TUM School of Computation, Information and Technology
Technische Universität München

Secure Decentralization of Cyber-Physical Systems for an Internet of Things without Clouds

Emanuel G. Regnath

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:

Prof. Dr.-Ing. Georg Sigl

Prüfende der Dissertation:

1. Prof. Dr. Sebastian Steinhorst
2. Prof. Dr. Marc-Oliver Pahl

Die Dissertation wurde am 12.04.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 07.11.2022 angenommen.

Colophon

This Dissertation was typeset using the LuaLaTeX typesetting system. LuaLaTeX is based on LaTeX, which is based on TeX originally created by Donald Knuth.

Typographical decisions follow some of the recommendations given in *The Elements of Typographic Style* by Robert Bringhurst.

Colors were chosen from TUM's corporate design guide.

Font families used in this document are:

Roman:	Libertinus Serif
Sans-Serif:	Libertinus Sans, Helvetica
Monospace:	`Inconsolata`
Math:	*Latin Modern Math*

The list of literature references was managed by BibLaTeX and citations are formatted according to alphabetic style.

Abstract

The fast development of silicon-based microprocessors and radio technology has enabled ubiquitous computation and connectivity among everyday objects. With an estimated number of 26 billion Internet-connected devices in 2022, the idea of an Internet of Things (IoT) has already become reality and is rapidly growing. By default, most IoT services that are being implemented today (e.g. smart home lighting) or are planned for the future (e.g. intelligent traffic management) rely on centralized cloud servers to process application data.

While such centralized architectures are well-understood and easy to implement, they have severe drawbacks regarding *scalability*, *efficiency* and *robustness*. First, the resources and bandwidth of the central instance need to scale simultaneously with the number of devices/requests to prevent increased processing delays. Second, central instances are often over-dimensioned to handle high-load periods, leading to less efficient use of resources. Furthermore, since all interactions are coordinated centrally, local interactions with shorter routes and less involved devices remain unused. Third, the central instance represents a single point of failure that, if attacked or malfunctioning, could lead to a collapse of the entire network and its services, which is an unacceptable risk on a global scale.

These three problems are mitigated in decentralized architectures where devices use Peer to Peer (P2P) connections to directly interact with each other on demand without relying on a central instance. Here, the available resources scale naturally with the number of devices and failures only affect individual connections, such that the rest of the network remains operational. Decentralization seems to be especially promising for distributed Cyber-Physical Systems (CPSs), which often interact with their physical environment locally and therefore do not require a global controller.

For example, Connected Autonomous Vehicles (CAVs) should communicate directly with their neighboring vehicles instead of relying on a central coordinator because a direct P2P communication does not require additional infrastructure, scales with the number of present vehicles, offers low latency, and keeps failures locally bounded.

However, decentralized architectures require a more sophisticated coordination of devices because they cannot easily access a global view on the current system state, which emerges naturally within a central controller that coordinates all state changes. Instead, devices need to exchange and propagate their local information in order to deduce parts of the global state. Any action that affects the global state needs to be synchronized among all involved devices. For example, vehicles performing an intelligent intersection scheduling need to know where all other approaching vehicles want to go in order to avoid collisions. Furthermore, devices that do not directly participate in state-changing processes might still need to retrieve and verify the state later from their peers, e.g. for monitoring, statistical analysis, or future state changes that depend on the previous one. While P2P communication increases the robustness of the network, it also increases communication complexity. In combination with the resource-constraints of many smart devices, such as low-cost sensors that operate only on an embedded micro-processor, we also need a lightweight authentication mechanism to ensure security despite a higher number of potentially untrusted communication partners.

In order to enable secure decentralization, this dissertation proposes new methods and system-level designs that can handle the increased complexity due to a missing central system state by solving the aforementioned core challenges: 1. the synchronization of state changes, 2. the certification and verification of the current state, and 3. the authentication of devices that manipulate the state.

Using CAVs as a representative IoT scenario involving physical systems, we propose two novel communication protocols to synchronize state information among vehicles using distributed consensus. One protocol coordinates driving maneuvers for platoons, the other enables vehicles to agree on a common schedule for crossing road intersections.

We then look into *blockchain* as a mechanism to certify state information and propose a resource-efficient method to enable even highly-constrained embedded devices to verify such information using just a few kilobyte of data. Furthermore, by verifying the timestamps stored in a blockchain, we have built a secure time synchronization scheme with an accuracy of one second that requires only a few hash operations. We also explore *smart contracts* as a mechanism to certify program execution. Here, we propose a new language to enable human verification of contract code by limiting confusing language constructs.

Finally, we improve *hash-based signatures* for lightweight message and device authentication by presenting two solutions that reduce their signature size without lowering security.

We were able to show that our proposed decentralized solutions can compete with or outperform their centralized counterparts in terms of performance, while at the same time our solutions are more lightweight, efficient, and require less infrastructure. We have also observed that cryptographic hash functions can be utilized as a versatile building block to provide security in several layers of the communication architecture. They are lightweight, quantum-resistant, and often hardware accelerated even on low-cost micro-processors.

We conclude that cloud-based services are still suitable for certain tasks that are computational intensive, but many IoT domains, such as intelligent traffic management, smart grids, and digital marketplaces, should be built on a decentralized system architecture. With the solutions developed in this dissertation, we further support the thesis that decentralization leads to better scalability and robustness for the Internet of Things.

Zusammenfassung (German Abstract)

Die rasante Entwicklung der letzten Jahrzehnte in der Chip-Herstellung führte zu immer effizienteren und günstigeren Mikroprozessoren, die über drahtlose Funktechnologie mit dem Internet vebunden werden können. Diese allgegenwärtige Konnektivität und Rechenleistung wird seitdem vermehrt in alltäglichen Geräten verbaut, um sie intelligenter zu machen.

Mit geschätzten 26 Milliarden Geräten, die seit 2022 mit dem Internet verbunden sind, ist das „Internet der Dinge“ (*Internet of Things* – IoT) bereits Realität geworden und wächst stetig weiter. Die meisten IoT Dienste, die derzeit eingesetzt werden (z.B. intelligente Gebäudebeleuchtung) oder sich in Planung befinden (z.B. automatisierter Straßenverkehr) setzen dabei standardmäßig auf eine zentrale Serverarchitektur um Anwendungsdaten zu verarbeiten.

Solche zentralisierten Architekturen sind zwar weit verbreitet und einfach zu implementieren, haben aber schwerwiegende Nachteile im Bezug auf Skalierbarkeit, Effizienz und Robustheit. Erstens müssen die Ressourcen und die Bandbreite des zentralen Servers gleichzeitig mit der Anzahl der Geräte wachsen, um Verzögerungen bei der Verarbeitung zu vermeiden. Zweitens sind die zentralen Instanzen oft überdimensioniert, um Hochlastzeiten zu bewältigen, was jedoch zu einer ineffizienten Nutzung der Ressourcen führt. Da außerdem alle Interaktionen zentral koordiniert werden, bleiben mögliche lokale Interaktionen mit kürzeren Wegen oft ungenutzt. Und drittens stellt die zentrale Instanz einen *Single Point of Failure* dar, der bei einem Angriff oder einer Fehlfunktion zu einem Zusammenbruch des gesamten Netzwerks und seiner Dienste führen könnte, was auf globaler Ebene ein inakzeptables Risiko darstellt.

Diese drei Probleme werden in dezentralen Architekturen entschärft, in denen Geräte über *Peer-to-Peer*-Verbindungen direkt miteinander interagieren. Hier skalieren die benötigten Ressourcen automatisch mit der Anzahl der verfügbaren Geräte und Ausfälle betreffen nur einzelne Verbindungen, so dass der Rest des Netzwerks funktionsfähig bleibt.

Die Dezentralisierung scheint daher besonders vielversprechend für verteilte Cyber-Physische Systeme (CPS) zu sein, die oft lokal mit ihrer Umgebung interagieren und daher keinen globalen Controller benötigen. Zum Beispiel sollten vernetzte autonome Fahrzeuge (Connected Autonomous Vehicles – CAVs) direkt mit ihren Nachbarfahrzeugen kommunizieren, anstatt jederzeit von einem weit entfernten, zentralen Cloud-Koordinator abhängig zu sein. Somit würde keine zusätzliche Infrastruktur benötigt werden, eine kurze Übertragunszeit wäre gewährleistet, und Ausfälle blieben lokal begrenzt.

Dezentralisierte Architekturen erfordern jedoch eine ausgefeiltere Koordination der Geräte, da sie nicht ohne Weiteres auf eine globale Ansicht des aktuellen Systemzustands zugreifen können, welche sich auf natürliche Weise in einem zentralen Controller ergibt, der alle Zustandsänderungen koordiniert. Stattdessen müssen die Geräte untereinander ihren lokalen Informationsstand

austauschen, um davon einen globalen Zustand abzuleiten. Dies erfordert auch, dass jede Aktion, die sich auf den globalen Zustand auswirkt, zwischen allen beteiligten Geräten synchronisiert wird. So müssen beispielsweise Fahrzeuge, die untereinander die Vorfahrt an einer Kreuzung aushandeln, wissen, wohin alle anderen ankommende Fahrzeuge fahren wollen, um Kollisionen zu vermeiden.

Die P2P-Kommunikation erhöht zwar die Robustheit des Netzwerks, erhöht aber auch die Komplexität der Kommunikation. In Kombination mit den Ressourcenbeschränkungen vieler intelligenter Geräte, wie z.B. preiswerter Sensoren, die nur mit einem schwachen Mikroprozessor arbeiten, benötigen wir auch eine effiziente Authentifizierungsmethode. Diese muss Kommunikationssicherheit trotz einer höheren Anzahl von potenziell nicht vertrauenswürdigen Kommunikationspartnern gewährleisten.

Um eine sichere Dezentralisierung zu ermöglichen, erarbeiten wir in dieser Dissertation neue Methoden und Entwürfe auf Systemebene, mit dem Ziel, die erhöhte Komplexität aufgrund eines fehlenden zentralen Systemzustands zu bewältigen, indem wir die oben genannten Herausforderungen lösen: 1. die Synchronisierung von Zustandsänderungen, 2. die Zertifizierung und Verifizierung des aktuellen Systemzustands und 3. die Authentifizierung von Geräten, die den Zustand manipulieren.

Anhand von autonomen Fahrzeugen als repräsentatives IoT-Szenario mit physischen Systemen entwerfen wir zwei neuartige Kommunikationsprotokolle zur dezentralen Synchronisierung mithilfe von Konsensus-Algorithmen. Ein Protokoll koordiniert Fahrmanöver von Fahrzeugkolonnen, das andere ermöglicht es Fahrzeugen sich auf einen gemeinsamen Zeitplan für das Überqueren von Kreuzungen zu einigen.

Anschließend untersuchen wir *Blockchain* als einen Mechanismus zur Zertifizierung von Zustandsinformationen und schlagen eine ressourceneffiziente Methode vor, mit der selbst stark eingeschränkte Geräte diese Informationen mit nur wenigen Kilobyte verifizieren können. Zum Beispiel kann dadurch die Gültigkeit von Zeitstempeln, die in einer Blockchain gespeichert sind, überprüft werden um eine sichere Zeitsynchronisation mit einer Sekunde Genauigkeit zu erreichen. Zuletzt schlagen wir noch zwei Verbesserungen für Hash-basierte Signaturen vor, um eine effiziente Authentifizierung von Nachrichten zu ermöglichen.

