized capital markets may transition from

small-scale pilots to mainstream adoption.

### 1.2.1 Case Study: So|bond Platform

The **so|bond**[1] platform, developed jointly by **Crédit Agricole CIB** and **SEB**, represents a significant step forward in the application of blockchain technology to the issuance and management of digital bonds within a sustainable and regulated framework. Its ambition is to create a decentralized infrastructure where bond issuers can raise funds and manage securities with increased efficiency, while simultaneously integrating environmental considerations into the core of the technological design.

The platform addresses two fundamental challenges faced by the financial sector: operational inefficiencies in bond markets and the growing need for environmentally sustainable solutions. By leveraging blockchain, so|bond aims to enhance transparency and traceability throughout the lifecycle of a bond, reducing the costs and settlement delays traditionally associated with intermediated financial infrastructures. From a regulatory perspective, the network is designed to comply with anti-money laundering (AML) and counter-terrorism financing (CFT) requirements, as all validator nodes are operated by entities under regulatory supervision. This characteristic is essential for institutional adoption, as it guarantee the traceability of token issuance and prevents participation by unidentified or non-compliant actors.

A key innovation of the so|bond platform is its integration of the **Proof-of-Climate awaReness (PoCR)** consensus protocol [12]. This mechanism derives from Ethereum's Clique Proof-of-Authority (PoA) consensus (EIP-225), which relies on a fixed set of identified validators to produce blocks in turn. PoCR introduces an additional layer of environmental accountability by ranking validators according to the ecological footprint of their IT infrastructure. Independent third-party auditors periodically assess each node's environmental performance, considering the full lifecycle of the hardware and data center resources involved, from construction and shipping to usage and end-of-life management. Rewards are then distributed according to this ranking: validators with a lower environmental footprint receive higher rewards, thereby creating a continuous incentive for node operators to improve their infrastructure. By explicitly linking financial incentives to environmental performance, PoCR transforms blockchain consensus into a driver of sustainability.

However, this approach also introduces specific limitations. Since PoCR is built upon a non-BFT PoA model, the system does not offer formal guarantees of fault tolerance against actively malicious behavior as discussed by Angelis *et al.* [13]. While the protocol

[1] Conflict of interest statement: It is important to disclose that **Finaxys**, the industrial partner supervising this CIFRE (Convention Industrielle de Formation par la REcherche) PhD, actively participated in the development of the **so|bond** platform, a blockchain project for the issuance of digital bonds, led by **Crédit Agricole CIB** (the corporate and investment banking entity of the Crédit Agricole banking and financial services group) and **SEB** (a Swedish bank). While this involvement may naturally influences the perspective and information presented in this chapter, the author did not personally contribute to the development of the platform. The objective remains to provide an objective and rigorous analysis, focusing on the technical, financial, and environmental aspects of the project, in full alignment with scientific integrity.

can tolerate node crashes or temporary network issues, even a single byzantine node could disrupt block production or censor transactions. Governance mechanisms, such as voting to exclude misbehaving validators, is essential in mitigating these risks; however, the absence of a cryptographic guarantee implies that the security of the network ultimately depends on the legal and contractual framework binding the participants. This stands in contrast to Byzantine Fault Tolerant protocols, such as IBFT or Tendermint[2], which can mathematically guarantee safety and liveness as long as fewer than one-third of the validators are byzantine. For a platform like so|bond, which operates in a highly regulated environment with a limited and carefully vetted set of participants, this trade-off may be acceptable in the initial deployment phase. Nevertheless, as the network grows or diversifies, the addition of a BFT layer could provide greater resilience and make the system robust to more adversarial scenarios.

Another critical element of PoCR lies in its reliance on external auditors to measure and verify the environmental footprint of nodes. These auditors act as oracles (see 2.3.1), introducing off-chain data into the blockchain consensus. While this design ensures that environmental metrics are grounded in real-world observations, it also creates a potential point of failure. If auditors are compromised, collude with certain node operators, or fail to apply the methodology consistently, the ranking system could be manipulated without detection. The governance model includes mechanisms such as staking requirements and community voting to sanction misbehaving auditors, but the effectiveness of these safeguards depends on vigilant oversight by the network's participants.

The smart contracts deployed on the so|bond platform automate key processes related to bond issuance and lifecycle management. For instance, coupon payments and redemption at maturity are executed automatically, with real-time synchronization of data among issuers, investors, and regulators. This automation reduces the administrative burden and eliminates many of the reconciliation issues that plague traditional financial infrastructures, thereby lowering operational risks and costs.

The benefits of the so|bond platform are therefore multifaceted. It delivers operational efficiency by streamlining issuance and settlement processes, enhances transparency by providing all stakeholders with real-time access to a single, immutable source of truth, and contributes to the advancement of sustainable finance by embedding environmental incentives directly into the consensus mechanism. However, these innovations also bring new challenges. The limited byzantine fault tolerance of PoCR places a strong emphasis on governance and legal compliance, requiring robust oversight to prevent collusion or misbehavior. In addition, the external audit process must be carefully managed to ensure its integrity and independence, as it plays a decisive role in the credibility of the entire system.

In the broader context of financial innovation, so|bond illustrates how blockchain technology can be aligned with environmental objectives and regulatory constraints. It demonstrates the feasibility of integrating sustainability into the very fabric of decentralized infrastructures. Looking ahead, the platform could evolve by incorporating Byzantine Fault Tolerant mechanisms to strengthen its resilience and by expanding its auditor ecosystem

[2]Tendermint has recently been renamed *Ignite*; however, in this manuscript, we continue to refer to it as Tendermint.

to enhance the robustness of environmental measurements. Such developments would position so|bond not only as a pioneer of green finance but also as a model for secure and sustainable blockchain adoption in the institutional financial sector.

### 1.2.2 Insurance

Blockchain technology is often presented as a transformative solution capable of reshaping the insurance industry by automating claims processes, improving transparency, and reducing fraud and operational costs. At its core, insurance relies heavily on accurate data verification, risk assessment, and efficient claims management. Blockchain's decentralized, transparent, and immutable characteristics promise to streamline these aspects significantly [14]. However, critical scrutiny reveals substantial challenges and limitations that temper the enthusiasm surrounding blockchain's integration into insurance processes.

**Automated Claims Processing and Limitations** Blockchain-based smart contracts enable automated insurance claims payments triggered by objective, externally verifiable events. Parametric insurance products, such as weather-based agricultural policies, exemplify the feasibility of blockchain to manage claims efficiently and transparently. Despite clear theoretical benefits, the reliability and accuracy of external data sources (oracles) remain critical weak points. If the oracle providing the triggering data is compromised or inaccurate, claims may be improperly validated, leading to unjustified payouts or denials. Consequently, the practical viability of such automated mechanisms still heavily depends on robust oracle infrastructure and data integrity assurances.

**Fraud Reduction: Promise and Complexity** Blockchain's tamper-evident nature facilitates fraud detection by maintaining transparent transaction histories accessible to multiple stakeholders. Yet, blockchain alone cannot fully eliminate fraud. Fraudulent activities often occur outside the blockchain, at the data input stage, where dishonest claims can still be initially validated and entered onto the ledger. In [15], the authors propose a multi-signature claim processing scheme aimed at preventing fraud in health insurance. Hence, blockchain solutions necessitate strong integration with identity verification systems and rigorous off-chain processes to effectively mitigate fraud.

**Micro-Insurance and Operational Challenges** Blockchain-based micro-insurance holds promise for financial inclusion, offering cost-effective coverage in regions underserved by traditional insurers. Platforms like Etherisc [16] have demonstrated early potential. Nevertheless, widespread adoption faces practical obstacles: consumer unfamiliarity with blockchain tools, infrastructural limitations such as internet connectivity, and regulatory uncertainty, especially in developing regions. Therefore, while blockchain technically enables micro-insurance products, considerable operational, infrastructural, and educational investments are required for successful deployment at scale.

**Regulatory and Technological Barriers** Furthermore, regulatory frameworks currently struggle to keep pace with blockchain innovations in insurance. Data privacy regulations (e.g., GDPR compliance) pose significant challenges for blockchain-based systems, given their immutable data structures. As such, reconciling blockchain's immutability with the legal necessity of data modification or deletion remains unresolved, potentially limiting broader adoption.

Ultimately, while blockchain offers innovative potential for insurance processes, it remains critical to address these significant operational, regulatory, and technological challenges explicitly before fully realizing such potential.

## 1.3 Decentralized Finance (DeFi)

Decentralized Finance, commonly abbreviated as **DeFi** refers to financial services and products built upon decentralized public blockchain networks, enabling automated and intermediary-free transactions through smart contracts.

DeFi represents a fundamental shift in the traditional financial landscape by leveraging blockchain technology to deliver financial services without reliance on centralized intermediaries such as banks or exchanges. DeFi aims to democratize finance by enhancing transparency, reducing transaction costs, and expanding accessibility, especially for underbanked populations. However, its reliance on public blockchains, primarily Ethereum, poses critical questions regarding scalability, energy efficiency, regulatory compliance, and systemic risk.

### 1.3.1 Technological Foundations of DeFi

DeFi is underpinned by several foundational technologies, among which blockchain stands as a critical infrastructure. Blockchains such as Ethereum facilitate trustless, automated transactions through smart contracts; self-executing programs that encode transaction rules and automatically enforce agreements among parties [17]. Smart contracts form the backbone of platforms like Aave and Compound, automating lending and borrowing processes and eliminating the need for traditional financial institutions. However, these smart contracts introduce substantial technological risks, including vulnerabilities to coding errors, exploits, and oracle manipulation [18].

Tokenization is another key element enabling digital representation of assets, including both cryptocurrencies and traditional financial instruments. Stablecoins, digital currencies pegged to fiat currencies, such as DAI or USDC, provide necessary stability in highly volatile cryptocurrency markets [19]. Despite their stability, stablecoins remain heavily reliant on centralized issuers or collateralized backing, paradoxically reintroducing elements of centralization and counterparty risk into decentralized ecosystems.

The critical role of oracles, data feeds (see 2.3.1) supplying external, real-world information to smart contracts, cannot be overstated. These tools ensure the accuracy and

timeliness of financial data required for transactions. Yet, they simultaneously represent significant points of centralization and vulnerability within DeFi protocols, susceptible to inaccuracies and malicious attacks, undermining the decentralized ideal [20].

### 1.3.2 Loans and Alternative Financing

Blockchain technology has significantly impacted traditional lending models through decentralized lending protocols and alternative financing mechanisms. Platforms using smart contracts and cryptographic assets offer novel ways for peer-to-peer lending and borrowing. However, despite clear advantages such as reduced intermediary costs and increased accessibility, the technology introduces substantial complexities and vulnerabilities that must be critically examined [4].

**Collateralization and Liquidation Risks** Decentralized lending predominantly employs over-collateralization, requiring borrowers to lock digital assets exceeding their loan value, thereby mitigating risk from price volatility. Yet, this protective mechanism is itself problematic. Severe price fluctuations in crypto markets frequently trigger liquidation cascades, where assets pledged as collateral are rapidly sold off, exacerbating market downturns. Such liquidation spirals can result in significant borrower losses and systemic risks, reflecting vulnerabilities inherent in decentralized collateral-based lending models.

Moreover, reliance on price oracles to trigger liquidations introduces additional risk vectors. Oracles (see 2.3.1), typically third-party data providers, represent potential points of failure or manipulation that could lead to incorrect liquidations, disputes, and financial losses. Therefore, the security of decentralized lending critically hinges on the robustness and integrity of oracle infrastructures.

**Interest Rate Volatility and Yield Farming Critique** Decentralized lending protocols typically offer dynamically adjusted interest rates based on supply and demand within liquidity pools. While theoretically efficient, these rates can fluctuate unpredictably, creating challenges for borrowers who require stable financing conditions. Furthermore, the popular practice of yield farming (i.e. leveraging deposits to earn compounded returns) often involves complex risk exposures, opaque risk assessments, and significant vulnerabilities to protocol exploits, posing considerable risks to user funds.

**Regulatory Uncertainty and Compliance Gaps** Regulatory uncertainties further complicate blockchain-based lending models. Decentralized lending protocols frequently struggle to comply with AML, KYC, and consumer protection regulations, raising critical legal and compliance issues. Given these limitations, regulatory bodies increasingly scrutinize decentralized finance platforms, potentially constraining their growth and operational viability unless significant compliance adaptations occur [5].

In conclusion, while decentralized lending and alternative financing hold significant disruptive potential, careful consideration of their systemic risks, security vulnerabilities,

and regulatory gaps is imperative. Addressing these critical challenges will determine the long-term viability and scalability of blockchain-based lending solutions.

### 1.3.3 Illustrative Examples

**Uniswap** [21], a leading decentralized exchange (DEX), exemplifies DeFi's automated market maker (AMM) model, enabling users to trade cryptocurrencies directly from their wallets without intermediaries. Uniswap relies on liquidity pools rather than traditional order books, providing continuous liquidity governed by the constant product formula ($x \cdot y = k$) [17]. Nevertheless, this model exposes liquidity providers to risks of "impermanent loss," which can occur when the price ratios of pooled assets fluctuate significantly.

**MakerDAO** [22] represents a significant DeFi application through its stablecoin, DAI. Users deposit volatile assets (e.g., Ether) into collateralized debt positions (CDPs) to mint DAI tokens, thus creating a decentralized stablecoin pegged to the US dollar [18]. While innovative, this model introduces substantial liquidation risks during periods of high volatility, potentially resulting in cascading liquidations and exacerbating market instability.

**Aave** [23] further expands lending possibilities by allowing depositors to earn interest on their digital assets, while providing borrowers access to liquidity against collateral. Although efficient, such lending protocols remain vulnerable to systemic risks, smart contract exploits, and oracle manipulation, underscoring ongoing security and risk management concerns [24].

### 1.3.4 Challenges and Confrontation with Traditional Finance

Despite significant innovations, DeFi faces numerous critical challenges. Regulatory ambiguity remains paramount, as decentralized structures complicate traditional enforcement mechanisms and jurisdictional oversight. Zetzsche *et al.* [24] highlight how decentralization paradoxically creates new concentrations of risk in less visible infrastructures such as cloud providers and dominant blockchain protocols.

Technologically, DeFi's heavy reliance on blockchain networks like Ethereum introduces significant scalability and sustainability concerns. Ethereum's current proof-of-stake consensus mechanism, although more energy-efficient than its previous proof-of-work model, continues to present scalability limitations and centralized tendencies in validator pools. This scenario contradicts the decentralized and sustainable ideals often attributed to blockchain-based financial services.

Moreover, security remains a persistent issue. High-profile hacks and exploits of smart contracts (e.g., the DAO exploit [25], recent flash loan attacks on DeFi protocols [26]) underline the urgent need for robust security audits, regulatory compliance, and ongoing technological enhancements to secure user assets and maintain trust.

### 1.3.5 DeFi in a Regulated Environment

Addressing these challenges requires a balanced synthesis between decentralization and regulatory oversight. "Embedded regulation," where regulatory compliance is integrated directly into DeFi protocols through smart contracts[3], offers a promising pathway [24]. Such an approach could enable automated, real-time compliance monitoring, bridging the gap between DeFi's innovation and traditional regulatory frameworks.

However, implementing embedded regulation necessitates considerable interdisciplinary collaboration, technical sophistication, and regulatory clarity; factors currently in nascent stages of development. The complexity of cross-border jurisdictional issues further complicates regulatory integration, requiring substantial international coordination and standardized frameworks.

In conclusion, DeFi holds transformative potential for global finance by promising greater financial inclusion, transparency, and efficiency. Yet, significant regulatory, technological, and environmental challenges must be addressed explicitly. The future of DeFi likely involves a hybrid model, combining decentralized efficiencies with rigorous oversight and embedded compliance measures, aligning innovation with systemic stability and sustainability objectives central to the financial sector's broader transformation.

## 1.4 Cross-Cutting Structural Transformations

Beyond specific financial applications, blockchain adoption also induces deeper structural transformations in how financial systems are organized and governed. These transformations affect core dimensions such as the role of intermediaries, mechanisms of risk management, integration of external data, and the interconnection of blockchain infrastructures. While not unique to finance, these dynamics are particularly impactful in this domain, where operational resilience, regulatory compliance, and systemic integration are essential. This section explores these cross-cutting issues, highlighting how they reshape institutional relationships and technical architectures in blockchain-based finance. For aspects involving more general technical design, such as oracle security and cross-chain protocols, references are made to dedicated technical chapters of this thesis.

### 1.4.1 Evolution of Intermediaries in Blockchain Ecosystems

Although blockchain technology is frequently promoted for its ability to reduce or eliminate traditional financial intermediaries, the practical implementation of blockchain-based financial services has revealed that intermediaries often evolve rather than disappear entirely. Indeed, blockchain ecosystems have given rise to a new generation of specialized

[3]This requires the ability to "modify" a smart contract over time. However, since smart contracts are immutable by nature, evolving the smart contract architecture becomes necessary in practice. A common approach is to use proxy smart contracts, which act as intermediaries delegating calls to interchangeable implementation contracts, thus enabling updates to the business logic without changing the main contract's address

actors such as digital asset custodians, cryptocurrency exchanges, stablecoin issuers, and payment gateways, each fulfilling specific needs that blockchain networks themselves cannot entirely satisfy.

One prominent category of new intermediaries is digital asset custodians. These companies provide secure storage solutions for blockchain-based assets, handling complex cryptographic key management processes. For example, firms such as **BitGo** and **Coinbase Custody** offer institutional-grade custody services, implementing advanced security measures like multi-signature schemes, Hardware Security Modules (HSMs), and insurance coverage against loss or theft of digital assets. Despite these rigorous safeguards, custodial providers reintroduce centralization and counterparty risks: clients must trust these entities to safeguard assets properly, exposing them to potential operational failures or security breaches, as illustrated by historical incidents like exchange hacks or insolvencies (e.g., Mt. Gox, QuadrigaCX, or FTX).

Centralized cryptocurrency exchanges, such as **Binance**, **Coinbase**, and **Kraken**, represent another critical intermediary role. These exchanges facilitate market liquidity by providing platforms for users to trade cryptocurrencies and fiat currencies. However, as centralized entities, they assume custody of user funds during trading activities, posing significant risks related to operational transparency, regulatory compliance, and solvency. The collapse of the FTX exchange in 2022, which resulted in substantial financial losses due to inadequate governance and misappropriation of customer funds, underscores the critical need for robust regulation and transparency measures.

Stablecoin issuers constitute yet another class of intermediaries, notably exemplified by companies such as **Circle** (issuer of USDC) and **Tether Limited** (issuer of USDT). These actors manage fiat currency reserves that back digital tokens, providing a stable digital medium of exchange suitable for payments and financial transactions. However, stablecoin issuers face continuous scrutiny regarding the adequacy, composition, and transparency of their reserves. Questions persist about Tether's USDT reserves, leading to ongoing regulatory attention and investor skepticism. Such intermediaries thus add additional layers of counterparty risk, despite blockchain's promise of decentralization.

Finally, payment gateways and on/off-ramps, including providers like **MoonPay**, **Ramp Network**, and **Simplex**, facilitate easy conversion between fiat currencies and blockchain-based digital assets. Although these services are critical for user adoption, particularly for retail and institutional participants unfamiliar with blockchain interfaces, they again reintroduce centralized actors, creating potential vulnerabilities and compliance challenges.

Overall, these evolving blockchain intermediaries fulfill indispensable operational functions, significantly contributing to blockchain technology's accessibility and practical adoption in financial markets. Nonetheless, their centralized nature poses critical risks that must be effectively managed through clear governance frameworks, regulatory oversight, and rigorous operational standards. Future blockchain implementations must therefore carefully balance the pursuit of decentralization against the pragmatic necessity of specialized intermediary services.

### 1.4.2 Risk Management and Regulatory Compliance

Regulatory compliance in finance encompasses a wide range of obligations, including *Know Your Customer* (KYC), *Anti-Money Laundering* (AML), data protection (e.g., GDPR), and country-specific securities laws. Traditional financial institutions enforce these rules through centralized systems, structured workflows, and institutional oversight. However, blockchain networks, particularly public and pseudonymous ones, pose specific challenges due to their decentralized and immutable nature [5].

To address these concerns, financial actors often turn to *permissioned* or *consortium* blockchains, where participation is limited to vetted nodes. These controlled environments allow for compliance rules to be embedded directly at the protocol or application layer. They also enable more predictable governance and auditing processes while retaining some of blockchain's core benefits such as transparency and tamper-evident records [7].

Despite these adaptations, blockchain systems introduce new categories of risk. Smart contracts, once deployed, cannot be easily modified and have been the target of significant exploits (e.g., The DAO hack in 2016). Mismanagement of cryptographic keys has also led to irreversible losses, highlighting the operational and custodial risks specific to blockchain-based financial systems. Moreover, legal uncertainties remain regarding liability in decentralized environments: it is still unclear whether validators or node operators could be held accountable for processing illicit transactions [27].

While zero-knowledge proofs (ZKPs) (see 3.5) were first introduced in 1985 by *Goldwasser et al.* [28], their integration into blockchain systems is a more recent development. These techniques provide a way to reconcile blockchain transparency with privacy and compliance requirements. ZKPs allow a party to prove compliance with a regulation, such as not exceeding a transaction threshold, without revealing the transaction's full details. Such methods are being explored to address regulatory needs without undermining the decentralization of blockchain systems [29].

Nonetheless, the cross-border nature of blockchain transactions amplifies regulatory complexity. What qualifies as legal in one jurisdiction may not be permitted in another, as exemplified by the FATF's "Travel Rule" for virtual asset service providers (VASPs). This rule requires cryptocurrency exchanges and similar platforms to collect and share customer information (sender and recipient details) for transactions above a certain threshold—similar to what traditional banks must do for wire transfers. However, enforcing this rule across decentralized platforms has proven technically difficult, spurring a wave of innovation and legal debates. [30].

Moreover, auditability in blockchain systems requires a dual approach. While transactions are verifiable on-chain, auditors must often reconcile this data with off-chain documentation, such as contracts or identity checks, to verify compliance. This hybrid model necessitates new tools and skillsets for compliance officers, particularly in institutional settings [31].

In summary, blockchain's transparency, immutability, and programmability offer powerful tools for improving compliance and risk management. However, these same features

introduce novel legal, technical, and operational challenges. Financial institutions looking to adopt blockchain must carefully navigate this terrain, balancing innovation with accountability and aligning new architectures with evolving global regulatory standards.

### 1.4.3 Integration of External Data: The Role of Oracles

While blockchains are inherently self-contained systems, many financial applications—particularly those found in decentralized finance (DeFi)—require timely access to off-chain data such as asset prices, interest rates, exchange rates, and macroeconomic indicators. Oracles are the components that enable this connection, acting as trusted bridges between on-chain smart contracts and external data sources.

In financial applications, oracles are critical for enabling functions such as automatic liquidations, dynamic pricing, and execution of parametric insurance contracts. However, they introduce centralization vectors and security vulnerabilities: the manipulation or failure of an oracle can compromise the integrity of otherwise decentralized systems. The **"oracle problem"** remains a key concern in blockchain-based finance, where external data feeds can become single points of failure [20].

Although this section highlights the importance of oracles in financial use cases, a comprehensive analysis of oracle architectures, their typologies (e.g., centralized, decentralized, hardware-based), and the security assumptions they rely on is provided in subsection 2.3.1, where the design and evaluation of oracle protocols are discussed in greater technical depth.

### 1.4.4 Interoperability and Cross-Chain Mechanisms

As blockchain ecosystems evolve, financial assets and services are increasingly distributed across multiple blockchain networks, including Ethereum, Solana, Cosmos, and others. This fragmentation poses a challenge to the fluidity and composability of decentralized financial services, where seamless movement of assets and data between chains is critical for liquidity and functionality.

Cross-chain interoperability mechanisms, such as bridges, wrapped tokens, and interchain messaging protocols, seek to address this challenge by enabling interoperability among heterogeneous blockchain environments. In finance, these tools facilitate multichain lending, trading, and portfolio management, and are foundational to constructing complex DeFi ecosystems.

However, cross-chain solutions raise substantial technical and security concerns. Many bridge protocols have been targets of large-scale exploits due to their complexity and reliance on trusted third parties or multisignature validators. As such, cross-chain infrastructure remains a key bottleneck to scalable and secure blockchain-based finance.

For a deeper exploration of cross-chain interoperability models and their associated tradeoffs, the reader is referred to subsection 2.3.2.

## 1.5 Challenges and Perspectives Specific to the Financial Sector

While blockchain technology opens promising avenues for financial innovation, its integration into the traditional financial sector brings forth domain-specific challenges that cannot be overlooked. These challenges are not merely technical, but also institutional, regulatory, and operational. This section examines two key dimensions; regulatory and compliance constraints, and operational and technical limitations—that must be addressed to enable a secure, scalable, and sustainable use of blockchain within financial infrastructures.

### 1.5.1 Regulatory and Compliance Challenges

As discussed in Section 1.4.2, financial regulation imposes stringent requirements on institutions regarding anti-money laundering (AML), know-your-customer (KYC), data protection, and reporting obligations. These obligations are structurally difficult to enforce in decentralized environments, where there is no central authority to ensure compliance and no clear jurisdictional boundaries.

In the context of financial sector adoption, these challenges take on a more acute character. Institutions are not only subject to local regulations, but also to international regulatory harmonization efforts, such as those led by the Financial Action Task Force (FATF) and the Basel Committee on Banking Supervision. Consequently, any blockchain-based solution adopted at scale in finance must meet interoperability and compliance standards across jurisdictions.

As previously discussed, proposed strategies to address these issues include permissioned blockchain models with embedded compliance logic, as well as **"embedded regulation"** mechanisms that integrate supervision into protocol-level operations [24, 29, 30]. However, implementing such approaches remains technically and institutionally challenging. Furthermore, the burden of demonstrating compliance under evolving legal frameworks continues to slow innovation in regulated financial environments.

In summary, regulatory challenges are not only technological but also deeply institutional. They require coordinated action from technologists, financial institutions, and regulators, as well as international consensus to create a sustainable compliance framework for blockchain-based financial systems.

### 1.5.2 Operational and Technical Challenges

Beyond legal concerns, blockchain deployment in finance also presents operational and technical barriers that limit its widespread adoption.

First, integrating blockchain systems with legacy infrastructures is complex and costly. Financial institutions rely on decades-old systems built for centralized architectures. Mi-

grating to, or even interfacing with, decentralized ledgers requires massive investments in IT, human capital, and governance redesign. Moreover, blockchain's latency and throughput constraints (especially in public chains) may not meet the real-time performance expectations of high-frequency financial operations.

Second, key management is a persistent operational risk. In blockchain-based finance, control over digital assets is tied to private keys, and the loss or compromise of these keys results in irreversible asset loss. Institutional custody solutions are emerging to mitigate this, but they introduce new centralized points of failure and regulatory concerns.

Third, blockchain platforms remain vulnerable to smart contract bugs, exploit vectors, and protocol design flaws. Despite ongoing improvements, the rate of critical incidents (e.g., flash loan attacks, bridge hacks) highlights a significant maturity gap. Formal verification and thorough auditing processes are essential, but they increase deployment cost and slow down innovation cycles [4].

Lastly, energy consumption, remains a strategic concern for financial institutions committed to ESG objectives. The choice of consensus mechanism and infrastructure design thus has direct implications on the alignment of blockchain-based systems with sustainable finance priorities.

In summary, while blockchain can enhance transparency, automation, and resilience in financial systems, its adoption is conditioned by substantial domain-specific constraints. A realistic path forward involves hybrid models that combine the strengths of decentralization with structured compliance frameworks, institutional controls, and energy-efficient protocols to ensure responsible innovation in finance.

## 1.6 Research Question and Contributions

Blockchain technology offers unprecedented opportunities for innovation in the financial sector, but its adoption faces significant barriers related to security, regulatory compliance, scalability, and environmental sustainability. Existing consensus mechanisms, while robust, often involve trade-offs between decentralization, performance, and energy efficiency. This tension motivates our investigation into new architectures specifically designed for the unique constraints of financial applications.

The central research question guiding this thesis is:

> *How can blockchain systems be designed to meet the strict requirements of the financial sector (i.e security, regulatory compliance, and high performance) while maintaining a low energy footprint?*

To address this question, we proceed in three steps. First, we analyze the technological and regulatory landscape of blockchain adoption in finance, identifying the challenges that current systems face. Second, we propose novel protocols that integrate **trusted hardware components** to enhance Byzantine fault tolerance, including an optimized

**Byzantine Reliable Broadcast** mechanism and **TenderTee**, a consensus protocol extending Tendermint. Finally, we evaluate these contributions through the lens of **energy efficiency**, comparing them with existing consensus models and discussing their implications for environmentally sustainable blockchain infrastructures.

## Manuscript organization

- Chapter 1 examined the role of blockchain in finance, highlighting both its promises and the regulatory and operational challenges it raises.

- Chapter 2 provides a concise overview of blockchain technology, with a focus on consensus mechanisms, smart contracts, and scalability solutions.

- Chapter 3 revisits the main cryptographic primitives underpinning blockchains and shows how they secure systems such as Bitcoin and Ethereum.

- Chapter 4 formalizes the system model and assumptions, and introduces trusted hardware and the Trusted Monotonic Counter as building blocks.

- Chapter 5 presents new Byzantine Reliable Broadcast protocols using trusted components to achieve optimal resilience with fewer messages. The contribution presented in this chapter has been published in [32].

- Chapter 6 introduces TenderTEE, a consensus protocol that leverages trusted attestations to cut validator requirements and communication costs while preserving strong guarantees. The contribution presented in this chapter has been published in [33].

- Chapter 7, the conclusion, discusses the contributions of this thesis, examines attacks on trusted hardware, and outlines future research directions.

The contributions proposed in Chapter 5 and Chapter 6 are published in the journal *Theoretical Computer Science, special issue dedicated to the conference Stabilization, Safety, and Security of Distributed Systems, SSS 2024* [34].

# Chapter 2

# Overview of Blockchain Technology

*The whole is something besides the parts.*
ARISTOTLE, *Metaphysics*

Having identified the financial use cases and regulatory challenges facing blockchain adoption, we now turn to the technical foundations of blockchain systems that underpin these applications. This chapter provides an overview of blockchain technology. It introduces the different agreement mechanisms designed to guarantee the integrity of the blockchain ledger, presents the concept of smart contracts, and discusses key challenges in the blockchain domain. Particular attention is given to issues especially relevant to the financial sector, including the use of oracles, cross-chain interoperability, and various scalability optimization strategies. While this chapter primarily focuses on *blockchain* systems, we also consider the broader class of *Distributed Ledger Technologies* (DLTs). A blockchain is a specific type of distributed ledger where data is grouped into ordered blocks that are cryptographically linked together, forming a single linear chain. However, not all distributed ledgers rely on this strict chain structure. For instance, alternative designs such as *Directed Acyclic Graphs* (DAGs) also maintain a decentralized and tamper-resistant ledger but allow for multiple branches or parallel updates instead of a single chain of blocks.

In this manuscript, the term *blockchain* will often be used for simplicity, but the concepts and models we present are, in many cases, applicable to DLTs more generally. When specific distinctions are required; for example, when a property holds for blockchains but not for DAG-based systems; we will explicitly state this difference.

The purpose of this chapter is to provide a concise overview, while some of these concepts will be explored in greater detail in the subsequent chapters.

## 2.1 Agreement Mechanisms in Blockchains

In the literature on distributed systems, the term *consensus* has a precise and classical meaning. The seminal work of Lamport, Shostak, and Pease (1982) [35] defines the *consensus problem* as the task of a set of processes reaching agreement on a single value, even in the presence of failures or Byzantine faults. This definition focuses on three fundamental properties:

- **Agreement:** all correct[1] processes must eventually decide on the same value.
- **Validity:** the decided value must be one of the values proposed by the processes.
- **Termination:** every correct process must eventually reach a decision.

In the context of blockchain systems, however, the term *consensus* is often used in a broader and less formal sense. While blockchains also require nodes to agree on a single, consistent view of the system's state, the problem extends beyond the classical consensus definition in several important ways:

1. The network is **open and decentralized**, meaning that participants can freely join or leave and are not known or fixed in advance.
2. There are often **economic incentives and adversarial behaviors**, where participants may act maliciously or selfishly to maximize their profit.
3. The protocol must ensure not only that nodes agree on a value, but also that they maintain a secure **global ordering of transactions**, validate the correctness of these transactions, and protect against attacks such as double-spending.

Because of these differences, many blockchain researchers prefer to use the term *agreement* mechanisms when referring to blockchain protocols, reserving the word *consensus* for the strict theoretical notion defined in classical distributed computing. Throughout this manuscript, we will adopt this distinction: "agreement" will refer to the practical protocols that secure and synchronize blockchain networks, while "consensus" will retain its formal meaning from *Lamport et al.*

Different agreement protocols have been developed to address the various challenges associated with achieving agreement among distributed nodes in a decentralized environment. In this section, we focus on agreement protocols used in Distributed Ledger Technology. Among the large variety of mechanisms that have been proposed, two families dominate today's blockchain ecosystem:

[1] processes that always follow the protocol, see subsection 4.1.5 for a definition of Byzantine and correct processes

- **Proof-of-Work (PoW)**, the original mechanism introduced by Bitcoin, which relies on solving computational puzzles to create blocks and secure the network.

- **Proof-of-Stake (PoS)**, an alternative that replaces computational effort with economic collateral, selecting block proposers based on the amount of cryptocurrency they lock as a stake.

### 2.1.1 Proof-of-Work (PoW)

Proof-of-Work [1] is the original agreement mechanism used by Bitcoin and several other early blockchains.

Proof-of-Work is the first method created to organize how participants in a blockchain agree on the order of transactions. The idea is that before someone can add new information to the shared ledger, they must prove that they have made a significant amount of effort. This effort takes the form of solving a problem that is deliberately made difficult and time-consuming for computers. Many participants try at the same time, and because the task requires both luck and effort, no one can predict who will succeed first. When a participant finally finds a valid solution, the rest of the network can check it almost instantly and confirm that the effort was indeed made. The winner is then allowed to add a new block of transactions to the chain and usually receives a reward.

This process makes it very expensive for anyone to attempt to cheat. For example, if an attacker wanted to change a past transaction, they would not only need to redo the effort for that transaction but also for all the blocks that came after it. Meanwhile, honest participants would continue building on top of the existing chain, making it practically impossible for the attacker to catch up. In this way, PoW creates a system where honesty is the most realistic strategy, because trying to cheat would require an enormous and impractical amount of computing resources.

In addition, every transaction in the blockchain is protected by cryptographic techniques. When a user creates a transaction, they use a digital signature that proves they are the legitimate owner of the funds being transferred. Other participants can easily verify this signature, but it cannot be forged by an attacker. This means that even if someone were able to redo the work and replace an older block, they could not simply invent new transactions or steal funds. They would still need the valid signature of the rightful owner, which provides an additional layer of protection on top of the Proof-of-Work mechanism.

While Proof-of-Work has demonstrated strong security and resilience, its high energy consumption raises concerns regarding large-scale deployment, especially in regulated sectors such as finance. This limitation is one of the central motivations for exploring alternative mechanisms throughout this thesis.

### 2.1.2 Proof-of-Stake (PoS)

The concept of Proof-of-Stake (PoS) was first introduced with Peercoin (initially PPCoin), proposed by Sunny King and Scott Nadal in 2012 [36]. Peercoin implemented a hybrid model that combined Proof-of-Work and Proof-of-Stake, establishing the foundation for later blockchains that adopted purely PoS-based agreement mechanisms.

Proof-of-Stake (PoS) is another method that blockchains use to determine who can add the next block of transactions. Instead of requiring participants to spend large amounts of computing power, PoS relies on the amount of cryptocurrency that each participant is willing to "lock" in the system as a guarantee of honest behavior. This locked amount is called a stake.

In practice, the network randomly selects one participant to create the next block, but the chances of being chosen are higher for those who have a larger stake. Once selected, this participant proposes a new block of transactions, which others can quickly verify. If the block is valid, it becomes part of the chain, and the participant receives a reward.

The security of PoS comes from the fact that anyone who tries to cheat risks losing their stake. Because their funds are directly tied to their behavior in the network, dishonest actions will result in financial loss. Moreover, just like in Proof-of-Work, transactions themselves remain protected by cryptography, so even a selected participant cannot create fake transfers without the valid approval of the rightful owners.

By reducing the reliance on raw computational power, Proof-of-Stake introduces a more energy-efficient alternative to PoW. Such characteristics make PoS particularly relevant for financial applications, where both sustainability and regulatory compliance are key requirements.

PoS has gained popularity for its ability to provide a more scalable and energy-efficient method of achieving agreement. It also reduces the risk of centralization, as there is less incentive for validators to concentrate geographically to reduce operational costs. The move from PoW to PoS in Ethereum 2.0 [37] highlights the increasing recognition and viability of this model.

### 2.1.3 Other agreement blockchain protocols

Beyond Proof-of-Work and Proof-of-Stake, several other agreement mechanisms have been proposed to improve the scalability, efficiency, and adaptability of blockchain systems. In the following, we briefly present the most important ones.

#### Delegated Proof-of-Stake (DPoS)

One of the most influential alternatives is Delegated Proof-of-Stake (DPoS), introduced by Larimer in 2014 [38]. In DPoS, token holders elect a limited number of delegates (also called witnesses or block producers) who are entrusted with validating transactions

and producing new blocks. This representative system significantly reduces the number of nodes directly involved in agreement, enabling higher throughput and lower latency compared to classical Proof-of-Stake. However, this gain in efficiency comes with a partial reduction in decentralization, as only a small set of elected participants takes an active role in block production [39].

### Practical Byzantine Fault Tolerance (PBFT)

Practical Byzantine Fault Tolerance (PBFT) [40], originally designed by Castro and Liskov, has been adapted to blockchain contexts where the number of validators is limited and partially trusted, such as in permissioned networks. PBFT ensures fast finality and high throughput, but it does not scale well to large, fully decentralized environments.

### Proof-of-Authority (PoA)

Proof-of-Authority (PoA) was introduced by Gavin Wood in 2017 as a agreement mechanism designed primarily for private and consortium blockchains [41]. Unlike Proof-of-Work or Proof-of-Stake, PoA relies on a limited set of pre-approved validators whose identity is known and trusted by the network. This approach significantly reduces the computational and energy costs of reaching agreement, making it suitable for applications where efficiency and throughput are prioritized over full decentralization. PoA has since been widely adopted in enterprise and permissioned blockchain settings, and its variants have been analyzed in the academic literature [42].

### Cardano and Proof-of-Stake Advancements

Cardano [43], founded by Charles Hoskinson, also an Ethereum co-founder, is based on Ouroboros Proof-of-Stake protocol which is designed to be more energy-efficient than traditional Proof-of-Work systems. By using a probabilistic mechanism to select a validator for each new block, Cardano achieves agreement without the need for intensive computational work, significantly reducing its carbon footprint.