Insgesamt konnten wir zeigen, dass die von uns vorgeschlagenen Lösungen fehlertoleranter und effizienter sind als zentralisierte Ansätze. Wir haben außerdem festgestellt, dass kryptografische Hash-Funktionen als vielseitiger Baustein genutzt werden können, um die Sicherheit der Kommunikationsarchitektur zu gewährleisten. Während die Cloud immer noch für bestimmte rechenintensive Aufgaben geeignet ist, sollten Anwendungen, wie intelligentes Verkehrsmanagement, intelligente Stromnetze und digitale Marktplätze, auf einer dezentralen Systemarchitektur aufgebaut werden. Unsere Ergebnisse stützen die These, dass Dezentralisierung zu einer besseren Skalierbarkeit und Robustheit im IoT Bereich führt.

Contents

Acronyms

AES	Advanced Encryption Standard.	117
AMSA	Adaptive Merkle Signature Architecture.	128, 130–136
API	Application Programming Interface.	148, 156
AWS	Amazon Web Services.	7
BFT	Byzantine Fault Tolerant.	24, 27, 29
CAV	Connected Autonomous Vehicle.	III–V, 4, 16, 34, 51, 54, 55, 60, 62, 139
CISCAV	Consensus-based Intersection Scheduling for Connected Autonomous Vehicles.	49, 53–56, 60–63
CPS	Cyber-Physical System.	III, V, 17, 19, 22, 23, 29, 32, 34–37, 40, 48, 65, 103, 105, 111, 114, 139, 140, 142, 149
CUBA	Chained Unanimous Byzantine Agreement.	40, 44–48, 63
DAG	Directed Acyclic Graph.	70, 71
DAO	Decentralized Autonomous Organization.	1, 137, 138, 140, 145
DLT	Distributed Ledger Technology.	68–71, 105, 144, 156, 157
DSL	Domain-Specific Language.	109, 110, 113, 114
ECC	Elliptic-Curve Cryptography.	15, 118, 134, 136
ECDSA	Elliptic Curve Digital Signature Algorithm.	120, 121, 133, 135, 136, 140, 145
FSM	Finite State Machine.	18, 22
HBS	Hash-Based Signature.	122, 126, 133–136, 145
IaaS	Infrastructure as a Service.	7
IoT	Internet of Things.	III, IV, VI, 1–3, 5–10, 13, 15–20, 32, 34, 66, 72, 88, 89, 99–101, 105, 120, 122, 131, 132, 137–139, 142, 143, 146–149
ITS	Intelligent Transportation System.	56, 139, 142, 147, 149
LAN	Local Area Network.	7
M2M	Machine to Machine.	3
MAC	Massage Authentication Code.	28, 30, 33, 116, 117, 141, 145
MTS	Many-Time Signature.	122, 123, 125, 128
NIST	National Institute of Standards and Technology.	121, 136
NSA	National Security Agency.	3
NTP	Network Time Protocol.	11, 12, 89, 93, 98, 100
OTS	One-Time Signature.	122–124, 130, 132

P2P	Peer to Peer.	III, IV, VI, 7, 11, 15, 27
PBFT	Practical Byzantine Fault Tolerance.	28–32, 54, 69
PKC	Public Key Cryptography.	15, 28, 33, 117, 118, 120, 121, 140, 141
PoS	Proof-of-Stake.	69, 70, 97, 104
PoW	Proof-of-Work.	9, 32, 69, 70, 76–78, 80, 82–89, 93, 94, 104, 105, 140, 144
PRNG	Pseudo Random Number Generator.	117
RAM	Random Access Memory.	72, 121, 133
RFID	Radio-Frequency Identification.	4, 141
RSA	Rivest–Shamir–Adleman.	15, 118, 120, 121, 135
RSU	Road-Side Unit.	50, 51
SMR	State Machine Replication.	27–29, 31, 154
SPV	Simplified Payment Verification.	84, 86
TCP	Transmission Control Protocol.	24
TLS	Transport Layer Security.	65, 119, 120
UTC	Coordinated Universal Time.	89, 90
UTXO	Unspent Transaction Output.	66, 68, 156, 157
VANET	Vehicular Ad-hoc Network.	17, 46, 49, 51, 142
WOTS	Winternitz One-Time Signature.	123–133, 135, 136

Chapter 1

Introduction

Contents

The "Internet of Things" has become a ubiquitous buzzword in the tech-industry and the claims made by its supporters about its potential impact on industry and society are quickly exceeding the threshold for sounding more like science fiction than current technological development.

Fridges that automatically order missing groceries, wearables that monitor health to notify about anomalies, and self-driving cars that avoid congestions are some applications that are already implemented and are currently tested in small projects – but we have yet to see when and to which extent these ideas will reach mass adoption.

Currently, the research in this field focuses on efficient and secure automation of processes with the vision of creating Decentralized Autonomous Organizations (DAOs), which consist of smart devices that work together seamlessly in the background to serve us humans.

1.1 The Internet of Things – Beyond the Buzzword

Internet of Things (IoT) in general refers to a paradigm in which every-day objects become smart and self-organized by incorporating sensing, processing, and communication ca-

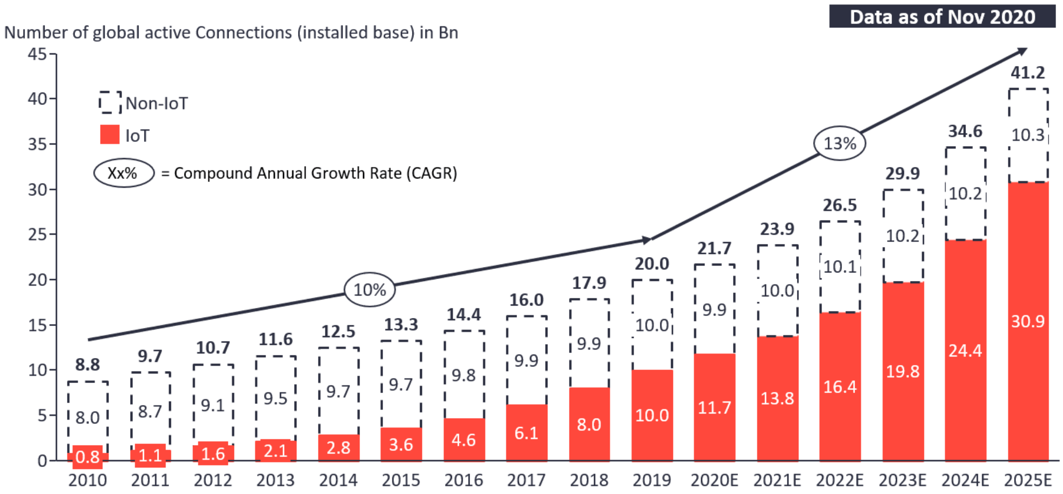

Figure 1: Annual growth of Internet connections. Numbers given in billions. Non-IoT includes phones, tablets, PCs, and laptops. IoT includes all consumer and B2B connections without human interaction. Taken with kind permission from *IoT-Analytics* [Lue20a].

pabilities via embedded processors that can connect to the Internet [Al-+15; ITU16]. A larger system composed of these smart devices will allow us humans to accurately monitor and also automate processes by simply specifying a desired behavior instead of manually triggering processes every time we need them.

However, a growing number of smart devices also increases the complexity of information exchange and requires new approaches to design and build systems on top of the existing Internet that was never intended for such a scenario.

In order to better understand the challenges that lie ahead, we first give an overview of the history, the current applications, and predicted development of IoT platforms.

1.1.1 Towards an Automated Internet

The Internet is growing rapidly. The Internet started in 1969 in form of the military *Advanced Research Projects Agency Network* (ARPANET) connecting four computers at different geographical locations. In June 1977, the ARPANET – now including companies and research institutions – connected 113 computers over 57 gateways [Pay78] and reached a growth rate of one computer per 20 days. Fast forward to 2020, there are now 21 billion Internet-connected devices[1] [Sta21] and 127 new devices are added to the Internet *every second* [McK18].

While the original ARPANET was decomissioned in 1990, its initial idea about fragmented message packets that are sent over multiple routes, is the foundation of Internet traffic, which paved the way for the IoT.

[1] Measured as individual connection.

IoT overtakes conventional Internet. As shown in Figure 1, already in 2019, half of all connected devices were IoT devices that establish Machine to Machine (M2M) connections without human interaction [Lue20a; Sta21]. However, another report from Cisco, estimates that the share of M2M connections will reach 50% only by 2023 [Cis20]. Among these M2M connections, the largest share of 48% belongs to smart home applications, such as smart lighting [Cis20]. Depending on the source, either smart factories or connected vehicles are the second largest and currently fastest growing application domain [Cis20; Lue20b].
Estimates of worldwide financial investment in IoT projects in 2020 ranges from $128.9 billion on enterprise solutions [Weg21] to a general spending of $749 billion [Vai21]. However, predictions agree on an *annual growth rate* of roughly 15 – 25% over the next five years.
In general, past predictions about the number of devices have been highly over-optimistic. In 2011, CISCO predicted 50 billion IoT devices by 2020 [Eva11], while in 2012 IBM made a very bold prediction of 1 trillion connected devices by the year 2015 [IBM12]. The wide range of these numbers illustrate the high expectations that the industry projects into the IoT paradigm and the high dynamics of change and innovation in this sector that make the predictions difficult.

IoT has risks, but also many benefits. Several movies, books and conspiracy theories have painted a dark future in which humanity is threatened by a globally connected and intelligent network of machines and devices. Either because the network of devices can be hacked and misused by a few powerful individuals to support only their own interests (e.g. *Orwell's 1984*) or because our high dependency on it can spark a violent chaos in case the network collapses (e.g. *Elsberg's Blackout*) or because the entire network can turn into a self-aware and malicious super-intelligence (e.g. *SkyNet* from *Terminator*). For the first scenario, there have already been several incidents ranging from individual hacked cars to the general surveillance practices of the National Security Agency (NSA) uncovered by Edward Snowden in 2013 [MD13], which would not have been possible to this extent without the increased automation and connectivity of smart devices.
However, if we look into the majority of existing IoT applications, we see many benefits as automated processes are more resource-efficient and more robust than manual processes.

1.1.2 Domains and Applications of IoT

The extend to which this new paradigm transforms conventional control systems to smart autonomous systems is illustrated by the following short list of applications domains together with concrete examples of use cases in which IoT-technology has been applied. The application shares of enterprise projects is shown in Figure 2.

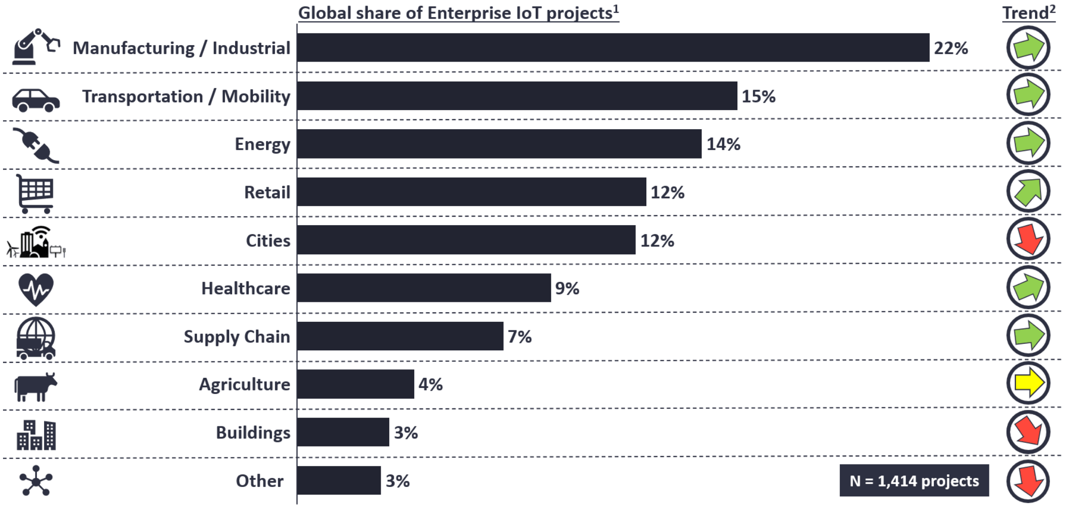

Figure 2: Application shares of 1414 enterprise IoT projects in 2020. ¹Enterprise projects do not include consumer IoT, such as smart-home automation or wearables. ²Trend shows a comparison against shares in 2018. Taken with kind permission from *IoT-Analytics* [Lue20b].

- **Intelligent Transportation Systems (ITS):** Using CAVs could improve the efficiency, availability and safety of traffic on roads, on water or in the air [Al-+15].

  In 2018, a pilot project by Siemens has used measurements of air quality and traffic congestion in Munich to suggest alternative routes to drivers. Over four weeks, the routes of 1600 drivers have been rerouted to save 83 kg of CO_2 and 633 km of travel distance [Gmb19].

  In 2019, a first real-world experiment of intelligent traffic signaling with 100% connected vehicles was conducted at the university of Calabria [Ast+20]. A three-leg intersection was tested with 5 connected vehicles, which uploaded their location and routes to the traffic light controller, while circling around and across the intersection. The results showed a 73% reduction of average travel time compared to tests with fixed time intervals for signaling.

- **Health-Care:** Wearables that monitor vital signs and automatically notify the user or even medical staff in case of anomalies could increase life expectancy. Another example is a project by IBM that uses Radio-Frequency Identification (RFID) technology at hospitals to track hand washing of staff after patient contact to avoid infections that cause about 90 000 deaths per year [IBM13] in the USA alone.

- **Energy and Water Grids:** Smart meters can help to monitor and balance the distribution of electricity, gas, and water.

  For example, in the UK, half a million smart water meters have been installed to monitor water flow. Together, they have detected and located over 28 000 leaks on private supply pipes, which have been repaired and are since then saving 43 million liter of water per day [Tha21]. In general, customers with a smart meter use 17% less water than customers without [Tha21].

- **Smart Buildings:** Intelligent control of light, heating, and ventilation according to human presence and actual sensor data is estimated to reduce energy consumption by 11 – 50% when compared to conventional buildings [MSO17; KP17].

  The U.S. Department of Energy plans to invest $61 million in 10 projects for "Smart Neighborhoods", which combines the ideas of a smart energy grid with smart buildings. Previous projects in Georgia with 46 houses and in Alabama with 62 houses have already demonstrated energy savings of 42-44% [Ene21; Buc+20].

  An independent and detailed study compared smart neighborhoods to conventional neighborhoods only on days of similar weather conditions and took the additional energy demand for the intelligent control into account. They confirmed that the smart neighborhoods do indeed reduce energy consumption and therefore help to cut energy costs by $700 per month for a neighborhood of 62 houses [Chi+19].

- **Industry 4.0:** Intelligent factories that combine machines with IoT and cloud computing can enable real-time traceability and controllability of production processes and logistics [Zho+17].

  For example, Alibaba's "Smart Warehouse" in Huiyang (China) is equipped with 60 WiFi-enabled robots that move goods around the warehouse and bring them to human workers for picking and packaging [You17]. Assisting humans with robots, the warehouse could increase its output efficiency by a factor of three.

  In 2020, Amazon was using a fleet of 200.000 robots in total with up to 800 robots moving on the floor at each warehouse. These battery-powered robots read QR-codes on the floor to locate themselves but are controlled via a centralized cloud computer that coordinates the route of every single robot [Tec20]. Despite these realizations of IoT technology, Amazon estimates that fully autonomous processing of all goods will not be possible within the next decade [Sta19].

- **Agriculture:** By monitoring weather data (temperature, luminosity, humidity, pressure), soil quality, and soil moisture in combination with a controlled supply of nutrients, water and pesticides, plants could receive precisely the right amount of these resources avoiding excessive waste [Far+19; NCP20]. Furthermore, animal's health and fertility could be monitored by sensing temperature, heart rate, motion, and digestion, which helps farmers to make better decisions [Far+19].

  In 2019, the project AgriTalk [Che+19] has tested IoT-based farming over six months in Taiwan by using various sensors and actuators to remotely manage the irrigation,

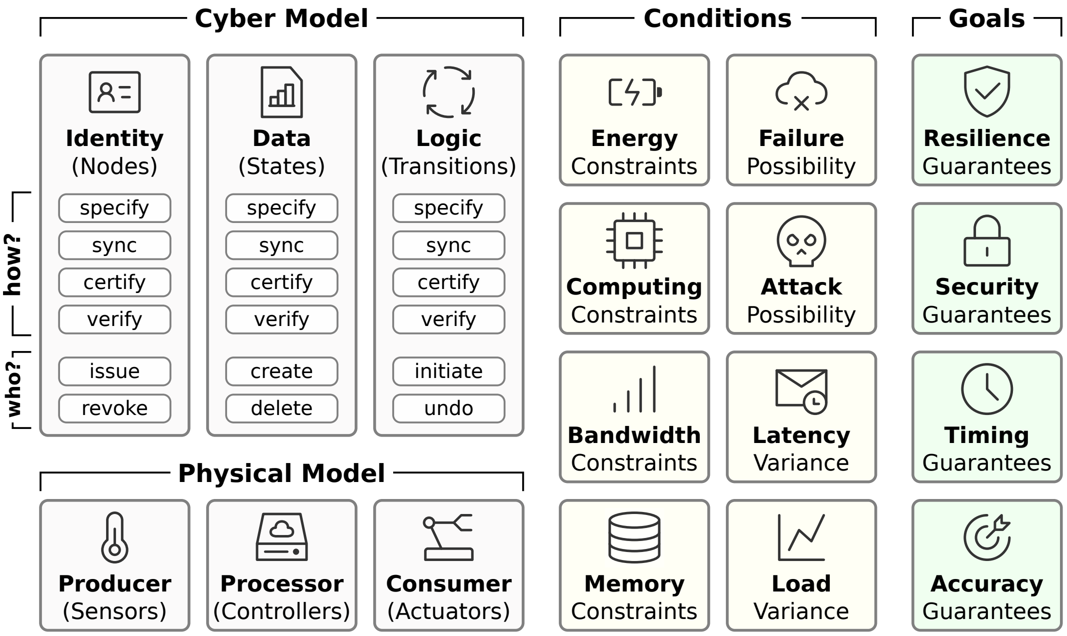

Figure 3: A high-level map of the research challenges and design space of the Internet of Things. Especially the cyber-space will be of interest and has open questions regarding **how** we can *specify, synchronize, certify,* and *verify* identities, data and logic, and **who** should be allowed to modify those entities.

fertilization, and pest control for the cultivation of turmeric. Compared to traditional farming, they have saved 70% water, multiplied the curcumin concentration in the turmeric by a factor of 5, and prevented acidification of the soil. With an initial investment of $60 000 and annual maintenance costs of $14 000 an additional annual revenue of $140 000 had been generated.

1.1.3 IoT Design Space

While the previously mentioned applications seem very broad and span multiple disciplines, they operate within similar constraints using related resources and methods and thus they can be mapped to a common high-level model of the general IoT design space, which is shown in Figure 3. The goal of most applications is to provide computational results with a certain accuracy and within a certain amount of time. Often sensors deliver the input for these computations, while the output of the computation is used to steer actuators in the physical environment.

The computational services that run the application logic should (in the best case) be available despite temporary failures and attacks. However, in the context of embedded IoT systems, the performance of any service is limited by the energy supply, computational power, and memory resources of the individual devices and the communication bandwidth between those devices.

While the full range of possibilities for IoT applications probably cannot be described in a single model, we will keep this simplified mental model of the IoT design space in mind in the following sections and chapters.

1.2 Decentralization: Scalable, Efficient, Robust

One important aspect of IoT applications – and the core topic of this dissertation – is the communication architecture, which can range from fully centralized to fully decentralized. Similar to [MSW18] and [CSB19], we divide this spectrum into four categories that are shown in Figure 4. The traditional single server represents a fully centralized architecture, while P2P networks, in which all devices are equal, represent a fully decentralized architecture.

Current implementations of IoT projects often leverage large cloud-based Infrastructure as a Service (IaaS) providers, such as Amazon Web Services (AWS) or Microsoft Azure, for the back-end processing of sensor and user data.

For example, if a user tells *Amazon's Alexa* to switch on a smart plug, the request will be processed in the AWS cloud, which then sends it to the cloud application of the smart plug, which then sends the command back to the smart plug in the user's room [ESS20]. For many IoT connections this vertical communication (as shown in Figure 4b) is enforced, despite both devices sitting in the same room and being connected to the same Local Area Network (LAN).

This general trend of centralized cloud computation becomes even more prominent when we look at the numbers. The worldwide market around centralized cloud services for IaaS has grown by 40% in 2020 alone [Gar21] and most organizations want to increase their IT spending on cloud computing [Cos21]. According to a survey from IBM with 651 developers in 2015, around 78% of developers think that cloud-based IoT is easier to implement and 77% are either considering or planning to use cloud-based platforms for their IoT services [IT15].

Till 2020, around 200 million smart speakers (Alexa, Google Assistant, Siri) have been sold and will further accelerate innovation around cloud-based home automation [Ste20], which makes up the largest part of all IoT applications at the moment [Cis20].

If we think about the properties of the IoT applications and how centralized communication works, the following questions emerge:

- if smart home devices, such as lights, are locally connected within one building, why should they communicate over cloud servers that are thousands of kilometers away?

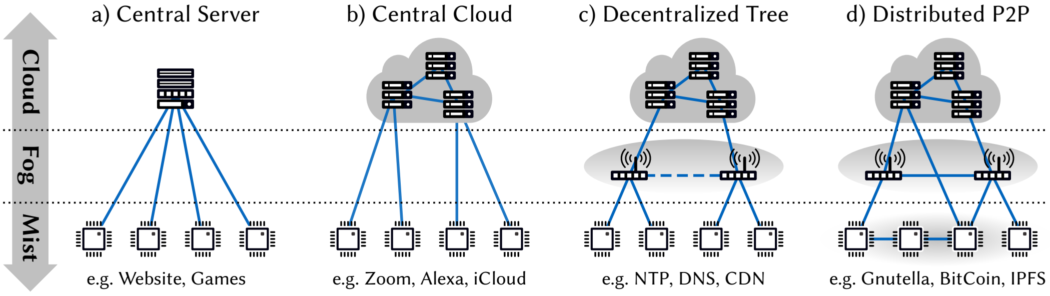

Figure 4: Different communication architectures over the three Internet layers *Cloud* (e.g. data centers), *Fog* (e.g. gateways, proxies), and *Mist* (e.g. sensors). Device symbols show where data is created, stored, or processed and blue lines indicate data flow. a) Simple websites use a single server to offer client connections. b) The cloud architecture hides a complex and scalable multi-server service behind a single central connection point. c) The decentralized tree has no central coordinator for every service request but processes and stores some of the data at fog level and some applications even utilize horizontal connections within the fog-layer (dashed line). d) A fully distributed peer-to-peer architecture has no hierarchy and there are horizontal connections within each layer. All participants run the same application and only differ in their available resources.

- if mobile devices, such as cars and robots, are spatially distributed and mostly interact with their neighbors, why should they rely on a central management?
- if personal health data is sensitive, why should we upload it to centralized cloud servers exposed to the Internet?

Just because there are *some* reasons for centralized architectures, such as ease of implementation and maintenance, they should not be seen as universal solution that is suitable for every IoT ecosystem.