Cardano's layered architecture separates the settlement layer, which handles transaction validation, from the computation layer, which runs smart contracts. This separation allows for greater flexibility and scalability, as changes can be made to the smart contract layer without affecting the core transaction ledger. Additionally, Cardano places a strong emphasis on formal verification methods to ensure the correctness and security of smart contracts, further distinguishing it from earlier blockchain systems.

### Solana and High-Performance Blockchains

Solana [44] represents another leap in blockchain performance, designed specifically to handle high throughput and low latency requirements. It achieves this through a unique

agreement mechanism called Proof-of-History (PoH), which provides a verifiable sequence of transactions without requiring all nodes to agree on the time and order of events. This approach allows Solana to process thousands of transactions per second, making it one of the fastest blockchains currently in operation.

The high performance of Solana makes it suitable for a wide range of applications, from high-frequency trading to complex decentralized applications that require fast and efficient transaction processing. Moreover, Solana's low transaction fees and energy-efficient design make it a compelling alternative to more resource-intensive blockchains.

### Directed Acyclic Graphs

Directed Acyclic Graphs (DAG) have emerged as a promising alternative to traditional blockchain architectures, often referred to as "Blockchain 3.0" [45, 46]. Unlike conventional blockchains that rely on a linear chain of blocks, DAG-based systems represent transactions as vertices of a directed graph, while approval relationships are expressed as edges. By design, the structure is acyclic, which prevents circular dependencies and ensures consistency in transaction ordering.

In DAG-based blockchains, each new transaction validates one or more previous transactions by including their hash values. This mechanism removes the necessity of grouping transactions into blocks and avoids bottlenecks associated with block confirmation. Consequently, DAG-based architectures enable faster transaction validation, higher throughput, and lower energy consumption compared to Proof-of-Work (PoW) or Proof-of-Stake (PoS) systems [45].

The main motivations behind DAG systems stem from several limitations of conventional blockchains:

- **Scalability.** While Bitcoin and Ethereum process approximately 10 [47] and 18 [48] transactions per second (TPS), respectively, DAG-based platforms can theoretically achieve several thousands of TPS. For example, IOTA reports a throughput of approximately 2 000 TPS [49].

- **Transaction fees.** In blockchains using PoW or PoS, fees are necessary to incentivize miners or validators, which makes microtransactions impractical. In contrast, DAG systems reduce transaction costs to near-zero, thereby enabling micro- and nano-payments [45].

- **Confirmation time.** Transaction finality in DAG systems is typically reached in seconds or even milliseconds, whereas Bitcoin confirmations may take up to several minutes.

- **Energy efficiency.** Since DAG validation mechanisms do not require extensive mining computations, they significantly reduce energy consumption compared to PoW-based blockchains [50].

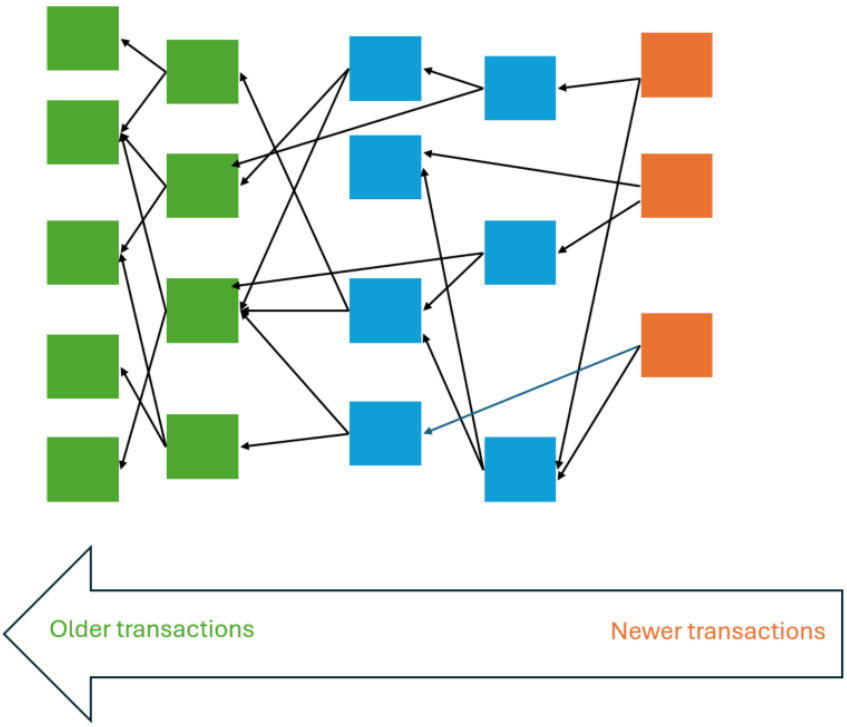



Figure 2.1: Illustration of a Directed Acyclic Graph (DAG)-based blockchain structure, each "new" transactions validate two "old" transactions

Several implementations illustrate the diversity of approaches within DAG-based distributed ledgers:

- **IOTA [50].** Introduces the *Tangle*, where each new transaction must approve two previous ones. IOTA employs a Markov Chain Monte Carlo (MCMC) algorithm to prevent lazy tips from dominating the network .

- **Nano [51].** Proposes a block-lattice architecture in which each account maintains its own blockchain. agreement is achieved via *Open Representative Voting* (ORV), enabling fast and low-cost peer-to-peer transactions .

- **Fantom [52].** Implements *Lachesis*, an asynchronous Byzantine Fault Tolerant (aBFT) consensus algorithm, and supports smart contracts through compatibility with the Ethereum Virtual Machine (EVM) .

- **Byteball/Obyte, DAGCoin, XDAG, CaixaPay [46, 45]**: These systems extend the DAG paradigm to domains such as tamper-proof data storage, privacy-preserving payments, and high-frequency microtransactions.

Despite their advantages, DAG-based blockchains still face challenges. Their security strongly depends on maintaining sufficiently high transaction rates; otherwise, the network may become vulnerable to large-weight attacks or double-spending attempts [53]. Moreover, some platforms (e.g., IOTA) rely on centralized components such as the “Coor-

dinator" to ensure stability in the early stages of adoption [50]. Finally, although Fantom and other projects support smart contracts, DAG-based blockchains generally lack the same maturity and ecosystem richness as Ethereum [45].

DAG-based architectures represent a promising path toward scalable, low-energy distributed ledgers, particularly suited for applications involving microtransactions and the Internet of Things (IoT). Their broader adoption, however, depends on overcoming current limitations related to decentralization, security under low activity, and generalized smart contract support.

In summary, various consensus mechanisms have been developed to ensure agreement in blockchain networks, each offering different trade-offs between **security**, **decentralization**, **scalability**, and **energy efficiency**. While PoW remains the most secure and battle-tested approach, its high energy consumption has motivated alternatives such as PoS, DPoS, and DAG-based architectures, which focus on sustainability and high throughput. Beyond reaching consensus on the order of transactions, blockchains also support **programmable logic** through *smart contracts*, enabling the automation of processes and the creation of decentralized applications. The next section explores smart contracts, their functioning, and their limitations.

## 2.2 Smart Contracts: A Powerful Tool with Inherent Limitations

Smart contracts are programs deployed on the blockchain that contain predefined rules expressed in code.

These contracts can encode agreements such as payments, exchanges of assets, or conditional transfers (for example, releasing funds only when both parties have met their obligations). Their reliability comes from the fact that execution is ensured by the agreement of the network rather than by intermediaries. However, smart contracts are strictly limited to the data available on the blockchain itself. They cannot directly access external information, which requires the use of additional mechanisms known as oracles (see section 2.3.1 for a more detailed explanation about oracles).

In the financial domain, this combination of strengths and limitations is particularly salient. On one hand, smart contracts enable automation in areas such as lending, trading, and settlement, reducing operational costs and dependence on centralized institutions. On the other hand, their constraints highlight the need for complementary infrastructures to bridge the gap between the blockchain and the real world. This dual nature illustrates both the promise and the challenges of integrating smart contracts into financial systems. For a more detailed explanation of smart contracts, see 3.7.2.

## 2.3 Challenges and trade-offs in the Blockchain domain: A quick overview

As blockchain technology evolves, new platforms have emerged to address some of the limitations of early implementations such as Bitcoin [1] and Ethereum [54], particularly regarding scalability, security, and environmental sustainability. These challenges are not unique to the financial sector: they affect a wide range of domains, including supply chain management, healthcare, identity systems, and voting. The financial sector, however, represents a particularly demanding context, with strict regulatory requirements, high throughput needs, and strong guarantees for privacy, auditability, and systemic resilience.

**Fundamental trade-offs.** Many of the difficulties faced by modern blockchain systems can be understood through a set of fundamental trade-offs that guide protocol design:

- **Scalability vs. Security:** increasing throughput and reducing latency often require relaxing certain security assumptions or reducing fault tolerance, while prioritizing strong security can limit scalability and performance.
- **Decentralization vs. Efficiency:** a highly decentralized network improves censorship resistance and resilience but introduces communication overhead and slower consensus, whereas optimizing for performance tends to concentrate power among a smaller set of participants.
- **Security vs. Environmental Sustainability:** mechanisms like Proof-of-Work provide robust security but consume vast amounts of energy, creating tension with environmental goals and operational costs.

Most innovations in blockchain architecture can be seen as attempts to improve one aspect of this triangle without overly compromising the others. For instance, moving from Proof-of-Work to Proof-of-Stake seeks to maintain strong security while reducing energy consumption, but it introduces new economic and governance risks. Similarly, sharding or Layer 2 protocols aim to increase throughput without reducing decentralization, at the cost of added complexity.

The following sections analyze several critical challenges that arise from these trade-offs and that are particularly relevant for financial applications:

1. **Reliable data acquisition through oracles**: blockchains cannot directly access external information, yet financial applications depend on real-time data such as exchange rates and market events. Oracles must balance decentralization, latency, trust assumptions, and cost, as discussed in Section 2.3.1.
2. **Cross-chain interoperability**: the growing diversity of blockchain platforms leads to fragmented liquidity and isolated ecosystems. Interoperability protocols

aim to connect these systems, improving composability and capital efficiency, but they introduce systemic security and regulatory risks, as explored in Section 2.3.2.

3. **Energy-efficient architectures**: the environmental footprint of blockchain networks remains a pressing concern, especially in financial contexts where operational costs and sustainability goals are paramount. Section 2.3.3 examines alternative designs to reduce energy consumption without compromising security.

### 2.3.1 Oracle Mechanisms

While blockchains provide secure and tamper-resistant infrastructures for executing on-chain logic, they are inherently limited in accessing external data. Many decentralized applications, particularly in finance, require real-time information about exchange rates, market prices, or real-world events. The need for oracles arises naturally with the advent of smart contracts, as blockchains cannot directly access external information.

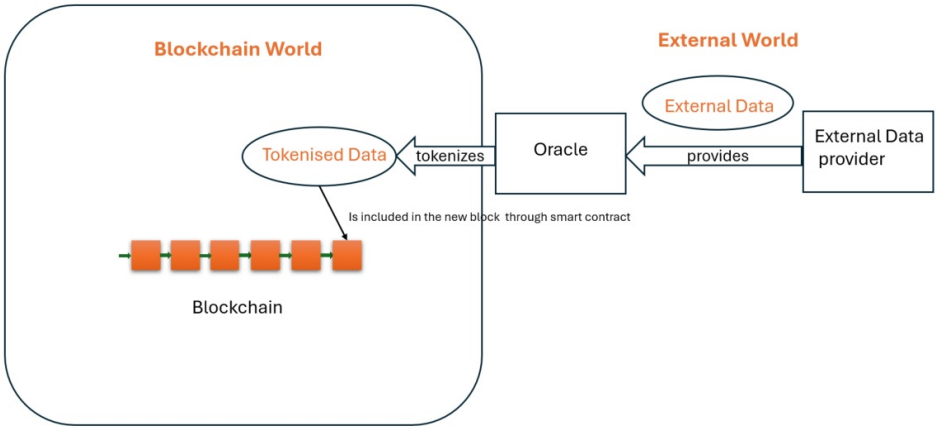



Figure 2.2: Illustration of how a simple centralized oracle works

Oracles come in various forms:

- **Centralized oracles**, operated by a single entity, are simple to implement but constitute a critical single point of failure.
- **Decentralized oracles**, such as *Chainlink* [55] or *Band Protocol* [56], aggregate data from multiple sources and use cryptoeconomic incentives to improve reliability and robustness.
- **Hardware oracles** connect physical devices (e.g., IoT sensors) to smart contracts, enabling automation in supply chains, insurance, or logistics.

Despite their importance, oracles remain one of the most vulnerable components of

blockchain systems, often described as the *oracle problem*. Since smart contracts blindly trust the data they receive, manipulating an oracle can result in widespread financial damage, as exemplified by several price manipulation attacks in DeFi. Ensuring oracle reliability involves trade-offs between decentralization, latency, trust assumptions, and cost. To the best of our knowledge, the first explicit formalization of the so-called *oracle problem* in the blockchain context was provided by Egberts [57].[2]

Advanced oracle designs employ techniques such as threshold signatures, trusted execution environments (TEEs), and reputation-based scoring. Nonetheless, integrating robust oracles remains a major research and engineering challenge, especially in regulated environments where data verifiability and auditability are crucial.

Oracles are particularly critical in financial applications, where real-time pricing feeds are used to trigger liquidations, execute trades, or calculate interest rates. Their integration must be carefully designed to balance responsiveness and resilience against manipulation or downtime.

### 2.3.2 Cross-Chain Interoperability Protocols

The proliferation of heterogeneous blockchain platforms has led to increased fragmentation of assets and applications across isolated networks. This lack of interoperability imposes significant constraints on liquidity, composability, and user experience—particularly in the context of decentralized finance (DeFi), where cross-chain strategies are increasingly employed to access diverse financial instruments and optimize capital allocation[58].

To address these limitations, several classes of interoperability protocols have been proposed. A first approach involves the use of *wrapped assets*, whereby tokens native to one chain are mirrored onto another through either custodial services or smart contract-based minting schemes. While operationally simple, such mechanisms introduce centralization risks and dependency on external trust assumptions[59].

Interoperability can also be achieved through protocol-level solutions designed for cross-chain asset exchange. Atomic cross-chain swaps are one such mechanism, enabling two parties to exchange assets across different blockchains without requiring a trusted intermediary. A notable recent proposal is R-SWAP, a relay-based atomic cross-chain swap protocol formalized by Lys et al. [60], which introduces the abstractions of relays and adapters to improve both security and performance . These design choices make R-SWAP a compelling approach for enabling efficient, trust-minimized swaps; an important consideration for financial use cases where asset exchange must be atomic, low-cost, and resilient.

Another strategy relies on *bridges*, which enable the direct transfer of assets or data between independent blockchains. These systems vary widely in architecture: some, such as the Polygon PoS Bridge [61], depend on trusted multisignature validators, while others, including Axelar or Wormhole, employ decentralized relayer networks or threshold

[2]Egberts' work is, to my knowledge, the first academic attempt to explicitly define and formalize the oracle problem in relation to blockchain systems.

signature schemes to verify cross-chain messages [62]. Despite their utility, bridges have repeatedly demonstrated security fragility, as evidenced by numerous high-profile exploits that have resulted from flawed assumptions or insufficient validator integrity [59].

More advanced interoperability is realized through *protocol-level communication frameworks*, such as the Inter-Blockchain Communication (IBC) protocol introduced within the Cosmos ecosystem. IBC enables sovereign chains to exchange standardized messages and tokens without relying on external intermediaries, offering improved guarantees in terms of trust minimization and extensibility [63].

Each of these models embodies trade-offs involving decentralization, latency, capital efficiency, and security. From a financial systems perspective, cross-chain interoperability enhances the ability to construct decentralized exchanges with broader liquidity, multi-chain lending markets, and diversified portfolios. These capabilities are essential for the development of robust and integrated decentralized financial infrastructures that can rival traditional systems in flexibility and reach. In particular, seamless interoperability is a prerequisite for achieving capital efficiency at scale, enabling composable financial primitives across heterogeneous platforms.

However, interoperability also introduces new systemic risks: the interconnection of blockchain networks through fragile bridges or shared trust dependencies can facilitate the propagation of faults or attacks across multiple platforms. Moreover, regulatory considerations are further complicated by the trans-jurisdictional nature of cross-chain transactions and the opacity of underlying validation processes, making compliance and auditing more difficult.

In the context of energy-efficient blockchain architectures for the financial sector, cross-chain protocols present a double-edged sword. While they can improve capital utilization and network specialization (thereby potentially reducing redundant computation across chains), they also increase protocol complexity and require additional infrastructure, which may offset energy gains. Ensuring secure, scalable, and efficient interoperability thus remains a fundamental challenge; one situated at the intersection of distributed systems design, cryptographic protocol engineering, and economic modeling. Its resolution is pivotal not only to technical robustness but also to the long-term sustainability and regulatory acceptance of blockchain-based financial ecosystems.

### Interoperability case-study: Polkadot

Polkadot [64], developed by Gavin Wood, co-founder of Ethereum, introduces a novel architecture designed to enable interoperability between different blockchains. Unlike traditional blockchains that operate in isolation, Polkadot [64] allows multiple blockchains to communicate and share data through its relay chain and parachains structure. This design not only enhances scalability by allowing parallel transaction processing across different chains but also fosters a more integrated blockchain ecosystem where assets and data can move seamlessly across diverse networks.

By facilitating interoperability, Polkadot [64] aims to overcome one of the critical limitations of earlier blockchains, enabling greater collaboration and innovation within the

blockchain space. Moreover, its Nominated Proof-of-Stake (NPoS) agreement mechanism optimizes security and scalability by involving a broader set of validators who are selected based on their reputation and stake in the network.

### 2.3.3 Scalability-Oriented Optimizations in Blockchain Architectures

Evaluating blockchain energy consumption accurately is a sophisticated and multifaceted task, influenced by a diverse array of variables including system architecture, consensus mechanism, transaction complexity, and network conditions. Traditional metrics commonly cited, such as average energy per transaction (kWh/tx), can provide preliminary insights but often fall short of capturing the full spectrum of energy demands inherent to blockchain systems [65].

A detailed assessment must consider three critical dimensions. Firstly, **Energy per transaction (kWh/tx)**, widely utilized as a straightforward metric, captures the immediate energy expenditure related to processing a single transaction. However, this metric tends to underestimate the total energy footprint by neglecting systemic overhead, including consensus algorithm execution, redundant operations by multiple validators, and network idle times [66].

Secondly, the **Energy per security unit** metric emphasizes the energy dedicated explicitly to achieving a defined security threshold, such as maintaining resistance to attacks, ensuring fault tolerance, and protecting against chain reorganizations. This metric is crucial as it directly ties energy consumption to security performance, reflecting the cost-effectiveness of a blockchain's security model [67].

Lastly, the **Energy per monetary unit settled (e.g., dollars)** metric offers an essential macroeconomic perspective, linking energy consumption directly to financial throughput. It provides an economic efficiency lens, particularly relevant within financial sector applications, allowing comparisons across traditional financial systems and various blockchain implementations [68].

Comprehensive measurement frameworks must thus integrate these metrics to offer a nuanced, actionable understanding of blockchain energy demands. Such an integrative approach facilitates informed decision-making by financial institutions, balancing operational performance with sustainability objectives.

Blockchain protocols continually evolve to satisfy increasing demands for performance and scalability. Beyond refining agreement mechanisms, several architectural enhancements are developed to improve transaction throughput, reduce latency, and increase operational efficiency. Although these optimizations do not explicitly target energy efficiency, their indirect impact often leads to reduced per-transaction energy consumption. We present here an in-depth exploration of key scalability-enhancing methodologies.

**Sharding.** Sharding is a technique that partitions the blockchain state and transaction processing into multiple "shards," each operating as a semi-independent chain segment. Each shard manages its ledger, transactions, and state transitions independently, significantly reducing computational overhead by enabling the parallel processing of transactions across different shards. Validators within this approach are designated specific shards, thus limiting their computational and storage requirements. Sharding is prominently featured in Ethereum 2.0's roadmap and implemented by platforms like Near [69] and MultiversX [70], demonstrating its practical viability and effectiveness[71]. (See Figure 2.3)

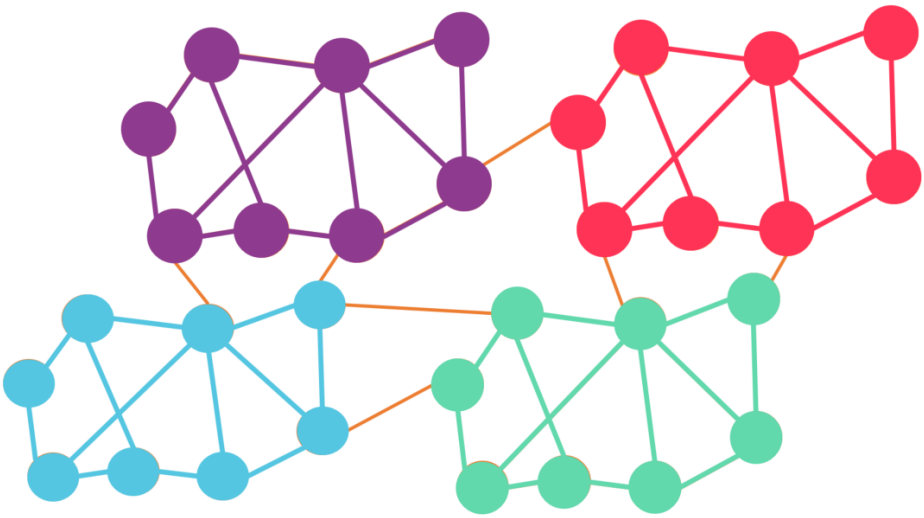


Figure 2.3: Illustration of a Blockchain divided into four shards

**Layer 2 Solutions.** Layer 2 (L2) solutions augment blockchain scalability by executing transactions off-chain while ensuring secure final state settlements on the base layer. Prominent categories include:

- **State Channels.** Protocols such as the Lightning Network facilitate off-chain micropayments by securing funds within multi-signature contracts. Transactions executed off-chain dramatically reduce latency and transaction fees, making frequent, small-value payments economically feasible. A state channel typically involves three on-chain interactions: (i) an opening (funding) transaction, which locks the participants' deposits, (ii) a series of off-chain state updates, and (iii) a closing (settlement) transaction that commits the final state to the blockchain. (see Figure 2.4)

- **Rollups.** Both Optimistic and Zero-Knowledge (ZK) rollups aggregate numerous transactions off-chain, periodically submitting batched transactions to the main blockchain in a compressed form. ZK-Rollups utilize cryptographic proofs (SNARKs/STARKs) for validity verification, offering stronger security assurances compared to Optimistic rollups, which rely on fraud proofs and challenge. [72]

- **Plasma Chains.** Introduced as an earlier scaling solution, Plasma employs child

chains that periodically commit Merkle root hashes to the Ethereum mainnet, relying on cryptoeconomic incentives and challenge mechanisms to maintain security.

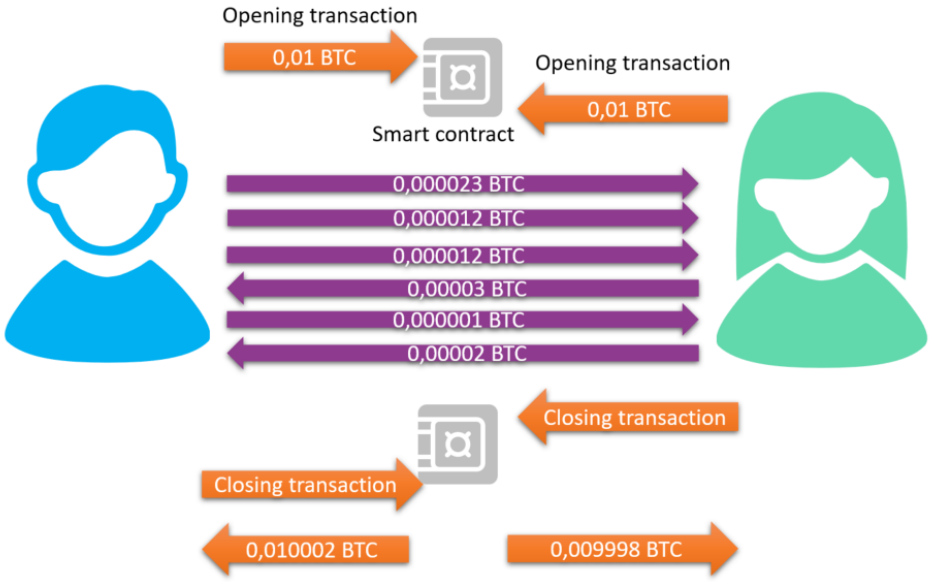



Figure 2.4: Example of a State Channel Lifecycle

Collectively, these Layer 2 approaches significantly reduce network congestion, improving scalability and user experience through faster, cheaper transactions.

**Execution-Level Optimizations.** Next-generation blockchain architectures, notably Solana [73] and Aptos [74], leverage transaction-level parallelism by organizing transaction execution to exploit multi-core CPU capabilities. These systems utilize sophisticated techniques such as non-overlapping memory accesses and deterministic transaction scheduling to concurrently process transactions without relying on secondary-layer solutions, significantly increasing transaction throughput and reducing latency.

**Efficiency Implications.** While scalability-focused enhancements reduce per-transaction energy consumption, their broader environmental and operational sustainability impact hinges on multiple interconnected factors:

- **Hardware and Infrastructure Demands.** Enhanced throughput and performance requirements may necessitate advanced CPU/GPU specifications and substantial network bandwidth, impacting total energy consumption.

- **Network Replication and Storage Costs.** Increased blockchain state sizes and transaction throughput amplify storage and synchronization demands, potentially elevating overall operational energy usage.

**Centralization Risks.** Certain scalability solutions, particularly those utilizing centralized sequencers or validators (e.g., certain rollup implementations), risk weakening blockchain decentralization and fault tolerance.

Detailed considerations regarding these trade-offs and their implications for blockchain sustainability are discussed in Chapter 7, providing a comprehensive analysis of energy consumption patterns and sustainability strategies associated with various blockchain architectures.

## From high-level introduction to technical foundations

In this chapter, we have introduced the foundational aspects of blockchain technology, presented the main agreement mechanisms, and outlined the key challenges and trade-offs shaping the evolution of distributed ledger systems. This overview provides the necessary conceptual background to understand how blockchain protocols are designed, as well as the motivations behind their continuous refinement.

Building on this foundation, the next chapter delves into the cryptographic primitives that underpin the security and integrity of blockchain systems. By examining asymmetric cryptography, digital signatures, hash functions, and related techniques in greater detail, we will move from a high-level understanding of blockchain operations to the technical foundations that make decentralized trust possible. Furthermore, the chapter will provide a more detailed explanation of agreement algorithms, with particular emphasis on Proof of Work and Proof-of-Stake, illustrated through the paradigmatic cases of Bitcoin [1] and Ethereum 2.0 [37].

Ultimately, this thesis aims to investigate how such mechanisms can be adapted to the specific constraints of the financial sector, with a particular focus on reducing their environmental footprint while preserving strong guarantiees of security and fault tolerance.

# Chapter 3

# Cryptographic Foundations of Blockchains

"God is in the details."

« Le bon Dieu est dans le détail. »
Attributed to Gustave Flaubert

Cryptography is the backbone of blockchain technology, ensuring both the integrity and security of data and transactions within decentralized systems. This chapter covers the fundamental cryptographic techniques that underpin blockchain systems, including asymmetric cryptography [75], digital signatures, cryptographic hashing, Merkle trees [76], or zero-knowledge proofs. Finally, we provide a detailed explanation of how these concepts are applied in practice by examining the mechanisms of two major blockchain platforms: Bitcoin and Ethereum.

## 3.1 Asymmetric cryptography and Digital Signatures

Asymmetric cryptography, and particularly digital signatures, plays a crucial role in blockchain systems by enabling secure and verifiable transactions between participants.

Introduced in 1976 by Diffie and Hellman, in their article *New Directions in Cryptography* [75][1], asymmetric cryptography involves a series of steps that enable secure communication between two parties without the need to share a secret key in advance. The steps of the Diffie-Hellman protocol are as follows:

1. **Public Parameter Agreement.** Both parties agree on public parameters: a large prime number $p$ and a generator $g$ such that $g$ is a primitive root modulo $p$, which define the finite mathematical group for the key exchange.

[1] Although earlier research conducted by the British Government Communications Headquarters (GCHQ) in the early 1970s was made public in 1997 [77].

2. **Private Key Generation.** Each party generates its private key by choosing a random number. For example, Alice selects a private key $a$, and Bob selects a private key $b$. It is crucial that these private keys are generated using a high-entropy source of randomness to ensure they cannot be guessed or brute-forced by attackers.

3. **Public Key Derivation.** Each party derives its public key from its private key using the agreed parameters. Alice computes her public key $A = g^a \mod p$, and Bob computes $B = g^b \mod p$. The public keys are designed to ensure that it is computationally infeasible in a reasonable time to derive the private key from the public key.

4. **Exchange of Public Keys.** The parties exchange their public keys over the insecure channel. Alice sends $A$ to Bob, and Bob sends $B$ to Alice.

5. **Shared Secret Computation.** Each party uses its private key and the public key of the other party to compute the shared secret. Alice computes $K = B^a \mod p$, and Bob computes $K = A^b \mod p$. Due to mathematical properties, both computations result in the same shared secret $K = g^{ab} \mod p$, which can then be used as the basis for secure communication.

Asymmetric cryptography can be used for two purposes:

- **Encryption and Decryption.** A sender encrypts a message using the recipient's public key. Only the owner (if the private key is kept secret properly) of the private key can decrypt this message. It ensures *confidentiality*.

- **Digital Signatures.** A sender generates a signature for a message using its private key. The recipient can verify the signature using the sender's public key, ensuring *authenticity* and *integrity* of the message.

Digital signatures are essential and highly used any blockchain systems, as they verify the identity of the sender and ensure that transactions have not been tampered with. In Bitcoin, for instance, every transaction is signed by the sender using their private key. The network nodes can then use the sender's public key to verify the signature and confirm the legitimacy of the transaction.

## 3.2 RSA algorithm

**RSA (Rivest–Shamir–Adleman)** [78] is one of the most well-known asymmetric cryptography algorithms. RSA encryption is based on the difficulty of factoring large prime numbers. The process starts with selecting two large prime numbers $p$ and $q$, which are

kept secret. The product of these primes is used to compute the modulus $n$:

$$n = p \times q$$

A public exponent $e$ is chosen such that e is a prime with $\phi(n) = (p-1)(q-1)$ and strictly smaller than $\phi(n)$. The public key consists of the modulus $n$ and a public exponent $e$. The public key $(n, e)$ is used to encrypt any message $m$, where $m < n$. Encryption is performed using the formula:

$$c = m^e \mod n$$

where $c$ is the ciphertext. The private key $(n, d)$ is derived from the prime factors $p$ and $q$, and an exponent $d$, which satisfies the equation:

$$d \times e \equiv 1 \mod \phi(n)$$

The private key $(n, d)$ is used to decrypt the ciphertext $c$ back into the original message $m$, using the decryption formula:

$$m = c^d \mod n$$

The security of RSA comes from the fact that while $n$ is public, it is computationally infeasible, in a reasonable time, to factor it into $p$ and $q$, and thus derive the private key $d$.

However, RSA requires significantly large key sizes to ensure robust security. For example, a 2048-bit RSA key offers a security level roughly equivalent to a 256-bit key in elliptic curve cryptography (ECC) which we explain in the next section. The increasing demand for key sizes in RSA leads to inefficiencies in both computational power and storage.

### Elliptic Curve Cryptography (ECC)

**Elliptic curve cryptography (ECC)** ([79], [80]) is a more efficient alternative that leverages the properties of elliptic curves over finite fields. An elliptic curve is defined by an equation of the form:

$$y^2 = x^3 + ax + b$$

where $a$ and $b$ are constants that define the specific curve used. The points on the curve, along with a special "point at infinity," form a group under an operation known as elliptic curve point addition.

In ECC, cryptographic strength comes from the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP). This problem can be stated as follows: given two points $P$ and $Q = kP$ on the curve, where $k$ is an integer and $kP$ represents the scalar multiplication of $P$, it is computationally infeasible to determine $k$, even when $P$ and $Q$ are known.

Elliptic curves provide equivalent security with much smaller key sizes compared to RSA. This makes ECC particularly useful in resource-constrained environments, such as blockchain networks.

In the context of blockchains, elliptic curve cryptography is used to secure transactions and generate digital signatures. For example, Bitcoin uses the *Elliptic Curve Digital Signature Algorithm* (ECDSA), which is based on the curve `secp256k1`, defined by the equation:

$$y^2 = x^3 + 7$$

Ethereum also employs ECDSA for its cryptographic operations, using the same elliptic curve. The choice of elliptic curve algorithms in blockchains allows for secure, efficient, and scalable transaction validation processes. We will provide more details on the use of asymmetric cryptography in blockchain later in dedicated subsections (3.7.1 for Bitcoin, 3.7.2 for Ethereum).

## 3.3 Cryptographic hash functions

Another essential component of blockchain technology is the use of cryptographic hash functions, which ensure data integrity and security through their unique properties. A cryptographic hash function is a mathematical algorithm that takes an input (or "value", or "message") and produces a fixed-size string of characters (often a hexadecimal value), typically a digest that appears random. Hash functions are deterministic, meaning the same input will always produce the same output, but even a small change in the input will produce a completely different hash (wich is called *the avalanche effect* - see Figure 3.1).

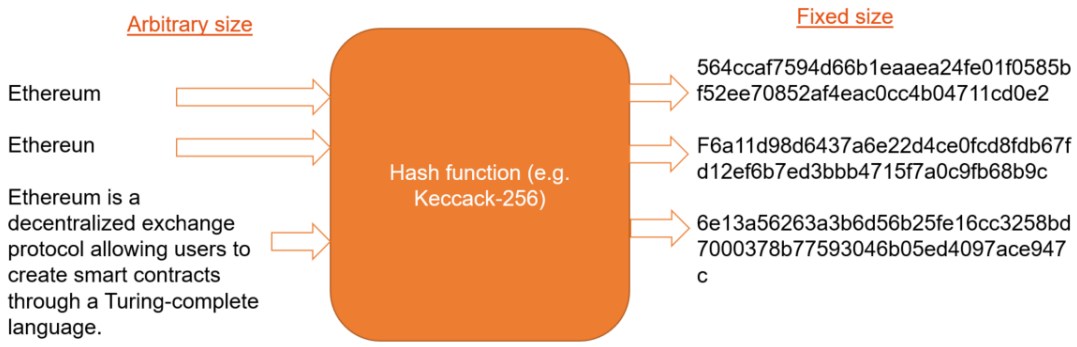



Figure 3.1: Illustration of the "avalanche effect" of a cryptographic hash function.

The security level of a cryptographic hash function (that we noted *hash*) has been defined by the three following properties [81].

- **Pre-image resistance.** Given a hash value $h$, it should be difficult to find a message $m$ such as $h = hash(m)$.
- **Second pre-image resistance.** Given a value $m_1$, it should be difficult to find an other value $m_2$ different of $m_1$ such that $hash(m_1) = hash(m_2)$.
- **Collision resistance.** It should be difficult to find a pair of two different value $m_1$ and $m_2$ such that $hash(m_1) = hash(m_2)$.

Trivially, as the output of a hash function has a fixed-size and since the input size is arbitrary, collisions exist. However, security relies on the computational difficulty of finding these collisions.

A hash function that satisfies these three properties is called 'a good hash function'. Throughout the document, we will assume that all hash functions referred to are 'good hash functions' and, for simplicity, will refer to them as simply 'hash functions', (or 'hash').

Hash functions are widely used in blockchain systems for ensuring data integrity and are a core component of Proof-of-Work (PoW) algorithm. In Bitcoin [1], the **SHA-256** [82] hash function is used to hash the block headers during the mining process. Miners must find a hash that is lower than a target value, a process that requires significant computational power, ensuring the security and immutability of the blockchain.

**SHA-256** [82] and **KECCACK-256** [83] are often chosen as hash functions in blockchain system. **KECCACK-256** is used in Ethereum, it's a variant of **KECCACK** wich has been choosen as a standard for SHA-3 [84] after a competition organised by the **NIST** (**N**ational **I**nstitute of **S**tandards and **T**echnology) to find a successor of SHA-2 that resists to post-quantum cryptographic attacks. SHA-3's design relies on a "sponge" construction, which processes data in two phases: absorbing input into an internal state and squeezing the hash output. This design is highly flexible and resistant to various cryptographic attacks in post-quantum era.

### Hashes as a Method for Storing Information in Blockchain

In blockchain systems, cryptographic hashes are not only used for securing and linking blocks but also for storing information in a compact and verifiable manner. By recording the hash of a piece of data on the blockchain, the original content is effectively timestamped, creating a proof of its existence

at a specific moment (Proof-of-anteriority). At a later time, the original content can be revealed to verify that it matches the stored hash, proving its authenticity and integrity. This approach ensures that the content remains private until the owner chooses to disclose it, while the blockchain guarantees its immutability and provides a trusted timestamp. Such a method is valuable for applications like intellectual property, legal proofs, or digital certifications, where preserving privacy and enabling selective disclosure are critical.

## 3.4 Merkle Trees and Data Integrity

Merkle trees [76] (or hash trees) are a critical data structure used in blockchains, enabling efficient and secure verification of large datasets. A Merkle tree is a binary tree where each leaf node represents the hash of a block of data, and each non-leaf node is the hash of the concatenation of the hashes of its child nodes. The root of the Merkle tree, called the Merkle root, summarizes all the transactions in a block, as shown in Figure 3.2. In

the Figure 3.2, D1...D7 represent any data, depending on the context Merle Tree is used it can represent transaction, an account state (see Ethereum account) or any other value. Merkle trees offer several key benefits for blockchain systems:

- **Efficient Verification.** Merkle trees allow blockchain nodes to verify individual transactions without needing to download the entire blockchain. A node can check the authenticity of a transaction by comparing the hashes along the path from the leaf to the Merkle root.

- **Data Integrity.** Since any change in the data of a single transaction will affect its hash, and in turn the hashes of all its ancestors up to the Merkle root, Merkle trees ensure that even the slightest modification to any part of the data can be detected. (thanks to the avalanche effect 3.3).

- **Scalability.** Merkle trees enable blockchains to scale by reducing the need for each node to store and process all transactions. Instead, nodes can store the Merkle root and use it to verify the integrity of any part of the blockchain.

In Bitcoin, Merkle trees are used to organize transactions within a block. The Merkle root is included in the block header and is part of the data that miners must hash when solving the Proof-of-Work puzzle. Any modification to a transaction will alter the Merkle root, thus ensuring that data tampering is easily detectable.