1.2.1 Centralized Architectures are Inefficient and Fragile.

In this section, we discuss three important disadvantages of centralized architectures. A general discussion and overview on the drawbacks of cloud-based architectures can be found in [CSB19].

Central servers are less efficient. In order to handle request peaks at the full scale of a network, central servers need to be equipped with the necessary resources.
In the past, providers of online services had the problem that they needed to decide in advance how much hardware resources they put into their dedicated central server. Using the average expected demand as baseline, leads to the problem that during periods of high demand, the server can not handle the huge amount of requests. For services, such as online marketplaces, this is not only an inconvenience for their users but also means financial loss due to missed sales. As a result, the resources of central services are often over-dimensioned for average request rates in order to tolerate a spontaneous increase in traffic load. At

periods where the demand is especially low, for example during nighttimes of a local online service, the central servers run idle most of the time leading to an accumulating waste of energy.

Data centers and clouds use a decentralized architecture of many servers internally but appear as a single instance for connecting clients and therefore represent a central server from a global system perspective. In 2009, the utilization of data centers was only around 20–30%, meaning that most of the time the servers did run idle but still consumed around 60% of their peak power [MGW09].

Over the period from 2010 – 2018 the global workload of data-centers has increased sixfold, the traffic tenfold, and their storage capacity 25-fold, but surprisingly the energy demand could be held almost constant (only 6% increase) due to strong improvements in hardware efficiency, power management, and virtualization [Mas+20; She+16]. However, since these mechanisms might soon be exhausted (e.g. power consumption will soon be almost proportional to utilization [She+16]), it is unclear for how long centralized efficiency improvements can continue to neutralize the rapid growth of IoT services.

The total electricity demand of data-centers reached 200 TW h (terawatt hours) per year in 2018, which corresponds to 1% of the global electricity demand and half of the global electricity used for transportation (2%) [Jon18]. Projecting the current development of IoT services into the future has lead to estimates of a demand between 3% and alarming 8% of the global electricity by the year 2030 [Jon18].

Removing the central servers and decentralizing IoT services could help to reduce the energy demand by 14% to 25% compared to a fully centralized architecture even when taking into account the increased communication complexity [AOL19]. [2]

While such an improvement alone will probably not fully handle the expected increase in energy demand, it could turn out to be a necessary pillar within a more holistic strategy to further scale Internet-based applications.

Central servers are more fragile. In a centralized architecture every message and transaction is routed through the central server. As a result, any failure of the central server affects the entire system.

For this reason, cloud computing has introduced redundancy against single failures by distributing application data and logic among several servers. What was previously a single server, has become a data-center with many servers and if one server fails, the others remain operational. However, even data-centers at several geographical locations are not immune to single failures as long as they have a central management.

[2] A short note on Bitcoin's energy consumption: The decentralized Bitcoin network consumes a huge amount of electricity. However, this is not a general characteristic of decentralized architectures but only occurs due to the Proof-of-Work (PoW) mechanism, which is explained in more detail in Section 3.1.2.

For example, Facebook could be considered decentralized with its 18 data-centers on 3 different continents [Fac21], however in October 2021, the company and all its web services went down for 5.5 hours – globally. Due to a mistake in their central BGP routing configuration, all their nameservers became unreachable [Cla21]. Within minutes, billions of people and businesses were unable to access any website hosted by Facebook and could also not communicate with each other over WhatsApp. Since Facebook accounts are also used as a general authentication mechanism for other web-services, many users found themselves locked out completely, including Facebook employees that were unable to access their offices [IF21].

In some countries, such as Myanmar and India, Facebook and WhatsApp have become synonyms for the Internet and almost all online communication is based on their services [IF21]. While the economical damage for all users is extremely difficult to estimate, the damage for Facebook alone was estimated to be over $60 million.

Overall, this incident illustrates our dependency on these platforms and the impact a single central failure could have. If we consider IoT applications for personal health monitoring, electrical grid, or traffic control, a complete and global shutdown of five hours could have catastrophic consequences.

Central servers are targets. Central services store a huge amount of sensitive data, such as passwords, personal information, or financial credentials, of millions or even billions of people [Bar18]. This fact makes central servers a profitable target for hackers. Once they manage to get access to a central database, they are able to download large parts or even the entire dataset of all users. Due to the high-speed connections of data-centers, this theft can be performed quickly and is difficult to detect among normal traffic [Mod+13].

Furthermore, the central servers also enable insider attacks from employees that have a more privileged access to the internal system [Mod+13] or the company itself might sell collected user data for profit.

In the first half of 2018 alone, over 4.5 billion sensitive data records have been stolen in 945 breaches [Bar18]. The largest data set came from social media platforms such as Facebook. In addition to the collection of personal data, central servers offer access to the entire network to perform malicious actions on a large scale. In 2016, a group of hackers from North Korea stole $81 million from the central bank of Bangladesh [Zet16]. The group had gained access to the bank's central SWIFT[3] system from which they initiated 35 transactions to transfer a total of $951 million from Bangladesh' reserves at the Federal Reserve Bank of New York. However, due to protection systems and observant employees, only $81 million went through and could be transferred out of reach [BBC21].

[3] SWIFT (Society for Worldwide Interbank Financial Telecommunications) is the messaging network used by banks to accurately and securely transfer large sums of money.

It has to be said that distributed systems are not inherently more secure than central architectures and individual nodes might even be less protected and easier to attack. However, when there are no highly privileged or trusted participants within the network, it becomes more difficult to compromise large parts of the system or steal multiple data sets at once. As a result, the implications of individual breaches are less severe, which would lower the incentive of attackers in the first place.

1.2.2 Decentralized Architectures: Types and Examples

Decentralized architectures, as shown in Figure 4c and 4d, do not rely on a single central instance to store and process all application data. They make use of shorter and faster links and vertically bounded processing, mostly due to two mechanisms [Que+19; CSB19]:

- **local processing** of information in the lower/outer layers of the networks. The terms *fog computing* and *edge/mist computing* describe this idea and promote the processing of data closer to their source, e.g. the sensors in the outer mist layer, instead of forwarding all data to the cloud.
- **horizontal communication** within a vertical layer to increase the possible communication paths. These additional and often faster links allow to distribute previously centralized computation tasks and data among devices within one layer. The term P2P describes this idea in which all devices communicate on the same level and do not connect to a higher instance for their coordination.

In this sense, cloud computing is already the first step of decentralization. Instead of a single central server, cloud computing uses multiple servers that are distributed at different geographical locations [MSW18; Fac21]. However, from the perspective of fog/mist devices, the cloud still appears as a single centralized instance as it illustrated in Figure 4.

Examples of Decentralized Systems

The Internet itself is a great example of decentralization and it would be very difficult to imagine an alternative history in which the global traffic of 9 exabytes per day [Cis16] was routed through a single central instance. Instead, distributed routers forward our messages over different routes to ensure robustness despite of failures and variance in traffic load. While examples of cloud computing have already been discussed, we want to give a few extra examples for systems that have a tree-like decentralization or are fully distributed.

The Network Time Protocol (NTP) is a tree-decentralized information provider. Accurate time synchronization is important for computer systems but there are too many computers and not enough atomic clocks available. The current solution is the Network

Time Protocol, which distributes time information in a tree-like fashion among many Internet-connected devices. Only a few computers are directly connected to the atomic clock but they propagate the time information to a larger set of time servers. These servers synchronize among each other (horizontal communication) and finally distribute the time to specific end devices. NTP has been in use since 1985 [Mil85] and is still the default synchronization technique, which illustrates the efficiency and robustness of this decentralized architecture.

Wikipedia uses tree-decentralized authority. Decentralization cannot only be applied to data storage and communication but also to the aspect of authorization. For example, the online encyclopedia Wikipedia runs on central servers but the authority to write and edit articles is distributed among all users. When it went online in 2001, it democratized authorship and authority by allowing anyone to edit articles [FLB09]. However, not all users are treated equal. Some users have higher privileges and can approve or undo changes of completely unprivileged users. Therefore, Wikipedia falls into the third category of a decentralized tree structure.

In 2009, Microsoft acknowledged that their own encyclopedia *Encarta*, which was the second largest online encyclopedia at that time but with a conventional centralized authorship, could not compete in terms of text quality and freshness with Wikipedia and, as a result, discontinued their service [Coh09].

Bitcoin is a fully decentralized data storage. In 2008, an unknown author with the pseudonym “Satoshi Nakomoto” proposed and implemented the idea of a decentralized crypto-currency called *Bitcoin* [Nak08]. In Bitcoin, transactions are verified and certified by every participant and not by a central bank. The history of all transactions and thus the exact balance of Bitcoins on every address is distributed and replicated by every participant of the network. There are no hierarchies/layers but every device running the Bitcoin software is treated equally, performs the same computations and stores the same information. As such, Bitcoin falls into the last category of a fully distributed peer-to-peer system. More details on Bitcoin will be discussed in Section 3.2. Within 13 years, the total market value of Bitcoin has increased from basically zero to around one trillion USD in February 2021 according to `coinmarketcap.com`, which might be an indicator for the success of decentralization in the financial sector.

1.3 Research Challenges: Synchronization, Certification, Authentication

In the previous section we have seen that we can distribute *data storage*, *data processing* and *authority* both vertically and horizontally and that this decentralization offers several advantages over centralized architectures. However, a decentralized system also means no global system state and increased communication complexity. As a result, more effort and care is required when designing such a system.

1.3.1 Why IoT Introduces new Challenges

While distributed systems have been studied extensively in the past, the conventional approaches are often not suitable for IoT applications due to three factors.

- **Large-Scale Network:** Previous studies on distributed systems assume a limited number of network participants in the range of 10 to 100 devices. However, as discussed in Section 1.1, the IoT consists of potentially billions of devices. Conventional approaches often do not scale efficiently enough to be used in large-scale networks [NL20; Yeo+17].
- **Resource-Constraints:** Previous studies on distributed systems often assume that devices have sufficient energy and computational power to perform advanced cryptography and sufficient storage to keep track of the states of all other devices. In IoT networks, the majority of devices are cheap sensors and actuators with very limited resources and thus existing security mechanisms and protocols might not run on these devices [CSB19].
- **Physical Reality:** Previous studies assume the processing of arbitrary data that is valid according to certain self-defined rules. As a result, the main goal is data integrity. For example, in the Bitcoin network any transaction that is stored in the blockchain is defined as valid [Nak08]. However, in IoT applications most of the data is bound to a physical reality (e.g. temperature measurements) and therefore inconsistencies might not just arise between copies of a data record but between the value of the record and the physical reality it should represent [Gre+19; Li+14].

In this section we will therefore analyze the technical challenges in general that arise when targeting a decentralized system architecture for IoT applications before we dive into the individual solution approaches in the chapters 2 to 4.

1.3.2 Technical Challenges We Explore

1. Decentralization requires state synchronization. As soon as we remove the central instance, we have no single place where the system state is stored and managed. Instead of applying state changes only in one place globally by an order that the trusted central instance defines, we have to store any state information locally at some edge device.
However, in a highly dynamic network, such as the Internet, devices might connect and disconnect frequently and thus the local states are replicated at different devices to ensure continuous operation [Bod+20]. However, as soon as we have multiple copies of state information, we need to synchronize state changes and apply the same changes in the same order to every copy [Yeo+17]. Otherwise, the system looses data integrity and will run into conflicting local states. Furthermore, when synchronizing many devices over the Internet, not all devices can be considered trustworthy but might act maliciously and our synchronization scheme needs to tolerate these adversaries.
For example, when playing an online multi-player game, the central server manages the locations and status of all players. The state of the game is fully determined by the information the server has received. If one player looses connection, it does not matter which action this player might take and the game will proceed as if the player did not make any moves. If we remove the central server, every player needs to keep track of all other players. Without proper synchronization, no single player can determine the global state of the game because each player might have received a different sequence of actions from the other players. There is also an incentive for the players to *cheat* and send wrong, conflicting, or delayed messages in order to gain an advantage in the game.