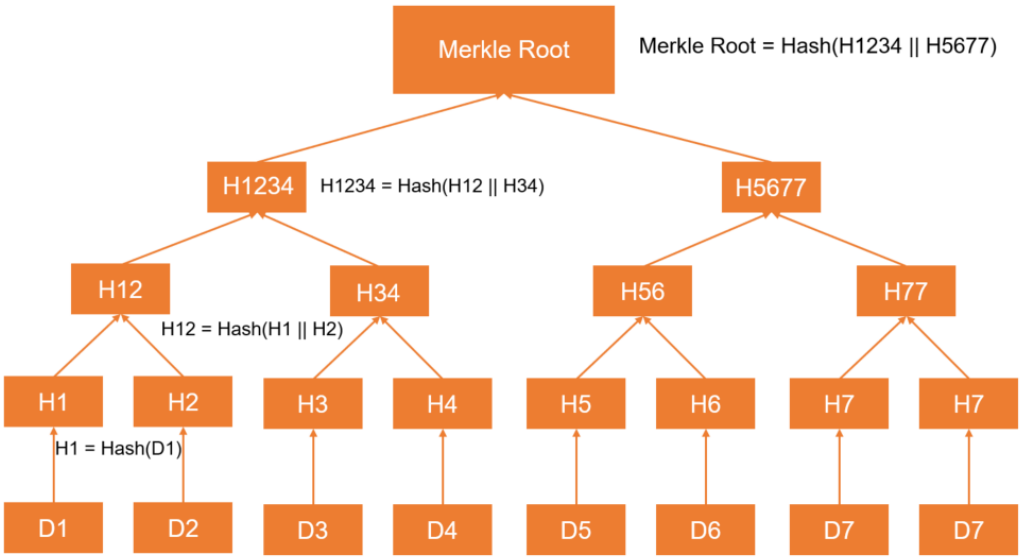



Figure 3.2: Merkle Tree Structure

## 3.5 Zero-Knowledge Proofs

Zero-Knowledge Proofs (ZKPs) were first introduced by Goldwasser, Micali, and Rackoff in 1985 [28]. They define a class of interactive proof systems where a prover can convince

a verifier that a statement is true without revealing any additional information beyond the validity of the statement itself. This concept was revolutionary, as it demonstrated that proofs could be both sound and privacy-preserving.

Although ZKPs have been well-established in theoretical cryptography for decades, their practical application has only recently gained traction, particularly in the blockchain space. Modern blockchain systems leverage ZKPs to reconcile the inherent transparency of distributed ledgers with privacy and regulatory compliance requirements. For example, ZKPs can be used to prove that a transaction does not exceed a specified threshold without revealing any details about the sender, receiver, or transaction amount.

This integration of ZKPs into blockchain protocols represents a recent and significant advancement, enabling scalable privacy-preserving solutions such as zk-SNARKs and zk-STARKs. These tools are now critical for applications like privacy-focused cryptocurrencies, secure voting systems, and cross-chain interoperability, providing a foundation for trustless yet compliant financial ecosystems.

## 3.6 Cryptographic Challenges in Blockchain

Despite its robustness, cryptography in blockchain faces several challenges. One of the primary concerns is the emergence of quantum computing, which threatens to break current cryptographic algorithms such as ECC and RSA. Quantum computers, once fully realized, could solve problems like the discrete logarithm problem and factorization, rendering many existing cryptographic protocols insecure. Some blockchains already use algorithms that are resistant to quantum attacks, such as hash functions based on SHA-3, which offer resilience against quantum computing threats. The majority of blockchain systems have not fully integrated quantum-resistant signature algorithms, which pose potential vulnerabilities to quantum attacks in the future.

In summary, cryptographic techniques are fundamental to the security and operation of blockchain systems. Asymmetric cryptography ensures secure transactions, cryptographic hash functions provide data integrity and security in agreement mechanisms, Merkle trees enable efficient verification of transactions and preserve data integrity, and elliptic curve cryptography ensures efficiency. Despite the advancements, challenges such as quantum security to push research toward even more robust cryptographic solutions for the blockchain space.

## 3.7 Understanding Key Blockchain Components: The Cases of Bitcoin and Ethereum

This section aims to introduce the fundamental concepts of blockchain. We will present the functioning of Bitcoin and Ethereum blockchains, which will allow us to explore their key components and gain a better understanding of their mechanisms.

### 3.7.1 Bitcoin: The First Practical Application of Blockchain Technology

Bitcoin, introduced by Satoshi Nakamoto in 2008 [1], is the first practical application of blockchain technology and was designed to facilitate peer-to-peer electronic cash transactions without the need for a central intermediary. The Bitcoin blockchain operates on a decentralized network where transactions are validated and recorded through a process known as mining. This process relies on a Proof-of-Work (PoW) agreement mechanism, in which miners compete to solve complex cryptographic puzzles. The first miner to solve the puzzle earns the right to add a new block to the chain and is rewarded with newly minted bitcoins. However, it is possible for two miners to simultaneously discover distinct valid blocks at the same height, where *height* refers to the distance of a block from the initial *genesis block* (i.e the first Bitcoin's block), measured as the number of links in the chain. When this occurs, the blockchain temporarily diverges into two competing branches, a situation known as a *fork*. To ensure eventual consistency, Bitcoin and other Proof-of-Work systems employ the *longest chain rule*, which dictates that nodes always adopt the branch with the *greatest cumulative proof-of-work* as the valid ledger. As additional blocks are mined, one branch will eventually grow faster than the other, becoming the dominant chain. Once this happens, all honest nodes converge on this longest chain, while the competing branch is discarded. Transactions contained exclusively in the discarded branch are returned to the pool of unconfirmed transactions (mempool), where they may later be included in future blocks. This mechanism enables decentralized networks to reach agreement on a single, authoritative ledger, even in the presence of temporary divergences.

Bitcoin is based on several fundamental concepts that form the backbone of the blockchain technology. In the following, we first present the fundamental components of Bitcoin separately, before illustrating how these elements interconnect to form the overall protocol and ensure the proper operation of the network.

**Transaction.** In Bitcoin, a transaction represents the transfer of value (in *bitcoins*, the cryptocurrency of the Bitcoin protocol) between participants. It is composed of *inputs*, which refer to the source of the bitcoins being spent, and *outputs*, which indicate the destination. Bitcoin operates on a model called UTXO (*Unspent Transaction Output*), which tracks the individual units of cryptocurrency available to be spent. Each UTXO represents an amount of Bitcoin that remains unspent after a transaction. A UTXO consists of two main components: the quantity of Bitcoin (denominated in satoshis, the smallest unit of Bitcoin), and a *ScriptPubKey*, a cryptographic script that defines the conditions for unlocking the UTXO. A UTXO is uniquely identified by the transaction ID (*TxID*) and an index, which differentiates it from other outputs in the same transaction. When a new transaction is made, one or more UTXOs are consumed as inputs, and new UTXOs are generated as outputs. This process continues with each transaction. The total balance of a Bitcoin address is simply the sum of all its unspent UTXOs.

This UTXO model provides several advantages: it is simple to understand, allows for the processing of multiple transactions in parallel, and offers enhanced security by preventing double-spending (in addition to the way transactions are validated through proof-of-work

which we introduce in the last chapter and that we explain in depth later in this chapter).

We can see a bitcoin as a chain of digital signatures that records the ownership transfers. Technically, you don't own bitcoins directly; instead, you hold transactions that have been sent to you. To spend bitcoins, you create new transactions based on those you have previously received. Users hold addresses, and transactions are executed from one address to another. Addresses are created through a process where the user selects a private key. The private key is used to derive a public key, and the address is formed by hashing the public key. In modern practice, two Bitcoin Improvement Proposals significantly enhance the management of private keys. BIP32 [2] (Bitcoin Improvement Proposal 32) introduces a method for deriving multiple addresses from a single master private key, while BIP44 [3] defines a hierarchical structure for handling multiple coins and accounts using that same master key.

This system allows for tracing the ownership history of a bitcoin (or a fraction of bitcoin). Consider the scenario where Alice sends a bitcoin to Bob, who then transfers it to Carole.

Alice creates a transaction based on a prior transaction, including:

- Bob's public key
- The hash of the previous transaction
- A signature of Bob's public key and the transaction hash, generated using Alice's private key (employing the ECDSA algorithm).

Now, Bob is considered the owner of that UTXO. For simplicity, we can say he "owns" the bitcoins associated with it. He can verify the authenticity of the sender by checking the transaction's signature using Alice's public key. To send the bitcoin to Carole, Bob would create a new transaction using the one he received from Alice. This new transaction would include:

- Carole's public key
- The hash of the previous transaction (this hash is generated by applying a hash function to Bob's public key, the prior transaction hash, and Alice's signature).
- A signature of the two elements above.

This is how Bitcoin transactions are formed, representing ownership transfers. Each person can only spend bitcoins they have previously received. Ownership can be verified by examining the transaction associated with a bitcoin, while the hash ensures data integrity, and the signature guarantees that only the bitcoin's owner can create valid

[2]BIP 32 specifications :https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
[3]BIP 44 specifications: https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki

transactions.

In practice, multiple received transactions can be combined or split into several new transactions. For example, someone could consolidate several received transactions into one or create multiple new transactions from a single input. Figure 3.3 illustrates this concept. Suppose there are seven Bitcoin addresses: A, B, C, D, E, F, and G. Address E has received four transactions from A, B, C, and D, amounting to 3, 6, 4, and 8 bitcoins, respectively. E wishes to send 9 bitcoins to F, 4 bitcoins to G, and 7 bitcoins to A. To achieve this, E would use the transactions they received, combining them as necessary. For instance, E might use the bitcoins from B's transaction to send 4 bitcoins to G, combine transactions from A and B to send 9 bitcoins to F, and use the 8 bitcoins from D to send 7 bitcoins back to A, keeping 1 bitcoin for future transactions.

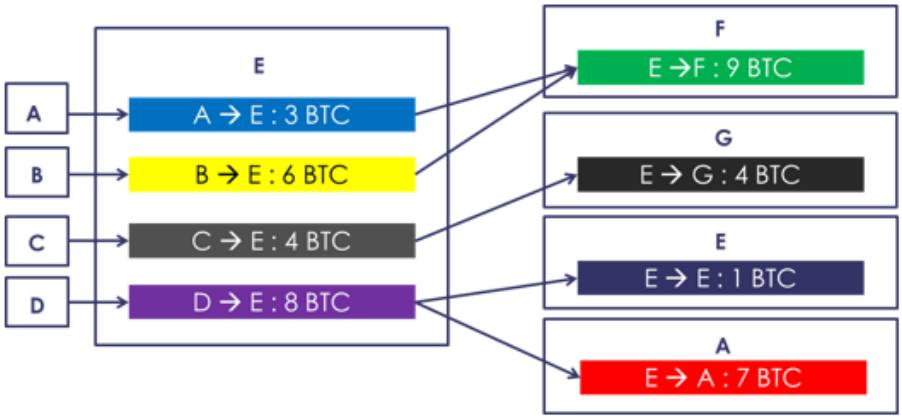



Figure 3.3: Illustration of UTXO Model.

In practice, in this case, E will send back to itself less than 1 BTC because it needs to include a fee (see paragraph "Mining and Rewards" 3.7.1) in the transaction to reward the validator. If E sets a fee of 0.001 BTC, it will send back to itself 0.999 BTC.

**Scripts.** A key feature of Bitcoin's transaction model is its scripting system, which defines the conditions under which a UTXO can be spent. Bitcoin uses a simple, stack-based, and intentionally non-Turing-complete scripting language called *Script* [1, 85]. The simplicity of this language is by design, providing enough flexibility to support different transaction types while avoiding the risks of unbounded loops or excessive computational complexity.

Each Bitcoin transaction output includes a *locking script* (or *ScriptPubKey*) that specifies the rules for spending that output. To unlock and spend the funds, the spender must provide an *unlocking script* (or *ScriptSig*) in the transaction input that satisfies the conditions set by the locking script. When a node validates a transaction, it concatenates the unlocking and locking scripts and executes them together on a stack. If the final result evaluates to `TRUE`, the transaction is considered valid.

The most common locking script type is the Pay-to-Public-Key-Hash (P2PKH), which

requires the spender to provide a public key and a digital signature corresponding to the hashed public key specified in the output. Other script types have been introduced through Bitcoin Improvement Proposals (BIPs), such as Pay-to-Script-Hash (P2SH, BIP16) [86], which allows more complex spending conditions like multi-signature schemes. For example, a 2-of-3 multisig wallet requires two valid signatures out of three possible signers to spend the funds, improving security and supporting organizational use cases.

More recently, the Taproot upgrade (BIP340-BIP342) [87, 88, 89] expanded the scripting capabilities of Bitcoin by introducing Schnorr signatures and Merklized Abstract Syntax Trees (MAST). These advancements enable more efficient multisignature transactions and enhanced privacy by allowing only the executed spending condition to be revealed on-chain, while others remain hidden.

Bitcoin's script system thus provides a flexible yet secure mechanism for expressing a variety of financial contracts, from basic payments to complex multi-party arrangements, without relying on a centralized intermediary. Despite its limited expressiveness compared to fully programmable blockchains like Ethereum, Bitcoin Script strikes a balance between functionality, security, and verifiability.

**Block.** A block in Bitcoin is a collection of verified transactions, which are then added to the blockchain through the Proof-of-Work mechanism (defined in paragraph 3.7.1). Each block contains a block *header* and a block *body*. The block *header* contains important metadata such as the hash of the previous block, the Merkle root summarizing the block's transactions, a timestamp, and a nonce, which is a number used in the mining process. The block *body* contains the list of transactions. Blocks are linked together (as the next block in the chain contains the hash of the previous block), forming a continuous, immutable ledger that records all transactions on the network.

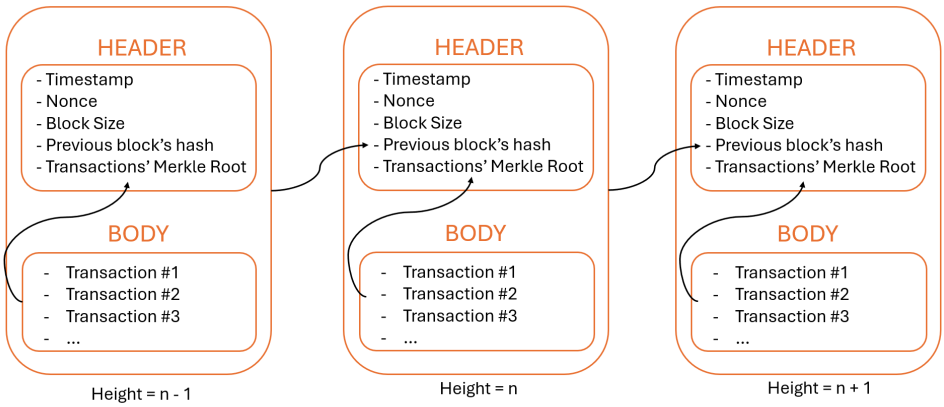



Figure 3.4: Illustration of three Bitcoin's block

See paragraph *How the Bitcoin Network works in practice?*, for a discussion about the immutability of the Bitcoin blockchain.

**Node.** A node refers to any device or computer that participates in the Bitcoin network by running the Bitcoin software. Nodes come in various forms, with the two primary types

being full nodes and lightweight nodes. Full nodes store the entire blockchain and are responsible for validating all transactions and blocks according to the agreement rules. In contrast, lightweight nodes only download the block headers and rely on full nodes to validate transactions. Both types of nodes ensure the decentralized nature of the network by propagating transactions and maintaining the integrity of the blockchain.

**Decentralized Peer-to-Peer Network.** Bitcoin operates on a decentralized peer-to-peer (P2P) network, where nodes communicate directly with each other without any central authority. This decentralized structure prevents any single entity from controlling the network, ensuring its censorship resistance and robustness against attacks. Transactions and blocks are verified and propagated across the network, allowing nodes to reach agreement on the current state of the blockchain.

**Mining and Rewards.** Miners (i.e nodes that participate in the block validation) play a pivotal role in the Bitcoin network by validating transactions and adding new blocks to the blockchain. As an incentive, miners receive a reward, which comprises two components. The first is the block reward, where a fixed number of bitcoins is awarded for adding a new block. This reward diminishes over time (from 50 BTC per block at Bitcoin launch and 3.125 BTC since the last halving at the date of writing in 2024), halving approximately every four years during an event known as the *halving*. The second part of the reward comes from transaction fees, paid by users for the inclusion of their transactions in the block. These rewards incentivize miners to participate in building new blocks, a process that inherently secures the network.

**How the Bitcoin Network works in practice?** The Bitcoin network works as a distributed system where agreement is reached through a process called *Proof-of-Work* (PoW). When a user broadcasts a transaction, full nodes on the network validate its authenticity by checking the inputs, signatures, and compliance with the network rules. Miners aggregate valid transactions, and attempt to validate them by proposing a new block.

In order to propose a new block, miners compete to solve a cryptographic puzzle by finding a nonce that, when combined with the block's data (header and transactions), generates a hash that meets a predefined target (the computed hash has to be inferior to a value known as *difficulty* which is a parameter of the block). This process, requiring significant computational power, ensures the security and integrity of the blockchain. When a miner successfully solves the puzzle, the new block is broadcast across the network. Other nodes validate the block, and if deemed valid, add it to their local copy of the blockchain.

However, forks (i.e when two different blocks are added at the same height in the blockchain, see Figure 3.5 that illustrate this situation where two nodes generate a valid block[4]) can occur due to network delays or, in some cases, malicious behavior where a node refuses to accept a valid block. Bitcoin resolves forks through the "longest chain

[4]The Figure 3.5 illustrates a fork situation in a blockchain network. Each circle represents a node maintaining its own local copy of the blockchain, and the colored arrows indicate the propagation of new blocks through the network. At some point, multiple miners (represented by the pickaxe icons) generate

rule," where nodes prioritize maintaining the longest valid chain. This process eventually resolves the fork, as only the blocks on the longest chain remain shared and accepted by the network.

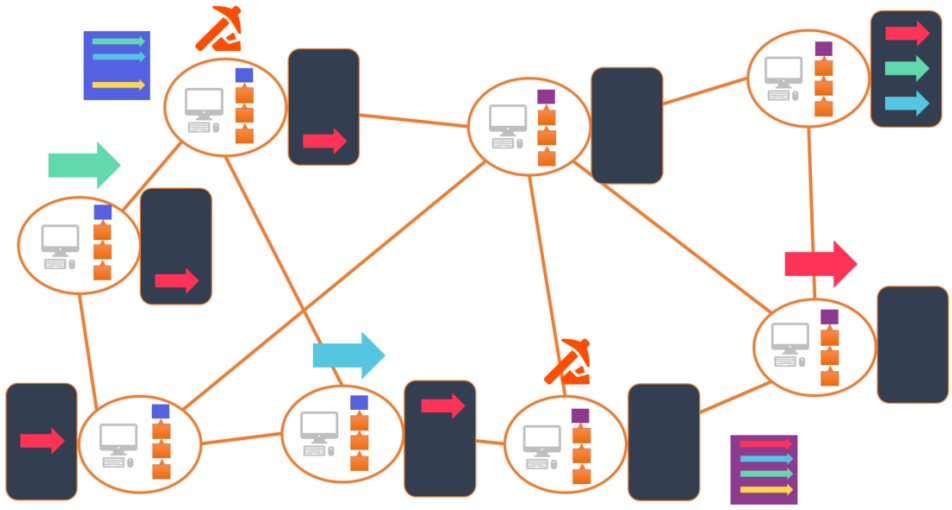


Figure 3.5: Illustration of a fork, where two differents nodes generate two valid blocks

The network reaches agreement when the majority of nodes accept the new block as part of the chain. This agreement is, however, probabilistic rather than deterministic in the classical sense of consensus as defined by Lamport et al. [35], and relies on eventual agreement rather than immediate finality. This decentralized mechanism ensures the blockchain's integrity and provides strong (though ultimately probabilistic) security. Transactions are often considered secure after six confirmations. Moreover, as specified by the original Bitcoin protocol rules, newly minted coins (coinbase rewards) cannot be spent until 100 blocks have been mined. This "coinbase maturity" requirement provides additional security, giving the network time to confirm the validity of newly created bitcoins before they can be spent.

These core concepts are foundational to Bitcoin and blockchain technology, enabling decentralized, secure, and transparent financial transactions.

The PoW mechanism creates a highly competitive environment among miners, who aim to solve cryptographic puzzles more quickly than their peers to earn the right to add new blocks and receive rewards. To enhance their chances of success, miners invest in high-performance computing hardware, such as Application-Specific Integrated Circuits (ASICs), which are specifically optimized for the complex calculations required in mining. Since there is no known shortcut to bypass these computations, having faster and more efficient hardware remains the only way to gain a competitive edge. This ongoing competition fosters a constant cycle of hardware upgrades to more powerful and energy-intensive models, resulting in an arms race for computational power. Consequently, Bitcoin mining consumes a substantial amount of electricity worldwide; some estimates suggest that the total energy consumption of the Bitcoin network exceeds that of several countries.

different blocks almost simultaneously. Because of network latency and partial connectivity, some nodes receive one block before the other, resulting in temporary divergence — two or more branches of the blockchain coexist.

In October 2025, the Cambridge Bitcoin Electricity Consumption Index [90] estimated Bitcoin's annual energy usage at between 109 TWh and 456 TWh, with a midpoint of around 211 TWh. Meanwhile, Digiconomist [91] placed that figure at about 196 TWh, ranking Bitcoin 23th in global energy consumption, between Thailand and Vietnam. The

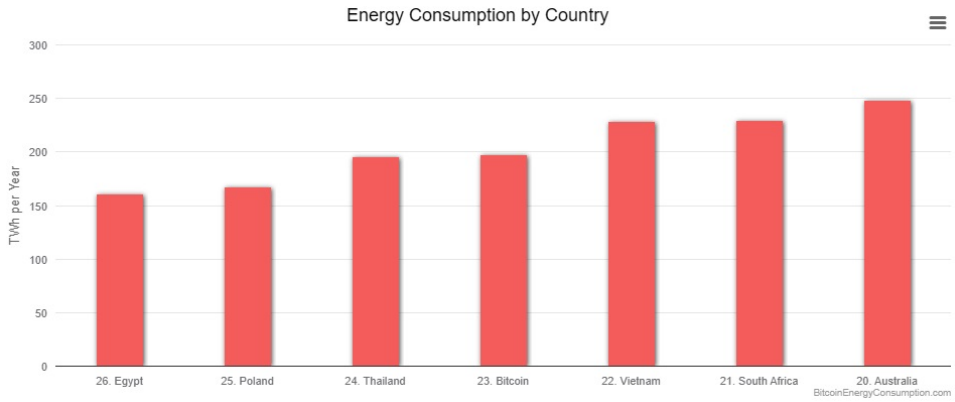



Figure 3.6: Comparative energy consumption of the Bitcoin network (source: Digiconomist Bitcoin Energy Consumption Index, accessed on October 6, 2025)

need for continuous operation of mining equipment, operating 24/7 to stay competitive, further amplifies this energy consumption, leading to significant electricity use and associated carbon emissions, particularly in regions reliant on fossil fuels and overall an important environmental impact. Recent studies have also highlighted the growing water footprint associated with Bitcoin mining, which is an often-overlooked aspect of its environmental impact. The "Bitcoin's Growing Water Footprint" study reveals that the cooling systems used in large-scale mining operations require vast amounts of water to prevent overheating of the mining equipment. In regions where water is scarce or where mining operations rely on freshwater sources, this additional demand can exacerbate local water shortages and negatively affect ecosystems [92]. The water usage associated with Bitcoin mining adds another layer of complexity to its environmental impact, extending beyond energy consumption and carbon emissions to include significant water resource implications.

### 3.7.2 The Ethereum Blockchain

Ethereum is a decentralized blockchain platform introduced in 2015 by Vitalik Buterin. Unlike Bitcoin, which primarily focuses on monetary transactions, Ethereum stands out for its flexibility and ability to execute smart contracts on an integrated virtual machine, the Ethereum Virtual Machine (EVM). This section explores Ethereum's foundational concepts with an emphasis of Ethereum 2.0, its role in the financial domain, and its broader implications, while briefly touching upon its energy efficiency improvements.

### Key differences between Ethereum 1.0 and Bitcoin

Ethereum is a public blockchain platform designed to go beyond simple peer-to-peer transactions by enabling complex decentralized applications. Like Bitcoin, it initially relied on the Proof-of-Work (PoW) agreement mechanism to validate transactions and maintain the integrity of the ledger. However, in September 2022, Ethereum underwent a major transition with the event known as "The Merge," which marked its shift to Proof-of-Stake (PoS). This change significantly reduced the network's energy consumption and laid the groundwork for future scalability enhancements.

A key distinction between Ethereum and Bitcoin lies in their state management models. Bitcoin operates using a UTXO (Unspent Transaction Output) model, where each transaction consumes previous outputs and generates new ones, creating a chain of value transfers. Ethereum, in contrast, adopts an account-based model. This model maintains a global state in which each account has an associated balance and, in the case of smart contracts, executable code. Ethereum distinguishes between externally owned accounts (EOAs), which are controlled by private keys, and contract accounts, which operate based on embedded smart contract logic. Transactions directly modify account balances and states, streamlining interactions and contract execution. While this model simplifies state transitions, it also introduces potential vulnerabilities, such as susceptibility to replay attacks if proper safeguards are not in place.

At the heart of Ethereum's programmability lies the Ethereum Virtual Machine (EVM), a runtime environment responsible for executing smart contract code. The EVM is a fully sandboxed virtual stack machine that provides a deterministic and platform-agnostic environment, allowing developers to deploy decentralized applications (DApps) with customizable behavior. This flexibility has established Ethereum as the primary platform for smart contract development.

The network is powered by ether (ETH), Ethereum's native cryptocurrency. Ether primarily serves as the platform's native token and functions as an incentive mechanism to align the interests of participants and secure the network. It is also used to pay for computational work in the form of "gas," a fee model that ensures fair resource allocation and mitigates denial-of-service risks, thereby supporting the proper operation of the Ethereum Virtual Machine (EVM).

### Smart Contracts

In Bitcoin, transactions are governed by a simple, stack-based scripting language that defines the spending conditions for each UTXO (see Section 3.7.1). These *scripts* are intentionally limited in expressiveness to ensure security and predictability. Ethereum extends this concept dramatically: instead of simple, predefined spending conditions, it introduces fully programmable *smart contracts*, enabling complex and dynamic logic to be executed directly on the blockchain.

The concept of a *smart contract* was first articulated by Nick Szabo in the mid-1990s, who envisioned self-executing agreements encoded in computer programs [93] and later

implemented in Ethereum (see the Ethereum Yellow Paper [94] for a technical specification). In the context of blockchain systems, a smart contract is a piece of code deployed on a distributed ledger that is executed deterministically by the network of participating nodes. Unlike traditional legal contracts, smart contracts are both *programs* and *stateful entities* residing on the blockchain.

Formally, a smart contract can be modeled as a finite state machine. Each contract has an associated state, which may include balances, mappings, or arbitrary variables. Transitions between states are triggered by transactions that carry data or value. These transactions are processed by the blockchain's agreement mechanism, ensuring that all nodes in the system update the contract state consistently.

Ethereum has popularized smart contracts through its *Ethereum Virtual Machine (EVM)*, a Turing-complete runtime environment where each node independently executes contract instructions. The EVM provides a sandboxed and deterministic environment, enforcing isolation between contracts and preventing side effects beyond the blockchain itself. Gas fees are introduced as a metering mechanism; each operation has a fixed gas cost (and ultimately, the gas price is set by the initiator of the transaction paid in Ether, ensuring that computational resources are fairly allocated and mitigating denial-of-service risks.

A key property of smart contracts is their *immutability*. Once deployed, the code cannot be altered, guaranteeing that the specified logic is preserved. This immutability, however, introduces risks: vulnerabilities in the code (e.g., reentrancy attacks, integer overflows) are permanent and can be exploited by adversaries. To mitigate these risks, formal verification, secure programming languages (e.g., Vyper [95]), and advanced auditing frameworks have been developed.

From a security standpoint, smart contracts leverage cryptographic primitives: all calls are digitally signed, ensuring authenticity and non-repudiation. Inter-contract communication is supported through message calls, enabling composability (when smarts contracts are on the same blockchain) but also increasing the surface area for cross-contract vulnerabilities.

In summary, smart contracts are verifiable and composable software artifacts that extend the capabilities of blockchains from simple asset transfers to fully programmable financial, organizational, and governance structures. Their emergence represents a paradigm shift, embedding logic and enforceability directly into distributed ledgers.

### Challenges and Limitations

Ethereum 1.0 faces several challenges related to its scalability, security, and competition. Scalability has been a critical issue, with high transaction fees and latency observed during periods of network congestion. Efforts to address this include the transition to Ethereum 2.0 and the implementation of solutions such as sharding, which aim to improve throughput and reduce costs. Security concerns are inherent in the ecosystem, as vulnerabilities in smart contract code can lead to financial losses, requiring rigorous audits and improved development tools.

### Ethereum 2.0

Ethereum 2.0, introduced with the Merge in September 2022, replaced Ethereum's Proof-of-Work (PoW) consensus with a Proof-of-Stake (PoS) protocol coordinated by the *Beacon Chain* [96]. This upgrade drastically reduced energy consumption (by more than 99%) and introduced a modular architecture where the *consensus layer* ensures ordering and finality of blocks, while the *execution layer* processes transactions and smart contracts inside *execution payloads*. The two layers communicate through the Execution–Consensus Engine API [97].

**Slots, epochs, and validators.** Time in Ethereum 2.0 is divided into *slots* of 12 seconds. In each slot, a validator is designated as *block proposer*, while a randomly selected committee of validators acts as *attesters*. A group of 32 slots is called an *epoch*, which is the unit for finality decisions. Validators are participants who lock up at least 32 ETH as a stake. Their voting power, called *effective balance*, is capped at 32 ETH and is used to weigh their attestations. Validators receive rewards for correct participation, while dishonest or inconsistent behavior (e.g., proposing two blocks in the same slot) results in penalties or slashing, i.e., the destruction of part of their stake.

**The LMD-GHOST fork-choice rule.** Ethereum 2.0 employs the *LMD-GHOST* (*Latest Message Driven Greediest Heaviest Observed SubTree*) rule to determine the canonical head of the chain. Unlike Bitcoin's *longest-chain rule*, which selects the branch with the most accumulated work, LMD-GHOST relies on validator attestations, weighted by stake.

The rule operates as follows:

1. The algorithm starts from the last *justified checkpoint* produced by the Casper FFG finality gadget. (see 3.7.2)

2. Each validator emits attestations (votes) for a block. Only the validator's *latest message* is considered (*Latest Message Driven*). This attestation contributes weight to the voted block and all its descendants.

3. The weight of a block is defined as the sum of the effective balances of validators whose latest votes support that block or one of its descendants.

4. Starting from the justified checkpoint, the algorithm repeatedly selects the child block with the greatest weight (*Greediest Heaviest SubTree*). This recursive process continues until a leaf is reached.

5. The final leaf is chosen as the canonical head of the chain.

Suppose that, from a justified checkpoint $C$, three competing blocks $A$, $B$, and $D$ are proposed. Validators cast their votes as follows:

- 60 ETH of stake for $A$ (or its descendants),
- 25 ETH of stake for $B$,
- 15 ETH of stake for $D$.

LMD-GHOST begins at checkpoint $C$ and selects child $A$ because it has the largest cumulative weight (60 ETH). Now assume $A$ has two children, $A_1$ and $A_2$, with:

- $A_1$: supported by 40 ETH,
- $A_2$: supported by 20 ETH.

The rule descends into $A_1$, again choosing the heaviest child. The process repeats until no further children are available. The resulting leaf is the canonical head of the chain. In this way, LMD-GHOST converges validators toward the most heavily supported branch, even under temporary forks.

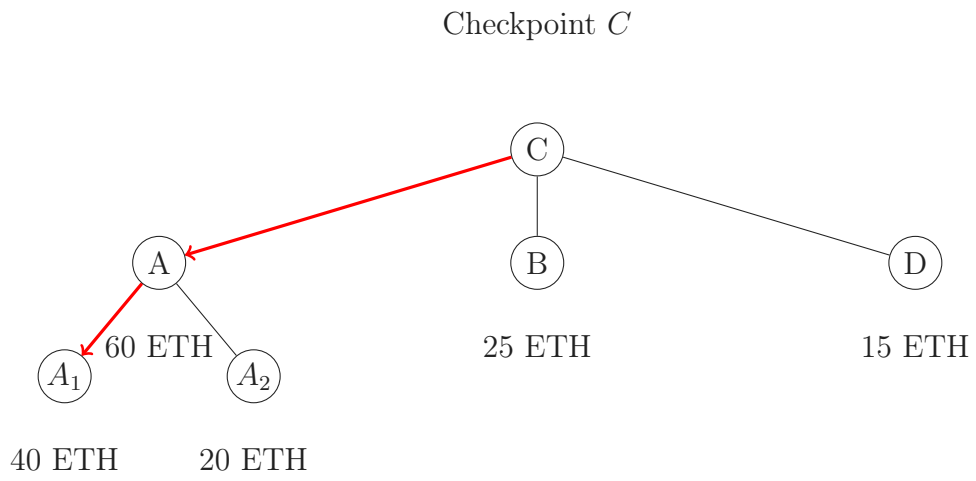



Figure 3.7: Illustration of the LMD-GHOST fork-choice rule. Starting from checkpoint $C$, the algorithm recursively selects the heaviest child (60 ETH $\rightarrow$ 40 ETH), leading to $A_1$ as the chain head.

**Casper FFG: finality through supermajority links.** Casper FFG (*Friendly Finality Gadget*) provides long-term economic finality based on *checkpoints* and stake-weighted validator votes [98]. Time is divided into *epochs* of 32 slots. In each epoch $e$, a *checkpoint* $C_e$ is defined as the block at the first slot of the epoch (or its nearest ancestor if that slot is empty).

Validators cast votes across epochs in the form of *source* $\rightarrow$ *target* links, denoted $(C_s \rightarrow C_t)$ with $s < t$. If at least 2/3 of the total stake supports such a link, the target $C_t$ becomes *justified*. If, at the same time, the source $C_s$ was already justified, then $C_s$ becomes *finalized*. Finalization makes all history up to $C_s$ irreversible, unless at least one-third of the total stake is slashed.

**Formal definitions.**

- **Checkpoint** $C_e$: the block at epoch $e$ (slot $32e$) or its ancestor.
- **FFG vote**: a pair $(C_s \rightarrow C_t)$ signed by a validator, weighted by its effective balance.
- **Justification**: $C_t$ is justified if $\sum_{v \in \mathcal{V}_{s \rightarrow t}} \text{stake}(v) \geq 2/3$ of the total stake.
- **Finalization**: if $C_s$ is justified and $(C_s \rightarrow C_t)$ gathers $\geq 2/3$ of the stake, then $C_s$ is finalized.

**Intuition.** Validator votes form a "chain of justification" across checkpoints. Once a link $(C_s \rightarrow C_t)$ gathers two-thirds of the stake and $C_s$ is already justified, the source $C_s$ becomes finalized. Finality therefore advances epoch by epoch. In practice, under normal network conditions, epoch $e$ is finalized when epoch $e{+}1$ is justified and a supermajority link validates $(C_{e+1} \rightarrow C_{e+2})$, which implies a latency of about two epochs.

**Safety and liveness.** Casper FFG guarantees (i) *safety*: two conflicting checkpoints cannot be finalized without slashing at least one-third of the total stake; and (ii) *liveness*: under partial synchrony, if the network stabilizes and at least two-thirds of validators participate honestly, new checkpoints will continue to be finalized [99].

**Finality progression over three epochs**

Consider checkpoints $C_e$, $C_{e+1}$, and $C_{e+2}$, with the total validator stake normalized to 1 (i.e., 100%). The aggregated votes per epoch are:

- **Epoch** $e$+1: at least 2/3 of the stake votes for the link $(C_e \to C_{e+1})$. The target $C_{e+1}$ becomes *justified*.
- **Epoch** $e$+2: at least 2/3 of the stake votes for $(C_{e+1} \to C_{e+2})$. Since $C_{e+1}$ was justified, the source $C_e$ becomes *finalized*.

**Reading.** Under honest participation and stable network conditions, finality typically progresses with a two-epoch latency: epoch $e$ is finalized once epoch $e$+1 is justified and a supermajority link validates $(C_{e+1} \to C_{e+2})$. Any attempt to finalize a conflicting checkpoint would necessarily result in slashing at least one-third of the total stake.

**Security and scalability mechanisms.** Several complementary mechanisms strengthen the protocol. Validator assignments are randomized each epoch using RANDAO-based randomness to prevent collusion [100]. Inactivity penalties guarantee progress even if many validators go offline, ensuring eventual finality. Lightweight synchronization is enabled by *sync committees*, rotating subsets of validators that collectively sign the head of the chain, allowing light clients to track progress efficiently [101]. On the scalability side, Ethereum 2.0 adopts a rollup-centric roadmap: the base layer provides security and data availability, while most transaction throughput is expected to come from layer-2 protocols.

**Summary.** Ethereum 2.0 thus combines (i) stake-weighted fork choice (LMD-GHOST) for fast head selection, (ii) checkpoint finality (Casper FFG) for strong safety guarantees, and (iii) additional mechanisms for randomness, penalties, and light-client support. Compared to Ethereum 1.0, this architecture achieves deterministic finality, higher energy efficiency, and compatibility with modular scaling solutions, reinforcing Ethereum's role as a programmable and sustainable blockchain platform.
Recent research by *Pavloff et al.* has further analyzed the underlying components of Ethereum's Proof-of-Stake consensus, such as its randomness mechanism (RANDAO) and penalty schemes (e.g., inactivity leaks), and highlighted potential Byzantine attack vectors that can impact liveness and safety [102, 103].

## 3.8 Conclusion

This chapter reviewed the cryptographic pillars of blockchain systems: asymmetric keys and digital signatures for authenticity, hash functions and Merkle trees for integrity and

efficient verification, and their composition in deployed protocols (e.g., Bitcoin's UTXO model, Ethereum's account model). We highlighted that security and performance arise not only from strong primitives but from how they are assembled under concrete network and governance assumptions.

Two brief takeaways: (i) cryptography is a *first-order dependency* that shapes guarantees, costs, and even energy footprint; (ii) application constraints (latency, throughput, auditability, compliance-govern acceptable protocol) trade-offs.

The next chapter introduces the system model and trusted components that form the foundation for the Byzantine fault tolerance mechanisms developed later in this manuscript.

# Chapter 4

# Blockchain System Model and Building Blocks

> "A map is not the territory it represents, but, if correct, it has a similar structure to the territory, which accounts for its usefulness."
> — Alfred Korzybski, *Science and Sanity* (1933)

Before delving into the Byzantine Reliable Broadcast as well as the Tendermint and TenderTee protocols, we first provide a rigorous definition of the system model that will serve as a foundation throughout the remainder of this manuscript.

The system model presented in Section 4.1 follows the formalization introduced by Amoussou-Guenou [104], which provides a comprehensive framework for reasoning about Byzantine fault-tolerant consensus protocols in blockchain settings. We adopt this model as it offers the necessary rigor and generality for our contributions. Section 4.2 then extends this foundation by introducing Trusted Execution Environments (TEEs) and shows how they can be integrated into consensus protocols.

## 4.1 System Model

We adopt the system model and notational conventions introduced in [104], with minor adaptations. This provides a clear and consistent foundation for describing Tendermint and for later extending it to the TenderTee protocol (both protocols are presented in Chapter 6). The model takes into account the number of participants in the system, their behavior, how connected they are, and how they can exchange information, *etc.*

### 4.1.1 Notations

In this section, we introduce some notations that we use throughout the manuscript.

Let $S$ be a set, we denote by $|S|$ the cardinality of $S$; and $2^S$ or $\wp(S)$ is the power set of $S$, the set of all subsets of $S$.

Let $b \in \{\texttt{false}, \texttt{true}\}$ be a boolean value. $\mathbb{1}_b$ denotes the indicator function, which outputs the value 1 if $b = \texttt{true}$, and 0 if $b = \texttt{false}$. Let $b_1, b_2 \in \{\texttt{false}, \texttt{true}\}$ be two boolean variables. The formula $b_1 \wedge b_2$ is evaluated at $\texttt{true}$ if and only if $b_1 = \texttt{true}$ and $b_2 = \texttt{true}$, and at $\texttt{false}$ otherwise. The formula $b_1 \vee b_2$ is evaluated at $\texttt{true}$ if $b_1 = \texttt{true}$ or $b_2 = \texttt{true}$; if $b_1 = b_2 = \texttt{true}$, $b_1 \vee b_2$ is also evaluated at $\texttt{true}$, i.e., $b_1 \vee b_2$ is evaluated at $\texttt{false}$ if and only if $b_1 = b_2 = \texttt{false}$. The formula $\neg b$ is evaluated at $\texttt{true}$ if and only if $b = \texttt{false}$.

Let $A$ be a non-empty set. We denote by $A^*$ the set of all finite concatenations of elements of $A$, i.e., if $u \in A^*$, then either $u = \varepsilon$, or $\exists k \geq 1 : u = u_1 \cdot \ldots \cdot u_k$, where $\forall i \in \{1, \ldots, k\}, u_i \in A$. The operation $\cdot$ represents the concatenation, the empty word $\varepsilon$ is such that $\forall a \in A, a \cdot \varepsilon = \varepsilon \cdot a = a$, and we write $A^+$ for the set $A^* \setminus \{\varepsilon\}$. Note the following exception, $\mathbb{N} = \{0, 1, \ldots\}$ is the set of natural numbers, and $\mathbb{N}^* = \mathbb{N} \setminus \{0\} = \{1, 2, \ldots\}$ is the set of all positive natural numbers.

### 4.1.2 Participants

A blockchain system is composed of a set $\Pi$ of *participants* or players, or nodes, or processes communicating by exchanging messages over a network. Messages represent (a batch of) transactions. We consider that messages have finite length and can be arbitrary sequences of information.

A participant models a computer program, the couple human/machine, *etc.*., that takes part in the blockchain system.

#### Execution Model

Each participant has an internal state and proceeds in rounds. At the end of each round, the participant goes to the next round and so on. A round is composed of three phases:

- A *send phase* at the beginning of the round. It is during the send phase that a participant can send messages to the other participants. The message to send should be prepared during the previous rounds, and as for the send phase of the first round, an initial (default) value may be sent. Following the model in [104], we consider that the send phase is executed atomically no matter the number of messages or the size of each message. Atomically means that if one message is sent then all messages prepared for this phase are also sent. After the send phase, the participant goes to the delivery phase.

- A *delivery phase*: During this phase, a participant collects from the network messages previously sent. These messages can be from the current round or the previous rounds. We consider that participants have an unbounded memory, i.e., they can store all messages they collect. When a participant effectively collects a whole message and can process it, we say that it has *delivered* the message, otherwise, even if part of the message is collected, it has not delivered the message. The delivery phase has a finite and positive duration. The duration represents the time the participant plans to wait for collecting messages sent. After its delivery phase, the participant goes to the compute phase.

- A *compute phase* at the end of the round. In compute phases, a participant uses all messages it delivered to update its internal state and to prepare the messages for the next round. After the compute phase, the participant goes to the send phase of the next round.

Hence, participants receive information from their environment by means of messages, they use the received information in their computations and they produce messages to the environment.

The duration of the send phase and the duration of the compute phase are negligible since they are absorbed by the duration of the messages exchanges. We then consider the duration of one round as being exactly the duration of its *delivery phase*. For the same or for different participants, different rounds can have different duration.

Formally, the execution of each participant can be viewed as an *input/output automaton* [105]. An input/output automaton $\mathcal{A}$ is a tuple $(Signature, States, Init, Transition)$, where:

- $Signature$ is the actions signature of the automaton. It is a non-empty set of internal, input, and output actions;

- $States$ is a non-empty set of states;

- $Init \subseteq States$ is the set of initial states;

- $Transition \subseteq States \times Signature \times States$ is the transition relation.

We assume that participants execute their automata in a *sequential* way, i.e., they execute one step of instruction (computational or communication steps) at a time. This does not prevent them from executing several threads of instructions with appropriate multiplexing.

Let $n > 0$ be a positive integer, and let $\mathcal{A}_i = (Signature_i, States_i, Init_i, Transition_i)$ be some input/output automata, $\forall i \in \{1, \ldots, n\}$. We say that the *composition* of the automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$, is the automaton $\mathcal{A} = (Signature, States, Init, Transition)$ if and only if:

- $Signature = \sqcup_{i=1}^{n} Signature_i$, where $\sqcup$ is the disjoint union;
- $States = \prod_{i=1}^{n} States_i$;
- $Init = \prod_{i=1}^{n} Init_i$;
- If $((s_1, \ldots, s_n), a, (d_1, \ldots, d_n)) \in Transition$, then $\exists i \in \{1, \ldots, n\} : a \in Signature_i$ and $(s_i, a, d_i) \in Transition_i$, and $\forall j \neq i, d_j = s_j$.

When the automaton of a participant runs, it generates *execution*s, where an execution is a sequence of alternating states and actions. The execution of the composition of automata of all participants in the system is called the *global execution*, or just the *execution* of the system.

The notion of participant synchrony refers to the speed of execution of the participants [106]. The participants are synchronous if they all take the same amount of time to execute an instruction. The participants are asynchronous if they are not synchronous; in more detail, one or more participants can remain at the same instruction for an arbitrarily long but finite amount of time, while the others continue their execution. In asynchronous systems, the speed of execution of one participant can differ from the speed of execution of another participant; the relative speeds of participants are not common knowledge and are unbounded.

We assume that there is a *fictional global clock* that takes values in $\mathbb{T}$, i.e., where $\mathbb{T} = \mathbb{N}$, the system evolves in discrete time. When we refer to a *point in time* or a *date*, it is with respect to the global clock. To represent the possible differences in participants' speed, we assume that each participant has a local clock. We note that participants do not have access to the global clock. When the participants are synchronous, their local clocks are synchronized with one another. When the participants are asynchronous, on the other hand, their local clocks are not necessarily synchronized among the participants.

### 4.1.3 Communication Network

In order to communicate, participants are linked to each other over a network. A network is a couple $(\Pi, E)$ where $\Pi$ is the set of participants in the system, and $E \subseteq \Pi \times \Pi$ is the set of bidirectional communication channels (edges) between participants, i.e., $\forall i, j \in \Pi$, if $(i, j) \in E$, then $(j, i) \in E$. Let $i, j$ be two participants, if $(i, j) \in E$, there exists a communication channel between them; we say that $i$ is a neighbour of $j$ and vice-versa. If $(i, j) \notin E$, there is no direct communication channel between $i$ and $j$, they are not neighbours of each other.

We say that there is a *path* between two participants $i$ and $j$ if $i$ and $j$ are neighbours, or if there exists a finite sequence of participants $\pi_1, \pi_2, \ldots, \pi_n$, where $n$ is the length of the path, such that the following conditions hold:

- $\pi_1 = i$, called the source;
- $\pi_n = j$, called the recipient; and
- $\forall k \in \{1, \ldots, n-1\}, (\pi_k, \pi_{k+1}) \in E$.

When between any two participants there exists a path, we say that the network is *connected*.

We say that two paths are *(node) disjoint* if they do not share any participant in common except the source and the recipient. A network is *k-connected* if and only if it contains at least $k$ disjoint paths between any two participants.

We assume that the networks considered in this manuscript are always connected.

Participants can send messages to their neighbours through this network. We assume *reliable channels* (i.e., messages are neither created nor corrupted by the network and, if sent infinitely often, are delivered infinitely often).

The networks presented are called *static*. A more general class of network exists where the communication channel between participants can vary over time (being present or not). We say that the network is dynamic. Dynamic networks can be modelled using *Time Varying Graphs* as defined by Casteigts *et al.* [107]. A Time Varying Graph is a tuple $(\Pi, E, \rho, \zeta)$, where:

- $(\Pi, E)$ is the underlying static graph. If $\exists i, j \in \Pi$ such that $(i, j) \in E$, it means that there is a point in time when $i$ and $j$ are neighbours.
- $\rho : E \times \mathbb{T}^+ \to \{0, 1\}$, is the *presence* function, such that $\rho(e, t) = 1$ if and only if the communication channel is active at time $t$.

  $\rho$ indicates whether the channel given in parameters is active at the given time.
- $\zeta : E \times \mathbb{T} \to \mathbb{T}$, is the *latency* function. It indicates the time it takes to cross a given communication channel at a given time.

  We note that the latency of a communication channel can vary over time.

The above notion of path cannot be applied in Time Varying Graph. The notion of time should be taken into account. We say that $\pi_1, \ldots, \pi_n$ is a *dynamic path* between participant $i$ and $j$ if $\exists t_1, \ldots, t_n \in \mathbb{T}$ such that:

- $\pi_1 = i$, $\pi_n = j$, and $\forall k \in \{1, \ldots, n-1\}, (\pi_k, \pi_{k+1}) \in E$;
- $\forall k \in \{1, \ldots, n-1\}, \forall t \in [t_k, t_k + \zeta((\pi_k, \pi_{k+1}), t_k)], \rho((\pi_k, \pi_{k+1}), t) = 1$; for all $k \leq n-1$ at date $t_k$, there exists for a sufficiently long enough time a channel for $\pi_k$ to send a message to $\pi_{k+1}$;

- $\forall k \in \{1, \ldots, n-1\}, \zeta((\pi_k, \pi_{k+1}), t_k) \leq t_{k+1} - t_k$, for all $k \leq n-1$, messages sent by $\pi_k$ to $\pi_{k+1}$ are received before $t_{k+1}$.

A static network $(\Pi, E)$ is a special class of a Time Varying Graphs $(\Pi, E, \rho, \zeta)$ where $\forall t \in \mathbb{T}, \forall e \in E, \rho(e, t) = 1$, the communication channels if they exist are always active. The latency or message transmission delay is discussed in Section 4.1.6.

### 4.1.4 Dynamicity of the System

The number of participants in a blockchain system can be known or not, and finite or not bounded. Blockchain systems can be classified into different classes, depending on who can access the system, and who can participate in maintaining it. In particular, blockchain systems are often open, i.e., new participants can enter in the system at any point in time.

A dynamic distributed system is *a continuously running system in which an arbitrarily large number of participants are part of the system during each interval of time and, at any time, any participant can directly interact with only an arbitrary small part of the system* [108]. Dynamic distributed systems can be categorised by the number of participants that can arrive (enter) in the system during its execution. That number is referred to as the *arrival model* [109] of the system. Participants can always leave the system. The arrival models are classified into three different categories:

- *n-arrival model*, where $n$ is a positive integer: the system is composed of exactly $n$ participants, and $n$ is known by the system. Therefore, $n$ can be used by the protocols.

- *Finite arrival model*: there is an infinite number of participants, but in each execution of the system, only a finite number of participants can arrive (enter in the system). The number of arrivals is however not known.

- *Infinite arrival model*: there is an infinite number of participants, and at each time in the execution of the system, new participants can always arrive in the system.

### 4.1.5 Failure Models

A distributed *protocol* or a sequence of instructions can be viewed as an assignment of automata to every participant in the system. When a participant executes its assigned automaton, we say that the participant follows the protocol; otherwise, we say that it deviates from the protocol. In this manuscript, we consider that participants can fail [110] at executing a given protocol. A participant can fail in different ways: for example, it can *crash*: from a point in time, it is not executing the protocol any more; it can fail by *omission*: it is not able to send or to receive messages any more; it can fail by sending contradictory messages, *etc.*. More generally, a participant can fail in any arbitrary way.

A participant that can fail in the system is called a *faulty* or a *Byzantine* participant [111]; Byzantine participants can control the network by modifying the order in which messages are received, but they cannot postpone forever message receptions. Moreover, Byzantine participants can collude to « pollute » the computation (e.g., by sending messages with different contents, while they should send messages with the same content if they were non-faulty). Participants that always follow the protocol are called *correct*. In particular, a correct participant is always present in the system since its beginning, and for the whole execution.

Since Byzantine participants can behave arbitrarily from a prescribed protocol, we do not make any assumption on their behaviour in our analyses. Byzantine faults are the worst faults that can occur and the most general, hence an interesting class of study. In particular, when a protocol is tolerant to Byzantine participants, we have the assurance that no fault can break it.

### 4.1.6 Time Assumptions

When a participant sends a message, other participants do not deliver the message instantaneously, there is a *message (transmission) delay* or a *transmission latency*. Let $\Delta \geq 0$ be the maximum message delay between any two participants, i.e., $\forall i, j \in \Pi$, if $i$ sends a message to $j$ at date (time) $t$, then at date $t + \Delta$, $j$ has delivered the message. The communication can be classified into different categories based on the finiteness of $\Delta$ and whether the participants know its value or not.

- **Synchronous Communication**: In synchronous communication systems, $\Delta$ is finite and known by the participants. If a participant broadcasts a message at a time $t$, then at time $t + \Delta$, all participants have delivered the message.

  Since the maximum delay $\Delta$ is known, participants can use that information in their execution; typically, they can match the duration of their delivery phase with $\Delta$ to ensure the reception of all messages sent at the beginning of the round.

- **Asynchronous Communication**: In asynchronous communication systems, $\Delta$ is infinite. That means that there is no upper bound on the message transmission delay. Messages sent are eventually delivered, since messages sent cannot be lost; but no one can predict an upper bound on when a message will be delivered.

- **Semi-Synchronous Communication**: Between synchronous and asynchronous communications, there exists a class of communication systems called semi-synchronous. In semi-synchronous systems, $\Delta$ is finite. We can further divide the class as follows:

  - *Partial Synchronous* Communication. As defined by [112], a communication system is partially synchronous if $\Delta$ is finite but unknown by the participants.

    Since $\Delta$ is not known, participants cannot use it directly, they may however try to guess it during their execution.

– *Eventual Synchronous* Communication. A communication system is eventually synchronous if there exists an unknown date $t \in \mathbb{T}$, called the *Global Stabilisation Time* (GST) such that all messages sent after date $t$ are received within the bound $\Delta$. There is no guarantee on message transmission delay for the messages sent before GST; however, the message will be eventually delivered since messages cannot be lost. The period before GST is called the *asynchronous period*, and the time after GST is called the *synchronous period*. It is important to note that since participants do not know when GST will be, they can only try to guess if GST is already reached during their execution. That is the case whether $\Delta$ the message transmission delay (hereafter GST) is known or not.

Most works in the literature assume that $\Delta$ is known, and only the GST is unknown. However, to the best of our knowledge, it is not known if that model is equivalent to having both $\Delta$ and GST unknown. If $\Delta$ is known, it seems that it can be used with a certain multiplicative factor to design the duration of the delivery phases of participants, while when $\Delta$ is unknown, it is not possible.

In this work, to be the most generic as possible, when using an eventual synchronous system, we assume that both GST and $\Delta$ are finite but unknown.

An important remark is that the synchrony assumptions of the communication hold no matter the size (finite) of the messages, i.e., no matter how long (but finite) a message is, it is delivered entirely with respect to the communication's time assumptions.

**Remark 1.** *Baldoni et al.[108] and Muñoz-Escoí et al.[113] discussed and proved that in infinite arrival models, only asynchronous communication is possible, whereas in finite arrival models it is possible to achieve a semi-synchronous communication, and synchronous communication can only be achieved in n-arrival models.*

Intuitively, in finite arrival models, there is a time from when no new participants enter the system, hence the maximum number of communication channels between any two participants can be bounded, and so can be the message transmission delay.

### 4.1.7 Communication Primitives

One of the needs in blockchain systems is to ensure that all participants have the same information and messages. To this end, we consider a broadcast primitive. A broadcast is a technique to send a message to all other participants. If a participant broadcasts a message $m$, it wants all other participants to eventually deliver $m$.

It is often sufficient to have a *Byzantine reliable broadcast* [114]. Following Bracha, [115], we define *Byzantine Reliable Broadcast* as follows:

**Definition 1** (Byzantine Reliable Broadcast problem)**.** *A broadcast is a Byzantine Reliable Broadcast if it satisfies:*

- *brb-CorrectInit: If the initiator is correct, all correct processes deliver the initiator's value.*

- *brb-ByzantineInit: If the initiator is Byzantine, then either no correct process delivers any value, or all correct processes deliver the same value.*

This means that all other correct participants eventually deliver all messages sent or delivered by at least one correct participant. It does not necessarily imply that the order in which messages are sent is the same order of messages delivered.

To disseminate a message to all participants, a *dissemination protocol* can be used. In such a protocol, if one participant receives a message for the first time, it sends the message to all its neighbours. Intuitively, if all participants are correct, eventually all participants will deliver the message. In contrast, when there are some Byzantine participants, such approach does not guarantee the property of reliable broadcast as shown by the following example.

**Example 1.** *Let the following dissemination protocol for broadcast: if one participant receives a message for the first time, it sends the message to all neighbours.* $(\Pi, E)$ *is the network such that* $\Pi = \{i, j, k\}$ *and* $E = \{(i, j), (j, k)\}$*. Let* $i$ *and* $k$ *be correct, and let* $j$ *be Byzantine.* $i$ *and* $j$ *are neighbours, and so are* $j$ *and* $k$*, but* $i$ *and* $k$ *are not.*

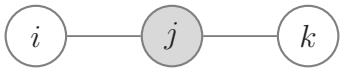



Figure 4.1: Communication Network of Example 1.

*Participant* $i$ *wants to broadcast the message* $m$*, and by following the protocol, it sends it to* $j$*.* $j$ *when receiving* $m$ *deviates and does nothing. Here,* $k$ *does not receive the message* $m$*.*

However, with some hypotheses on the communication network, it is possible to achieve a reliable broadcast in the presence of Byzantine participants as shown by the following remark.

**Remark 2.** *Let* $n$ *be the total number of participants and* $f$ *be the maximum number of Byzantine participants. In a static network with Byzantine participants, Dolev [116] shows that when the network is* $(2f + 1)$*-connected, it is possible to guarantee that all messages sent by correct sources are delivered by correct participants. Similarly, Maurer et al. [117] shows that there exists a class of Time Varying Graphs where it is possible to guarantee that all messages sent by a correct source are eventually delivered by correct participants; the result of Maurer et al. [117] also depends on the maximum number of Byzantine participants in the system. Both propositions however do not satisfy the property brb-Termination-2. If* $n = 3f+1$*, Bracha [115] and Raynal [110] show that there exist broadcast protocols where all participants share the same set of messages, satisfying brb-Termination-2. Hence, by combining the results of Dolev [116], Maurer et al. [117], Bracha [115] and Raynal [110], Byzantine reliable broadcast is possible.*

Intuitively, theses results require that the network is « sufficiently connected », such that there always exist enough (dynamic) paths with no Byzantine participants between any two correct participants. In the same spirit, Komatovic *et al.*[118] proposes a protocol for dynamic Byzantine reliable broadcast.

In the following presentation of Tendermint, we assume the existence of a reliable broadcast that is used by participants, and we do not discuss the network nor the broadcast protocols. However, as explained in Remark 2, the existence of such a primitive implies some restrictions on the connectivity of the communication network.

To broadcast a message $m$, a participant uses the primitive broadcast($m$) during one of its send phase $m$. During a delivery phase, a participant receives a broadcast of a message by executing the primitive delivery().
Later in this manuscript, in Chapter 5, we will introduce a new communication primitive that implements Byzantine Reliable Broadcast with a $(f)$-fault tolerance, for a network of size $2f + 1$.

### 4.1.8 Blockchain Consensus

Consensus, introduced by Lamport, Shostak, and Pease in their seminal work on the Byzantine Generals Problem [35], is one of the fundamental problems in fault-tolerant distributed computing. In the consensus problem, a set of $n$ nodes, also referred to as participants or processes, must reach agreement on a single value, even in the presence of up to $t$ malicious or faulty nodes. These faulty nodes may behave arbitrarily or maliciously, posing a significant challenge to achieving consensus.

The consensus problem is critical in distributed systems because it ensures all non-faulty nodes agree on a common state or value, which is essential for maintaining consistency and reliability. The agreement must satisfy three key properties:

**Definition 2.** *(Consensus):*

- ***Termination**: Every correct node decides on a value (a block).*
- ***Agreement**: If two correct node decide respectively on values $B$ and $B$', then $B = B'$*
- ***Validity**: A decided value by any correct node is valid; it satisfies a predefined predicate.*

We use the consensus definition use in Amoussou-Guenou thesis [104] that use the concept of *external validity* introduce by Cachin *et al.* in [119]. The validity predicate must be known by all participants and is defined by the given application. External validity was later adapted by Crain *et al.* in [120] as a well-suited validity concept for blockchains.

These properties ensure that the system can continue to function correctly even if some nodes fail or act maliciously, a feature that is indispensable for distributed systems and blockchain networks.

### 4.1.9 Blockchain Repeated Consensus

The repeated production of blocks by the different committees can be seen as multiple instances of agreement, where at each height, exactly one block should be decided.

Many abstractions related to multiple consensus have been studied. The concept of *multi-consensus* is presented in [121], where the authors assume that only the faulty participants can postpone the decision of correct participants; additionally, the consensus is reached a finite number of times. The *long-lived consensus* presented in [122] studies the consensus when the inputs are changing over time; their specification aims to study under which conditions the decisions of correct participants do not change over time. However, none of the specifications previously cited are appropriate for blockchain systems. In [123], Delporte-Gallet *et al.* defined the *repeated consensus* as an infinite sequence of consensus instances, where the input values may be completely different from one instance to another; but all correct participants have the same infinite sequence of decisions.

Blockchain consensus can be viewed as a repeated consensus problem. However, in [123] only crash failures were considered. Amoussou-Guenou proposes in [104], Definition 3 a generalization of the *repeated consensus* resilient to Byzantine failures. The main difference is that we do not predicate on the faulty participants, including the Byzantine ones. Each correct participant outputs an infinite sequence of decisions; each decision corresponds to a block in the blockchain analogy. We call that sequence of decisions the *output* of the participant.

**Definition 3** (Repeated Consensus)**.** *An algorithm implements a Byzantine repeated consensus if and only if it satisfies the following properties:*

- brc-Termination. *Every correct participant has an infinite output;*
- brc-Agreement. *If the $i^{th}$ value of the output of an correct participant is $B$, and the $i^{th}$ value of the output of another correct participant is $B'$, then $B = B'$;*
- brc-Validity. *Each value in the output of any correct participant is valid; it satisfies a predefined predicate.*

Alternatively, brc-Termination can be stated as follows: « $\forall k \in \mathbb{N}$, every correct participant eventually outputs the $k^{\text{th}}$ value ». However, we use the definition above to stick with the definition of Delporte-Gallet *et al.*[123].

If clear from the context, we simply write Termination, Agreement and Validity instead of respectively brc-Termination, brc-Agreement and brc-Validity.

If an algorithm implements the repeated consensus then each correct participant will have an infinite sequence of decisions (blocks); any two correct participants will have the exact same sequence (the same blockchain), and all blocks in the sequence will be valid with respect to the application-dependant predicate. This abstraction corresponds to the idea of a blockchain, as the output of each participant is its local sequence of blocks.

### 4.1.10 FLP impossibility result

The FLP impossibility result, established by Fischer, Lynch, and Paterson in 1983 [124], is a fundamental theorem in the field of distributed computing that demonstrates the limitations of achieving consensus in asynchronous systems. The result states that in an asynchronous distributed system, where at least one process can fail by crashing, there is no deterministic algorithm that can guarantee consensus (no deterministic algorithm can satisfy all three consensus properties at the same time) among the correct processes. This is because, in such an environment, a protocol cannot distinguish between a slow process and a failed process, making it impossible to guarantee termination without potentially violating agreement or validity.

The FLP impossibility result has significant implications for blockchain and other distributed ledger technologies that rely on consensus mechanisms. Many modern consensus protocols, such as Ouroboros, Tendermint, and HotStuff, are designed with these limitations in mind. They operate under different assumptions to circumvent the impossibility result:

- **Partial Synchrony.** Protocols like Tendermint and HotStuff assume a partially synchronous network model (introduce by C. Dwork in [112]), where there are known bounds on message delays after an unknown global stabilization time.

- **Randomization and Probabilistic Guarantees.** Some consensus protocols, including those based on Proof-of-Stake (e.g., Ouroboros), incorporate randomization or probabilistic approaches to achieve consensus. By relaxing the deterministic requirement, these protocols can achieve consensus with high probability, even in the presence of failures or network asynchrony.

- **Fault Tolerance and Byzantine Assumptions.** Protocols like Byzantine Fault Tolerance (BFT) variants assume a certain number of Byzantine (arbitrarily malicious) faults and use cryptographic techniques and voting-based mechanisms to achieve consensus despite faulty behavior. These protocols often require a supermajority of honest nodes (e.g., 2/3 or more) to maintain security and liveness.

The FLP impossibility result highlights the inherent challenges in designing fault-tolerant consensus algorithms for distributed systems. By understanding these limitations, blockchain researchers and developers can better design protocols that provide the necessary trade-offs between safety, liveness, and fault tolerance. These considerations are essential for developing robust and secure blockchain systems that can operate effectively in real-world, adversarial environments.

**Fault Tolerance Bound.** The classical bound stating that a consensus protocol can tolerate at most $f$ Byzantine faults if and only if $n \geq 3f + 1$ was first established by Lamport, Shostak, and Pease in their seminal paper on the Byzantine Generals Problem [35]. This result relies on several strong assumptions. First, the system is modeled

as a fully connected network where every process can communicate directly with every other process through reliable but *unauthenticated* channels. In this setting, there are no cryptographic signatures, which means that faulty processes can *equivocate* by sending conflicting messages to different honest participants without being detected. Second, all non-faulty processes are deterministic and follow the protocol exactly, while Byzantine processes may behave arbitrarily, including colluding with each other. Finally, consensus must satisfy the two classical properties: *safety* (no two honest nodes decide on different values) and *liveness* (all honest nodes eventually reach a decision).

Under these assumptions, the system must contain strictly more than two-thirds honest processes to guarantee that honest messages form a supermajority. This ensures that when a node collects $2f+1$ consistent messages, at least $f+1$ of them come from honest processes, preventing faulty nodes from breaking safety. Formally, this requirement leads to the inequality $n - f \geq 2f + 1$, which simplifies to $n \geq 3f + 1$. If this condition is not met, Byzantine processes can exploit equivocation to partition the network and make honest nodes reach conflicting decisions.

Research in distributed computing has continually sought to improve the resilience of consensus protocols beyond the $\frac{n}{3}$ limit. While the theoretical resilience bound remains a significant constraint, various approaches have been explored to enhance the fault tolerance of consensus algorithms. For example, protocols that leverage cryptographic techniques, such as digital signatures or threshold cryptography, have been developed to provide stronger guarantees in environments with authentication.

Further advancements include hybrid consensus protocols that combine Byzantine Fault Tolerance (BFT) with other mechanisms, such as Proof-of-Stake (PoS) or Proof-of-Work (PoW), to improve scalability and security in blockchain applications. These innovations have enabled consensus protocols to tolerate a higher number of faulty nodes while maintaining performance and security standards.

The consensus problem, as defined by Lamport, Shostak, and Pease, remains a cornerstone of distributed computing and blockchain technology. The resilience bound of one-third faulty nodes highlights the inherent challenges in achieving agreement in decentralized networks, particularly in environments without authentication.
To address these limitations, we next turn to Trusted Execution Environments (TEEs), which provide hardware-based guarantees of integrity and non-equivocation, and can thus be leveraged to strengthen consensus protocols.

## 4.2 Trusted Components for Enhancing Byzantine Fault Tolerance

The use of trusted components represents a promising direction for improving the fault tolerance of blockchain systems. The notion of a *Trusted Execution Environment* (TEE) is broad and can be implemented either purely in software or with dedicated hardware support. While both approaches aim to create a secure and isolated space for executing sensitive operations, we argue that hardware-assisted TEEs generally provide stronger

protection against a wide range of attacks. We focus primarily on hardware-based TEEs, as they offer a more robust foundation for mitigating Byzantine faults, while acknowledging that such solutions are not immune to vulnerabilities; potential weaknesses are discussed later in Section 7.3. This robustness largely stems from the fact that hardware TEEs typically require *physical access* to the device in order to mount successful attacks. In a distributed setting, this implies that an adversary would need to gain physical access to multiple, potentially geographically dispersed nodes, which significantly raises the cost and complexity of a coordinated compromise and thereby strengthens the overall security guarantees [125, 126].

### 4.2.1 Trusted Hardware explained

Trusted hardware refers to specialized hardware components designed to provide secure, tamper-resistant environments for performing sensitive computations. The fundamental property of trusted hardware is its ability to maintain the integrity and confidentiality of the computations it executes, even in the presence of powerful adversaries with control over the operating system, hypervisor, or other system components. Typically, trusted hardware is implemented through hardware-based secure modules such as Trusted Platform Modules (TPM), Hardware Security Modules (HSM), or Trusted Execution Environments (TEE). examples of these include Intel SGX (Software Guard Extensions) and ARM TrustZone. (see subsection 4.2.3 for an evaluation of the different trusted hardware).

One critical capability of trusted hardware is the provision of remote attestation, allowing external entities to verify the authenticity and integrity of the computational environment remotely. Through attestation protocols, an external verifier can reliably determine whether computations are executed within an authentic secure module without being compromised.

#### Attested Append-only Memory (A2M)

Attested Append-only Memory (A2M), introduced by Chun *et al.* [127], is a specialized trusted hardware abstraction designed explicitly to prevent Byzantine nodes from equivocating, i.e., sending conflicting messages to different participants in a distributed system. A2M achieves this through the use of secure hardware-based logs that maintain an immutable, append-only record of messages.

In detail, A2M provides the following properties:

- **Append-only logs**: Each log maintained by an A2M instance can only be appended, not overwritten or modified. This property ensures immutability and prevents tampering.

- **Attestation**: Every log entry receives a cryptographic attestation, allowing external nodes to verify that the entry was produced within a valid A2M instance and

that it respects the append-only ordering.

- **Ordering guarantees**: The logs maintain an explicit ordering of messages, providing a reliable sequence that prevents conflicting messages.

**A2M interface.** We denote $q$ the append-only log of an A2M instance, the interface exposed by the A2M instance is defined as follows.

- **A2M-append**$(q, x)$: Appends a value $x$ to the append-only log $q$, extending it with a new entry at the end and return an attestation.
- **A2M-advance**$(q, n, x)$: Advances the log $q$ to position $n$ by appending value $x$ and return an attestation, under the condition that $n$ is greater than or equal to the current end of $q$; otherwise, the monotonicity property would be violated.
- **A2M-lookup**$(q, n)$: Returns the value stored at position $n$ in the log $q$, if such a position exists.
- **A2M-end**$(q, x)$: Returns the last position of the log $q$ if the last value equals $x$, enabling verification of the current state of the log.

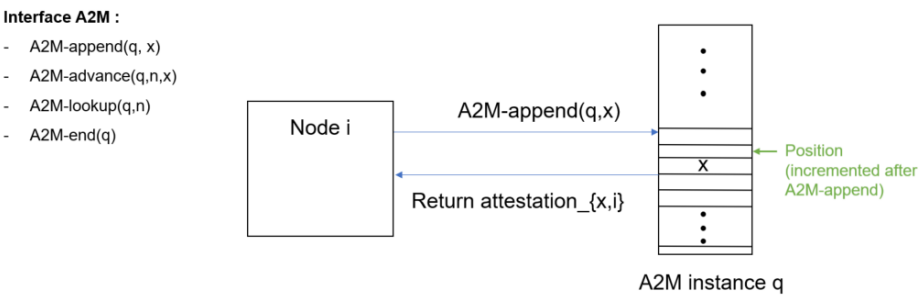



Figure 4.2: Execution of an A2M-append call

A2M's capabilities enable the creation of enhanced consensus protocols, such as PBFT-EA [127], which can tolerate up to half of the nodes being Byzantine. By binding messages to verifiable, ordered logs, A2M reduces the possibility of equivocation, significantly increasing the resilience of consensus protocols.

### Trusted Incrementer (TrInc)

Trusted Incrementer (TrInc), introduced by Levin *et al.* [128], provides a simpler and more focused trusted hardware abstraction compared to A2M. TrInc offers a secure monotonic counter within a tamper-proof hardware module, incrementing its value upon request and providing cryptographic attestations for each increment.

The critical properties of TrInc include:

- **Monotonicity**: The counter strictly increases, ensuring no rollback or repetition of previously attested values.

- **Unique attestations**: Each increment produces a unique cryptographic attestation linked to the specific counter value, preventing nodes from generating conflicting or ambiguous messages.

- **Minimal hardware requirements**: TrInc's simplicity allows it to be efficiently implemented within existing hardware modules with minimal complexity and cost.

TrInc has been demonstrated to serve as a foundational building block for constructing more advanced trusted components like A2M. Specifically, Levin *et al.* showed that an A2M module could be implemented using multiple TrInc instances, demonstrating the versatility and fundamental nature of monotonic counters in distributed system security [128].

#### Unique Sequential Identifier Generator (USIG)

The Unique Sequential Identifier Generator (USIG), introduced by Veronese *et al.* [129], extends the monotonic counter concept further. USIG provides trusted hardware-based functionality designed to issue cryptographically signed messages coupled with strictly increasing sequential identifiers. This feature ensures message authenticity and prevents equivocation by tying each message uniquely to its specific identifier.

The essential properties of USIG include:

- **Sequential identifiers**: Each message is assigned a unique sequential identifier, which strictly increases with each new message.

- **Cryptographic signing**: Messages are cryptographically signed, providing strong guarantees of authenticity and integrity.

- **Non-equivocation**: By enforcing the strict sequencing of identifiers, USIG prevents nodes from sending conflicting messages, significantly mitigating Byzantine behaviors.

Due to its powerful properties, USIG is particularly valuable in consensus protocols, providing robust protection against Byzantine faults and message equivocation.

### 4.2.2 Trusted Hardware in Blockchain Systems

## Increasing Consensus Fault Tolerance with Trusted Hardware

*Consensus*, as defined by Lamport, Shostak, and Pease [35], is a fundamental problem in fault-tolerant distributed computing. In the consensus problem, $n$ nodes attempt to agree on a value despite the potential malicious behavior of up to $t$ of them. The quality of a consensus protocol is often measured by its *resiliency*, or the fraction of faulty nodes it can tolerate. In traditional settings without authentication, Byzantine consensus protocols require a resilience bound of one-third, meaning they can tolerate up to $t = \lfloor \frac{n-1}{3} \rfloor$ Byzantine faults [35]. This bound was later proved to hold even in models with local authentication [130].

To improve consensus resilience beyond this one-third limit, researchers have explored the use of trusted hardware that provides a secure environment for computation. Trusted hardware components offer a set of trusted services that can enhance the security and fault tolerance of blockchain networks.

## Early Use of Trusted Hardware in Consensus Protocols

One of the early approaches in using trusted hardware for consensus was proposed by Correia *et al.* with the Trusted Timely Computing Base (TTCB) [131, 132]. The TTCB is a distributed component that includes local parts (local TTCBs) within nodes and a secure communication channel with bounded delays. By using this "wormhole," the authors demonstrated that Byzantine Fault Tolerance (BFT) could tolerate up to half of the nodes being Byzantine. However, the practical implementation of this approach is challenging due to the extensive requirements for trusted components and secure communication channels.

To address these implementation challenges, Chun *et al.* proposed Attested Append-Only Memory (A2M) [127], a more focused and practical trusted hardware solution. A2M prevents faulty nodes from equivocating (i.e. sending different messages to different nodes). An A2M-equipped node has a set of trusted, ordered append-only logs that provide attestation for each entry. By using A2M, the authors developed PBFT-EA, a variant of the Practical Byzantine Fault Tolerance (PBFT) protocol [40] that attaches attestations to each message exchanged. This design increases resilience to up to half of the nodes being Byzantine without requiring stronger network assumptions than the original PBFT protocol.

An alternative approach is to use a monotonic counter implemented in a tamper-proof module. Levin *et al.* introduced TrInc [128], a trusted monotonic counter designed to prevent equivocation in large distributed systems by providing once-in-a-lifetime attestations. They also proved that TrInc could be used to implement A2M. The monotonic trusted counter concept was further developed by Veronese *et al.* in [129], who proposed the Unique Sequential Identifier Generator (USIG), a local service available on each node that signs messages and assigns them unique, sequential counter values. This service must be implemented within a tamper-proof module to ensure security.

### Speculative Methods and Trusted Environments

Another line of research combines speculative execution methods with trusted environments. For example, CheapBFT [133] and ReBFT [134] use trusted hardware to enhance performance in fault-tolerant systems. Under normal conditions, these protocols require only $f+1$ nodes to guarantee agreement. However, in the presence of suspected Byzantine nodes, the protocol switches to a PBFT-like approach with trusted hardware, activating additional passive replicas to maintain security. While these methods are effective in reducing the number of active replicas needed under benign conditions, their integration into blockchain environments may lead to transient forks, which complicate the consistency model.
Such optimistic approaches may have potential applicability in highly regulated domains such as finance. In these settings, Byzantine behavior is more likely to stem from external adversaries rather than insider nodes, which could justify assuming more benign conditions in practice. This assumption might enable the system to achieve higher efficiency and reduced replication overhead during normal operation, while preserving the ability to switch to a more resilient mode under attack. Although this idea is not formally analyzed in this thesis, it may constitute an interesting avenue for further exploration.

### Recent Advances in Trusted Hardware for Blockchain Resilience

The application of trusted hardware to enhance blockchain resilience is relatively recent. One notable example is the enhancement of the HotStuff consensus protocol with trusted components to tolerate a minority of corruptions [135]. In this work, the authors assume the existence of underlying expander graphs (i.e a sparse graph that has strong connectivity properties) that provide probabilistic guarantees on consensus properties. Another recent development is Damysus [136], a BFT protocol that uses trusted environments to improve HotStuff's resilience. This protocol introduces two trusted services—Checker and Accumulator—that respectively increase resilience and reduce latency. However, these BFT protocols typically require strong assumptions or significant modifications to the original protocol to integrate trusted hardware effectively.