2. Decentralization requires certification of data. As soon as we have synchronized the state among a certain set of agreeing devices, we need to synchronize the state further with devices that did not participate in the agreement but are also interested in the current state. Furthermore, since edge devices often dynamically connect and disconnect, the reconnecting devices that participate in the agreement process also need to learn about the current state in order to make future decisions.
In our game example, once the players have agreed on the state of the game, they can play the game and determine the outcome. However, there might be third-party spectators that are also interested in the outcome of the game and want to verify the current state. There could be some functionality, which relies on that information (e.g. betting on players), but does not actively participate in the game and its state agreement. For such services, we would need to generate some type of certificate that a game ended with a certain score.
When we think further in this direction, we realize that in general we would like to have a mechanism to certify the correctness of any type of data. Especially for physical measurements from sensors that are locally generated, we might not be able to rely on

agreement but need another method to verify the validity of sensor readings. Currently, this type of certification is often achieved by trusting certain sensors to work correctly and only authenticating their messages.

3. A P2P network of resource-constrained devices requires efficient authentication. In a centralized architecture, each device that connects to the server only has to authenticate the server and no other device. The server, on the other hand, has to authenticate every connected device but since a central instance usually is a powerful cloud center with almost unlimited resources, this overhead is not a serious obstacle.

In a decentralized network, each device needs to authenticate the messages of all its peers and therefore also store keys for each peer device. For large-scale networks, the management of symmetric keys would be very complex and resource consuming, and thus most applications rely on asymmetric Public Key Cryptography (PKC), such as digital signatures. However, PKC is also computationally expensive and consume precious time and energy on battery-powered micro-processors. In combination with the increased number of messages within a decentralized network, it is often very difficult to achieve proper authentication and security in all layers of the system.

Another challenge arises from the accelerating development of quantum computers, which will break all of the currently used PKC schemes, such as Rivest–Shamir–Adleman (RSA) and Elliptic-Curve Cryptography (ECC) [Sho94; Mav+18]. Only an authentication scheme that is both lightweight and quantum secure would be suitable as a long-term solution for IoT applications.

In the upcoming Section 1.4.1, we will put the challenges into perspective with our contributions.

1.3.3 Technical Challenges We Assume Solved

Besides the aforementioned challenges, there are many other important aspects, questions, and challenges, which would deserve their own thesis. In order to focus on the decentralization aspect, we have developed the approaches in this dissertation under the assumption that the following challenges are already solved in a suitable way.

- **Connectivity:** How can devices connect to the Internet? Which wireless radio-protocols, such as WiFi, ZigBee, or LoRa, offer the best combination of power efficiency, range, and bandwidth?
- **Routing:** How can messages be routed to their destinations despite of changing network topologies?
- **Interoperability:** How can a group of heterogeneous devices work together and exchange data even when they are not produced by the same manufacturer?

- **Flexible Configuration:** How can devices be updated with new functionality and configurations to dynamically change their role and behavior?
- **Power Management:** How are devices powered? Which combination of batteries, wires, or energy harvesting is suitable for an application?
- **Human-Machine Interface:** How can devices communicate their identity, status, events, and abilities to humans and how can humans trigger actions on these devices?

1.4 Meta-Parameters of this Dissertation

This section clarifies the *Scope*, *Contributions*, *Methods*, and *Related Author Publications* of this dissertation and should help the reader to judge the relevance of the covered aspects according to personal interests as well as help to identify and navigate to the corresponding sections.

Scope: The goal of this dissertation is to investigate how resource-constrained devices that are connected via the Internet can provide an automated and collaborative system functionality within a decentralized system architecture. We analyze this problem from a system-level perspective with the constraints and conditions depicted in Figure 3 (p. 6). In this sense, we assume that power, memory, and connectivity are given but limited and that devices are in principle able to communicate with each other but messages might get lost or delayed. We consider the device classes 1–4 in the taxonomy for resource-constrained devices [BEK21][4], which covers several types of microcontrollers. This work focuses in detail on secure and resource-efficient interaction and communication protocols in the context of self-organized IoT applications, such as intelligent transportation systems consisting of CAVs. Overall, this work provides a methodological overview as well as specific technical strategies on how to address the challenges that arise from decentralized architectures in order to harvest their benefits.

1.4.1 Structure and Contributions

The current **Chapter 1** provides an introduction to the IoT and its domains, and motivates the need for decentralization. Especially in Section 1.3, we have formulated three research challenges (synchronization, certification, authentication) for decentralizing the IoT and explained why they are not solved by previous studies on decentralized systems. The following three Chapters 2 – 4 cover our core contributions to tackle the challenges 1 – 3.

[4] Examples: Arduino Zero (Cortex M0+) would be Class 1/2, Espressif's ESP32 would be Class 4.

Chapter 2 tackles challenge 1 and explores synchronization of distributed data via consensus protocols and how these protocols can be used in CPS – a domain for which they were never designed. In the previous decades, consensus has been extensively studied for database systems in which a few powerful computers replicate abstract data records. However, in the context of IoT, we have thousand or millions of resource-constrained devices that want to agree on physical-information that often has local differences in the environment. This mismatch of conditions requires adjustments and novel mechanisms to ensure that agreement guarantees can still hold.
We contribute two communication protocols [RS19; RBS21] that enable cooperative vehicle maneuvers using distributed consensus over Vehicular Ad-hoc Network (VANET). The first protocol (Section 2.3) targets the management of vehicle platoons that drive behind each other on the same lane. Using an unanimous consensus that is tailored to the topology of platoons allows us to safely decide on synchronized driving maneuvers. The second protocol (Section 2.4) uses consensus to let vehicles agree on crossing schedules at intersections. Vehicles directly communicate in onion-like layers when approaching the intersection and perform a more efficient scheduling compared to conventional traffic rules.

Chapter 3 tackles challenge 2 and explores the certification and provability of agreed-upon data via blockchain. In centralized systems, the central instance is often trusted and the information it provides is seen as valid. Thus certification is achieved by message authentication of the central instance. However, in decentralized systems, agents are not trustworthy by default but need to prove their honesty and the validity of the data they provide. A shared data structure that is validated and enforced by all participants is a promising solution and blockchain is one candidate for this data structure.
We contribute a generic verification scheme [RS18a] for data stored in blockchains that is highly memory efficient (Section 3.2). By adding a single additional reference hash to the blockheader, we create an interlink pattern that allows the traversal of any blockchain with logarithmic complexity. We also propose a secure but lightweight method [Reg+20] to synchronize IoT devices to a global reference time using the timestamps in public blockchains (Section 3.3). By only passively listening to the stream of newly generated block headers, we are able to securely estimate the correct date and time to an accuracy of one second.

Chapter 4 tackles challenge 3 and explores lightweight and secure authentication of agents and messages. Currently, digital signatures are used in almost all Internet applications to authenticate devices and messages. However, digital signatures are computational expensive and require the storage of keys. In centralized architectures, devices in the *mist* layer (see Figure 4) only need to authenticate the central instance which is barely feasible on cheap micro-processors. However, in decentralized architectures, devices are required to authenticate several peers and an increased amount of messages, which quickly exceeds their hardware capabilities.

We contribute an improved variant of a hash-based authentication scheme [RS20] for fast and quantum-secure message authentication with a smaller signature size than related schemes (Section 4.3). The public keys for peers are only the size of the security strength in bits, e.g. 128 bit key for 128 bit security, and the signature size could be reduced to roughly 2 kB.

Chapter 5 combines the results of all previous chapters, discusses their implications, and draws the final conclusion. The main high-level findings of this dissertation are summarized in Section 5.1.

The appendix A provides further details, such as algorithms or additional related work, that are not necessary to follow the narrative of the thesis.

1.4.2 Methods for Modeling and Evaluation

In general, we model IoT applications as collaborative multi-agent systems that interact via message passing. *Agents* are cyber-physical devices that execute instructions based on algorithms and protocols to fulfill a certain functionality. Each device consists of a cyber layer, which represents abstract data and logic in form of software, and a physical layer that considers the location of hardware within the real world allowing it to sense and manipulate its environment. We also focus on resource-constrained devices of class 1–4 according to the system proposed in [BEK21], which refers to different types of microcontrollers.

The system logic is modeled on the basis of Finite State Machines (FSMs), where behavior is determined by a sequence of states and the transitions between states are decided by the input provided to the system, which could be either an event in the cyber layer or a sensor reading from the physical layer.

The information exchange between agents is modeled as message passing network, which means that information is quantized into chunks (the messages) and then send from one specific agent to one other specific agent using an addressing scheme. Each message is either fully delivered or lost. Furthermore, timing is considered and the delays between sending and receiving a message are unknown, variable, and non-negligible compared to the processing time of atomic instructions.

Communication protocols and the behavior of individual agents are developed and presented as pseudo-code that specifies the instructions each agent would execute based on received messages or changes in the system state. Agent are not trusted by default and can also show malicious behavior.

The security of communication protocols is analyzed using threat models, in which we first specify assumptions on the capabilities and possible actions that malicious agents can take and afterwards show that the success probability of attacks under these assumptions is close to zero.

Overall, evaluation methods focus on quantitative analysis, simulations with well established and mature frameworks, as well as some experimental measurements on representative hardware using prototypical software implementations. All obtained results are compared against related work to determine the quantitative improvement of our approaches. If measurement uncertainties are present, statistical analysis is used to estimate and reduce the margin of error.

1.4.3 Author Publications

The previously mentioned contributions have also been made public in the following peer-reviewed first-author publications:

Author Publications		**Section**
[RS19]	E. Regnath, S. Steinhorst: *"CUBA: Chained Unanimous Byzantine Agreement for Decentralized Platoon Management"* in IEEE Conf. on Design, Automation and Test in Europe (DATE), 2019	2.3
[RBS21]	E. Regnath, M. Birkner, S. Steinhorst: *"CISCAV: Consensus-based Intersection Scheduling for Connected Autonomous Vehicles"* in IEEE Conf. on Omni-Layer Intelligent Systems (COINS), 2021	2.4
[RS18a]	E. Regnath, S. Steinhorst: *"LeapChain: Efficient Blockchain Verification for Embedded IoT"* in ACM Conf. on Computer-Aided Design (ICCAD), 2018	3.2
[Reg+20]	E. Regnath, N. Shivaraman, S. Shreejith, A. Easwaran, S. Steinhorst: *"Blockchain, what time is it? Trustless Datetime Synchronization for IoT"* in IEEE Conf. on Omni-Layer Intelligent Systems (COINS), 2020	3.3
[RS18b]	E. Regnath, S. Steinhorst: *"SmaCoNat: Smart Contracts in Natural Language"* in IEEE Forum on specification and Design Languages (FDL), 2018	3.4
[RS20]	E. Regnath, S. Steinhorst: *"AMSA: Adaptive Merkle Signature Architecture"* in IEEE Conf. on Design, Automation and Test in Europe (DATE), 2020	4.3

The conferences have been selected according to their coverage of the paper topics and their general quality ranking. For example, the conferences DATE and ICCAD received the highest *Qualis*-Score[5] of A1 according to conferenceranks.com and have an acceptance rate of $< 25\%$. The COINS conference is a very good match for IoT-related topics that focus on CPS and decentralization. Since it started in 2019, which was only 3 years before writing this dissertation, it has not received a ranking yet. However, prominent researches in the

[5] Ranking system for conferences with scores A1 (highest), A2, B1, B2, B3, B4, and B5 (lowest)

domain of decentralized IoT, such as the inventor of the *Hashgraph* protocol [BL20] have also published at this conference, indicating solid quality.