In contrast, our approach (presented in Chapters 5 and 6) involves minimal modifications to the original protocol. We demonstrate that by incorporating a specific authenticated broadcast service, such as TMC-Broadcast (Algorithm 4) or A2M-broadcast (Algorithm 10), into the Tendermint protocol, we can increase its resilience without extensive changes. This methodology can be applied to the repeated consensus abstraction with similar improvements in resilience, showing that trusted hardware can provide substantial benefits in enhancing the security and fault tolerance of blockchain systems. Notably, our approach addresses repeated consensus in trusted environments, a topic not thoroughly explored in previous research.

### 4.2.3 Temporal Overhead Introduced by Trusted Hardware

It is important to recall that the "costly" operation in this setting is the *generation* of attestations, which requires the involvement of trusted hardware. In contrast, the subsequent *verification* of these attestations can be performed outside the TEE, at negligible computational cost.

Trusted Hardware can enhance system security by isolating critical computations in protected hardware regions. However, this additional security comes at a cost: the introduction of temporal overhead due to enclave transitions, memory encryption, and restricted access to privileged system calls.

**Notation.** We introduce the notation $N_{TEE}^{Protocol}$ to count the number of TEE calls in a protocol execution, allowing us to analyze the overhead introduced by trusted components.

Formally, for a given *Protocol*:

$$N_{TEE}^{Protocol} = \sum_{i=1}^{k} c_i,$$

where $k$ is the number of distinct phases of the protocol involving the TEE and $c_i$ is the number of TEE calls performed in phase $i$.

This notation allows us to directly compare protocols in terms of their reliance on trusted hardware.

**Time evaluation.** Now that we have introduced a formal notation to count the number $N_{TEE}^{protocol}$ of TEE calls performed by a given protocol, we proceed to analyze the temporal cost associated with each of these calls. This requires examining the latency characteristics of the underlying trusted hardware, as the duration of a single TEE invocation depends heavily on the specific technology being used.

In the following, we present the typical transition and execution latencies for several widely deployed TEEs, including Intel SGX, ARM TrustZone, AMD SEV, and Keystone for RISC-V platforms.

- **Intel SGX.** Intel SGX incurs significant temporal overhead primarily due to the cost of transitions between the untrusted host and the secure enclave environment (ECALL and OCALL), as well as due to the encryption and decryption of memory pages via the Memory Encryption Engine (MEE). Costan and Devadas [126] report that ECALL/OCALL transitions require approximately **8,000 to 17,000 CPU cycles**, equivalent to **2 to 5 µs** on a 3.4 GHz processor. Chen *et al.* [137] further detail that memory encryption and cache misses in SGX can lead to up to **5x slowdown** in data-intensive workloads compared to unprotected execution. For cryptographic primitives such as AES-128 or RSA-2048, execution time within the enclave is typically in the range of **10 to 500 µs**, depending on the implementation

and data size. A recent study [138] have shown that the average enclave transition (entry or exit) takes approximately **3 to 18 microseconds** on modern Intel processors under typical conditions. In more demanding workloads with substantial memory or I/O contention, this latency can increase dramatically, sometimes exceeding one millisecond. This increased latency is primarily attributed to the additional micro-architectural security mechanisms, such as cryptographic memory protections, TLB and cache flushes, and the secure state store and restore process mandated by SGX during enclave entry and exit. These factors must be considered when evaluating SGX for latency-sensitive embedded or real-time systems.

- **ARM TrustZone.** The performance overhead associated with context switching in ARM TrustZone technology varies depending on the processor architecture. For the ARM Cortex-A8, which operates at 300 MHz, experimental results indicate that the latency for switching between the secure and non-secure worlds is approximately 1.3 microseconds, corresponding to about 400 processor cycles. In contrast, for ARM Cortex-M4 platforms, which feature three pipeline stages and typically lack built-in cache, the TrustZone world switch is notably more efficient. Measurements show a switching latency ranging from 100 to 400 nanoseconds, largely due to the optimized implementation that avoids a dedicated monitor mode and leverages lightweight function veneers. These results suggest that TrustZone context switching introduces minimal overhead in embedded applications where real-time response is required, thus making it a viable option for security-critical systems. [139]

- **AMD SEV.** AMD's Secure Encrypted Virtualization introduces latency primarily due to memory encryption for virtual machines. Measurements by Bahmani *et al.* [140] indicate a transition latency of **3 to 10 µs** for VM exits and memory access overheads of approximately **5 to 15%**, depending on memory intensity. SEV does not provide enclave-style fine-grained protection but applies encryption at the VM boundary.

- **Keystone (RISC-V).** Keystone is an open-source TEE framework for RISC-V platforms. According to Lee *et al.* [141], ECALL latency in Keystone is around **3 to 10 µs** for a typical microcontroller-scale configuration, with cryptographic operations (e.g., SHA256) ranging from **10 to 500 µs**. The overall overhead remains comparable to Intel SGX in the worst-case, but can be lower depending on the trust model and hardware design.

These implementations of trusted hardware are not immune to attacks; see Section 7.3.

## 4.3 Trusted Monotonic Counter Object

Trusted Execution Environments (TEEs) introduced in Section 4.2 (e.g. A2M [127], TrInc [128], USIG [129]) are reputed to be powerful tools for avoiding equivocation. Although the trusted component abstraction makes protocols immune to equivocation (where the initiator sends different messages to different processes), *Clement et al.* show

that non-equivocation is not enough to provide $n \geq 2f + 1$ resilience nor to support the equivalent of digital signatures [142].

We now define the Trusted Monotonic Counter Oracle abstraction TMC-Object the core of our novel Byzantine Reliable Broadcast protocol that supports $t$ Byzantine failures among $n$ processes, where $n \geq 2t + 1$, an improvement on the classical $n \geq 3t + 1$ algorithms.

In short, TMC-Object provides a non-falsifiable, verifiable, unique, monotonic, and sequential counter. In particular, TMC-Object provides each process with a read-only local variable, called trustedCounter. Whenever the TMC-Object is invoked, it returns a value for trustedCounter that is strictly greater than any previous value it returned. The difference between two successive counter values is exactly 1, so when a process $p$ receives two messages stamped with counter values, it can detect whether there have been intermediate messages.

The TMC-Object supports the operation **get_certificate()**. A process $p$ invokes **get_certificate(m)** with a message $m$. The object returns a *certificate* and a *unique identifier*. The certificate certifies that the returned identifier was created by the tamper-proof TMC-Object object for the message $m$. The unique identifier is essentially a reading of the monotonic counter trustedCounter, incremented whenever **get_certificate(m)** is called.

The TMC-Object object guarantees the following properties:

- **Uniqueness:** TMC-Object will never assign the same identifier to two different messages.
- **Sequentiality:** TMC-Object will always assign an identifier that is the successor of the previous one.

## 4.4 Conclusion

This Chapter established the system model and notational conventions and introduced the role of trusted hardware together with the *TMC-object*, which will serve as a building block in subsequent constructions. With the system model and building blocks established, we now present our first contribution: Byzantine Reliable Broadcast protocols that leverage the TMC object to achieve optimal resilience.

# Chapter 5

# Byzantine Reliable Broadcast with Trusted Monotonic Counters

*«Nescit vox missa reverti.»*
*A word once uttered can never return.*
HORACE, *Ars Poetica*

This chapter presents a study of Byzantine Reliable Broadcast using trusted components[1]. First, we show that naively replacing message sends in the Byzantine Reliable Broadcast protocol of Bracha [115] with message sends leveraging monotonic counters in trusted hardware does not improve resilience. Simply adding trusted components to Bracha's algorithm only prevents equivocating messages but does not address blocking or liveness issues caused by Byzantine processes. As a result, the fault tolerance bound remains at $n \geq 3t + 1$, and improving resilience requires modifying the protocol itself. Instead, we propose a novel algorithm that uses a single (optimal number) trusted component to implement asynchronous Byzantine reliable broadcast with $n$ processes, $n \geq 2t+1$ where $t$ is an upper bound on the number of Byzantine processes. We leverage the *Trusted Monotonic Counter Object* that was introduced in 4.3.

**Chapter organisation.** This chapter is organized as follows. Section 5.1 defines the execution model and presents the specification of the Byzantine Reliable Broadcast problem. Section 4.3 introduces the key component of our Byzantine Reliable Broadcast implementation, the Trusted Monotonic Counter Object. Section 5.2 discusses the impossibility of improving Bracha's Byzantine Reliable Broadcast resilience even when each process is equipped with a trusted component. Section 5.3 presents our algorithm for Byzantine Reliable Broadcast using a single Trusted Monotonic Counter Object at the initiator.

[1]This chapter comes from a conference paper co-authored with Yackolley Amoussou-Genou, Maria Potop-Butucaru, and Maurice Herily, published at the SSS24 conference [32] held in Nagoya, Japan.

## 5.1 System model and Problem Definition

Using the system model defined in Section 4.1, in this Chapter, we consider a set of $n$ asynchronous sequential processes. Up to $t$ processes can be *Byzantine*, meaning they can deviate from the given protocol. The others are *correct* processes.

Processes communicate by exchanging messages through an asynchronous network. We make the assumption that there is a public key infrastructure (PKI) where public keys are distributed, each process has his unique (universally known) public key and a matching private key, moreover, each message is signed by its creator. Messages are not lost or spuriously generated. Each process can send messages directly to any other process, and each process can identify the sender of every message it receives.

We assume processes have access to a broadcast primitive, broadcast($m$), which ensures that the message $m$ is received by every correct process in a finite (but unknown) time. When a process initiates a broadcast instance, we call that process the *initiator*.

## 5.2 Bracha's Byzantine Reliable Broadcast with Trusted Components

In this section, we investigate whether trusted hardware can improve the resilience of Byzantine Reliable Broadcast protocols. Byzantine Reliable Broadcast is a fundamental building block for consensus algorithms such as Tendermint, so any improvement in its resilience would directly translate to better fault tolerance in higher-level protocols.

A natural first question is: can we simply augment existing Byzantine Reliable Broadcast algorithms with trusted hardware components to achieve better resilience? To explore this, we examine Bracha's broadcast protocol [115] and consider what happens when nodes leverage trusted monotonic counters when sending messages. The intuition seems appealing: if nodes can cryptographically attest to the order in which they send messages using trusted hardware, perhaps this additional guarantee could help reduce the classical $3f + 1$ resilience requirement.

We analyze this approach both through concrete scenarios and formal proofs to understand whether such naive integration of trusted hardware can effectively improve fault tolerance. This investigation will inform how trusted hardware should be properly integrated into broadcast protocols. In this section, we prove that modifying the Byzantine Reliable Broadcast protocol introduce by Bracha [115] by (only) equipping each process with a trusted component (here, TMC-Object) does not change the tolerance threshold of Byzantine processes, which remains 1/3, we do so by using the modular formalism introduced by Raynal [143]. First, we present the Byzantine Reliable Broadcast protocol of Bracha, using the formalism introduced by Raynal, see Algorithm 1.

Then, we propose a new algorithm that replace all message by TMC attested message. To send a message $u$ certified with TMC-Object, a process $p$ first invokes TMC-Object, which creates a certificate $\mathcal{C}_p$ corresponding to the value of the trustedCounter $c_p$, then the

**Algorithm 1** Reliable Broadcast of Bracha

```
1: procedure RB-BROADCAST(u)
2:     Step 0
3:       if p is the initiator then
4:         Send ⟨initial, u⟩ to all
5:     Step 1
6:       Wait until receipt of
7:         1 message ⟨initial, v⟩ message, or
8:         α ⟨echo, v⟩ messages
9:         for some v
10:      Send ⟨echo, v⟩ to all
11:    Step 2
12:      Wait until receipt of
13:        α ⟨echo, v⟩ messages
14:          (including messages received in Step 1)
15:        for some v
16:      Send ⟨ready, v⟩ to all
17:    Step 3
18:      Wait until receipt of
19:        β ⟨ready, v⟩ messages
20:          (including messages received in Steps 1 and 2)
21:        for some v
22:      rb-Deliver v
```

process sends the tuple $(u, \mathcal{C}_p, c_p)$, which can be verified by any other process receiving the message. Each invocation to TMC-Object increments the value of the trustedCounter $c_p$ of process $p$, and the initial value of the counter is 0. In the following, that sequence of operation is simply called **TMC-Object-Send** $u$. A message sends through TMC-Object-Send is called a **TMC-Object-message**, a TMC-Object-message is considered valid if the value it contins is valid
When a process receives a message sent by TMC-Object-Send, it verifies that the signature is valid; this verification can be implemented outside the trusted enclave. (Algorithm 2)

**Algorithm 2** Secure messages management for node $i$

```
1: upon reception of ⟨TYPE, h, e, message, att_message, j⟩ from node j do
2:    if ∄c : (⟨TYPE, h, e, c⟩, j) ∈ messagesSet ∧verify(att_message,j, message, pubKey_j) then
3: ▷ We suppose the existence of a function verify that takes an attestation, a message and a pubKey and returns true if
   attestation is valid for message and is produced by j's TMC-Object
4: messagesSet ← messagesSet ∪ (⟨TYPE, h, e, message⟩, j)
```

In the following, we describe Algorithm 3 which is Bracha's reliable broadcast where each process uses the TMC-Object-Send operation instead of a Send operation. In more detail, the protocol works in sequential steps. In the broadcast primitive described in Algorithm 3, there are three types of messages used in the protocol: *initial*, *echo*, and *ready*. All these messages are sent using the TMC-Object-Send operation. In the initial step (Step 0) of the protocol, when a process $p$ wants to broadcast a value $u$, it TMC-Object-Sends an initial message for $u$ ($\langle initial, u\rangle$) to all other processes, therefore incrementing its trusted counter. Recall that the process initiating the broadcast is called the *initiator*.

In Step 1, upon receiving a valid[2] initial message with value $v$ from the initiator, a process TMC-Sends an echo message for $v$ ($\langle echo, v\rangle$). An echo message is also sent if instead of receiving the initial message, the process receives enough (here, $\alpha$) echo messages for the same value from different processes, implying that many processes saw the initiator message. After, and only after the TMC-Send operation, the process moves to Step 2.

[2]Here, valid TMC-Object-message. If the value of the counter is strictly greater than 2, the initiator may have equivocated (having already TMC-sent another message).

**Algorithm 3** Reliable Broadcast with a Trusted Environment

```
1: procedure RB-BROADCAST(u)
2:     Step 0
3:       if p is the initiator then
4:         TMC-Object-Send ⟨initial, u⟩ to all
5:     Step 1
6:       Wait until receipt of
7:         1 valid TMC-Object-message ⟨initial, v⟩ message, or
8:         α ⟨echo, v⟩ messages
9:         for some v
10:      TMC-Object-Send ⟨echo, v⟩ to all
11:    Step 2
12:      Wait until receipt of
13:        α ⟨echo, v⟩ messages
14:          (including messages received in Step 1)
15:        for some v
16:      TMC-Object-Send ⟨ready, v⟩ to all
17:    Step 3
18:      Wait until receipt of
19:        β ⟨ready, v⟩ messages
20:          (including messages received in Steps 1 and 2)
21:        for some v
22:      rb-Deliver v
```

In Step 2, each process waits to receive echo messages for the same value, say $v$, from at least $\alpha$ different processes sent from Step 1. When that is the case, the process TMC-Object-Sends a ready message for the value $v$ ($\langle ready, v \rangle$). After the send operation, the process moves to Step 3.

Step 3 is similar to Step 2. The process waits for $\beta$ ready messages, when the $\beta$ ready messages are received for the same value, say $v$, the process delivers value $v$ and finishes the instance of broadcast.

The broadcast is successful if all correct processes rb-Deliver the same value. Thus, rb-Broadcast and rb-Deliver provide us with a pair of communication primitives. (see Figure 5.1 that illustrates messages exchanged during Step 0, 1, 2 and 3).

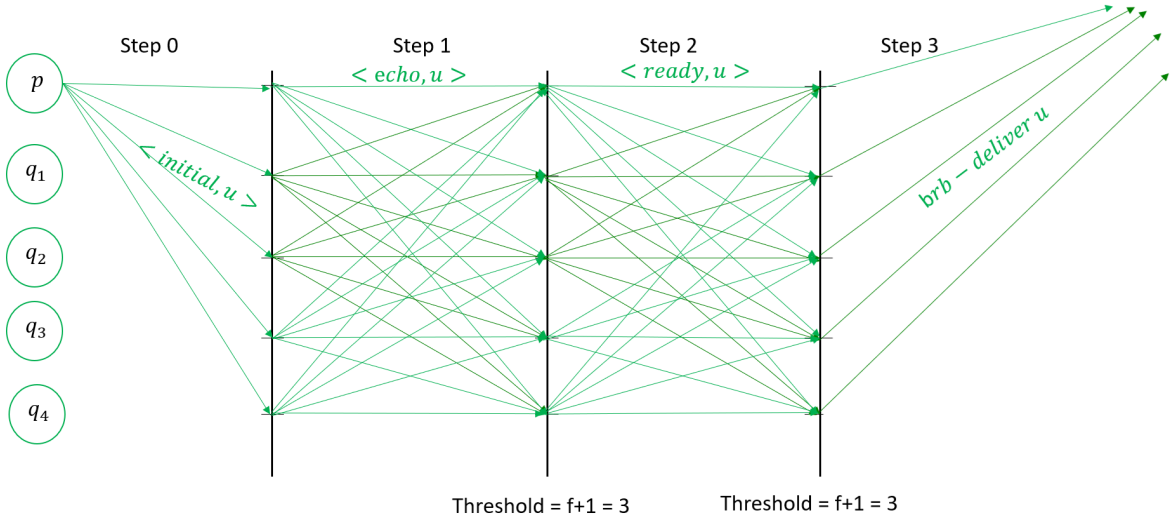



Figure 5.1: Messages exchange for a Byzantine Reliable instance with 5 nodes.

**Lemma 1.** *Consider Algorithm 3 with parameter $n \geq \alpha > t$ and $\alpha \geq n/2 + 1$ where $t$ is the number of Byzantine processes. If two correct processes TMC-Object-Send $\langle echo, v \rangle$ and $\langle echo, u \rangle$ messages, respectively, then $u = v$.*

*Proof.* The proof will be conducted by contradiction. Assume there exist two correct processes that TMC-Object-Send $\langle echo, v\rangle$ and $\langle echo, u\rangle$ messages, respectively, with $u \neq v$. Let $q$ be the first correct process that TMC-Object-Sends an $\langle echo, v\rangle$ message, and let $r$ be the first correct process that TMC-Object-Sends an $\langle echo, u\rangle$ message.

- Case 1: Process $q$ receives an initial message $\langle initial, v\rangle$ and process $r$ receives an initial message $\langle initial, u\rangle$. If the initiator is correct, then this situation is impossible since a correct initiator sends only one initial value. If the initiator is Byzantine, then either $q$ or $r$ rejects the initial value since the TMC-Object nominal sequence is invalid (the counter associated with one of these values is strictly greater than 1, hence the message is not valid).

- Case 2: Process $q$ must have received $\alpha$ $\langle echo, v\rangle$ messages, and process $r$ must have received $\alpha$ $\langle echo, u\rangle$ messages. Notice that a correct process can send only one echo. Since $\alpha > t$, where $t$ is the number of Byzantine processes in the system, among the $\alpha$ messages some come from correct processes. Since $\alpha \geq n/2 + 1$ then there is at least one correct process that TMC-Object-Sent $\langle echo, v\rangle$ and $\langle echo, u\rangle$ messages which is impossible since the process is correct.

□

**Lemma 2.** *Consider Algorithm 3 with parameter $n \geq \alpha > t$ and $\alpha \geq n/2 + 1$ where $t$ is the number of Byzantine processes. If two correct processes TMC-Object-Send $\langle ready, v\rangle$ and $\langle ready, u\rangle$ messages, respectively, then $u = v$.*

*Proof.* Proof by contradiction. Assume there exist two correct processes which TMC-Object-Send $\langle ready, v\rangle$ and $\langle ready, u\rangle$ messages, with $u \neq v$. Let $q$ be the first process that TMC-Object-Sends a $\langle ready, v\rangle$ message, and let $r$ be the first process that TMC-Object-Sends a $\langle ready, u\rangle$ message. Process $q$ must have received more than $\alpha$ $\langle echo, v\rangle$ messages and process $r$ must have received more than $\alpha$ $\langle echo, u\rangle$ messages. Since $\alpha > t$ and $\alpha \geq n/2 + 1$ it follows that at least one correct process must have TMC-Object-Sent $\langle echo, v\rangle$ and at least one correct process must have TMC-Object-Sent $\langle echo, u\rangle$ messages. Following Lemma 1, we then have $u = v$. □

**Lemma 3.** *Consider Algorithm 3 with parameter $n \geq \alpha > t$ and $\alpha \geq n/2 + 1$ where $t$ is the number of Byzantine processes. If two correct processes, $q$ and $r$, deliver the values $v$ and $u$, respectively, then $u = v$.*

*Proof.* If $q$ delivers the value $v$ then it must have received $\alpha$ $\langle ready, v\rangle$ messages, and therefore a $\langle ready, v\rangle$ message is from at least 1 correct process. Similarly, $r$ must have received a $\langle ready, u\rangle$ message from at least 1 correct process. By Lemma 2, $u = v$. □

We now show that Algorithm 3 satisfies the property brb-CorrectInit of the Byzantine reliable broadcast.

**Theorem 1.** *Consider Algorithm 3 with parameters $\alpha$ and $\beta$. If the initiator is a correct process Algorithm 3 satisfies the brb-CorrectInit property if $\alpha = \beta$ and $n/2 + 1 \leq \alpha \leq n$*

*and $n \geq 2t+1$ where $t$ is the number of Byzantine processes and $n$ is the total number of processes.*

*Proof.* The proof follows directly from Lemmas 1, 2 and 3. Let $p$ be the initiator. Since the initiator is correct, the value broadcast by $p$, $u$, will eventually be received by all other correct processes (at least $n-t = t+1$ correct processes). These processes will echo that value $u$ (TMC-Object-Send an echo message for $u$). Since all correct processes echo the same value (Lemma 1), and their number is sufficient to make the protocol advance (there are at least $n/2+1$ correct processes), each correct process will receive enough echoes to send a ready message, and the same one (Lemma 2). By the same argument and applying Lemma 3, all correct processes will receive enough ready messages for the initiator value $u$, and then will rb-Deliver the initiator message. □

Unfortunately, the following result shows that in the presence of a Byzantine initiator, Alorithm 3 could produce undesirable behaviour, and hence does not implement the Byzantine reliable broadcast.

Before formally proving that Algorithm 3 does not satisfy BRB-ByzantineInit when the number of nodes exceeds $2t+1$, we first exhibit a concrete counterexample. Consider a scenario with 5 nodes where two are Byzantine (shown in red in Figure 5.2): $p$ and $q_3$ are Byzantine, while $q_1$, $q_2$, and $q_4$ are correct.

The Byzantine nodes $p$ and $q_3$ selectively send their messages only to $q_1$ and $q_2$, systematically excluding $q_4$ at each step of the protocol. Consequently, $q_1$ and $q_2$ receive enough messages to reach the quorum threshold at the end of each step and progress to the next phase. However, $q_4$ only receives messages from $q_1$ and $q_2$, which is insufficient to meet the quorum requirement. As a result, $q_4$ never gathers enough messages to decide on value $u$, thereby violating the BRB-ByzantineInit property. Now that we have built the intuition through this counterexample, we formalize this result in the following lemma.

**Lemma 4.** *Let $n$ be the number of processes, and $t$ be an upper bound on the Byzantine processes with $n \geq 2t+1$. Consider Algorithm 3 with parameters $\alpha$ and $\beta$ with $\alpha = t+1$ and $t+1 \leq \beta < 2t+1$ Algorithm 3 does not satisfy the brb-ByzantineInit property when the initiator is Byzantine.*

*Proof.* If the initiator is a Byzantine process, Byzantine processes could force a subset of correct processes to deliver a value, and another subset of correct processes to never deliver any value. Note that even though all processes use a TMC-Object abstraction such that Byzantine processes cannot equivocate, Byzantine processes still can send a message to some processes but not to others.

Let $p$ be the Byzantine initiator. $p$ TMC-Object-sends a value $u$ to $1 \leq x \leq t$ correct processes $q_1, q_2 \dots q_x$ but not to the other $n-t-x$ processes. Denote by $Q$ this set of $x$ correct processes receiving the initiator's initial message. Since processes in $Q$ receive the message from the Byzantine initiator, they TMC-Object-Send an echo message for $u$. Assume now that all the Byzantine processes TMC-Object-Send an echo message for value $u$ only to processes in $Q$ but not to the others. It follows that all correct processes

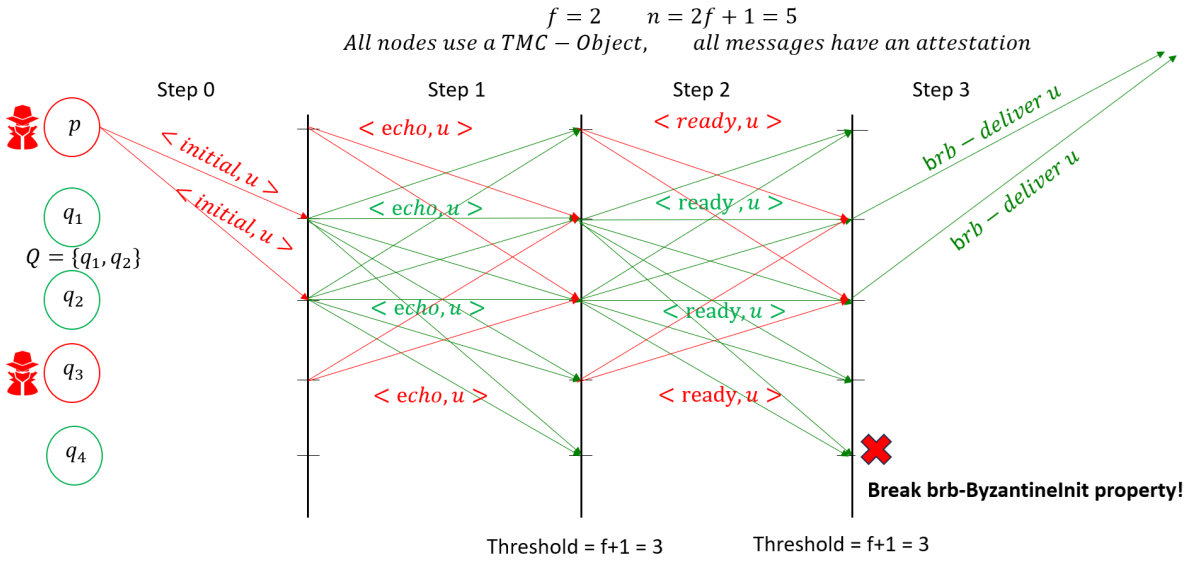



Figure 5.2: Counterexample with 5 nodes where Byzantine nodes $p$ and $q_3$ (in red) selectively send messages only to $q_1$ and $q_2$, preventing $q_4$ from reaching quorum.

but those in $Q$ have no message from the initiator and only the $\langle echo, u\rangle$ from processes in $Q$. Those processes cannot advance past Line 6 of Algorithm 3 since they need at least $\alpha = t + 1 > x$ echo messages.

All correct processes but those in $Q$ have only $x$ echo messages that come from the processes in $Q$. Processes in $Q$ on the other hand would have the echo messages from all Byzantine processes in addition to their own echo messages, which sums to $t + x$ echo messages for the value $u$. Therefore, processes in $Q$ will advance and TMC-Object-Send a ready message for $u$.

In the same spirit, Byzantine processes can TMC-Object-Send a ready message for value $u$ to processes in $Q$ only. The other correct processes are still blocked at Line 6 of Algorithm 3. In addition to the ready messages from the Byzantine processes, processes in $Q$ also get their own ready messages for value $u$, so each process $q \in Q$ has a total of $t + x$ ready messages and hence delivers the value $u$ (rb-Delivery). The other correct processes only receive the values TMC-Object-Sent by processes in $Q$, meaning $x$ echo message and $x$ ready message, both for $u$, hence, they can never reach an acceptance decision in Algorithm 3. It follows that Algorithm 3 does not satisfy the brb-Byzantinelnit property when the initiator is Byzantine. □

When $\alpha = \beta = t + 1$, Lemma 4 violates the brb-Byzantinelnit property of the Byzantine reliable broadcast (Definition 1), and, hence, does not satisfy the Byzantine reliable broadcast (Definition 1) as stated by the following Corollary.

**Corollary 1.** *Let $n$ be the number of processes, and $t$ an upper bound of the Byzantine processes. If $n \geq 2t+1$, Algorithm 3 does not implement the Byzantine reliable broadcast.*

However, for Algorithm 3 to implement the Byzantine reliable broadcast, we show in Theorem 2 that we must have $\beta = 2t + 1$.

**Theorem 2.** *Necessary conditions for 3 with parameters $\alpha$ and $\beta$ to implement the Byzantine reliable broadcast are $\alpha = t+1$, $\beta = 2t+1$, and $n-t \geq \beta$, where $t$ is the upper bound of the number of Byzantine processes and $n$ is the total number of processes.*

*Proof.* If the initiator is correct, all correct processes decide to deliver the initiator message, by Theorem 1.

It remains to show that when the initiator is Byzantine, either no correct process delivers any value or all correct processes deliver the same value.

- By Lemma 3, if two correct processes deliver a value, they must deliver the same one.

- Now, let us turn to the case where only one correct process reaches a decision.

  Assume that process $q$ reaches a decision and delivers a value $u$. It means that $q$ received at least $\beta$ ready messages, from which at least $\beta - t$ are from correct processes.

  At least $\beta - t$ correct processes sent a ready message. However, to TMC-Object-Send a ready message, a correct process must have reached Step 2, and must have completed Step 1 of Algorithm 3. In fact, if a correct process does not TMC-Object-Send an echo message, it cannot enter Step 2. Therefore, we know that at least $\beta - t$ correct processes have TMC-Object-Sent an echo message. By Lemma 1, all correct processes that TMC-Object-Sent an echo message have TMC-Object-Sent it for the same value, hence it means that they all TMC-Object-Sent an echo message for the value $u$.

  We would like to have that, with at least $\beta - t$ correct processes TMC-Object-Sending an echo message for the same value, say a value $u$, all correct processes must have received at least $\alpha$ echo messages for value $u$. Hence, we have that $\beta - t \geq \alpha \implies \beta \geq \alpha + t \geq 2t + 1$. For lower bounds, now assume that $\alpha = t + 1$ and $\beta = 2t + 1$. The rest of the proof shows that it is sufficient for Algorithm 3 to implement the Byzantine reliable broadcast.

  Hence they will all TMC-Object-Send an echo message for $u$. In that case, all correct processes (at least $2t + 1$ processes) TMC-Object-Sent an echo for $u$, all correct processes (at least $2t + 1$) will then TMC-Object-Send a ready message for $u$, which leads to all correct processes eventually delivering value $u$. Hence, if one correct process delivers a value $u$, all other correct processes eventually deliver the same value $u$.

□

We showed that a straightforward integration of trusted hardware into Bracha's protocol, where nodes invoke monotonic counters when sending messages, fails to improve the resilience bound. The protocol still requires 3f+1 nodes because Byzantine behavior in message processing remains unconstrained. This negative result establishes that trusted

hardware cannot be simply "plugged in" to existing Byzantine protocols; algorithmic innovation is necessary.

## 5.3 Byzantine Reliable Broadcast with optimal TMC-Object

**Algorithm 4** Byzantine Reliable Broadcast with a single Trusted Environment for the initiator

```
1: procedure TMC-Broadcast(u)
2:     Step 0:
3:     if p is the initiator then
4:         TMC-Object-Send ⟨initial, u, id_initiator⟩ to all
5:     Step 1:
6:     Wait until receipt of
7:       1 valid TMC-Object message ⟨initial, v, id_initiator⟩
8:       for some v
9:     Send ⟨echo_in, v, (⟨initial, v, id_initiator⟩, C_init, c_init)⟩ to all
10:              ▷ the process broadcasts back the initiator's message withe the associated certificate and trusted counter.
11:    Step 2:
12:    Wait until receipt of
13:      t+1 messages ⟨echo_in, v⟩
14:                                   ▷ Projection of the echoes received keeping only the value.
15:      (including messages received in Step 1)
16:      for some v
17:    Send ⟨ready, v⟩ to all
18:    Step 3:
19:    Wait until receipt of
20:      t+1 messages ⟨ready, v⟩
21:      (including messages received in Steps 1 and 2)
22:      for some v
23:    brb-Deliver v
```

In this section, we present Algorithm 4, which contains a small modification of Bracha's algorithm [115]. Algorithm 4 solves the reliable broadcast problem tolerating $t < n/2$ Byzantine processes. Algorithm 4 uses the trusted component setups to increase the security threshold of Byzantine reliable broadcast from 1/3 of Byzantine processes to 1/2.

Moreover, to reduce the use of the trusted component, which can be resource-intensive, only the initiator is required to send certified messages. The other processes simply check the validity of the message and its certification but do not require the equipment to send certified messages. In this section, we consider the use of TMC-Object defined in Section 4.3.

In Algorithm 4, only the initiator sends a message using the light TMC-Object abstraction for its send operation. We say that the initiator TMC-Object-Sends a message. The other processes send their message classically. The other difference with Algorithm 3 is that during Step 1, when a process receives the initiator's message, say for value $u$, it sends an echo message for $u$ coupled with the initiator message (meaning the message, and the certificate and counter sent by the initiator). In such a way, it ensures that all other processes will eventually receive the initiator's certified message. The rest of the algorithm proceeds as in Algorithm 3 where the TMC-Object-Sends are replaced by classical Send operations).

The broadcast is successful if all the correct processes brb-Deliver the same value, say

$u$. Thus, TMC-Broadcast and brb-Deliver provide a pair of communication primitives resilient to $t < n/2$ Byzantine processes.

We can now prove the correctness of Algorithm 4 against the Byzantine reliable broadcast abstraction.

Recall that we say that a process *accept*s a message if it receives and adds the "valid" messages in terms of the validity of the TMC-Object, meaning that there was no message before in that same category, hence the value of the trusted counter is the lowest. Notice that if the message received is not expected to be a TMC-Object message, and is indeed not part of a TMC-Object operation (e.g., a send which is not TMC-Object-Send), such a message is valid by default and, therefore, is accepted.

Finally, for any value $v$, the message $\langle echo_in, v, (\langle initial, v, id_initiator\rangle, \mathcal{C}_{\text{initiator}}, c_{\text{initiator}})\rangle$ should be understood as two messages bundled together, i.e., the echo message $\langle echo, v\rangle$ sent after the reception of the initiator message and sending back the initiator's message $\langle initial, v, id_initiator\rangle$, along with the certificates and trusted counter, i.e., $\mathcal{C}_{\text{initiator}}, c_{\text{initiator}}$. The message is exactly the initiator's message, the TMC-Object message will be correctly validated.

**Lemma 5.** *In any execution of Algorithm 4, with $n \geq 2t + 1$ where $t$ is the number of Byzantine processes, a correct process sends each type of message (initial, echo_in, ready) at most once.*

*Proof.* In Steps 1 and 2 of Algorithm 4, the protocol requires sending exactly 1 message, and then moving to the subsequent phase. A correct process cannot send more messages.

In Step 0, a correct process TMC-Object-Sends a message if and only if it is the initiator, and after that moves to Step 1. If the process is not the initiator, it does not (TMC-Object-)Send anything, but moves directly to Step 1. Hence, in Step 0, at most 1 message is sent.

□

**Lemma 6.** *Consider Algorithm 4 with $n \geq 2t+1$ where $t$ is the number of Byzantine processes. If two correct processes $p$ and $q$ receive and accept respectively $\langle initial, u, id_initiator\rangle$ and $\langle initial, v, id_initiator\rangle$, then $u = v$.*

*Proof.* This holds thanks to the properties of the TMC-Object. Since equivocation is not possible at the initiator level, thanks to the use of the counter in TMC-Object if two correct processes $p$ and $q$ accept an initiator message, it means they received the same message, and they then echo the accepted initial message (Line 9 of Algorithm 4). Therefore, they receive and accept the same message. □

By Lemma 6, we know that if two correct processes accept an initiator message, then they accept the same message. The only thing that could happen is for one correct process to receive the initiator message, while another process does not receive such a message.

**Lemma 7.** *Consider Algorithm 4 with $n \geq 2t + 1$ where $t$ is the number of Byzantine processes. If one correct process sends $\langle echo_in, v\rangle$ for some $v$, then all correct processes will eventually send $\langle echo_in, v\rangle$.*

*Proof.* Let $p$ and $q$ be processes. Without loss of generality, assume that $p$ is the first correct process to do an echo. If a correct process echoes a message, it means it accepts the initiator message, and will send the echo along with the TMC-Object-Send of the initiator (Line 9 of Algorithm 4). Two cases can arise. Either the initiator sent the initial message to both $p$ and $q$ or the initiator did not send a message to $q$. Notice that it is not possible for the initiator not to have TMC-Object-Sent a message to $p$, since $p$ echoed the initiator message.

First, consider the case where $p$ received the initiator message, but not $q$. The process $p$ sent an echo message containing the initiator's initial message respecting the TMC-Object format. By assumption, a message sent by a correct process will eventually be received by all the other correct processes. Therefore, eventually, $q$ will receive $p$'s echo message, containing the initial message. $q$ will be able to assess the validity of the initial message (according to the initiator signatures), will accept it, and will send an echo for the message too. Because before receiving $p$'s echo message, $q$ is still waiting in Step 1.

Finally, if the initiator sends a message to both $p$ and $q$, therefore, either it sends the same message to $p$ and $q$, and so $q$ will echo that same message (by Lemmas 5 and 6, it is not possible for $q$ to echo something else), or the message sent to $q$ is invalid and not accepted. That last case is equivalent to the above situation, since an invalid message is not registered nor considered and is equivalent to not having received a message. □

Thanks to Lemmas 5 and 7, we know that whenever a correct process sends an *echo_in* message, all other correct processes will also echo a message (and, more accurately, the same message).

**Lemma 8.** *If two correct processes $p$ and $q$ send respectively $\langle ready, v\rangle$ and $\langle ready, u\rangle$, then $u = v$.*

*Proof.* By contradiction. Assume $u \neq v$. Without loss of generality, let $p$ be the process that sends a $\langle ready, v\rangle$ message, and let $q$ be a process that sends a $\langle ready, u\rangle$ message. To send a ready message for value $x$, a correct process must have received from at least $t+1$ different processes the message $\langle echo_in, x\rangle$ (Line 17 of Algorithm 4). Therefore, $p$ must have received the message $\langle echo_in, v\rangle$ from at least $t+1$ different processes, and process $q$ must have received the message $\langle echo_in, u\rangle$ from at least $t + 1$ different processes. Since there are at most $t$ Byzantine processes, at least one correct process must have sent an echo message for both $u$ and $v$, which is impossible by Lemma 5. Therefore, it is impossible to have $u \neq v$. □

**Lemma 9.** *Consider Algorithm 4 with $n \geq 2t + 1$ where $t$ is the number of Byzantine processes. If two correct processes, $p$ and $q$, brb-deliver the values $v$ and $u$, respectively, then $u = v$.*

*Proof.* This proof is similar to the proof for 8. We proceed by contradiction. Assume two messages containing respectively values $u$ and value $v$ such that $u \neq v$. Without loss of generality, let $p$ be the process that brb-delivers the message containing $v$, and let $q$ be a process that brb-delivers $u$. To brb-deliver a value $x$, a correct process must have received from at least $t + 1$ different processes the message $\langle ready, x \rangle$ (Line 23 of Algorithm 4). Therefore, $p$ must have received the message $\langle ready, v \rangle$ from at least $t + 1$ different processes, and process $q$ must have received the message $\langle ready, u \rangle$ from at least $t + 1$ different processes. Since there are at most $t$ Byzantine processes, it means that at least one correct process sent ready messages for both values $u$ and $v$, which is impossible by Lemma 5. Therefore, it is impossible to have $u \neq v$. □

**Lemma 10.** *Consider Algorithm 4 with $n \geq 2t + 1$ where $t$ is the number of Byzantine processes. If a correct process $p$ delivers the value $v$ then every other correct process will eventually deliver $v$.*

*Proof.* If $p$ brb-Deliver $v$ then $p$ received the message $\langle ready, v \rangle$ from at least $t+1$ different processes. Since there are at most $t$ Byzantine processes, it means that at least one correct process sent a message $\langle ready, v \rangle$. Since one correct process sent a ready message for $v$, it means that it received an $\langle echo, v \rangle$ message from at least $t + 1$ different processes; hence, (since there are at most $t$ Byzantine processes) it means that at least one correct process sent a message $\langle echo, v \rangle$. Therefore, by Lemma 7, all other correct processes will (eventually) send an echo message for value $v$. Those will be received by all the correct processes. This will lead to having at least $t+1$ different processes sending it. All correct processes will, therefore, eventually send a ready message for value $v$ (by Lemma 8, since we already know that one correct process sent a ready for $v$). Hence, at least $t + 1$ ready messages will be received by all correct processes, which will lead them to brb-Deliver $v$. □

**Lemma 11.** *Consider Algorithm 4 with $n \geq 2t + 1$ where $t$ is the number of Byzantine processes. If a correct process $p$ broadcasts $v$ then all correct processes brb-deliver $v$.*

*Proof.* The proof of the lemma is straightforward. If a correct process broadcasts an initial message, it does so to all processes. All processes in Step 1 will echo_in the initiator message containing value $v$, thanks to Lemma 7. Since correct processes are the majority, and the network is eventually synchronous, they will all eventually receive at least $t + 1$ echo_in messages for value $v$ and send each a ready message for value $v$. Thanks to Lemma 8, since one correct process sends a ready for $v$, $v$ is the only value correct processes will send a ready for. That value will, therefore, be present in at least $t + 1$ ready messages, hence in Step 3, a correct process will brb-deliver $v$. By Lemma 10, all correct processes will eventually brb-deliver $v$. □

We can now prove that the algorithm implements the Byzantine reliable broadcast.

**Theorem 3.** *Let $n$ be the number of processes, and $t$ an upper bound of the Byzantine processes. If $n \geq 2t + 1$, Algorithm 4 implements Byzantine reliable broadcast.*

*Proof.* By Lemma 11, when a correct initiator broadcasts a value, all correct processes brb-deliver that value.

By Lemma 7, if a correct process brb-delivers an initiator message (even if the initiator is Byzantine), all correct processes will eventually brb-deliver the same initiator message. In that case, the rest of the proof follows thanks to Lemma 11.

Otherwise, no correct process brb-delivers any value. In more detail, if a Byzantine initiator does not send an initial message to any correct process, no correct process will deliver anything. That is because no correct process will send an echo message (then none will send ready messages). Since all advances require $t + 1$ messages from different processes, and Byzantine processes are at most $t$, the correct processes will be stuck in Step 1 and will make no decision.

□

**Theorem 4.** *Let $n$ be the number of processes, and $t$ be an upper bound of the Byzantine processes. If $n \geq 2t + 1$, in Algorithm 4, the number of TMC-Object used is optimal, in the sense that if we remove the only TMC-Object (initiator), Algorithm 4 does not implement the Byzantine reliable broadcast.*

*Proof.* In Algorithm 4, only 1 TMC-Object is used, the one at the initiator. If the TMC-Object is removed (instead of doing a TMC-Object-Send, the initiator does only a Send operation), then the algorithm resembles Bracha's Byzantine reliable broadcast protocol [115] where instead of a $2t + 1$ bound to advance, we have only a $t + 1$ bound. Since Bracha's Byzantine reliable broadcast protocol is optimal in the number of faults [115, 143], Algorithm 4 with no TMC-Object cannot implement Byzantine reliable broadcast.
□

## 5.4 Impact of Trusted Components on Byzantine Reliable Broadcast Complexity

To analyze the communication cost of distributed protocols, we count the number of point-to-point messages exchanged between processes.

**Notation** For a protocol executed by $n$ processes, we denote by $M(n)$ the *message complexity*: the total number of point-to-point messages sent during one successful instance of the protocol.

### 5.4.1 Message complexity

Both Algorithm 3 (Bracha-style BRB with TMC-Object-Send at every process) and Algorithm 4 (BRB with a single TMC at the initiator) follow the same three-message pattern *(initial, echo, ready)* for one broadcast instance.[3] Concretely, one initiator sends *initial* to all, then all processes broadcast *echo*, then all processes broadcast *ready*.

[3]See the pseudocode and step structure in Algorithm 4

**Per-instance message count.** Let $n$ be the number of processes. The total number of point-to-point messages for one broadcast instance is

$$M(n) \;=\; (n-1) \;+\; n(n-1) \;+\; n(n-1) \;=\; (n-1)(2n+1) \;=\; 2n^2 - n - 1 \;=\; \Theta(n^2).$$

This expression applies to both Algorithm 3 and Algorithm 4, since the message pattern is identical (the TMC only changes who certifies, not how many messages are sent).[4]

**Impact of resilience threshold on $n$.** Algorithm 3 (Bracha with naive TMC at all processes) does *not* improve the classical fault threshold and still requires $n \geq 3t+1$ to tolerate $t$ Byzantine faults.[5] In contrast, Algorithm 4 achieves Byzantine reliable broadcast with $n \geq 2t+1$.[6]

**Same tolerance $t = f$: quantitative savings.** For a fixed fault tolerance $t = f$, Algorithm 3 used $n_1 = 3f+1$ while Algorithm 4 used $n_2 = 2f+1$. Plugging these into $M(n)$:

$$M_1 \;=\; M(3f+1) \;=\; 18f^2 + 9f, \qquad M_2 \;=\; M(2f+1) \;=\; 8f^2 + 6f.$$

Hence the absolute saving is

$$\Delta M \;=\; M_1 - M_2 \;=\; 10f^2 + 3f,$$

and the relative reduction is

$$\frac{\Delta M}{M_1} \;=\; \frac{10f^2+3f}{18f^2+9f} \;=\; \frac{10f+3}{18f+9} \;\xrightarrow[f\to\infty]{}\; \frac{10}{18} \;=\; \frac{5}{9} \;\approx\; 55.6\%.$$

Thus, achieving the same Byzantine fault tolerance with $n = 2f+1$ instead of $n = 3f+1$ reduces the message volume by roughly **56%** asymptotically.

**Concrete savings for $f = 1$.** For a single Byzantine fault ($f = 1$), Algorithm 3 requires $n_1 = 3f+1 = 4$ processes and Algorithm 4 requires $n_2 = 2f+1 = 3$. Using $M(n) = (n-1)(2n+1)$:

$$M_1 = M(4) = 27, \qquad M_2 = M(3) = 14, \qquad \Delta M = 13.$$

Thus, Algorithm 4 sends 13 fewer messages per broadcast instance, which corresponds to a $\frac{13}{27} \approx 48.15\%$ reduction. Note that this finite-$f$ saving is close to, but below, the asymptotic reduction of $\approx 55.6\%$ obtained when $f$ grows.

[4] Algorithm 3 keeps Bracha's pattern with certificates on all messages; Algorithm 4 certifies only the initiator's *initial* and keeps classical sends for the rest.

[5] See Section 5.2: adding trusted components to all processes without protocol changes does not lift the 1/3 bound.

[6] Theorem 4 in Chapter 5 establishes correctness of Algorithm 4 for $n \geq 2t+1$.

**Takeaways.**

- **Asymptotic class.** Both algorithms remain $\Theta(n^2)$ per broadcast instance.
- **Constant-factor gain.** Moving from $3f+1$ to $2f+1$ processes for the same $t = f$ brings a $\approx 56\%$ reduction in messages.
- **TEE usage.** Algorithm 4 attains this gain *and* minimizes TEE usage to a single call at the initiator per instance (cf. temporal overhead subsection 5.4.2).

### 5.4.2 Temporal overhead: Access to TEE

As discussed in subsection 4.2.3, each call to a Trusted Hardware has a temporal overhead. The enhanced Byzantine Reliable Broadcast protocol *TMC-Broadcast* uses the TEE exclusively to initialize a single **Trusted Monotonic Counter (TMC)** at the very beginning of the broadcast instance. Once this secure counter is created, the rest of the protocol does not require any additional interaction with the TEE. Consequently, the temporal overhead induced by the TEE is essentially constant and negligible:

$$N_{TEE}^{TMC-Broadcast} = 1$$

per broadcast instance and independent of the number of processes or messages exchanged. This design highlights one of the advantages of our approach: the security benefits of trusted hardware are obtained without adding significant runtime cost to the broadcast itself.

## 5.5 Conclusion

In this chapter, we focus on improving Byzantine Reliable Broadcast fault tolerance using trusted components. Firstly, we show that adding a TMC-Object to all processes to prevent equivocation does not improve the security threshold or security guarantees of Bracha's Byzantine Reliable Broadcast, thanks to the formalism introduced in [143]. Secondly, we propose an optimal trusted component-based algorithm that implements Byzantine Reliable Broadcast in asynchronous settings with resilience $n \geq 2t + 1$. Our algorithm employs a very simple trusted component that provides a trusted monotonic counter. Moreover, our solution needs a single trusted monotonic counter at the initiator. This broadcast primitive constitutes a fundamental building block that will be leveraged in the next chapter to enhance the fault tolerance of the BFT protocol Tendermint.

# Chapter 6

# TenderTee Consensus

*« On a souvent besoin d'un plus petit que soi. »*
*We often need someone smaller than ourselves.*
Jean de La Fontaine, *Le Lion et le Rat*,
*Fables*, Livre II, fable XI (1668)

While Chapter 5 demonstrated how TMC enables optimal Byzantine Reliable Broadcast, we now extend this approach to the consensus problem, where processes must agree on a sequence of values rather than a single broadcast. In the context of applying blockchain technology to the financial sector, the choice of the Tendermint[1] [144] protocol for this thesis is both strategic and practical. Tendermint is a Byzantine Fault Tolerant (BFT) consensus algorithm designed to offer high throughput, fast finality, and robustness in a decentralized environment. Its design makes it particularly suitable for financial applications where transaction speed, reliability, and security are paramount. The primary reason for studying Tendermint in this thesis is its potential to provide a more efficient and environmentally friendly alternative to traditional Proof-of-Work-based blockchains like Bitcoin, which are known for their high energy consumption and scalability issues. In the following, we first present the Tendermint algorithm based on the formalization proposed in [104]. In the second part, we introduce our contribution TenderTee[2] that enhances Tendermint's Byzantine fault tolerance. TenderTee protocol, which we originally introduced in [33], and we subsequently proposed a new variant in [145] that exploits the *TMC-Object* we introduced in [32] (see section 4.3).

[1]Reminder: Tendermint has recently been renamed *Ignite*; however, in this manuscript, we continue to refer to it as Tendermint.

[2]This chapter is based on a conference paper co-authored with Maria Potop-Butucaru and José Alfaro, published at ICDCN 2023 in Kharagpur, India [33]. A preliminary version of this work appeared in the proceedings of ICDCN 2023; the extended version is published in *Theoretical Computer Science, special issue dedicated to the conference Stabilization, Safety, and Security of Distributed Systems, SSS 2024* [34].

## 6.1 Tendermint

As introduced in Chapter 2, Tendermint [144] is a BFT consensus protocol. We now present it using the formalism introduced by Amoussou-Guenou *et al.* [146].

### 6.1.1 Tendermint Consensus

Tendermint is a Byzantine Fault Tolerant (BFT) consensus protocol inspired by PBFT but adapted to the blockchain setting. Its design follows the rotating coordinator paradigm: for each block height, a proposer is deterministically selected among validators to suggest new block. The protocol proceeds in epochs, each of which is divided into three rounds: *PRE-PROPOSE*, *PROPOSE*, and *VOTE* [146].

During the PRE-PROPOSE round, the designated proposer broadcasts a candidate block. In the PROPOSE round, validators evaluate the proposal, checking both its validity and consistency with previously locked values, and then propagate it to other validators. If a validator receives at least $2f + 1$ matching proposals for a value $v$, it "locks" on $v$, ensuring that it will not support conflicting proposals in later epochs.
In the final VOTE round, validators broadcast their votes; if at least $2f + 1$ votes are gathered for the same value $v$, $v$ becomes the decided block.
The formalization of Synchronous TendermintBFT (Algorithm 5) and Partially Synchronous TendermintBFT (Algorithms 6 and 7) was introduced by Yackolley Amoussou-Guenou in [104] where he proves Theorem 5.

**Theorem 5.** *In partially synchronous system, TendermintBFT implements the consensus specification.*

**Algorithm 5** Synchronous TendermintBFT for height $h$ executed by $i$

1: **Initialization:**
2: $e_i \leftarrow 0$ ▷ Current epoch number
3: $decision_i \leftarrow nil$ ▷ Decision of process $i$
4: $lockedValue_i \leftarrow nil$; $validValue_i \leftarrow nil$
5: $proposal_i \leftarrow \mathsf{getValue}()$ ▷ Value that $i$ will (pre-)propose
6: $v_i \leftarrow nil$ ▷ Local variable holding a delivered pre-proposal
7: $vote_i \leftarrow nil$

8: **Round PRE-PROPOSE**$(e_i)$
9: **Send phase:**
10: **if** $decision_i \neq nil$ **then**
11: **for all** $v, j$ **such that** $(\langle \text{VOTE}, h, e_i, v\rangle, j) \in messagesSet_i$ **do**
12: $\mathsf{Broadcast}\big(\langle \text{VOTE}, h, e_i, v\rangle\big)$
13: **return**
14: **if** $\mathsf{proposer}(h, e_i) = i$ **then**
15: $\mathsf{Broadcast}\big(\langle \text{PRE-PROPOSE}, h, e_i, proposal_i\rangle\big)$
16: **Delivery phase:**
17: **while** $\neg\mathsf{expired}(timerPrePropose)$ **do**
18: **if** $\exists v :$ $\mathsf{sentByProposer}(h, e_i, v)$ **then**
19: $v_i \leftarrow v$ ▷ $v$ is the value sent by the proposer
20: **end while**
21: **Compute phase:**
22: **if** $\neg\mathsf{isValid}(v_i)$ **then**
23: $proposal_i \leftarrow nil$ ▷ $\mathsf{isValid}(nil) \equiv \texttt{false}$
24: **else**
25: **if** $lockedValue_i = nil \ \vee \ v_i \in \{lockedValue_i, validValue_i\}$ **then**
26: $proposal_i \leftarrow v_i$
27: **else**
28: $proposal_i \leftarrow nil$

29: **Round PROPOSE**$(e_i)$
30: **Send phase:**
31: **if** $proposal_i \neq nil$ **then**
32: $\mathsf{Broadcast}\big(\langle \text{PROPOSE } (\textsc{Prevote}), h, e_i, proposal_i\rangle\big)$
33: **Delivery phase:**
34: **while** $\neg\mathsf{expired}(timerPropose)$ **do**
35: ▷ Collect messages
36: **end while**
37: **Compute phase:**
38: **if** $\exists v :$ $\mathsf{2f{+}1}\big(\langle \text{PROPOSE } (\textsc{Prevote}), h, e_i, v\rangle\big) \ \wedge \ \mathsf{isValid}(v) \ \wedge \ \mathsf{sentByProposer}(h, e_i, v)$ **then**
39: $lockedValue_i \leftarrow v$
40: $validValue_i \leftarrow v$
41: $vote_i \leftarrow v$
42: **else**
43: $vote_i \leftarrow nil$

44: **Round VOTE**$(e_i)$
45: **Send phase:**
46: **if** $vote_i \neq nil$ **then**
47: $\mathsf{Broadcast}\big(\langle \text{VOTE } (\textsc{Precommit}), h, e_i, vote_i\rangle\big)$
48: **Delivery phase:**
49: **while** $\neg\mathsf{expired}(timerVote)$ **do**
50: ▷ Collect messages
51: **end while**
52: **Compute phase:**
53: **if** $\exists v' :$ $\mathsf{2f{+}1}\big(\langle \text{PROPOSE } (\textsc{Prevote}), h, e_i, v'\rangle\big) \ \wedge \ \mathsf{isValid}(v') \ \wedge \ \mathsf{sentByProposer}(h, e_i, v')$ **then**
54: $validValue_i \leftarrow v'$
55: **if** $\exists v_d, e_d :$ $\mathsf{2f{+}1}\big(\langle \text{VOTE } (\textsc{Precommit}), h, e_d, v_d\rangle\big) \ \wedge \ \mathsf{isValid}(v_d) \ \wedge \ decision_i = nil$ **then**
56: $decision_i \leftarrow v_d$
57: **else**
58: $e_i \leftarrow e_i + 1$
59: $v_i \leftarrow nil$
60: **if** $validValue_i \neq nil$ **then**
61: $proposal_i \leftarrow validValue_i$
62: **else**
63: $proposal_i \leftarrow \mathsf{getValue}()$

**Algorithm 6** (part 1) Partially Synchronous TendermintBFT for height $h$ executed by $i$

1: **Initialization:**
2: $e_i \leftarrow 0$ ▷ Current epoch number
3: $decision_i \leftarrow nil$ ▷ Decision of process $i$
4: $lockedValue_i \leftarrow nil$; $validValue_i \leftarrow nil$
5: $lockedEpoch_i \leftarrow -1$; $validEpoch_i \leftarrow -1$
6: $proposal_i \leftarrow$ getValue() ▷ Value that $i$ will (pre-)propose
7: $v_i \leftarrow nil$ ▷ Local variable holding delivered pre-proposal
8: $validEpoch_j \leftarrow nil$ ▷ Proposer's validEpoch (local)
9: $vote_i \leftarrow nil$ ▷ Value process $i$ will vote for
10: $timeoutPropose \leftarrow \Delta_{\text{Pre-propose}}$; $timeoutPrevote \leftarrow \Delta_{\text{Propose}}$; $timeoutPrecommit \leftarrow \Delta_{\text{Vote}}$

11: **Round PRE-PROPOSE**
12: **Send phase:**
13: **if** $decision_i \neq nil$ **then**
14: **for all** $v, j$ **such that** $(\langle \text{PRECOMMIT}, h, e_i, v\rangle, j) \in messagesSet_i$
15: Broadcast$\big(\langle \text{PRECOMMIT}, h, e_i, v\rangle\big)$
16: **return**
17: **if** proposer$(h, e_i) = i$ **then**
18: Broadcast$\big(\langle \text{PRE-PROPOSE}, h, e_i, proposal_i, validEpoch_i\rangle\big)$
19: **Delivery phase:**
20: $timerPrePropose \leftarrow timeoutPropose$
21: **while** $\neg$expired$(timerPrePropose) \ \wedge \ \neg\big(\exists v_j, e_j : \text{sentByProposer}(h, e_i, v_j, e_j)\big)$ **do**
22: **if** $\exists v_j, e_j :$ sentByProposer$(h, e_i, v_j, e_j)$ **then**
23: $v_i \leftarrow v_j$ ▷ $v_j$ sent by proposer
24: $validEpoch_j \leftarrow e_j$ ▷ $e_j$ is the proposer's validEpoch
25: **if** $\neg\big(\exists v, epochProp :$ sentByProposer$(h, e_i, v, epochProp)\big)$ **then**
26: $timeoutPropose \leftarrow timeoutPropose + 1$
27: **Compute phase:**
28: **if** 2f+1$\big(\langle \text{PREVOTE}, h, validEpoch_j, v_i\rangle\big) \ \wedge \ validEpoch_j \geq lockedEpoch_i \ \wedge \ validEpoch_j < e_i \ \wedge$ isValid$(v_i)$ **then**
29: $proposal_i \leftarrow v_i$
30: **else**
31: **if** $\neg$isValid$(v_i) \ \vee \ (lockedEpoch_i > validEpoch_j \ \wedge \ lockedValue_i \neq v_i)$ **then**
32: $proposal_i \leftarrow nil$ ▷ isValid$(nil) \equiv$ `false`
33: **if** isValid$(v_i) \ \wedge \ (lockedEpoch_i = -1 \ \vee \ lockedValue_i = v_i)$ **then**
34: $proposal_i \leftarrow v_i$

**Algorithm 7** (part 2) Partially Synchronous TendermintBFT for height $h$ executed by $i$

1: **Round PROPOSE**
2: **Send phase:**
3: **if** $proposal_i \neq nil$ **then**
4: Broadcast$(\langle \textsc{Prevote}, h, e_i, proposal_i \rangle)$
5: Broadcast$(\langle \mathsf{HeartBeat}, \textsc{Prevote}, h, e_i \rangle)$
6: **Delivery phase:**
7: $timerPropose \leftarrow timeoutPrevote$
8: **while** $\neg$expired$(timerPropose) \ \wedge \ \neg$ 2f+1$(\langle \mathsf{HeartBeat}, \textsc{Prevote}, h, e_i \rangle)$ **do**
9: ▷ HeartBeat messages must come from distinct committee members
10: **if** $\neg$ 2f+1$(\langle \mathsf{HeartBeat}, \textsc{Prevote}, h, e_i \rangle)$ **then**
11: $timeoutPrevote \leftarrow timeoutPrevote + 1$
12: **Compute phase:**
13: **if** $\exists v' :$ 2f+1$(\langle \textsc{Prevote}, h, e_i, v' \rangle) \ \wedge \ \mathsf{isValid}(v') \ \wedge \ \mathsf{sentByProposer}(h, e_i, v')$ **then**
14: $lockedValue_i \leftarrow v'$
15: $lockedEpoch_i \leftarrow e_i$
16: $validValue_i \leftarrow v'$
17: $validEpoch_i \leftarrow e_i$
18: $vote_i \leftarrow v'$
19: **else**
20: $vote_i \leftarrow nil$

21: **Round VOTE**
22: **Send phase:**
23: **if** $vote_i \neq nil$ **then**
24: Broadcast$(\langle \textsc{Precommit}, h, e_i, vote_i \rangle)$
25: Broadcast$(\langle \mathsf{HeartBeat}, \textsc{Precommit}, h, e_i \rangle)$
26: **Delivery phase:**
27: $timerVote \leftarrow timeoutPrecommit$
28: **while** $\neg$expired$(timerVote) \ \wedge \ \neg$ 2f+1$(\langle \mathsf{HeartBeat}, \textsc{Precommit}, h, e_i \rangle)$ **do**
29: ▷ Collect messages
30: **if** $\neg$ 2f+1$(\langle \mathsf{HeartBeat}, \textsc{Precommit}, h, e_i \rangle)$ **then**
31: $timeoutPrecommit \leftarrow timeoutPrecommit + 1$
32: **Compute phase:**
33: **if** $\exists v'' :$ 2f+1$(\langle \textsc{Prevote}, h, e_i, v'' \rangle) \ \wedge \ \mathsf{isValid}(v'') \ \wedge \ \mathsf{sentByProposer}(h, e_i, v'')$ **then**
34: $validValue_i \leftarrow v''$
35: $validEpoch_i \leftarrow e_i$
36: **if** $\exists v_d, e_d :$ 2f+1$(\langle \textsc{Precommit}, h, e_d, v_d \rangle) \ \wedge \ \mathsf{isValid}(v_d) \ \wedge \ decision_i = nil$ **then**
37: $decision_i \leftarrow v_d$
38: **else**
39: $e_i \leftarrow e_i + 1$
40: $v_i \leftarrow nil$
41: **if** $validValue_i \neq nil$ **then**
42: $proposal_i \leftarrow validValue_i$
43: **else**
44: $proposal_i \leftarrow$ getValue()

### 6.1.2 Tendermint Repeated Consensus

In this section, we present Tendermint's implementation of repeated consensus (Definition 3). As previously discussed, a blockchain can be conceptualized as a repeated consensus problem, where each height $h$ corresponds to a consensus instance producing a block. Tendermint's repeated consensus algorithm coordinates these successive instances while maintaining blockchain consistency across the distributed system.

Tendermint's repeated consensus builds upon the consensus algorithm presented in Section 6.1, but extends it to handle an infinite sequence of decisions. At each height, a new committee of validators is deterministically selected, and these validators execute a consensus instance to decide on the block to be added to the blockchain. Participants who are not part of the committee for a given height wait to receive sufficient evidence (*commits*) before accepting the decided block.

Amoussou-Guenou shows in [104], Theorem 6:

**Theorem 6.** *In an eventually synchronous system, and assuming that the consensus function is correct, the Tendermint Repeated Consensus algorithm implements the Repeated Consensus.*

**Data structures.** The integer $h$ represents the current height of the participant. $C$ is the current set of committee members. $B$ is the variable that will be set to the block to be appended. $commitsReceived_i^h$ is the set containing all the commits $i$ delivered for the height $h$. $toReward_i^h$ is the set containing the committee members from which $i$ delivered commits for the height $h$. *TimeOutCommit* represents the time a committee member has for collecting commits after an instance of consensus. *TimeOutCommit* is set to the default value $\Delta_{\text{Commit}}$.

**Functions.** Let $Height = \mathbb{N}^*$ be the set of all heights, let *Commits* be a set of all possible commits, and let $\mathbb{B}$ be the set containing all possible blocks. We also recall that $\Pi_\rho$ is the set of participants in the system run.

- $\mathsf{committeeMembers} : \Pi \times Height \to 2^{\Pi_\rho}$ is an application dependent and deterministic selection function which gives the set of committee members for a given height with respect to the blockchain history. We have $\forall h \in Height, |\mathsf{committeeMembers}(h)| = n$.

- $\mathsf{consensus} : Height \times 2^{\Pi_\rho} \times 2^{Commits} \to \mathbb{B}$ is a consensus algorithm. It outputs for the participant the decision of the consensus (Definition 3) among the committee members.

- $\mathsf{MoreThanThird} : \mathbb{B} \times 2^{Commits} \to \{\texttt{false}, \texttt{true}\}$ is a predicate which checks if there are commits from at least $f + 1$ different committee members for a given block in the given set.

- $\texttt{isValid} : \mathbb{B} \to \{\texttt{false}, \texttt{true}\}$ is an application-dependent predicate that is satisfied if the given block is valid. If there is a block $B$ such that $\texttt{isValid}(B) = \texttt{true}$, we say that $B$ is valid. We note that for any non-block, we set $\texttt{isValid}$ to false, (i.e., $\texttt{isValid}(\perp) = \texttt{false}$), the validity of the block depends on the blockchain and the application, and $\texttt{isValid}$ is known by all participants.

Note that in Tendermint, the reward for the height $h$ is allocated during the height $h+1$, and to a subset of committee members who committed the block for $h$ (Line 18 of Figure 6.1).

**Function** repeatedConsensus($\Pi$); ▷ Repeated consensus for the set $\Pi$ of participants

**Init**:
(1) $h \leftarrow 1$; $B \leftarrow \perp$; $C \leftarrow \perp$; ▷ *Height* and Set of committee members
(2) $commitsReceived_i^h \leftarrow \emptyset$; $toReward_i^h \leftarrow \emptyset$; $TimeOutCommit \leftarrow \Delta_{\text{Commit}}$;

---

**while** (true) **do**
(3) $B \leftarrow \perp$;
(4) $C \leftarrow$ committeeMembers($h$); ▷ Application and blockchain dependant
(5) **if** $(i \in V)$ **then**
(6) $B \leftarrow$ consensus($h, V, toReward_i^{h-1}$); ▷ Consensus function for the height $h$
(7) **trigger** broadcast $\langle \texttt{COMMIT}, (B, h)_i \rangle$;
(8) **else**
(9) **wait until** $(\exists B' : |\text{MoreThanThird}(B', commitsReceived_i^h)|)$;
(10) $B \leftarrow B'$;
(11) **endif**
(12) set $timerCommit$ to $TimeOutCommit$;
(13) **wait until**($timerCommit$ expired);
(14) **trigger** decide($B$);
(15) $h \leftarrow h + 1$;
**endwhile**

---

**upon event** delivery $\langle \texttt{COMMIT}, (B', h')_j \rangle$:
(16) **if** $(((B', h')_j \notin commitsReceived_i^{h'}) \wedge (j \in \text{committeeMembers}(h')))$ **then**
(17) $commitsReceived_i^{h'} \leftarrow commitsReceived_i^{h'} \cup (B', h')_j$;
(18) $toReward_i^{h'} \leftarrow toReward_i^{h'} \cup j$;
(19) **trigger** broadcast $\langle \texttt{COMMIT}, (B', h')_j \rangle$;
(20) **endif**

Figure 6.1: Tendermint Repeated Consensus algorithm for Obedient Participant $i$.

**Detailed description of the algorithm.** We describe in Figure 6.1 the Tendermint algorithm for the repeated consensus (Definition 3). For a participant $i$, the algorithm proceeds as follows:

- $i$ computes the set of committee members for the current height;
- If $i$ is a committee member, then it calls the consensus function solving the consensus for the current height, then broadcasts the decision, and sets $B$ to that decision;
- Otherwise, if $i$ is not a committee member, it waits for at least $n/3$, or equivalently $f + 1$, commits from the same block and sets $B$ to that block;

- In any case, it sets the timer to *TimeOutCommit* to collect more commits and lets it expire. Then $i$ decides $B$ and goes to the next height.

Whenever $i$ delivers a commit, it broadcasts it (Lines 16 - 20 of Figure 6.1).

### 6.1.3 Tendermint limitations

Tendermint's consensus mechanism operates through a two-thirds majority voting system, which allows it to achieve consensus with significantly lower energy consumption compared to PoW systems. This characteristic aligns well with the financial sector's requirements, where reducing operational costs, including energy expenses, is crucial. Moreover, Tendermint's ability to provide near-instantaneous transaction finality is advantageous for financial applications that demand fast and secure transaction processing.

However, while Tendermint addresses some of the core challenges in blockchain applications for finance, such as scalability and energy efficiency, it has its limitations. The use of Trusted Hardware could provide additional layers of security, improve fault tolerance, and further reduce energy consumption by optimizing the consensus process. This integration could offer a more resilient and energy-efficient solution for blockchain applications in the financial sector, meeting both the operational and sustainability goals of modern financial institutions.

One key challenge is scalability: as the number of validators grows, the communication overhead and the complexity of coordinating decisions increase significantly. This limits Tendermint's ability to support very large, decentralized networks, which is particularly relevant in financial-grade systems where both security and performance must scale together. It can also be limiting in smaller, three-party networks (where Tendermint cannot tolerate even a single Byzantine process), as will be discussed in more detail later (see tri-party scenarios in Section 6.4).

In the following, we show how the introduction of trusted hardware in Tendermint can increase the tolerance of the system to Byzantine faults beyond traditional limits while simplifying the coordination process. This improvement not only strengthens the protocol's resilience but also reduces the overall message complexity, allowing validators to reach agreement more efficiently.

## 6.2 TenderTee

While Tendermint addresses many challenges in blockchain consensus, one fundamental barrier remains: the $n = 3f+1$ requirement means that tolerating even a single Byzantine fault demands at least four validators. This constraint severely limits practical deployments in financial settings where tri-party arrangements (three independent institutions sharing validation duties) are ubiquitous. With Tendermint, such arrangements offer no fault tolerance whatsoever.

**Can we improve this fault tolerance?** This question motivates the core contribution of this thesis.

### 6.2.1 TenderTee Consensus

We present a novel consensus protocol that breaks through the $n = 3f + 1$ barrier by carefully integrating Trusted Execution Environments. TenderTee achieves Byzantine fault tolerance with $n = 2f + 1$ validators, the optimal threshold for crash fault tolerance, while maintaining full BFT safety and liveness properties.

We now detail the TenderTee protocol, starting with the one-shot consensus algorithm and building up to the full repeated consensus mechanism.

Algorithms 8 and 9 proceed in 3 rounds for a given epoch $e$ at height $h$. Each protocol message (pre-propose, propose or vote) is sending through TMC-Broadcast (Algorithm 5.3) which broadcasts the message value to the other processes in a certified manner and increases the corresponding counter. We use the notation TMC-Broadcast($TYPE$, content) to mean that the certified broadcast is done on the counter for the messages of type $TYPE$.

The epoch $e_i$ is initially set to 0.

*Counters*: Each process has three certified counters, one for each round type. Counters are initially set to 0.

- Round PRE-PROPOSE: Firstly, nodes verify if a decision has been taken in a previous round, if it is the case they only broadcast the Votes that helped them reach a decision. If the validator $p_i$ is the proposer of the epoch, it pre-proposes its proposal value, otherwise, it waits for the proposal from the proposer. If a validator $p_j$ delivers the pre-proposal from the proposer of the epoch, $p_j$ checks the validity of the pre-proposal and that certified attestation is valid and if both conditions are verified, it accepts it with respect to the values in $validValue_i$, $lockedValue_i$, $validEpoch_j$ and $lockedEpoch_i$. If the pre-proposal is accepted and valid, $p_j$ sets its proposal $proposal_j$ to the pre-proposal, otherwise it sets it to $nil$.

- Round PROPOSE: During the PROPOSE round, each validator broadcasts (through TMC-Broadcast primitive) its proposal, and collects the proposals sent by other validators (only the one that have a valid attestation). After the Delivery phase, validator $p_i$ has a set of proposals, and checks if $v$, pre-proposed by the proposer, was proposed by at least $t+1$ different validators. If it is the case, and the value is valid, then $p_i$ sets $vote_i$, $validValue_i$ and $lockedValue_i$ to $v$ and updates $lockedEpoch_i$ to the current epoch $e_i$, otherwise it sets $vote_i$ to $nil$.

- Round VOTE: In the round VOTE, a correct validator $p_i$ votes $vote_i$ and broadcasts (using TMC-Broadcast primitive) all the proposals it delivered during the current epoch. Then $p_i$ collects all the messages that were broadcast. First $p_i$ checks if it

has delivered at least $t+1$ proposals for a value $v'$ pre-proposed by the proposer of the epoch, in that case, it sets $validValue_i$ to that value then it checks if a value $v'$ pre-proposed by the proposer of the current epoch is valid and has at least $t+1$ votes, if it is the case, then $p_i$ decides $v'$ and goes to the next height; otherwise it increases the epoch number and updates the value of $proposal_i$, with respect to $validValue_i$.