The sections that are based on publications are marked with a footnote at the end of their heading. Furthermore, some parts of the introductions of the chapters 2–4 are also based on individual paragraphs from the aforementioned author publications but are not explicitly marked as such for simplicity.

Chapter 2

Distributed Consensus for Cyber-Physical Systems

Contents

What should we have for lunch? is the most important question to answer once the clock strikes 12:00 in our office. Vietnamese curry, Italian pasta or Bavarian Schnitzel are common suggestions and while there is always some discussion, we typically reach a consensus among our team in a few tens of seconds. In our small office, reaching agreement is fairly easy.

However, the complexity of this problem escalates quickly once we scale the situation to a global network of thousands of devices while at the same time restricting all participants to communicate blindly over an unreliable message passing system with arbitrarily high dynamics in latency and bandwidth, eliminating face-to-face authentication and instead introduce the possibility of masquareaded lying agents that pretend to belong to the team. Reading the previous sentence again might sound like an exaggerated and only hypothetical puzzle for communication engineers, full of uncertainties and impossibilities; and yet, this scenario seems to hold appropriate and realistic assumptions for analyzing distributed systems that communicate over the Internet.

In this chapter we will explore consensus protocols, why we need them in decentralized systems, and how we can utilize their properties in CPS – a domain for which they were originally not designed.

Why Consensus enables Decentralization Most applications can be modeled as a FSM that describes a certain behavior as a sequence of transitions between system states. The system state contains the set of input variables that are needed to check for transitions and the current output variables.

For example, a simple heating system could be modeled using the two states "on" and "off" and the transition between them happens when the temperature falls below or exceeds the target temperature value. Each state then contains the current temperature as input and the opening of the water valve as output.

In a centralized architecture, the system state and its transitions are managed by a central coordinator and thus the system state is well defined as long as the coordinator is working correctly. In a decentralized architecture, however, the global system state depends on the local states of multiple devices. For example, if we would have three temperature sensors and two heaters, how are we going to decide when to turn on each heater? We can define a deterministic rule for each heater as soon as all five variables ($3\times$ temperature, $2\times$ valve) are known and therefore all devices need to communicate and synchronize their states with each other. The tricky part about synchronizing the state variables arises from three effects:

1. **Data Dependency:** some variables, e.g. the valve, will change depending on the value of the other variables because this will allow complex system behavior. This means that the exact order in which local variables are changed matters.
2. **Message Delay:** One key characteristic of distributed systems is that "the message transmission delay is not negligible compared to the time between events in a single process" [Lam78]. Therefore, the message delay time might be longer than the time between variable changes.
3. **Local Clocks:** Each agent runs a local clock that has different phase and frequency [Lam78]. Clock synchronization is possible but either assumes a symmetric delay when sending messages or already uses consensus. Even then, the local clocks will not be perfectly in sync and the difference still might cause errors.

The combination of these effects makes it impossible to trust single messages about state variables because upon reception, the state could have already been changed by another agent.

Consensus protocols are communication rules that solve the problem of synchronizing state variables among a set of agents despite message delays or failures and ensure that variables will not change before they are synchronized. Therefore, consensus protocols are one important building block for applications in which the system state is distributed among individual agents.

2.1 Consensus: History, Models, and Implementations

In this section, we will discuss the basic principles of conventional consensus protocols, before we adjust and apply them to CPS in the next section.
In general, the goal of consensus is to reach agreement on a common decision among a set of agents. Each agent is able to send votes for a value to all other agents. The agents somehow "discuss" by several rounds of voting and agents may also change their vote. At some point, each agent needs to "decide" on a value and is not allowed to change this decision afterwards.

2.1.1 Evolution of Conventional Consensus

In the following, we give a short overview of the most important results from literature that have pushed forward the development of consensus protocols over the last decades.

Generals and their Problems

The discussion about synchronizing data in a distributed system is historically often started by an illustration of generals that try to coordinate an attack of a city. In this fictive scenario, the attack is only successful if all generals attack simultaneously and will fail if only a subset of the generals attack. To complicate this task, the generals can only communicate via messengers on horses and these messengers might get captured. Using this scenario, we present some of the most important early results in literature around 1980 before we continue with our system model.

The Two General's Termination Problem One of the earliest descriptions of an agreement problem was published in 1975 by E. Akkoyunlu *et al.* [AEH75] and considers the case when there are only two communication partners. Let us assume two honest generals Alice and Bob, which both need to agree whether they will attack or not. They could simply exchange only one type of vote (e.g. "yes") and then decide once they receive a "yes" from the other general. Unfortunately, each general cannot know whether the other general has read their message. So when Alice receives a "yes" and has sent a "yes", she cannot decide because she needs to know whether Bob has also received her "yes" and if he will therefore

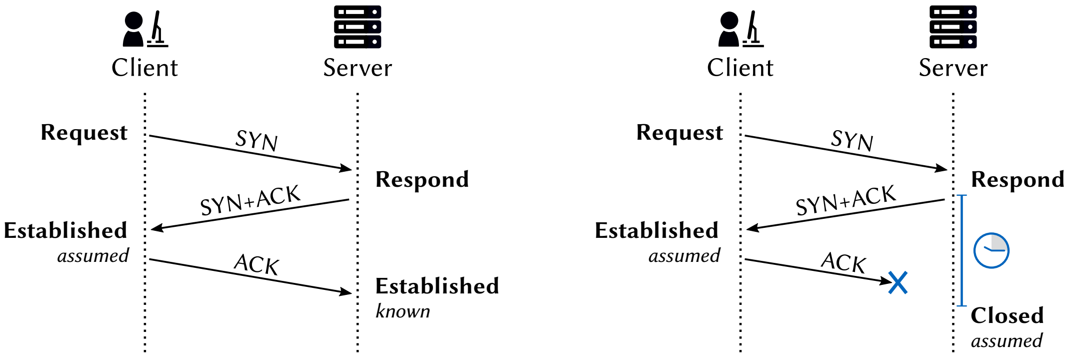

Figure 5: Illustration of the Transmission Control Protocol (TCP) handshake, which does not solve the Two General's Problem because both sides can never be sure when the connection is established. When the client does not receive SYN+ACK, it will assume a timeout and try again. Once the client sends an ACK, it will assume the connection is established (left). However, if the server does not receive the ACK, it will assume that the client timed out and closes the connection (right). The state is not consistent in this case!

decide the same value as she. If they have agreed to exchange another acknowledgment message (e.g. "ACK") before they decide, then they have the exact same problem in this second round. As a result, there can never be a last acknowledgment message, if one general cannot be sure it will be received and read. As shown in Figure 5, this result implies that even the common internet protocol TCP cannot guarantee state consistency between client and server.

The Byzantine General's Problem After the state consistency problem with two participants, Leslie Lamport *et al.* started a discussion on consensus protocols that can tolerate arbitrary faults [LSP82; PSL80]. In contrast to the *Two General's* problem, not all generals involved are honest but there are traitors among them. These traitors may not just remain silent, but have the ability to send arbitrary messages, especially conflicting messages. This arbitrary behavior is called *Byzantine Fault* and is more difficult to handle than crash faults. However, the authors showed that it was possible to synchronize the state among the honest generals, if the number of traitors f, which can forge messages, is below one third of the total number of generals [PSL80]. More formally, consensus is possible if $f < N/3$, which means that for $N = 4$, there must be only one traitor. In light of the result from the *Two General's Termination Problem*, they made the critical assumption that if an honest general sends a message, it will be received within a certain constant time and thus the absence of a message can be detected [LSP82]. This assumption is often referred to as *synchronous communication*. While this does not solve the Termination Problem for asynchronous communication, their protocol guarantees that an honest general will never decide for the value of the traitor and thus they guarantee integrity of the agreement. Based on these results, the class of protocols that can handle Byzantine faults are called Byzantine

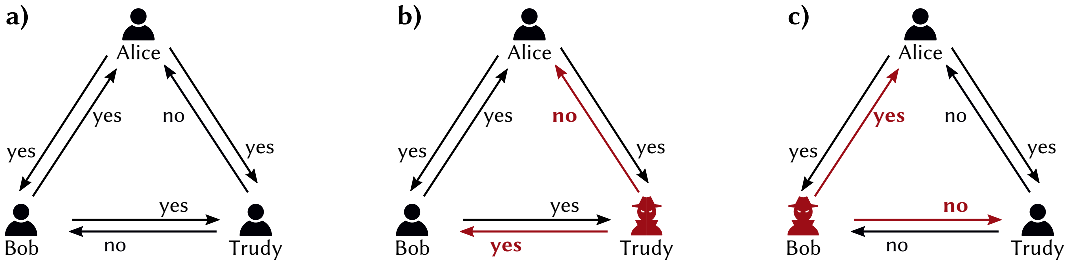

Figure 6: Depiction of some problems with Byzantine Generals. a) All honest generals either send or receive one "no" and thus no general attacks (consensus reached). b) Trudy is a traitor and sends conflicting messages. Bob will attack, while Alice retreats (consensus failure). c) Bob is the traitor but Alice and Trudy agree to retreat (consensus reached). In all three scenarios, Alice receives the same messages and thus it is impossible for her to tell if there is a traitor and who it is.

Fault Tolerant (BFT).

The FLP Impossibility Result After showing that agreement tolerating faults is possible with synchronous communication, the search began for a generic solution in the asynchronous setting where the delay of messages can be arbitrarily high and thus agents cannot distinguish between the absence of a message and a slow message. However, a paper by Fischer, Lynch, and Paterson (FLP) in 1985 destroyed this hope [FLP82]. In a simplified version, the FLP impossibility states that "consensus cannot be deterministically solved in an asynchronous system if a single process can crash" [Cor+11]. The proof was constructed by showing that for every consensus protocol there exists a sequence of events that keeps the agents in an undecidable state forever. For each round in a consensus protocol there exists a *critical* message that brings the protocol closer to agreement. However, according to the assumptions, this message could take arbitrarily long or the process responsible for sending the message could crash shortly before doing so. No matter how the rules are specified, in every round the critical message could not arrive. The result does not only apply to *Byzantine Fault* but to single crash fault and is thus in line with the *Two General's Termination Problem.* To circumvent this problem, most consensus approaches slightly modify the system model in such a way that the FLP result does not apply. Two common modifications are

1. allow non-determinism: By using randomness in the case of undecidability, the agents could by chance reach a state that is decidable without relying on a critical message.
2. partly-synchronous communication: Instead of assuming fully asynchronous communication, it is assumed that some messages will arrive within a certain time bound. Thus by repeating messages, the protocol will make progress once the messages are delivered within the time bound.

Partial Synchrony The scientific community quickly realized that fully asynchronous communication without any time bounds is almost an equally unrealistic assumption as continuously synchronous communication. In combination with the FLP result that made deterministic consensus impossible in the fully asynchronous setting, the search began for a timing model that lies in between those two extremes. In 1988, Dwork *et al.* [DLS88] proposed a *partial synchronous* timing model, which has two variants:

1. There is a maximum finite message delay δ_{max} but it is *unknown* to participants. The solution is to gradually increase the timeout until all messages are delivered within δ_{max}.
2. There is no maximum delay δ_{max} but nodes simply agree on an upper timeout time δ_{to}, which will eventually hold after some unknown global stabilization time T_{GST} for some time duration t_{Cons}. t_{Cons} is the time required to perform one consensus round. The solution is to repeat messages until there is a synchronous round.

The second variant is more commonly used as it also tolerates dynamically changing delays as long as there exists some time window t_{Cons} for every round in which all messages can be delivered within the agreed upon upper bound δ_{to}. In other words: Partial synchrony requires consensus protocols to ignore late messages until all messages required to complete the consensus arrive synchronously within time bound δ_{to}. This prevents the system to consider a correct agent as faulty, simply because some of its message arrive too late.

Conventional Consensus Problem Definition

The results of these early studies were often based on slightly different definitions of what the consensus problem actually is. It is clear that agents should agree on some value but with the varying system assumptions, it was unclear what formal requirements need to be satisfied.

Over time there emerged some consensus about the definition of the consensus problem. Largely in line with the early work [FLP82; DLS88], the consensus problems can be considered solved if three properties hold:

- Agreement: All correct agents decide the same value.
- Integrity: All correct agents decide only once.
- Termination: All correct agents decide eventually.

Agreement and Integrity are safety properties, Termination is a liveness property. An agent is *correct* if it follows the consensus protocol. However, many papers use different terms, such as “consistency”, or slightly different definitions of those.

It should also be mentioned that besides the consensus problem, similar problems have been studied extensively in literature over the last four decades. An overview on the differences of the common three (Byzantine Agreement [LSP82], Distributed Consensus [FLP82], and Interactive Consistency [PSL80]) can be found in Appendix A.1.1.

The most common extension or application of a consensus protocol and also the focus of the remaining chapter is *State Machine Replication* [Sch90; CL99]. In State Machine Replication (SMR), we are not only interested in agreeing on a single value but continuously agree on state transitions, such that each agent processes the same data in the same order. SMR allows to run an application in a distributed fashion and achieves fault tolerance because if one agents fails, the application will continue its operation from the same state on the other agents. To achieve state replication, the agents must operate *deterministically* and establish a global *total order* of all communicated state transitions before executing them [CL99].

2.1.2 Consensus Protocols

Since the first description of the general's problem and the consensus problem, hundreds – if not thousands – of protocols have been proposed to solve them. This huge amount of protocols arises from the fact that even slight changes in the system properties and network assumptions offer many variants and trade-offs in the design of protocols and thus they can easily be tailored to specific use cases.

While a complete overview of all protocols is infeasible, we will categorize them according to their characteristic assumptions and features.

Our classification system

As part of this dissertation, we have developed a classification and notation scheme for consensus protocols. The notation is divided into the three aspects *system assumptions*, *protocol choices*, and *result guarantees*.

Our classification system is based on related work [Bod+20; NL20]. Especially [NL20] has proposed a taxonomy for blockchain consensus protocols and has identified 7 dimensions: *Fault Tolerance* (BFT/Fail-Stop), *Network Timing* (Sync/Async), *Scarce Resource* (time, vote, storage), *Block Proposal*, *Transaction Finality* (deterministic/probabilistic), *Network Accessibility* (public/private), and *Network Communication* (gossip, P2P). We have adjusted and extended this system to general consensus protocols and not only those that are used in blockchain networks. Additionally, we propose a short notation to make protocols easily comparable.

Dimensions: We classify consensus protocols according to the following dimensions

1. **Fault Type [B/F]:** Describes the types of faults that can occur. **B**yzantine fault means that agents might send malicious messages. **F**ail-Stop faults are only those faults where agents will stop sending messages over a longer period of time.
2. **Timing [A/S]:** The communication between agents can either be **A**synchronous or **S**ynchronous. Synchronous also includes any system with *partial synchrony* as it still requires some synchronous communication to proceed.
3. **Equality [E/Q/L]:** The distribution of responsibilities. **E**qual means that all agents have equal capabilities and every agent can propose new states. **Q**uorum (committee) means that a subset of agents is selected to run the current round or make decisions. **L**eader means that a single agent has special responsibilities.
4. **Resource [V/A/H]:** The resource that inherits the right to vote. **V**ote means that every agents has exactly one vote and the same voting power. **A**sset means that the voting power scales proportional according to a virtual agreed-upon asset, such as *stake* or *time*. In this sense a vote is a special type of asset but since it is so common, it gets its own letter. **H**ardware means that voting power scales proportional with physical hardware properties, which could be computational power or memory size.
5. **Authentication [O/W/T]:** The authentication mechanism for messages. **O**ral messages use symmetric Massage Authentication Codes (MACs) that are only verifiable between two agents. **W**ritten messages use PKC and signatures, which can be verified by any agent even after the message has been relayed several times. **T**hreshold signatures are special signatures that require multiple agents to cooperatively sign a message before the signature is valid.
6. **Conflict Resolution [D/P]:** The mechanism used to converge to agreement in case the protocol gets into a state that is undecidable. **D**eterministic approaches will always find the correct majority while **P**robabilistic approaches use some randomness to converge.
7. **Certainty [C/x]:** Certainty is a guarantee that is satisfied if a transaction is certainly accepted after running the protocol. Non-certain protocols have asymptotic finality, where transactions could still change but the probability for a change approaches 0 and the probability for validity approaches 1 over time.
8. **Fairness [F/x]:** Fairness is a guarantee that is satisfied if valid transactions are always applied in the same ordered as they have been received by a majority of agents. This property is only possible for SMR protocols. For example, Bitcoin is not fair because miners can select which transactions will be included in a block.

An overview of consensus protocols according to these dimensions is given in Table 2.2. For example, the well known protocol Practical Byzantine Fault Tolerance (PBFT) has the

Name	Year	Prob.	Notation	Faults	Messages
LSP-O [LSP82]	1982	bA	BS – LVOD – Cx	$3f+1$	–
LSP-W [LSP82]	1982	bA	BS – LVWD – Cx	$2f+1$	–
Rabin [Rab83]	1983	bC	BA – EVWP – Cx	$10f+1$	$O(n^2)$
Ben-Or [Ben83]	1983	bC	BA – EVOP – Cx	$5f+1$	$O(2^n)$
Paxos [Lam98]	1998	C	FS – LVOD – Cx	$2f+1$	$O(n)$
PBFT [CL99]	1999	SMR	BS – LVOD – Cx	$3f+1$	$O(n^2)$
ABBA [CKS00]	2000	bA	BA – EVTP – Cx	$3f+1$	$O(n^2)$
PoW [Nak08]	2008	SMR	BA – E**H**WP – xx	$2f+1$	$O(n)$
Aardvark [Cle+09]	2009	SMR	BS – LVWD – CF	$3f+1$	$O(n^2)$
PoS [KN12; Sal20]	2012	SMR	BA – E**A**WP – xx	$2f+1$	$O(n)$
RBFT [AM13]	2013	SMR	BS – EVOD – Cx	$3f+1$	$O(n^3)$
BChain [Dua+14]	2014	SMR	BS – LVOD – Cx	$3f+1$	$O(n)$
Raft [OO14]	2014	SMR	FS – LVOP – Cx	$2f+1$	–
Hashgraph [Bai16]	2016	SMR	BA – EVWP – CF	$3f+1$	$O(n^2 \log n)$
Tendermint[Buc16]	2016	SMR	BS – LVWD – Cx	$3f+1$	–
HoneyBadger [Mil+16]	2016	SMR	BA – EVTP – CF	$3f+1$	$O(n^2 \log n)$
Aleph [Gąg+19]	2019	SMR	BA – EVTP – CF	$3f+1$	$O(n^2 \log n)$

Table 2.2: Overview of a subset of consensus algorithms and their characteristic features. The solved **Prob**lem can be Binary Agreement (bA), Binary Consensus (bC), or State Machine Replication (SMR). **Notation** is our notation mentioned in Section 2.1.2. **Faults** number of required nodes, if f faulty nodes should be tolerated. **Messages** is the size complexity of data being transmitted.

notation BS-LVOD-Cx, which means it tolerates Byzantine faults (B) and assumes partially synchronous communication (S). Furthermore, PBFT relies on a leader (L) and uses voting (V) with symmetric message authentication (O) to deterministically (D) reach agreement. It can guarantee certainty (C) but not fairness (x instead of F).

In the following we will explain the details of PBFT to provide a better understanding on how consensus protocols in general solve the consensus problem.

Practical Byzantine Fault Tolerance (PBFT)

PBFT, which was developed in 1999 by [CL99], is one of the most important BFT consensus algorithms and serves as the basis for many other algorithms developed later. Understanding the principles of PBFT is vital for understanding the mechanisms that make a protocol tolerate Byzantine faults. We will therefore illustrate the working principle of PBFT before continuing with consensus protocols for CPS.

PBFT solves the SMR problem and consists of a network of N nodes. Every node runs a local state machine and the state machines are synchronized by making sure that all agents apply the same operations to their local machine. All these operations are stored in an ordered list, such that any new node could reach the current system state by applying the list of operations the same order on a fresh state machine.

In PBFT one node is selected as primary. The primary is responsible for selecting and

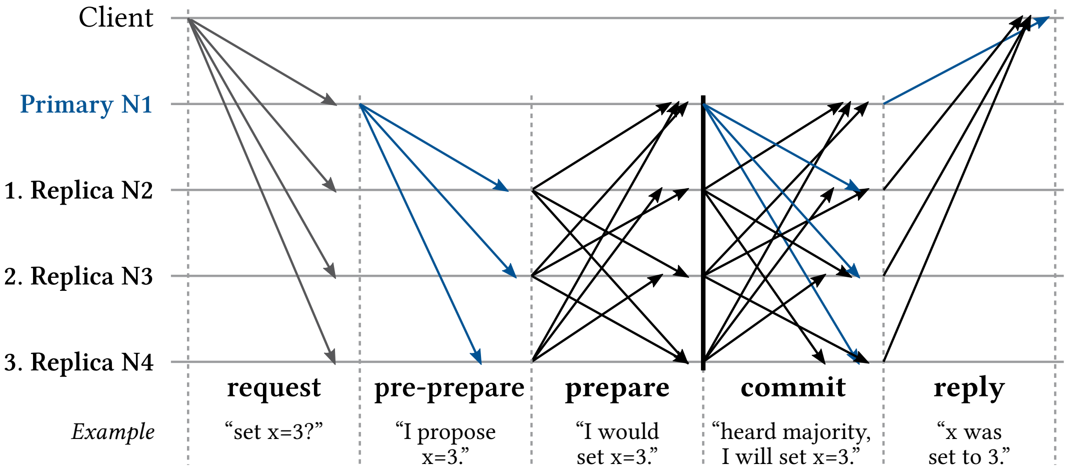

Figure 7: Illustration of the PBFT consensus protocol with one client and four nodes. Once the client sends a request, the primary starts a new consensus round, which consists of the three phases *pre-prepare*, *prepare* and *commit*. After a node commits, it will send its reply to the client. N1 is the primary node and N2-N3 are the replicas.

proposing the next operation that should be applied to the local state machine by all agents. The node index i of the primary is stored in a variable v called *view* and all agents need to know the current view number. Thus, the protocol starts with a pre-defined view and the view is also synchronized with the operations. The protocol runs in rounds and the current round number is r. In each round, the nodes try to agree on an operation x by exchanging messages. Each round has 5 phases: *Request*, *Pre-Prepare*, *Prepare*, *Commit*, and *Response*. However, the phases *Request* and *Response* are not directly involved in the consensus round.

PBFT Protocol Sequence The sequence of message transmissions for normal operation is illustrated in Figure 7. As an example, imagine the client C wants to execute the operation $x = 3$ in round $r = 7$ where N1 is the primary (view $v = i = 1$). Then the protocol would run as described in following. Note that for simplicity we assume that messages are signed with a signature σ, which is also a valid mode of operation for PBFT. However, PBFT does not require signatures and MACs could also be used.