**Algorithm 8** Partial Synchronous TenderTee for height $h$ executed by $i$: Round Pre-Propose

1: ▷ There are 3 counters for process $i$, one for each round: Prepopose, Propose, and Vote
2: **Initialization:**
3: $e_i \leftarrow 0$ ▷ Current epoch number
4: $decision_i \leftarrow nil$ ▷ Store the decision of the process $i$
5: $lockedValue_i \leftarrow nil$; $validValue_i \leftarrow nil$
6: $lockedEpoch_i \leftarrow -1$; $validEpoch_i \leftarrow -1$
7: $proposal_i \leftarrow$ getValue() ▷ Store the value the process will (pre-)propose
8: $v_i \leftarrow nil$ ▷ Local variable stocking the pre-proposal if delivered)
9: $validEpoch_j \leftarrow nil$ ▷ Local variable stocking the propose's validEpoch
10: $vote_i \leftarrow nil$ ▷ Store the value the process will vote for
11: $timeoutPropose \leftarrow \Delta_{\text{Pre-propose}}$; $timeoutPropose \leftarrow \Delta_{\text{Propose}}$; $timeoutVote \leftarrow \Delta_{\text{Vote}}$
12: **Round** PRE-PROPOSE:
13: **Send phase:**
14: **if** $decision_i \neq nil$ **then**
15: $\forall v, j : (\langle \text{VOTE}, e_i, v \rangle, j) \in messagesDelivered_i$, Broadcast$(\langle \text{VOTE}, e_i, v, \mathcal{C}_j, c_j \rangle, j)$
16: ▷ Send back all the VOTE messages (as well as the corresponding certification (attestation) $_j$ and counter $c_j$) the process received that made it decides
17: **return**
18: **if** proposer$(e_i) = i$ **then**
19: **TMC-Broadcast**$(< \textbf{PRE-PROPOSE}, e_i, \textbf{\textit{proposal}}_i, \textbf{\textit{validEpoch}}_i, \textbf{voteMessagesDelivered}_i(e_i) >)$
20: ▷ This broadcast gives an certified attestation, and increment the counter of the pre-propose, such that other processes can know if there were other pre-proposal.
21: **Delivery phase:**
22: $timerPrePropose \leftarrow timeoutPrePropose$
23: **while** $\neg$expired$(timerPrePropose) \wedge \neg(\exists v_j, e_j :$ sentByProposer$(e_i, v_j, e_j))$ **do**
24: **if** $\exists v_j, e_j :$ sentByProposer$(e_i, v_j, e_j)$ **then**
25: $v_i \leftarrow v_j$ ▷ $v_j$ is the value sent by the proposer
26: $validEpoch_j \leftarrow e_j$ ▷ $e_j$ is the $validEpoch$ sent by the proposer
27: **if** $\neg(\exists v, epochProp :$ sentByProposer$(e_i, v, epochProp))$ **then**
28: $timeoutPrePropose \leftarrow timeoutPrePropose + 1$
29: **Compute phase:**
30: **if** t+1$(\langle \text{PROPOSE}, validEpoch_j, v_i \rangle)$
31: $\wedge validEpoch_j \geq lockedEpoch_i$
32: $\wedge validEpoch_j < e_i$
33: $\wedge$isValid$(v_i)$ **then**
34: $proposal_i \leftarrow v_i$
35: **else**
36: **if** $\neg$isValid$(v_i) \vee (lockedEpoch_i > validEpoch_j \wedge lockedValue_i \neq v_i)$ **then**
37: $proposal_i \leftarrow nil$ ▷ Note that isValid is set to `false`
38: **if** isValid$(v_i) \vee (lockedEpoch_i = -1 \vee \ lockedValue_i = v_i)$ **then**
39: $proposal_i \leftarrow v_i$

**Algorithm 9** Partial Synchronous TenderTee for height $h$ executed by $i$ Rounds Propose and Vote

1: **Round** PROPOSE:
2: **Send phase:**
3: **if** $proposal_i \neq nil$ **then**
4: **TMC-Broadcast**$(< \mathbf{PROPOSE}, e_i, \boldsymbol{proposal}_i, \boldsymbol{validEpoch}_i, \textbf{preproposeMessagesDelivered}_i(e_i) >)$
5: **else**
6: **TMC-Broadcast**$(< \mathbf{PROPOSE}, e_i, \text{HeartBeat}, \boldsymbol{validEpoch}_i, \textbf{preproposeMessagesDelivered}_i(e_i) >)$
7: **Delivery phase:**
8: $timerPropose \leftarrow timeoutPropose$
9: **while** $\neg\text{expired}(timerPropose) \wedge \neg\text{t+1}(\langle(\text{HeartBeat}, \text{PROPOSE}) \mid \text{PROPOSE}, e_i\rangle)$ **do**{}
10: ▷ HeartBeat messages must originate from distinct processes.
11: **end while**
12: **if** $\neg\text{t+1}(\langle(\text{HeartBeat}, \text{PROPOSE}) \mid \text{PROPOSE}, e_i\rangle)$ **then**
13: $timeoutPropose \leftarrow timeoutPropose + 1$
14: **Compute phase:**
15: **if** $\exists v' :$ t+1$(\langle\text{PROPOSE}, e_i, v'\rangle) \wedge \text{isValid}(v') \wedge \text{sentByProposer}(e_i, v')$ **then**
16: $lockedValue_i \leftarrow v'$; $lockedEpoch_i \leftarrow e_i$
17: $validValue_i \leftarrow v'$; $validEpoch_i \leftarrow e_i$; $vote_i \leftarrow v'$
18: **else**
19: $vote_i \leftarrow nil$
20: **Round** VOTE:
21: **Send phase:**
22: **if** $vote_i \neq nil$ **then**
23: **TMC-Broadcast**$(< \mathbf{VOTE}, e_i, \boldsymbol{vote}_i, \boldsymbol{validEpoch}_i, \textbf{proposeMessagesDelivered}_i(e_i) >)$
24: **else**
25: **TMC-Broadcast**$(< \mathbf{VOTE}, e_i, \text{HeartBeat}, \boldsymbol{validEpoch}_i, \textbf{proposeMessagesDelivered}_i(e_i) >)$
26: **Delivery phase:**
27: $timerVote \leftarrow timeoutVote$
28: **while** $\neg\text{expired}(timerVote) \wedge \neg\text{t+1}(\langle(\text{HeartBeat}, \text{VOTE}) \mid \text{VOTE}, e_i\rangle)$ **do**{}
29: **end while**
30: **if** $\neg\text{t+1}(\langle(\text{HeartBeat}, \text{VOTE}) \mid \text{VOTE}, e_i\rangle)$ **then**
31: $timeoutVote \leftarrow timeoutVote + 1$
32: **Compute phase:**
33: **if** $\exists v'' :$ t+1$(\langle\text{PROPOSE}, e_i, v''\rangle) \wedge \text{isValid}(v'') \wedge \text{sentByProposer}(e_i, v'')$ **then**
34: $validValue_i \leftarrow v''$; $validEpoch_i \leftarrow e_i$
35: **if** $\exists v_d, e_d :$ t+1$(\langle\text{VOTE}, e_d, v_d\rangle) \wedge \text{isValid}(v_d) \wedge decision_i = nil$ **then**
36: $decision_i \leftarrow v_d$
37: **else**
38: $e_i \leftarrow e_i + 1$ ; $v_i \leftarrow nil$
39: **if** $validValue_i \neq nil$ **then**
40: $proposal_i \leftarrow validValue_i$
41: **else**
42: $proposal_i \leftarrow \text{getValue}()$

*Broadcast*: When a process TMC-Broadcasts a value, it broadcasts in the same message all the messages received at the previous step as well as its vote message from the previous epochs (and *nil* when in epoch 0), proving that his message is causally related to the previous messages, notice that theses messages are also accompanied by the corresponding certificates and counters. Therefore, when a process delivers a TMC-Broadcast message, in addition to checking the certificates and counters, the process also checks that the message is causally possible according to the accompanying messages. When a message does not satisfy the causality property, it is kept, but not yet used until it delivers messages allowing to satisfy such property.

### 6.2.2 Correctness of TenderTee

We now prove the correctness of TenderTee (Algorithms 8 & 9) in a partially synchronous system. We suppose that $n = 2t + 1$ and that each protocol's message $m$ sent by a node $i$ is sent with an attestation $< att_{m,i}, m >$ produced by TMC-Object.

**Lemma 12** (Validity)**.** *In a partially synchronous system, TenderTee satisfies the following property: A decided value satisfies the predefined predicate denoted as* `isValid`().

*Proof.* The proof directly follows by construction. When a correct process decides (Line 36 of Algorithm 9), it checks before if that value is valid (Line 35 of Algorithm 9). Therefore, a correct process only decides valid values. □

**Lemma 13** (Integrity)**.** *In a partially synchronous system, TenderTee satisfies the following property: No correct process decides twice.*

*Proof.* The proof follows by construction. Before deciding (Lines 35 - 36), a correct process $i$ checks if there is not already a value decided ($decision_i = nil$) for the current height (*i.e.* line 35). If there is already a value decided ($decision_i \neq nil$), there is no decision (Lines 37 - 42). Moreover, a correct process exits the algorithm after it has decided (Line 16 of Algorithm 8). No correct process decides twice. □

**Lemma 14.** *In a partially synchronous system, TenderTee satisfies the following property: A correct process proposes and votes only once per epoch.*

*Proof.* We prove this lemma by construction. In Algorithm 9, a correct process proposes (Line 4) and votes only once during the corresponding round (Line 23 or Line 25). At the end of the VOTE round, a process changes epoch (Line 38). Therefore, it cannot propose nor vote for that epoch any more. □

**Lemma 15.** *If a correct process accepts a propose (or vote) message from another process, only that message can be accepted by the other correct processes. Somehow, it means that processes can* validly[3] *propose and vote at most once.*

[3]Validly here means that the vote message can be accepted by correct processes, meaning it satisfies the TMC-Object guarantees.

*Proof.* By Lemma 14, we have that correct processes propose and vote at most once per epoch. Now, we can focus on the Byzantine processes. Assume that a Byzantine process proposes (resp. votes) for two different value. Broadcast should be done with the TMC-Broadcast. If the process does not use the TMC-Object for the broadcast, there will be no certificate, hence the corresponding message will not be considered by correct processes, since it is not valid. If the process uses the TMC-Broadcast for both propose messages (resp. vote messages) $v$ and $v'$, the two messages cannot have the same counter number, however, only one could be valid and accepted by correct processes, all other values cannot be accepted thanks to the TMC-Object guarantees. The Byzantine process can, however, also not propose (resp. vote) any message. □

**Lemma 16.** *In a partially synchronous system, TenderTee satisfies the following property: At most one value can be proposed by at least $t+1$ processes per epoch, and at most one value can be voted at least $t+1$ times per epoch.*

*Proof.* We prove this lemma by contradiction. Let $v, v'$ such that $v \neq v'$. Since there are $2t+1$ processes in the system, if $v$ or $v'$ gets at least $t+1$ proposals (resp. votes), it means that at least 1 process proposes (resp. votes) for both $v$ and $v'$ which contradicts Lemma 15. □

**Lemma 17.** *Let $v$ be a value, and let $e$ an epoch. In a partially synchronous system, TenderTee satisfies the following property: If at least $t+1$ different processes (validly) vote for $v$ during epoch $e$, then no correct process $i$ will have $lockedValue_i \neq v \wedge lockedEpoch_i \geq e$, at the end of each epoch $e' > e$; moreover, all correct processes who voted $v$ during epoch $e$ can only proposes $v$ or nil for each epoch $e' > e$.*

*Proof.* Let $v$ be a value, and let $e$ be an epoch, and $L^{v,e} = \{i : i \text{ valid votes for } v \text{ during epoch } e\}$, we assume that $|L^{v,e}| \geq t+1$. We prove the theorem by induction on epoch number.

- *Initialisation:* At the end of epoch $e$, by assumption $|L^{v,e}| \geq t+1$. There is a correct process in that set, say $i$ ($i \in L^{v,e}$). It means that $i$ updates $lockedValue_i$ to $v$ during epoch $e$, therefore $i$ delivered $t+1$ proposals for the value $v$ (Lines 15 - 16 of Algorithm 9). By Lemma 16, at most one value can have at least $t+1$ proposals during epoch $e$, and since $v$ has at least $t+1$ proposals, no correct process $j$ can update $lockedValue_j$ to a value $v' \neq v$ during epoch $e$. At the end of $e$, for any correct process $j$, $lockedValue_j = v \vee lockedEpoch_j < e$.

- Induction: Let $a \geq 1$, we assume that $\forall i \in L^{v,e}$, $lockedValue_i = v$ at the end of each epoch between $e$ and $e+a$, we also assume that if a value was proposed at least $t+1$ times during these epochs it was either $v$ or *nil*. We prove that at the end of epoch $e+a+1$, no correct process $j$ will have $lockedValue_j = v' \wedge lockedEpoch_j = e+a+1$ with $v' \neq v$.

  Let $i \in L^{v,e}$ such that $i$ delivers a pre-proposal for $v$, then $i$ will set $proposal_i$ to $v$; it will propose $v$ since $lockedValue_i = v$ (Lines 33 - 39 of Algorithm 8 & Line 4 of Algorithm 9), in any other case, if $i$ does not deliver a pre-proposal or deliver a pre-proposal for a value $v' \neq v$, it will set $proposal_i$ to *nil* and will propose *nil*

(Lines 33 - 39 of Algorithm 8 & Line 4 of Algorithm 9), since $\texttt{isValid}(nil) = \texttt{false}$ and by assumption, there is no $e' \in \{e, \ldots, e+a\}$ where there were at least $t+1$ proposals for a value $v' \neq v$, and $lockedEpoch_i \geq e$. All processes in $L^{v,e}$ will then propose $v$, *nil* or not propose at all during epoch $e+a+1$. By Lemma 14, correct processes only propose once per epoch, at least $t+1$ processes propose $v$ or *nil*; since messages cannot be forged, the only values that can get at least $t+1$ proposals for the epoch $e+a+1$ are $v$ and *nil*. If a correct process $j$ delivers at least $t+1$ proposals for $v$, it sets $lockedValue_j$ to $v$ and $lockedEpoch_j$ to $e+a+1$ (Lines 15 - 16 of Algorithm 9); otherwise, it does not change $lockedValue_j$ nor $lockedEpoch_j$ (Line 19 of Algorithm 9). At the end of epoch $e+a+1$, there is no correct process $j$ such that $lockedValue_j \neq v \wedge lockedEpoch_j = e+a+1$. Moreover, processes in $L^{v,e}$, only propose $v$ or *nil* during epoch $e+a+1$.

We proved that if $|L^{v,e}| \geq t+1$, no correct process $i$ will have $lockedValue_i \neq v \wedge lockedEpoch_i \geq e$; moreover a process in $L^{v,e}$ only validly proposes $v$ or *nil* for each epoch $e' > e$. □

**Lemma 18** (Agreement)**.** *In a partially synchronous system, TenderTee satisfies the following property: If there is a correct process that decides a value $v$, then eventually all the correct processes decide $v$.*

*Proof.* Let $i$ be a correct process. Without loss of generality, assume that $i$ is the first correct process that decides and assume that it decides value $v$ during epoch $e$. At time $t$ where $i$ decided, no other node has decided, even those having a bigger epoch number. To decide, $i$ delivered at least $t+1$ votes for $v$ for epoch $e$. Since there are less than $t$ Byzantine processes, and since by Lemma 14 correct processes can only vote once per epoch, so at least 1 correct process voted for $v$ during epoch $e$, so we have $L^{v,e} = \{i : i$ validly votes for $v$ during epoch $e\}$, we assume that $|L^{v,e}| \geq t+1$. By Lemma 17 a process in $L^{v,e}$ only proposes $v$ or *nil* during each epoch after $e$, and no correct process $j$ will have $lockedValue_i \neq v \wedge lockedEpoch_i \geq e$.

Thanks to the broadcast guarantees (Definition 1), all correct processes will eventually deliver the $t+1$ votes for $v$ from epoch $e$ that made $i$ decides; especially so because when a correct process decides, it sends back all votes it delivered that make it decide (Line 14 of Algorithm 8).

If a correct process $j$ does not decide before delivering these votes, when eventually it delivers them, it will decide $v$ (Lines 35 - 36 of Algorithm 9). Otherwise, it means that $j$ decides before delivering the votes from epoch $e$.

By contradiction, we assume that $j$ decides a value $v' \neq v$ during an epoch $e' > e$, so $j$ delivered at least $t+1$ votes for $v'$ during epoch $e'$ (Lines 35 - 36 of Algorithm 9). Since a correct process only votes once by Lemma 14, there are less than $t$ Byzantine processes and the messages are unforgeable, at least 1 correct process voted for $v'$ during epoch $e'$.

A correct process votes a non-*nil* value if that value was proposed at least $t+1$ times during the current epoch (Lines 15 - 25 of Algorithm 9). By Lemma 15 processes validly proposes at most once, there are less than $t$ Byzantine processes and the messages are unforgeable,

so at least 1 correct process proposed $v'$ during $e'$. Since $e' > e$ and $|L^{v,e}| \geq t+1$, by Lemma 17 there are at least 1 correct process that proposed $v$ or *nil* during epoch $e'$. Even if all the $t$ processes remaining proposes $v'$, there cannot be $t+1$ proposals for $v'$, which is a contradiction. So, $j$ cannot decide $v' \neq v$ after epoch $e$ since by assumption, $e$ is the first epoch where a correct process decides. □

**Lemma 19.** *In a partially synchronous system, if there is an epoch after which, when a correct process broadcasts a message during a round, it is delivered by all correct processes during the same round, TenderTee satisfies the following property: If a correct process $i$ updates $lockedValue_i$ to a value $v$ during epoch $e$, then at the end of the epoch $e$, all correct processes have $validValue = v$ and $validEpoch = e$.*

*Proof.* We prove this lemma by construction.

Let $e$ be the epoch after which when a correct process broadcasts a message during a round $r$, it is delivered by all correct processes during the same round $r$. Let $i$ be a correct process, we assume that at the end of epoch $e' \geq e$, $i$ has $lockedValue_i = v$ and $lockedEpoch_i = e'$, it means that $i$ delivered at least $t+1$ proposals for $v$ during epoch $e'$ (Lines 15 - 16 of Algorithm 9). Thanks to the reliable broadcast guarantees, and since all messages are propagated, all correct processes will deliver these proposals for $v$ in the worst-case in the VOTE round. Let $j$ be a correct process since $j$ will deliver at least $t+1$ proposals for $v$ and epoch $e'$ during the VOTE round, it will set $validValue_j = v$ and $validEpoch_j = e'$ (Lines 33 - 34 of Algorithm 9). □

**Lemma 20** (Termination)**.** *In a partially synchronous system, TenderTee satisfies the following property: Every correct process eventually decides a value.*

*Proof.* By construction, if a correct process does not deliver more than $t+1$ messages (or 1 from the proposer in the PRE-PROPOSE round) from different processes during the corresponding round, it increases the duration of its round. So eventually, during the synchronous period of the system, all the correct processes will deliver the pre-proposals, proposals and votes from correct processes respectively during the PRE-PROPOSE, PROPOSE and the VOTE round; and messages delivered by a correct process will be delivered by the others at most in the following round. Let $e$ be the first epoch after that time.

If a correct process decides before $e$, by Lemma 18 all correct processes will eventually decide, which ends the proof.

Otherwise, at the beginning of epoch $e$, no correct process decides yet. Let $i$ be the proposer of epoch $e$. First, we assume that $i$ is correct and pre-proposes $v$; $v$ is valid since getValue() always gives a valid value (Line 7 of Algorithm 8 & Line 42 of Algorithm 9), and $validValue_i$ is always valid (Lines 15 & 33 of Algorithm 9). We have two cases:

- Case 1: At the beginning of epoch $e$, $|\{j : j \text{ correct} \wedge (lockedEpoch_j \leq validEpoch_i \vee lockedValue_j = v)\}| \geq t+1$.

  Let $j$ be a correct process where the condition $lockedEpoch_j \leq validEpoch_i \vee lockedValue_j = v$ holds. After the delivery of the pre-proposal $v$ from $i$, $j$ will update $proposal_j$ to $v$ (Lines 33 - 39 of Algorithm 8). During the PROPOSE round,

$j$ proposes $v$ (Line 4 of Algorithm 9), since there are at least $t+1$ similar correct processes (included $j$), they will all propose $v$, and all correct processes will deliver at least $t+1$ proposals for $v$ (Line 8 of Algorithm 9).

Correct processes will set their variable *vote* to $v$ (Lines 15 - 4 of Algorithm 9), then will vote $v$, and they will deliver all the votes (at least $t+1$) from this epoch (Lines 23, 25 & 26 of Algorithm 9). Since no correct process has decided yet, and since each of them delivers at least $t+1$ votes for $v$, they will decide $v$ (Lines 35 - 36 of Algorithm 9).

- Case 2: At the beginning of epoch $e$, $|\{j : j \text{ correct} \wedge (lockedEpoch_j \leq validEpoch_i \vee lockedValue_j = v)\}| < t+1$.

  Let $j$ be a correct process where the condition $lockedEpoch_j > validEpoch_i \wedge lockedValue_j \neq v$ holds. When $i$ will make the pre-proposal, $j$ will set $proposal_j$ to *nil* (Line 37 of Algorithm 8) and will propose *nil* (Line 4 of Algorithm 9).

  By counting only the proposed values of the correct processes, no value will have at least $t+1$ proposals for $v$. There are two cases:

  - No correct process delivers at least $t+1$ proposals for $v$ during the PROPOSE round, so they will all set their variable *vote* to *nil*, then they will vote *nil* and go to the next epoch without changing their state (Lines 19 & 23 - 26 & 37 - 42 of Algorithm 9).

  - If some correct processes deliver at least $t+1$ proposals for $v$ during the PROPOSE round, i.e., some Byzantine processes send proposals for $v$ to those processes.

    As in the previous case, they will vote for $v$, and since there are $t+1$ of them, all correct processes will decide $v$. Otherwise, there are less than $t+1$ correct processes that deliver at least $t+1$ proposals for $v$. Only them will vote for $v$ (Line 23 or 25 of Algorithm 9). Without Byzantine processes, there will be less than $t+1$ votes for $v$, no correct process will decide (Lines 35 - 36 of Algorithm 9) and they will go to the next epoch; if Byzantine processes send votes for $v$ to a correct process such that it delivers at least $t+1$ votes for $v$ during VOTE round, then the correct process will decide (Lines 35 - 36 of Algorithm 9), and by Lemma 18 all correct processes will eventually decide.

    Let $k$ be one of the correct processes that delivers at least $t+1$ proposals for $v$ during the PROPOSE round, it means that $lockedValue_k = v$ and $lockedEpoch_k = e$. It follows by Lemma 19 that at the end of epoch $e$, all correct processes will have $validValue = v$ and $validEpoch = e$.

  If there is no decision, either no correct process changes its state, or all correct processes change their state and have the same *validValue* and *validEpoch*; therefore, eventually, a proposer of an epoch will satisfy Case 1, and that ends the proof.

If $i$, the proposer of epoch $e$, is a Byzantine process and more than $t+1$ correct processes delivered the same message during PRE-PROPOSE round, and that pre-proposal is valid, the situation is the same as if $i$ was correct. Otherwise, there are not enough correct

processes that delivered the pre-proposal, or if the pre-proposal is not valid, then there will be less than $t + 1$ correct processes that will propose that value, which is similar to case 2.

Since proposers are selected in a round-robin fashion, a correct process will eventually be the proposer, and correct processes will decide. □

**Theorem 7.** *In a partially synchronous system, TenderTee implements the consensus specification*[4].

*Proof.* The proof follows directly from Lemmas 12, 13, 18 and 20. By Lemma 12, we show that TenderTee satisfies Validity; by Lemma 13, we show that TenderTee satisfies Integrity; by Lemma 18, we show that TenderTee satisfies Agreement; and by Lemma 20, we show that TenderTee satisfies Termination. □

### 6.2.3 TenderTee Repeated Consensus

TenderTee Repeated Consensus reuses the **Data Structures** and **Functions** defined for Tendermint Repeated Consensus (Subsubsection 6.1.2).

#### TenderTee Repeated Consensus Algorithm

In the repeated consensus module, we rely on A2M-Broadcast (Algorithm 10) instead of the TMC-Broadcast (Algorithm 4) used in the one-shot consensus. The key reason is that the repeated consensus requires explicit progression across epochs and heights: at the end of each consensus instance, every process must update its logs (`logPrepropose`, `logPropose`, and `logVote`) in order to guarantee that the next epoch starts from a consistent state (see Lines 16–18 in Figure 6.2). The A2M-advance operation, which securely increments the log index inside the TEE, is tightly bound to the use of A2M-Broadcast. This mechanism ensures that each broadcast message carries evidence of the sender's position in the log and enforces monotonicity across epochs. In contrast, TMC-Broadcast does not update the log position and therefore cannot provide the guarantees needed for consistent state advancement across multiple consensus iterations. For this reason, the repeated consensus algorithm mandates the use of A2M-Broadcast when broadcasting commit messages and updating the logs at each height.

**Algorithm 10** A2M-Broadcast

1: **A2M-Broadcast**(*round*,*height*, *epoch*,*log*,*message*, *validEpoch*):
2: **broadcast** $(< round, height, epoch, message, validEpoch, A2M - advance(log, height \times SizeEpoch + epoch, message) >)$

**Detailed description of the algorithm** We describe in Figure 6.2 the TenderTee algorithm for the repeated consensus (Definition 3). Each message is sent through A2M-

[4]Definition 2, Subsection 4.1.8

**Function** TenderTee-Repeated-Consensus($\Pi$);
▷ Repeated consensus for the set $\Pi$ of nodes

**Init**:
(1) $h \leftarrow 1$; $B \leftarrow \perp$; $C \leftarrow \perp$ ▷ *Height* and set of committee members;
(2) $commitsReceived_i^h \leftarrow \emptyset$; $toReward_i^h \leftarrow \emptyset$; $TimeOutCommit \leftarrow \Delta_{\text{Commit}}$;

---

**while** (true) **do**
(3) $B \leftarrow \perp$;
(4) $C \leftarrow$ committeeMembers($h$); ▷ Application and blockchain dependant
(5) **if** ($i \in C$) **then**
(6) $B \leftarrow$ TenderTee-consensus($h, C, toReward_i^{h-1}$);
(7) ▷ Consensus A2M function for the height $h$
(8) **trigger** A2M-Broadcast $\langle \texttt{COMMIT}, (B,h)_i, att_i \rangle$;
(9) **else**
(10) **wait until** ($\exists B' : |\text{MoreThanHalf}(B', commitsReceived_i^h)|$); ▷ Wait f+1 (verified) commits for the same block
(11) $B \leftarrow B'$;
(12) **endif**
(13) set *timerCommit* to *TimeOutCommit*;
(14) **wait until**(*timerCommit* expired);
(15) **trigger** decide($B$);
(16) A2M-advance($logPrepropose_i$, h*SizeEpoch - 1, digest, 0);
(17) A2M-advance($logPropose_i$, h*SizeEpoch - 1, digest, 0);
(18) A2M-advance($logVote_i$, h*SizeEpoch - 1, digest, 0);
(19) ▷ Update log $h$
(20) $h \leftarrow h + 1$;
**endwhile**

---

**upon event** delivery $\langle \texttt{COMMIT}, (B',h')_j, att_j \rangle$:
(21) **if** $((B',h')_j \notin commitsReceived_i^{h'}) \wedge (j \in \text{committeeMembers}(h'))$ $\wedge$ verify($\texttt{COMMIT}, (B',h')_j, att_j, pk_j$)) **then**
(22) $commitsReceived_i^{h'} \leftarrow commitsReceived_i^{h'} \cup \{(B',h')\}_j$;
(23) $toReward_i^{h'} \leftarrow toReward_i^{h'} \cup \{j\}$;
(24) **trigger** A2M-Broadcast $\langle \texttt{COMMIT}, (B',h')_j, att_j \rangle$;
(25) **endif**

Figure 6.2: TenderTee Repeated Consensus algorithm for Correct node sis.

Broadcast (Algorithm 10) and each message received is verified with Algorithm 2 (verify A2M's attestation validity). For a node $i$, the algorithm proceeds as follows:

- $i$ computes the set of committee members for the current height;
- If $i$ is a committee member, then it calls the TenderTee-consensus function (the one-shot TenderTee consensus) solving the consensus using A2M for the current height, then broadcasts the decision (using A2M-Broadcast function), and sets $B$ to that decision;
- Otherwise, if $i$ is not a committee member, it waits for at least $n/2$, or equivalently $f + 1$, commits from the same block and sets $B$ to that block;
- In any case, it sets the timer to *TimeOutCommit* to collect more commits and lets it expire. Then $i$ decides $B$ and goes to the next height.

Whenever $i$ delivers a commit, it broadcasts it using A2M-Broadcast function (Lines 21 - 25 of Figure 6.2).

### 6.2.4 Correctness of TenderTee Repeated Consensus

In this section, we prove the correctness of the repeated consensus module of TenderTee, when assuming that the consensus algorithm used (Line 6 of Figure 6.2) is correct (Definition 3). In the proofs below, the lines mentioned refer to the lines in the algorithm presented in Figure 6.2.

**Lemma 21** (brc-Termination)**.** *In a partially synchronous system, and assuming that TenderTee one-shot consensus is correct, the TenderTee Repeated Consensus algorithm satisfies the following property: Every correct node has an infinite output.*

*Proof.* By contradiction, let $i$ be a correct node and we assume that $i$ has a finite output. Two scenarios are possible, either $i$ cannot go to a new height, or from a certain height $h$ it outputs only $\perp$.

- If $i$ cannot progress, one of the following cases is satisfied:
  - The function TenderTee-consensus does not terminate (Line 6), which is a contradiction since it violates the Termination property of the TenderTee-consensus (Termination in Definition 3).
  - $i$ waits an infinite time for receiving enough commits (Line 10), which cannot be the case thanks to the A2M-Broadcast guarantees and the partial-synchronous assumption, all the correct committee members terminate the function TenderTee-consensus and broadcast their commit.

- If $i$ does not decide at each height (Line 15), it means that from a given height, $i$ only outputs $\perp$. Let height $h$ be the first such height, and let $h' > h$; (i) either $i$ is a committee member for $h'$ and the function TenderTee-consensus returns $\perp$ (Lines 5 & 6), or (ii) $i$ is not a committee member for $h'$ but delivered at least $f+1$ commits for $\perp$ (Lines 10 & 24).

  (i) If TenderTee-consensus returns the value $\perp$, it means by the Validity property (Definition 12) of the consensus that `isValid`($\perp$) = `true`, which is a contradiction with the definition of the function `isValid`.

  (ii) Only committee members commit, and each of them broadcasts its commit (Lines 5 - 8), and $f < n/2$. Since non-committee members collect at least $n/2$ commits, it means that $i$ delivered a commit from at least one correct committee member. By the Validity property of the TenderTee-consensus (Definition 12), correct nodes only decide/commit on valid value, and $\perp$ is not valid, which is a contradiction.

Therefore, if $i$ is a correct node, then it has an infinite output. □

**Lemma 22** (brc-Agreement)**.** *In a partially synchronous system, and assuming that the TenderTee-consensus function is correct (i.e. it satisfies the one-shot consensus specification), TenderTee Repeated Consensus Algorithm satisfies the following property: If the $h^{th}$ value of the output of a correct node is $B$, then $B$ is the $h^{th}$ value of the output of any other correct node.*

*Proof.* We prove this lemma by construction. Let $i$ and $j$ be two correct nodes, and let $h$ be a height. Two cases are possible:

- $i$ and $j$ are committee members for the height $h$, so both calls the function TenderTee-consensus (Lines 5 & 6). By Agreement property of the TenderTee-consensus (Lemma 18), $i$ and $j$ decide the same value and then output that same value (Line 15).

- At least one of $i$ and $j$ is not a committee member for the height $h$. Without loss of generality, we assume that $i$ is not a committee member for the height $h$. Since all the correct committee members commit the same value, let's say $B$, thanks to the Agreement property of the TenderTee-consensus (Definition 3), and since they broadcast their commit (Line 8), eventually there will be more than $f+1$ commits for $B$. So no other value $B' \neq B$ can be present at least $f+1$ times in the set $commitReceived_i^h$. Therefore, $i$ outputs the same value $B$ as correct committee members (Line 10). If $j$ is a committee member, that ends the proof. If $j$ is not a committee member, then by the same argument, $j$ outputs the same value $B$. Hence, both $i$ and $j$ output the same value $B$ for height $h$.

□

**Lemma 23** (brc-Validity)**.** *In a partially synchronous system, and assuming that* TenderTee-consensus *function is correct (i.e. it satisfies the one-shot consensus specification), TenderTee Repeated Consensus Algorithm satisfies the following property: Each value in the output of any correct node is valid; it satisfies the predefined predicate denoted* `isValid`*.*

*Proof.* We prove this lemma by construction. Let $i$ be a correct node and assume that the $h^{\text{th}}$ value of the output of $i$ is $B$. If $i$ decides a value (Line 15), then that value has been set during the execution and for that height (Line 3):

- If $i$ is a committee member for the height $h$, then $B$ is the value returned by the function TenderTee-consensus. By the Validity property of the TenderTee-consensus (Definition 12), we have $\texttt{isValid}(B) = \texttt{true}$.

- Let $h$ be a height, and $B$ be the $h^{\text{th}}$ value of the output of a node $j$. If $j$ is not a committee member for the height $h$, it means that it delivered more than $f + 1$ signed commits from the committee members for the value $B$ (Lines 5 - 8 and 21 - 25), hence at least one correct committee member committed $B$, and by the Validity property of the TenderTee-consensus (Definition 12), we have $\texttt{isValid}(B) = \texttt{true}$.

Each value in the output of a correct node satisfies the predicate `isValid`. □

**Theorem 8.** *In a partially synchronous system, and assuming that the* TenderTee-consensus *function is correct (i.e. it satisfies the one-shot consensus specification), the TenderTee Repeated Consensus algorithm implements Repeated Consensus*[5]*.*

*Proof.* Assuming that the TenderTee-consensus function satisfies the one-shot consensus specification, the proof follows directly from Lemmas 21, 22 and 23. By Lemma 21, we show that the TenderTee Repeated Consensus algorithm satisfies brc-Termination; by Lemma 22, we show that the TenderTee Repeated Consensus algorithm satisfies brc-Agreement; and by Lemma 23, we show that the TenderTee Repeated Consensus algorithm satisfies brc-Validity. □

[5] Definition 3, Subsection 4.1.9

## 6.3 Protocol comparison: TenderTee vs. Tendermint

### 6.3.1 Message Complexity

**Model.** Let $f$ be the maximum number of Byzantine processes tolerated. Tendermint requires $n = 3f + 1$ validators, while TenderTee tolerates $f \leq \lfloor \frac{n-1}{2} \rfloor$ (i.e., $n = 2f + 1$). Both protocols run a successful decision in one *epoch* composed of three communication rounds: *(i)* proposer reliable broadcast (PRE-PROPOSE), *(ii)* all validators reliable broadcast (PROPOSE), *(iii)* all validators reliable broadcast (VOTE).

Each reliable broadcast involves three phases: SEND, ECHO, and READY. We do not count self-sent messages.

**Cost of a single Reliable Broadcast.** For Tendermint, using the classical Byzantine Reliable Broadcast (Bracha-style) with $n = 3f + 1$ validators:

- SEND phase: $(n - 1) = 3f$ messages
- ECHO phase: $n(n - 1) = (3f + 1)(3f) = 9f^2 + 3f$ messages
- READY phase: $n(n - 1) = 9f^2 + 3f$ messages

Total per RB: $3f + 9f^2 + 3f + 9f^2 + 3f = 18f^2 + 9f$ messages.

For TenderTee, using TMC-Reliable Broadcast with $n = 2f + 1$ validators:

- SEND phase (with TMC attestation): $(n - 1) = 2f$ messages
- ECHO phase: $n(n - 1) = (2f + 1)(2f) = 4f^2 + 2f$ messages
- READY phase: $n(n - 1) = 4f^2 + 2f$ messages

Total per TMC-RB: $2f + 4f^2 + 2f + 4f^2 + 2f = 8f^2 + 6f$ messages.

**Number of Reliable Broadcasts per successful epoch.** Both protocols execute:

- 1 RB in PRE-PROPOSE (proposer only)
- $n$ RBs in PROPOSE (all validators)
- $n$ RBs in VOTE (all validators)

Total number of RBs: $1 + 2n$.

For Tendermint: $1 + 2(3f + 1) = 6f + 3$ reliable broadcasts.

For TenderTee: $1 + 2(2f + 1) = 4f + 3$ reliable broadcasts.

**Total message complexity per successful epoch.**

$$M_{\mathrm{TM}}(f) = (6f + 3) \times (18f^2 + 9f) = 108f^3 + 108f^2 + 27f,$$

$$M_{\mathrm{TT}}(f) = (4f + 3) \times (8f^2 + 6f) = 32f^3 + 48f^2 + 18f.$$

**Concrete numbers for $f = 1$.**

$$n_{\mathrm{TM}} = 4, \quad M_{\mathrm{TM}}(1) = 9 \times 27 = 243 \text{ messages},$$
$$n_{\mathrm{TT}} = 3, \quad M_{\mathrm{TT}}(1) = 7 \times 14 = 98 \text{ messages}.$$

**Breakdown ($f$=1):**

- Tendermint ($n$=4): 9 reliable broadcasts, each costing 27 messages. Total: $9 \times 27 = 243$.
- TenderTee ($n$=3): 7 TMC-reliable broadcasts, each costing 14 messages. Total: $7 \times 14 = 98$.

**Real-world gain.** For $f = 1$, TenderTee uses 98 transmissions vs. 243 for Tendermint, i.e.,

$$\frac{M_{\mathrm{TT}}(1)}{M_{\mathrm{TM}}(1)} = \frac{98}{243} \approx 0.403,$$

representing approximately **60% fewer** messages for a successful decision.

As $f$ grows, the ratio tends to

$$\lim_{f \to \infty} \frac{32f^3 + 48f^2 + 18f}{108f^3 + 108f^2 + 27f} = \frac{32f^3}{108f^3} = \frac{32}{108} = \frac{8}{27} \approx 0.296,$$

i.e., approximately **70% fewer** messages at large scale.

**Takeaway.** Both protocols have $\Theta(n^3)$ message complexity in terms of $f$ per successful epoch, but TenderTee achieves optimal resilience with $n = 2f+1$ instead of $3f+1$. This results in a substantial reduction in communication overhead: approximately 60% fewer messages for $f$=1, approaching 70% reduction as $f$ increases. This gain stems from two factors: (i) fewer validators needed ($2f + 1$ vs. $3f + 1$), and (ii) more efficient reliable broadcast primitive enabled by TMC attestations ($8f^2+6f$ vs. $18f^2+9f$ per broadcast).

### 6.3.2 Temporal overhead: Access to TEE

The TenderTee protocol operates in three phases during each consensus round:

- **Round PRE-PROPOSE.** The **proposer** performs a single TEE call to securely generate and sign its proposal before broadcasting it to the network.
- **Round PROPOSE.** Each **validator** performs a TEE call to securely validate the received proposal and lock the corresponding value before sending its vote.
- **Round VOTE.** Each **validator** again performs a TEE call to verify and finalize its vote, ensuring the integrity and consistency of the decision before committing to the block.

Assuming there are $n$ validators and $r$ rounds required to reach consensus, the total number of TEE calls is given by:

$$N_{TEE}^{TenderTee} = N_{TEE}^{BRB} \times r \times (1 + 2n) = r \times (1 + 2n),$$

where $N_{TEE}^{BRB} = 1$ corresponds to the one-time initialization overhead described previously (see subsection 5.4.2), the term 1 represents the single call made by the proposer in the PRE-PROPOSE phase, and $2n$ accounts for the calls made by all validators during the PROPOSE and VOTE phases.

## 6.4 TenderTee for Financial-Grade Blockchains

As explained in Chapter 1, financial infrastructures demand *verifiable finality*, *multi-actor governance*, and *operational efficiency*. TenderTee augments a Tendermint-like BFT core with *trusted, non-equivocating* attestations (A2M/TEE), replacing large $2f+1$ quorums by $f+1$ in a committee of size $n=2f+1$, while retaining BFT-style safety under asynchrony and liveness after GST [32, 144, 33]. The result is the *same asymptotic order* ($\Theta(n^2)$ per successful round) but strictly *smaller constants* (fewer validators, fewer transmissions, reduced energy), and *cryptographic accountability* out-of-the-box.

**Beyond complexity: enabling tri-party operating models.** Beyond reducing message complexity and the associated energy footprint, TenderTee enables realistic *tri-party* deployment patterns that are ubiquitous in finance. By combining anti-equivocation through trusted attestations and smaller quorums ($f+1$ with $n=2f+1$), TenderTee makes it practical for multiple independent institutions to co-validate a shared ledger. This lowers adoption barriers in regulated environments while preserving separation of duties, auditability, and compliance.

## Typical tri-party scenarios examples in finance (who does what).

**Tri-party repo (short-term loan with collateral).** There are three roles. The *dealer* needs cash for a few days and promises to buy it back later; to secure this promise, the dealer hands over safe assets (the *collateral*). The *cash investor* provides the cash and expects to get it back with a small fee. The *triparty agent* is a neutral bank that holds the collateral and checks every day that its value still covers the loan. With TenderTee, each action is recorded and co-signed: when the dealer posts collateral, when the agent checks its value, when the investor asks for more collateral if prices fall, and when the loan is closed. No one can secretly change these records later, so all three always see the same final state.

**Client–Asset Manager–Custodian (who owns, who decides, who holds).** The *client* is the real owner of the assets. The *asset manager* decides which assets to buy or sell for the client. The *custodian* is a bank that actually holds the assets and executes the manager's orders. In practice: the manager sends an order, the custodian executes it and updates the positions, and the client must be able to check that the holdings match what was ordered. With TenderTee, the order, the execution, and the updated positions are all co-signed by the three parties and written once. This prevents hidden changes and makes later checks straightforward.

**Issuer–Underwriter–Investor (selling a new bond or share).** The *issuer* needs to raise money by selling a new financial product (a bond or a share). The *underwriter* (a bank or a group of banks) organizes the sale and collects orders from many buyers. The *investors* submit how much they want to buy. The underwriter decides who gets how much and at what price, then the issuer receives the money and delivers the product. With TenderTee, each step is fixed in place: the investor's order, the final allocation, and the payment/delivery. If someone later asks "who got what at which price?", everyone can point to the same signed record.

**Bank–Central Risk Manager–Clearing Member (making trades safe after they happen).** A *bank* trades. A *central risk manager* (a specialized institution) stands in the middle and guarantees that trades will be honored. A *clearing member* supports the bank and passes money back and forth with the central risk manager to cover risk each day. Concretely: the trade is registered, the daily risk is computed, and if needed, extra cash is posted to keep the system safe. With TenderTee, registering the trade, asking for extra cash, and confirming payment are all co-signed events that cannot be duplicated or edited later. This reduces disagreements and speeds up daily checks.

**Originator–Servicer–Trustee (pooling loans and paying investors).** An *originator* groups many small loans into one pool. A *servicer* collects monthly payments from the borrowers. A *trustee* watches over the process on behalf of investors to make sure rules are followed. Each month, the servicer receives money from borrowers and splits it according to simple rules (for example, first cover expenses, then pay investors). With TenderTee, receiving each payment, applying the rules, and sending money to investors are recorded as signed steps. The trustee and investors can verify that the steps were followed exactly, using the same unchangeable history.

**Openness with assurance.** With TenderTee, *each actor* can operate its own validators (or enclaves) and *co-sign* the shared state without relying on a single operator. Anti-equivocation (one attested message per step/round) and $f+1$ quorum certificates lower entry costs: (i) *fewer* validators for a given safety level; (ii) *fewer* messages per phase, hence less bandwidth and energy; (iii) *locally verifiable* attestations that auditors and regulators can check independently. The outcome is *pragmatic openness*: heterogeneous institutions can join consensus while retaining their own control perimeters and regulatory duties (segregation of duties, internal controls, audit trail).

**Positioning vs. PoA and classical BFT.** Proof-of-Authority (PoA) schemes (e.g., Aura/Clique) are lightweight but typically provide heuristic finality and are susceptible to equivocation or timing manipulation under adverse scheduling (as discussed in the so|bond case study, see 1.2.1); accountability often rests on off-chain governance [147, 148]. Classical BFT (e.g., Tendermint/IBFT) offers strong finality and proofs but needs $n \geq 3f+1$ with $2f+1$-sized quorum certificates, increasing validator counts and messaging [144, 149]. TenderTee keeps BFT-style guarantees but enforces non-equivocation in hardware, allowing *honest-majority* bounds ($n=2f+1$) and *smaller quorums* ($f+1$), which improves constants in validator provisioning, communication, and energy while preserving cryptographic evidence for audits [127, 33, 150].

**Practical consequence for permissioned finance.** In permissioned networks targeting interoperability (cross-application workflows, compliance compartments), these tri-party designs decrease onboarding friction (i.e. no central authority must be blindly trusted for timestamping or immutability) while maintaining safety on par with the state of the art. In short, TenderTee couples *efficiency* (fewer messages, lower energy) with *open governance* (multi-actor, tri-party), aligning with real financial operating models and removing a key barrier to production-grade blockchain adoption.

## 6.5 Conclusion

This chapter introduced **TenderTee**, a consensus protocol that integrates Trusted Execution Environments to enhance Byzantine fault tolerance in blockchain systems. Starting from the Tendermint foundation, TenderTee achieves consensus with $n = 2f + 1$ validators, rather than the classical $3f + 1$, while preserving safety and liveness under partial synchrony. Through hardware-backed attestations (TMC-Broadcast and A2M-Broadcast), the protocol ensures non-equivocation, causal consistency, and verifiable message ordering.

We proved that TenderTee satisfies the four fundamental consensus properties: *Validity*, *Integrity*, *Agreement*, and *Termination*. Its repeated consensus variant further guarantees correct state advancement across epochs and heights, thus enabling a practical blockchain construction with verifiable and energy-efficient finality.

A quantitative comparison demonstrated that TenderTee reduces the message complex-

ity by nearly half compared to Tendermint, without weakening fault tolerance guarantees. Beyond these asymptotic improvements, TenderTee offers qualitative advantages for financial-grade blockchains: it enables realistic tri-party and multi-institutional deployments, where consensus can be reached securely among a small number of independent actors.

In summary, TenderTee bridges the gap between theoretical Byzantine consensus and the operational needs of regulated financial infrastructures. It combines the robustness of BFT protocols with the verifiability of trusted hardware, paving the way for more efficient, auditable, and sustainable blockchain systems.

# Chapter 7

# Synthesis and Conclusion

## 7.1 Synthesis of Research Findings

This thesis has investigated the design of blockchain consensus protocols with a low energy footprint, specifically adapted to the requirements of the financial sector. The central research question, posed in Section 1.6, was the following:

> *How can blockchain systems be designed to meet the strict requirements of the financial sector (i.e security, regulatory compliance, and high performance) while maintaining a low energy footprint?*

To address this question, our work unfolded along three complementary dimensions: a critical analysis of blockchain adoption in finance (Chapter 1), the formalization of system models and trusted components (Chapter 4), the design of a novel Byzantine Reliable Broadcast protocol tolerating up to nearly one half of Byzantine faults (Chapter 5), and finally the development of *TenderTEE*, a consensus protocol that builds upon this broadcast abstraction by integrating trusted hardware to achieve consensus with $n = 2f + 1$ validators, thereby reducing both message complexity and validator requirements while maintaining strong security guarantees (Chapter 6).

The financial sector represents unique constraints that distinguish it from other domains of blockchain application. As discussed in Chapter 1, financial institutions demand deterministic finality, verifiable auditability, multi-party governance structures, and alignment with Environmental, Social and Governance (ESG) criteria. Traditional Proof-of-Work protocols, while robust, fail to satisfy the efficiency demands of this context. Pure Proof-of-Stake approaches, although more energy efficient, introduce centralization risks and remain vulnerable to specific attack vectors. Classical BFT protocols such as Tendermint guarantee safety but require $n \geq 3f + 1$ validators and generate $O(n^2)$ messages with large constant factors, which reduces scalability and efficiency.

Our contributions respond to these limitations through the systematic and minimal integration of trusted hardware. Chapter 5 shows the work published in the SSS 2024

conference [32] that introduced a Byzantine Reliable Broadcast protocol that leverages a single Trusted Monotonic Counter (TMC) at the initiator. This design achieves optimal resilience of $n \geq 2f + 1$ and reduces message complexity by approximately 48% for $f = 1$ (from 27 to 14 messages per broadcast instance) and 55% for larger $f$. Importantly, our analysis showed that naïvely adding trusted components to existing protocols, such as Bracha's, does not automatically improve resilience: the protocol logic must be carefully redesigned to exploit the non-equivocation guaranties of the hardware.

Chapter 6 shows the work published in ICDCN 2023 conference [33] that introduce *TenderTEE*, which extends Tendermint consensus with TMC-based attestations. By preventing equivocation at the protocol level, TenderTEE achieves Byzantine consensus with $n = 2f + 1$ validators instead of $3f + 1$, thereby reducing both the validator set size and message complexity. For $f = 1$, TenderTEE requires only 14 messages per successful round compared to Tendermint's 27, a reduction of nearly half. As the system scales, the asymptotic ration converges to $\frac{4}{9}$, corresponding to a reduction of 56%. This improvement translates directly into lower energy consumption, reduced bandwidth usage, and lower operational costs. Furthermore, TenderTEE is particularly well suited to the deployment models common in finance, such as tri-party repo arrangements, client-asset manager-custodian structures, or issuer-underwriter-investor setting. The combination of cryptographic attestations with smaller quorum requirements lowers adoption barriers for institutions while preserving regulatory and governance requirements.

The results presented in Chapters 5 and 6 are published in the journal *Theoretical Computer Science, special issue dedicated to the conference Stabilization, Safety, and Security of Distributed Systems, SSS 2024* [34].

## 7.2 Contributions to the Field

This thesis advances the state of blockchain consensus in three principal ways.

First, it demonstrates that minimal integration of trusted hardware can achieve significant improvements. A single TMC at the broadcast initiator suffices to guarantee optimal Byzantine Reliable Broadcast resilience with $n \geq 2f + 1$. This result (Theorem 3, Chapter 5) shows that trusted hardware can be used sparingly and strategically rather than being deployed across all nodes. This minimality reduces hardware costs, simplifies audits, and limits the attack surface of trusted components.

Second, we introduce TenderTEE as the first consensus protocol explicitly tailored to financial applications that achieves deterministic finality, optimal resilience with $n = 2f+1$, reduced message complexity approaching $\frac{4}{9}$ of classical BFT, cryptographic accountability through attestations, and compatibility with tri-party governance structures. TenderTee satisfies both the consensus specification (Theorem 7) and the repeated consensus specification (Theorem 8) under partial synchrony.

Finally, the thesis contributes to the practical viability of consensus protocols in regulated environments. By enabling smaller validator sets while maintaining strong security, our protocols significantly reduce operational complexity and energy consumption, bringing

blockchain infrastructures closer to ESG commitments. The auditability offered by TMC attestations provides regulators and auditors with verifiable evidence of correct execution, facilitating compliance and forensic accountability.

## 7.3 Limitations

While the proposed approaches demonstrate strong theoretical and practical potential, they are not without limitations. This section outlines the main constraints and open challenges that remain, addressing in particular the security assumptions on trusted hardware, implementation considerations, and scalability issues. .

### Risks and Attacks on Trusted Hardware

While the results presented here provide solid theoretical foundations, several limitations remain. Most notably, the security of our protocols ultimately depends on the integrity of Trusted Execution Environments. Although trusted hardware can provide substantial benefits, it is not immune to risks: numerous vulnerabilities and attacks have been documented against different TEE implementations, revealing important limitations to their reliability.

- **Plundervolt** [151]: A software-based fault injection attack targeting Intel SGX by manipulating CPU voltage levels, requiring privileged software access and specific hardware configurations.
- **CLKscrew** [152]: An attack exploiting dynamic frequency scaling capabilities on ARM TrustZone processors, allowing fault injection and cryptographic key extraction, requiring kernel-level privileges.
- **Rowhammer** [153]: A software-based attack that leverages repeated memory accesses to cause unintended bit flips in adjacent memory cells, potentially enabling privilege escalation or data corruption without physical access.
- **Spectre** and **Meltdown** [154]: Side-channel attacks exploiting speculative execution to leak sensitive data from trusted execution environments without requiring physical access.
- **Foreshadow** [155]: Specifically targeting Intel SGX, this speculative execution attack leaks sensitive enclave information, including cryptographic keys, typically requiring privileged system-level access.

These attacks usually assume significant attacker capabilities, such as privileged kernel-level access or direct physical proximity to the hardware. Therefore, their practical

implications in decentralized blockchain systems may be limited, but they nevertheless emphasize the need for multi-layered security strategies combining hardware robustness, software protection, and vigilant operational practices.

### Implementation consideration

Another limitation concerns implementation. This thesis provides formal protocol definitions and complexity analyses but lacks implementation and empirical benchmarking. Deploying TenderTEE in practice would require addressing challenges such as TEE initialization and key management, synchronization under realistic network conditions, integration with existing blockchain frameworks, and operational procedures for validator onboarding and recovery.

### Scalability

Scalability also remains an open challenge. TenderTEE significantly reduces complexity for small validator sets typical of consortium blockchains, but the underlying $O(n^2)$ message complexity remains a bottleneck for large public networks. Future work may investigate hierarchical committee structures, sharding approaches supported by TEEs, or randomized validator subsampling with TEE-based verifiable randomness.

## 7.4 Future Reasearch Directions

In the short term, the priority is implementation and benchmarking. Developing production-quality implementations of both TMC-Broadcast and TenderTEE, integrating them with frameworks such as the Cosmos SDK, and evaluating them against existing protocols like Tendermint, HotStuff, and PBFT would provide practical validation.
A systematic framework for transforming generic BFT protocols into TEE-enhanced versions could also be developed, providing protocol designers with rules, templates, and performance comparisons prior to implementation (see Recipe: Hardware-Accelerated Replication Protocols [156]). Comprehensive benchmarking suites should assess throughput, latency, energy usage, bandwidth, and TEE overhead in order to measure practical gains.

In the medium term, several research avenues emerge. One direction is to reduce TenderTEE to a single-phase protocol, exploiting stronger TEE guarantees to eliminate intermediate communication (cf. OneShot, which achieves consensus in one communication phase under normal conditions [157]). Another is to apply TEE to blockchain sharding (see [158] for a use of TEE to improve sharding), addressing challenges of cross-shard state verification and atomicity (e.g., TEE-based sharding methods in SoK: Public Blockchain Sharding [159]). Privacy-preserving protocols are also a promising direction, where TEEs combined with zero-knowledge proofs could enable confidential transactions compatible with regulation. Finally, mechanized verification of TenderTEE using theorem provers such as Coq or Isabelle would provide stronger assurance than hand-written proofs.

In the long term, we envision research on quantum-resistant consensus with TEE acceleration, interoperability protocols where TEEs act as trust anchors, adaptive consensus where TEE usage is dynamically adjusted to conditions, and sustainability standards where blockchain energy use is measured and verified through hardware-based accountability.

## 7.5 Broader Implications

This work contributes to resolving the enduring blockchain trilemma of decentralization, security, and efficiency. By introducing minimal trusted hardware assumptions, we demonstrate that regions of the design space previously proven unattainable in purely software-based models can in fact be reached. For the financial sector, the thesis outlines a concrete technical pathway toward blockchain infrastructures that meet institutional requirements for security, governance, and sustainability. More broadly, the methodology of strategic trusted component integration may inform distributed system design well beyond blockchains, in databases, multi-party computation, and decentralized infrastructures where Byzantine resilience is required but communication costs must be minimized.

Beyond the specific protocols developed in this work, some broader guidelines emerge regarding the characteristics that an efficient blockchain for the financial sector should ideally exhibit. These guidelines are not prescriptive, as the diversity of financial applications makes it unlikely that a single design will suit all use cases. Rather, they highlight recurring themes that appear throughout the analysis conducted in this thesis.

Energy efficiency should be a primary design goal. The transition from Proof-of-Work to Proof-of-Stake has already demonstrated that significant reductions in energy consumption are achievable. Building on this foundation, consensus protocols intended for finance should continue to reduce overhead through lower message complexity, smaller validator sets, and the reduction of redundant computation.

Deterministic finality and auditability are equally essential. Financial institutions cannot tolerate probabilistic settlement or ambiguity in transaction status. Protocols should therefore offer clear and irreversible finality, combined with cryptographic evidence of correct execution, ideally in a form compatible with audit and compliance frameworks.

Institutional adoption further depends on governance and separation of duties. Blockchains for finance should accommodate heterogeneous roles and enable arrangements such as tri-party structures, while ensuring accountability and preventing the concentration of power.

The strategic use of trusted components can provide efficiency gains and strengthen accountability, but these must be integrated minimally and transparently. The results of this thesis show that even limited use, such as a single Trusted Monotonic Counter at a broadcast initiator, can have substantial impact. However, trusted components should not be viewed as a universal solution but as one element in a broader toolkit.

Finally, efficiency must be considered in a wider ecosystem. Sharding, layer-two protocols, and optimistic execution represent orthogonal approaches that can complement trusted hardware integration. In practice, future financial blockchains will likely combine several of these techniques. Any such architecture must remain aligned with regulatory requirements and sustainability objectives, ensuring institutional viability.

Taken together, these considerations suggest that efficient blockchains for finance must carefully balance performance, governance, accountability, and sustainability, rather than optimizing for a single dimension.

## 7.6 Final Remarks

Integrating blockchain into the financial sector requires more than theoretical protocols: it demands systems aligned with institutional realities, regulatory frameworks, and sustainability goals. This thesis has argued that Trusted Execution Environments, despite their limitations, offer a practical mechanism to design consensus protocols that meet these multifaceted requirements. The road from theory to practice remains substantial, but the central insight holds: by judiciously leveraging minimal trusted components, it is possible to build blockchain systems that are more efficient, more resilient, and better adapted to financial applications than existing approaches. The future of blockchain in finance will likely involve hybrid architectures and heterogeneous trust models. The protocols presented here provide one piece of that broader puzzle and illustrate how trusted hardware can serve as a bridge between decentralization ideals and the pragmatic needs of regulated, energy-conscious financial infrastructure.

# Bibliography


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

[2] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. Bitcoin and cryptocurrency technologies. In *Bitcoin and Cryptocurrency Technologies*, pages 1–308. Princeton University Press, 2016.

[3] Philip Treleaven, Richard Brown, and Danny Yang. Blockchain technology in finance: The future of financial infrastructure. *Journal of Financial Transformation*, 45:114–123, 2017.

[4] Campbell R. Harvey, Ashwin Ramachandran, and Joseph Santoro. Defi and the future of finance. *Journal of Financial Economics*, 146(2):281–313, 2022.

[5] Dirk Andreas Zetzsche, Ross Buckley, and Douglas Arner. The distributed liability of distributed ledgers: Legal risks of blockchain. *University of Illinois Law Review*, 2018.

[6] Michel Rauchs, Apolline Blandin, Reuben Bear, and Patrick McKeon. Distributed ledger technology systems: A conceptual framework. Technical report, Cambridge Centre for Alternative Finance, 2018.

[7] Karl Wüst and Arthur Gervais. Do you need a blockchain? *2018 Crypto Valley Conference on Blockchain Technology*, 2018.

[8] Erik Brynjolfsson. The productivity paradox of information technology: Review and assessment. *Communications of the ACM*, 36(12):66–77, 1993.

[9] David Yermack. Corporate governance and blockchains. *Review of Finance*, 21(1):7–31, 2017.

[10] Société Générale. Société générale issues its first covered bond as a security token on a public blockchain, 2019.

[11] Ripple Labs. Ripple: Real-time cross-border payments, 2021. https://ripple.com/.

[12] Guénolé de Cadoudal. Ethereum pocr repository. https://github.com/ethereum-pocr, 2025. Accessed: 2025-09-17.

[13] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. In Elena Ferrari, Marco Baldi, and Roberto Baldoni, editors, *Proceedings of the Second Italian Conference on Cyber Security, Milan, Italy, February 6th - to - 9th, 2018*, volume 2058 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

[14] Philip Boucher. Blockchain for insurance: an overview of promising use cases and challenges. In *European Parliament Research Service (EPRS)*, 2017.

[15] Md Al Amin, Rushabh Shah, Hemanth Tummala, and Indrajit Ray. Utilizing blockchain and smart contracts for enhanced fraud prevention and minimization in health insurance through multi-signature claim processing, 2024.

[16] Etherisc. Etherisc. https://etherisc.com/, 2025. Accessed: 2025-09-29.

[17] Fabian Schär. Decentralized finance: On blockchain- and smart contract-based financial markets. *SSRN Electronic Journal*, 2020.

[18] Yan Chen and Cristiano Bellavitis. Decentralized finance: Blockchain technology and the quest for an open financial system. *Journal of Business Venturing Insights*, 2019.

[19] Robert Leonhard. Decentralized finance on the ethereum blockchain. *SSRN Electronic Journal*, 2019.

[20] Angela Walch. *Crypto Assets: Legal and Monetary Perspectives*. Oxford University Press, 2019.

[21] Uniswap Labs. Uniswap: Decentralized exchange protocol, 2021. https://uniswap.org/.

[22] MarkerDAO. Markerdao. https://makerdao.com/, 2025. Accessed: 2025-09-29.

[23] Aave. Aave: Open source liquidity protocol, 2021. https://aave.com/.

[24] Dirk A. Zetzsche, Douglas W. Arner, and Ross P. Buckley. Decentralized finance. *Journal of Financial Regulation*, 6(2):172–203, 2020.

[25] Muhammad Izhar Mehar, Charles Louis Shier, Alana Giambattista, Elgar Gong, Gabrielle Fletcher, Ryan Sanayhie, Henry M. Kim, and Marek Laskowski. Understanding a revolutionary and flawed grand experiment in blockchain, 2021.

[26] Abdulrahman Alhaidari, Balaji Palanisamy, and Prashant Krishnamurthy. Protecting defi platforms against non-price flash loan attacks, 2025.

[27] Primavera De Filippi and Samer Hassan. The legal framework of blockchain technology: Challenges and opportunities. *Journal of Cyber Policy*, 2020.

[28] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC '85)*, pages 291–304. ACM Press, 1985.

[29] Vitalik Buterin. An introduction to zk-snarks, 2018. https://vitalik.ca/general/2017/06/09/zeroknowledge.html.

[30] Financial Action Task Force. Guidance for a risk-based approach to virtual assets and virtual asset service providers, 2019. https://www.fatf-gafi.org/media/fatf/documents/recommendations/RBA-VA-VASPs.pdf.

[31] Basel Committee on Banking Supervision. Prudential treatment of cryptoasset exposures, 2021. https://www.bis.org/bcbs/publ/d519.htm.

[32] Yackolley Amoussou-Guenou, Lionel Beltrando, Maurice Herlihy, and Maria Potop-Butucaru. Byzantine reliable broadcast with one trusted monotonic counter. In Toshimitsu Masuzawa, Yoshiaki Katayama, Hirotsugu Kakugawa, Junya Nakamura, and Yonghwan Kim, editors, *Stabilization, Safety, and Security of Distributed Systems - 26th International Symposium, SSS 2024, Nagoya, Japan, October 20-22, 2024, Proceedings*, volume 14931 of *Lecture Notes in Computer Science*, pages 360–374. Springer, 2024.

[33] Lionel Beltrando, Maria Potop-Butucaru, and José Alfaro. Tendertee: Increasing the resilience of tendermint by using trusted environments. In *24th International Conference on Distributed Computing and Networking, ICDCN 2023, Kharagpur, India, January 4-7, 2023*, pages 90–99. ACM, 2023.

[34] Yackolley Amoussou-Guenou, Lionel Beltrando, Maurice Herlihy, and Maria Potop-Butucaru. Byzantine reliable broadcast and tendermint consensus with trusted components. *Theoretical Computer Science*, 1066, 2025. Part of special issue: Stabilization, Safety, and Security of Distributed Systems (SSS 2024).

[35] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[36] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. White paper, 2012. Accessed: 2025-09-08.

[37] Ethereum Foundation. Ethereum 2.0: The next evolution of ethereum blockchain, 2021. https://ethereum.org/en/eth2/.

[38] Daniel Larimer. Delegated proof-of-stake (dpos). White paper, 2014. Accessed: 2025-09-08.

[39] Rui Xu, Shijie Zhang, Ling Liu, and Yanzhi Dou. A survey of consensus protocols in blockchain-based distributed systems. *ACM Computing Surveys*, 54(6):1–38, 2021.

[40] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999.

[41] Gavin Wood. Proof-of-authority chains. Parity Technologies, Technical Report, 2017. Accessed: 2025-09-08.

[42] Jianliang Xie, Yulei Wu, Geoffrey C. Fox, Shiyong Lu, and Yiming Yang. A survey of proof-of-authority consensus mechanisms for blockchain. *Future Internet*, 11(12):210, 2019.

[43] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889, 2017. Cardano white paper (Ouroboros protocol).

[44] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0.8.13, 2018. White Paper.

[45] Anand Devarajan and Erkan Karabulut. Directed acyclic graph based blockchain systems. *arXiv preprint arXiv:2312.09816*, 2023.

[46] Huma Pervez, Muhammad Muneeb, and Muhammad Irfan. A comparative analysis of dag-based blockchain architectures. In *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*, pages 27–34. IEEE, 2018.

[47] Blockchain.com. Blockchain.com - charts - transaction rate per second. https://www.blockchain.com/explorer/charts/transactions-per-second, 2025. Accessed: 2025-09-25.

[48] etherscan. Etherscan daily transactions chart. https://etherscan.io/chart/tx, 2025. Accessed: 2025-09-25.

[49] IOTA Explorer. Iota explorer. https://explorer.iota.org/, 2025. Accessed: 2025-09-25.

[50] Serguei Popov. The tangle. Technical report, IOTA Foundation, 2018. Whitepaper.

[51] Colin LeMahieu. Raiblocks distributed ledger network. https://raiblocks.net/, 2014. Whitepaper.

[52] Fantom Foundation. Fantom whitepaper v1.6. Technical report, Fantom Foundation, 2018. Whitepaper.

[53] Ethan Heilman and Graeme Percival. Cryptanalysis of curl-p and other attacks on the iota cryptocurrency. *IACR Cryptology ePrint Archive*, 2019:344, 2019.

[54] Vitalik Buterin. Ethereum whitepaper.

[55] Sergey Nazarov, Steve Ellis, and Ari Juels. Chainlink: A decentralized oracle network. White paper, 2017. Accessed: 2025-09-08.

[56] Soravis Srinawakoon, Paul Nattapatsiri, and Sorawit Suriyakarn. Band protocol: Decentralized data governance. White paper, 2020. Accessed: 2025-09-08.

[57] Jan Egberts. The oracle problem — an analysis of how smart contracts can obtain trustworthy data. Master's thesis, University of Amsterdam, 2017. Master's Thesis.

[58] Rafael Belchior, Andre Vasconcelos, Sergio Guerreiro, and Miguel Correia. Blockchain interoperability: State-of-the-art and open challenges. *Future Generation Computer Systems*, 117:529–547, 2021.

[59] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, and George Danezis. Sok: Communication across distributed ledgers. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 509–526. IEEE, 2021.

[60] Léonard Lys, Arthur Micoulet, and Maria Potop-Butucaru. R-swap: Relay based atomic cross-chain swap protocol. In *Algorithmic Aspects of Cloud Computing (ALGOCLOUD 2021)*, volume 12709 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2021.

[61] Polygon. Polygon portal. https://portal.polygon.technology/, 2025. Accessed: 2025-10-02.

[62] Sergey Gorbunov, Dahlia Malkhi, Kevin Gailly, and Abhi Shelat. Axelar: A decentralized interoperability network, 2021.

[63] Cosmos IBC Working Group. The inter-blockchain communication protocol: An overview. https://ibcprotocol.org, 2020. Accessed: 2025-07-25.

[64] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. White paper, 2016. Accessed: 2025-09-08.

[65] Ulrich Gallersdörfer, Lena Klaaßen, and Christian Stoll. Energy consumption of blockchain networks. *Nature Sustainability*, 3(11):870–877, 2020.

[66] Alex De Vries and Christian Stoll. Energy consumption of blockchain: A critical review. *Joule*, 6(6):1121–1138, 2022.

[67] Max J. Krause and Thabet Tolaymat. Quantification of energy and carbon costs for blockchain security. *Environmental Research Letters*, 17(8):084013, 2022.

[68] Johannes Sedlmeir, Hans Ulrich Buhl, Gilbert Fridgen, and Robert Keller. The energy consumption of blockchain technology: Beyond myth. *Business & Information Systems Engineering*, 62(6):599–608, 2020.

[69] NEAR Protocol. Near protocol. https://www.near.org/, 2025. Accessed: 2025-09-25.

[70] MultiversX. Multiversx. https://multiversx.com/, 2025. Accessed: 2025-09-25.

[71] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948. ACM, 2018.

[72] Vitalik Buterin. An incomplete guide to rollups. https://vitalik.eth.limo/general/2021/01/05/rollup.html, 2021. Accessed April 2025.

[73] Solana. Solana. https://solana.com/fr, 2025. Accessed: 2025-09-25.

[74] Aptos Foundation. Aptos foundation. https://aptosfoundation.org/, 2025. Accessed: 2025-09-25.

[75] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 1976.

[76] Ralph C. Merkle. A digital signature based on a conventional encryption function. *Lecture Notes in Computer Science*, 1987.

[77] J. H. Ellis. The possibility of secure non-secret digital encryption. *COMMUNICATIONS-ELECTRONICS SECURITY GROUP*, 1970.

[78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of The ACM*, 1978.

[79] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 417–426, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[80] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[81] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[82] National Institute of Standards and Technology. FIPS PUB 180-2: Secure Hash Standard (SHS). Technical report, U.S. Department of Commerce, August 2001.

[83] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013.

[84] National Institute of Standards, Technology (NIST), and Morris J. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 00:08:00 2015.

[85] Andreas M. Antonopoulos. Mastering bitcoin: Programming the open blockchain, 2017.

[86] Gavin Andresen. Bip 16: Pay to script hash. https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki. Accessed: 2025-09-17.

[87] Jonas Nick Pieter Wuille and Tim Ruffing. Bip 340: Schnorr signatures for secp256k1. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki. Accessed: 2025-09-17.

[88] Anthony Towns Pieter Wuille, Jonas Nick. Bip 341: Taproot: Segwit version 1 spending rules. https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki. Accessed: 2025-09-17.

[89] Anthony Towns Pieter Wuille, Jonas Nick. Bip 342: Validation of taproot scripts. https://github.com/bitcoin/bips/blob/master/bip-0342.mediawiki. Accessed: 2025-09-17.

[90] Cambridge Centre for Alternative Finance (CCAF). Cambridge blockchain network sustainability index: Cbeci. https://ccaf.io/cbnsi/cbeci, 2025. Accessed: 2025-03.

[91] Digiconomist. Bitcoin energy consumption index. https://digiconomist.net/bitcoin-energy-consumption, 2025. Accessed: 2025-06.

[92] Alex de Vries and Christian Stoll. Bitcoin's growing water footprint. *Cell Reports Sustainability*, 1(1):100005, 2023.

[93] Nick Szabo. The idea of smart contracts. *Extropy: Journal of Transhumanist Thought*, 1997. Accessed: 2025-09-08.

[94] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Technical report, Ethereum Project Yellow Paper, 2014.

[95] Vyper Community. Vyper: Pythonic smart contract language for the ethereum virtual machine, 2025. Accessed: 2025-09-09.

[96] Ethereum Foundation. The merge. https://ethereum.org/en/roadmap/merge/, 2022. Accessed: 2025-09-15.

[97] Ethereum Foundation. Execution-consensus engine api specification. https://github.com/ethereum/execution-apis, 2025. Accessed: 2025-09-15.

[98] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. Technical report, Ethereum Foundation, 2017. arXiv:1710.09437.

[99] Joachim Neu, Ertem Nusret Tas, Ersin Kalayci, and David Tse. Gasper: Combining ghost and casper for accountable finality. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1693–1707, 2020.

[100] Vitalik Buterin. Randao seeds and committee generation. https://notes.ethereum.org/@vbuterin/SkeyEI3xv#Aside-RANDAO-seeds-and-committee-generation, 2019. Accessed: 2025-09-15.

[101] Ethereum Foundation. Altair: Beacon chain upgrade (sync committees). Ethereum Foundation Blog, 2021. Accessed: 2025-09-15.

[102] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Ethereum proof-of-stake and the probabilistic bouncing attack. *Distrib. Ledger Technol.*, January 2025. Just Accepted.

[103] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Byzantine attacks exploiting penalties in ethereum pos. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 53–65, 2024.

[104] Yackolley Amoussou-Guenou. *Gouvernance des biens communs dans les blockchains*. PhD thesis, Sorbonne Université, 2020. Thèse de doctorat dirigée par Maria Potop-Butucaru, Sorbonne Université.

[105] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[106] Danny Dolev, Cynthia Dwork, and Larry J. Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, 1987.

[107] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distributed Syst.*, 27(5):387–408, 2012.

[108] Roberto Baldoni, Marin Bertier, Michel Raynal, and Sara Tucci-Piergiovanni. Looking for a definition of dynamic distributed systems. In *Parallel Computing Technologies, 9th International Conference, PaCT 2007, Pereslavl-Zalessky, Russia, September 3-7, 2007, Proceedings*, volume 4671 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2007.

[109] Marcos Kawazoe Aguilera. A pleasant stroll through the land of infinitely many creatures. *SIGACT News*, 35(2):36–59, 2004.

[110] Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018.

[111] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[112] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[113] Francesc D. Muñoz-Escoí and Rubén de Juan-Marín. On synchrony in dynamic distributed systems. *Open Comput. Sci.*, 8(1):154–164, 2018.

[114] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

[115] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.

[116] Danny Dolev. Unanimity in an unknown and unreliable environment. In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 159–168. IEEE Computer Society, 1981.

[117] Alexandre Maurer, Sébastien Tixeuil, and Xavier Défago. Communicating reliably in multihop dynamic networks despite byzantine failures. In *34th IEEE Symposium on Reliable Distributed Systems, SRDS 2015, Montreal, QC, Canada, September 28 - October 1, 2015*, pages 238–245. IEEE Computer Society, 2015.

[118] Jovan Komatovic, Rachid Guerraoui, Petr Kuznetsov, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, and Andrei Tonkikh. Dynamic Byzantine Reliable Broadcast. In *24th International Conference on Principles of Distributed Systems, OPODIS 2020, December 14-16, Strasbourg, Online, France*, 2020.

[119] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 524–541, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[120] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. (leader/randomization/signature)-free byzantine consensus for consortium blockchains. *CoRR*, abs/1702.03068, 2017.

[121] Amotz Bar-Noy, Xiaotie Deng, Juan A. Garay, and Tiko Kameda. Optimal amortized distributed consensus. *Inf. Comput.*, 120(1):93–100, 1995.

[122] Shlomi Dolev and Sergio Rajsbaum. Stability of long-lived consensus. *J. Comput. Syst. Sci.*, 67(1):26–45, 2003.

[123] Carole Delporte-Gallet, Stéphane Devismes, Hugues Fauconnier, Franck Petit, and Sam Toueg. With finite memory consensus is easier than reliable broadcast. In *Principles of Distributed Systems, 12th International Conference, OPODIS 2008, Luxor, Egypt, December 15-18, 2008. Proceedings*, volume 5401 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2008.

[124] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983, Colony Square Hotel, Atlanta, Georgia, USA*, pages 1–7. ACM, 1983.

[125] Sérgio Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys*, 51(6):1–36, 2019.

[126] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.

[127] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 189–204. ACM, 2007.

[128] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA*, pages 1–14. USENIX Association, 2009.

[129] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Veríssimo. Efficient byzantine fault-tolerance. *IEEE Trans. Computers*, 62(1):16–30, 2013.

[130] Malte Borcherding. Levels of authentication in distributed agreement. In *Distributed Algorithms,10th International Workshop, WDAG '96, Bologna, Italy, October 9-11, 1996*, pages 40–55, 1996.

[131] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. How to tolerate half less one byzantine nodes in practical distributed systems. In *23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil*, pages 174–183. IEEE Computer Society, 2004.

[132] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. BFT-TO: intrusion tolerance with less replicas. *Comput. J.*, 56(6):693–715, 2013.

[133] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. Cheapbft: resource-efficient byzantine fault tolerance. In *European Conference on Computer Systems, Proceedings of the Seventh EuroSys Conference 2012, EuroSys '12, Bern, Switzerland, April 10-13, 2012*, pages 295–308. ACM, 2012.

[134] Tobias Distler, Christian Cachin, and Rüdiger Kapitza. Resource-efficient byzantine fault tolerance. *IEEE Trans. Computers*, 65(9):2807–2819, 2016.

[135] Sravya Yandamuri, Ittai Abraham, Kartik Nayak, and Michael K. Reiter. Communication-efficient BFT protocols using small trusted hardware to tolerate minority corruption. *IACR Cryptol. ePrint Arch.*, page 184, 2021.

[136] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. DAMYSUS: streamlined BFT consensus leveraging trusted components. In *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 1–16. ACM, 2022.

[137] Tianwei Chen, Zhiqiang Chen, Baolei Zang, and Haibo Zhang. Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *Proceedings of the Asia-Pacific Workshop on Systems (APSys)*. ACM, 2016. SGX microbenchmarking results presented under S̈GXbenchc̈ontext.

[138] Peter Yuhala, Céline Göttel, Julien Ménétrey, Valerio Schiavoni, Daniel Kozhaya, and Pascal Felber. On real-time guarantees in intel sgx and tdx. In *ECRTS 2025: 37th Euromicro Conference on Real-Time Systems*, 2025.

[139] Naiwei Liu, Meng Yu, Wanyu Zang, and Ravi Sandhu. Cost and effectiveness of trustzone defense and side channel attacks on arm platform. *Journal of Wireless Communications and Networking*, 11(4), 2022.

[140] Ramin Bahmani, Matthias Fischer, and Stefan Egger. Sev-watt: Benchmarking the performance impact of amd sev. In *Proceedings of the 17th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, pages 101–114, 2021.

[141] Dayeol Lee, Jaebaek Ko, Sangho Kim, Heejin Kim, Taesoo Kim, Haesoo Lee, and Dawn Xiaodong Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the 15th European Conference on Computer Systems (EuroSys)*, pages 1–16, 2020.

[142] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. On the (limited) power of non-equivocation. In *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 301–308. ACM, 2012.

[143] Michel Raynal. On the versatility of bracha's byzantine reliable broadcast algorithm. *Parallel Process. Lett.*, 31(3):2150006:1–2150006:9, 2021.

[144] Jae Kwon. Tendermint: Consensus without mining. https://tendermint.com/static/docs/tendermint.pdf, 2014. [Online; accessed 2020 July 10].

[145] Yackolley Amoussou-Guenou, Lionel Beltrando, Maurice Herlihy, and Maria Potop-Butucaru. Byzantine reliable broadcast and tendermint consensus with trusted components. Cryptology ePrint Archive, Paper 2025/622, 2025.

[146] Yackolley Amoussou-Guenou, Antonella Del Pozzo, Maria Potop-Butucaru, and Sara Tucci Piergiovanni. Dissecting tendermint. In *Networked Systems - 7th International Conference, NETYS 2019, Marrakech, Morocco, June 19-21, 2019, Revised Selected Papers*, volume 11704 of *Lecture Notes in Computer Science*, pages 166–182. Springer, 2019.

[147] Muhammad Khan et al. A comparative analysis of proof-of-authority consensus algorithms: Aura vs clique. Preprint, 2022. Accessed: 2025-09-10.

[148] Xiaorui Zhang, Zekun Li, Yuntian Feng, Tianxiang Sun, Zhiheng Xi, and Yang Liu. Breaking fairness against proof of authority aura. *Proceedings of the ACM on Management of Data*, 2023.

[149] Henrique Moniz, Nuno Neves, and Paulo Verissimo. The istanbul bft consensus algorithm. *arXiv preprint arXiv:2002.03613*, 2020.

[150] Christian Cachin et al. Vivisecting the dissection: On the role of trusted components in bft protocols. *arXiv preprint arXiv:2312.05714*, 2023.

[151] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against intel sgx. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 1466–1482. IEEE, 2020.

[152] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. Clkscrew: Exposing the perils of security-oblivious energy management. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1057–1074. USENIX Association, 2017.

[153] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*, pages 361–372. IEEE, 2014.

[154] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 1–19, 2019.

[155] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008. USENIX Association, 2018.

[156] Dimitra Giantsidi, Emmanouil Giortamis, Julian Pritzi, Maurice Bailleu, Manos Kapritsos, and Pramod Bhatotia. Recipe: Hardware-accelerated replication protocols, 2025.

[157] J. Decouchant et al. Oneshot: View-adapting streamlined bft protocols with tees. In *Proceedings of IPDPS 2024*, 2024. Uses TEEs to achieve one-phase consensus under normal conditions.

[158] Hung Dang, Tien Tuan Anh Dinh, Ee-Chien Chang, and Beng Chin Ooi. Chain of trust: Can trusted hardware help scaling blockchains? *CoRR*, abs/1804.00399, 2018.

[159] Md Mohaimin Al Barat, Shaoyu Li, Changlai Du, Y. Thomas Hou, and Wenjing Lou. Sok: Public blockchain sharding. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, page 766–783. IEEE, May 2024.