1. **Request:** The client C sends a request with an operation x at time t to all nodes: $m_1 = \langle \textsc{Req}, x, t, C, \sigma_C \rangle$
2. **Pre-Prepare:** The primary $N1$ sends a pre-prepare message to all replicas: $m_2 = \langle \textsc{PrePre}, v = 1, r = 7, h(x), N1, \sigma_{N1} \rangle$, where $h(x)$ is the hash digest (digital fingerprint) of x.
3. **Prepare:** Each replica Ni with index i that accepts the PrePre message, responds with a prepare message to all other replicas (including the primary): $m_3 = \langle \textsc{Pre}, v =$

$1, r = 7, h(x), i, \sigma_i\rangle$

4. **Commit:** All replicas that have received $2f$ agreeing Pre messages, will apply the operation x and broadcast a commit message: $m_4 = \langle\textsc{Commit}, v = 1, r = 7, h(x), i, \sigma_i\rangle$.
5. **Response:** The client waits for $f + 1$ replies with valid signatures from different replicas. The replicas that have committed the operation will send a reply directly to the client: $m_5 = \langle\textsc{Reply}, v = 1, t, C, i, \sigma_i\rangle$.

Correctness of PBFT. PBFT solves the SMR problem despite of failures by guaranteeing *agreement*, *integrity*, and *termination*. We will outline the ideas of the proofs which can be found in the original paper [CL99]. First, *agreement* is achieved by requiring $2f$ agreeing prepare messages before committing. Since a maximum of f replicas can experience faults (crash or malicious), we have at least $2f + 1$ correct replicas following the protocol correctly and thus every replica will receive at least $2f$ honest messages. If we start from a synchronized state, then those $2f + 1$ correct replicas will all agree. From the 5 phases above, it is easy to see that they will all commit the same value. If the correct replicas are split into different states, e.g. f replicas are in state A and $f + 1$ replicas are in state B, then the malicious replicas can only convince the $f + 1$ replicas in state B to have a strong majority of $2f + 1$ agreeing replicas. However, the malicious replicas can only form a weak majority of $2f$ replicas with group A because every node in group A would receive only $2f - 1$ agreeing messages. Thus, f malicious replicas can never convince two correct replicas to commit conflicting operations.

Second, *integrity* is easily achieved by not allowing correct nodes to sent another prepare or commit message with different content within the same round r and view v. Only the same message might be repeated. Thus, if a correct replica commits an operation, it is final and will not change.

Third, *termination* (liveness) is ensured by reuniting partitioned replicas and replacing faulty primaries via *view changes*. In case the correct replicas were split into A and B and the malicious replicas helped to commit B, then at least $f+1$ replicas will broadcast commit messages. Once the partition is resolved, the replicas in group A will receive the missing prepare messages from group B and also commit B. If instead the malicious replicas remain silent or the primary is slow or faulty, we will not make progress and have no commit phase. In this case, the correct replicas will wait until a timeout timer expires and then send a *view change* request to switch to a new primary. If $f + 1$ view change requests are received, the replica enters the *view-change* subprotocol. Replicas in the *view-change* mode will stop accepting consensus messages until the *view-change* subprotocol has moved to a new view $v + 1$ and a new primary sends a new pre-prepare message within the new view. When the new primary has collected a majority of view change votes, it broadcasts a

New-View message as a reply to all other nodes that confirms the change.

Essentially, the timeouts will ensure progress in case of a slow or faulty primary but they are also the reason PBFT consensus is a partial synchronous protocol.

Evolution of Consensus Protocols

As illustrated by Table 2.2, consensus protocols have been evolving over time with different properties. While the table only shows a selection of protocols and furthermore cannot capture every detail about each consensus protocol, some trends are visible.

Since asynchronous communication is the weaker assumption compared to synchronous communication, it is often the choice of later protocols. However, due to the FLP impossibility, a consensus protocol that assumes asynchronous message passing, must either sacrifice *certain finality* of transactions (e.g. PoW) or use a random selection of transactions (e.g. Ben-Or) to reach agreement. Recent protocols, such as HoneyBadger, utilize a *random common coin*, which is basically a random value generated with cryptographic threshold signatures. This common coin is used to resolve conflicts and ensures that certain finality can still be guaranteed.

Furthermore, there is a trend towards leaderless protocols because a leader can always abuse its special responsibilities to degradate the performance of the network. If protocols do not rely on leaders, it is also easier to guarantee *fairness* because every agent can broadcast its transactions at any time and the honest majority of receivers will include valid transactions. More details on PoW will be presented in Section 3.1.2 and HoneyBadger is described in Section A.2.1. In the following, we will discuss how consensus protocols like PBFT can be use in a CPS.

2.2 Using Consensus Protocols for Cyber-Physical Systems

As stated in the introduction, the IoT does not only consist of a virtual world but is closely connected to the physical world via sensors and actuators within smart devices, which are then referred to as Cyber-Physical System (CPS). However, the relation to physical processes has not been considered in most of the previous work on consensus protocols that is shown in Table 2.2. Conventional consensus protocols only focus on the synchronization and integrity of arbitrary data among homogeneous nodes without considering the potential physical meaning of the data.

This is also true for existing consensus implementations in control systems. For example, the ARINC 659 SAFEbus used in the airplane *Boeing 777* uses consensus to agree on the calculation from three redundant processors for flight control [Yeh01], but the consensus protocol uses (simple) median selection on the final output without considering its physical meaning[1]. Furthermore, the same input is provided to all processors and thus the consensus is not considering any differences in the sensing capabilities but focuses only on tolerating faults in the computation or the communication.

In this section, we will investigate if and how we could incorporate the physical aspects into consensus protocols and discuss the general assumptions and challenges, before we propose specific approaches in the next sections.

2.2.1 Our System Model for Consensus

In the remainder of this chapter we will in general use the following system assumptions and notation.

A consensus system consists of N agents $a_i \in \mathbb{A} = \{a_1, a_2, ...a_N\}$ that communicate via message passing. In the message-passing model we assume

- **Packet Loss:** Messages can get lost with a certain probability < 1 but when messages are repeated continuously, eventually one of the messages will get delivered successfully.
- **Partial Synchrony:** Message delays have no upper bound δ_{max} but messages will be delivered within a chosen timeout interval δ_{to} after some unknown stabilization time T_{GST}.
- **Authentication:** the sender of a message can be authenticated by the receiver, which means that the receiver is able to verify the integrity of the message and the identity of the sender. This is possible by PKC or MACs.

Furthermore, we make two more assumptions about the characteristics of agents:

- **Known participants:** Each agent $a_i \in \mathbb{A}$ knows all other agents within $\mathbb{A}$ and can authenticate their messages. If $\mathbb{A}$ changes (join or leave), this has to be synchronized among all agents in $\mathbb{A}$ before the next consensus round.
- **Faulty agents:** Up to f of these N agents might be faulty. Depending on specific scenario, a fault might be either a crash of the agent's process or any malicious behavior, which means a faulty agent might deliberately send incorrect or confusing messages. Following this convention, there are $N - f$ correct agents, which will execute the protocol correctly.

Our Consensus Problem Definition

Under the previously stated system assumption, the set of correct agents tries to decide on a common value. As we have mentioned before, the general consensus problem is considered

[1] The processors itself might perform physical validation but this is not considered a part of the consensus.

solved if three properties hold:

- Agreement: All correct agents decide the same value.
- Integrity: All correct agents decide only once.
- Termination: All correct agents decide eventually.

The above properties ensure that all correct agents will decide the same value but they will not ensure that the decided value is correct by any correctness function. If we think back to our lunch example from the beginning of this chapter, then the above consensus properties are sufficient because we just want to reach a common agreement but there is no "correct" or "wrong" lunch location.

However, if we think about IoT applications that manipulate physical processes, then there is an implied correctness for some values. For example, if we have several temperature sensors that should agree on the average room temperature then we want them to agree on the *actual* average temperature value and not just on any temperature value.

Therefore, for most IoT applications, the decision should also be correct with respect to some policy or desired behavior. Derived from the notion of [SB12], we require two additional properties to solve the Consensus problem:

- Validity: The decision of a correct agent was validated by a correct agent.
- Provability: The validity of a decision can be verified by any agent.

Traffic Management as CPS Scenario

For exploring consensus protocols in the general context of CPS, we have chosen traffic management with Connected Autonomous Vehicles (CAVs) as a specific but representative example for any type of mobile agent that can sense and move within a physical environment, such as robots, drones, or autonomous boats. We see moving agents as weaker assumption compared to static agents and thus most results with moving agents should also be applicable to static agents. Otherwise it should in principle be easier to transfer the results for moving agents to static agents than in the other direction.

Within this traffic management context, we assume that vehicles cooperate in two scenarios:

1. **Platoon Coordination:** Vehicles on the same lane travel together in small groups called *Platoons*. Vehicles within the same platoon drive closely together to reduce air drag and try to coordinate certain maneuvers, such as lane change or velocity change [Jia+16; Amo+15].
2. **Intersection Scheduling:** When reaching an intersection, vehicles will not follow conventional right-of-way rules, but will try to agree on an alternative crossing

schedule in order to reduce the total waiting time and required velocity changes [CE16].

The details of these two scenarios and how consensus can be applied, will be further described in the following Section 2.3 and Section 2.4. For now, it is just important to know the general ideas in order to understand the potential implications for consensus protocols within these scenarios.
While one could argue that vehicles are not directly embedded systems, this assumption relies mostly on the fact that their current implementation is powered by gasoline. However, smaller delivery bots, which could even be solar powered would in principle fall into the category of embedded and resource-constrained devices.

2.2.2 New Challenges within CPS

Solving consensus in self-organized CPS is more difficult than solving consensus for database systems due to several reasons that will be presented in the following in the context of traffic management.

Heterogeneous Agreement

In contrast to conventional consensus systems where each agent can validate and vote on all possible requests, not all vehicles might be able to validate the same set of requested transitions [Xu+17]. The main reason for this problem is the locality of physical objects and physical data. If we need to agree on physical data at a certain location then often only sensors that are placed near or directly at this location can measure and validate the physical data. For example, a vehicle approaching the tail of a platoon can be sensed easier and with more details by the last vehicle of the platoon. We therefore need to find consensus protocols that can account for these differences in validation capabilities.

Real-Time Requirements

Another aspect are real-time requirements of physical processes. Conventional consensus protocols focus mostly on safety and consistency and less on availability and performance. For database systems, the integrity of the data is most important and it often does not harm the system if reaching consensus takes longer than expected.
However, in the physical domain, some operations need to be finished in time because physical processes will continue regardless of the data and states in the cyber domain [CSB19]. For example, vehicles approaching an intersection need to reach an agreement on whether they can continue with the current velocity, have to slow down, or stop at the intersection *before* they reach the intersection at full speed.

If we want to reach consensus about such physical operations with hard deadlines, we should not rely on optimistic protocols that could take much longer in the case of conflicts. At least, we need to specify clear default rules, in case a decision can not be reached until a certain critical point in time. In automotive domain, we will solve this challenge by assuming that it is safe to stop the vehicle at certain points, e.g. *in front* of an intersection, where we can then wait for the consensus protocol to finish. However, in other domains there might not be such a simple solution. For example, in aviation, we cannot simply stop an airplane during flight and thus real-time requirements in consensus protocols remain a difficult challenge.

Zero Fault Tolerance

Conventional consensus algorithms, which are mostly applied to databases, can tolerate a certain amount of f failures as long as a sufficient majority $(2f + 1)$ operates correctly. Since, the correct state of such a system is determined by this majority, each agent can determine the current state by receiving a strong majority of correct votes. The remaining votes of unresponsive, outdated, or malicious agents can simply be ignored. As a result, the final decisions of faulty agents have no influence on the decisions of correct agents, allowing correct agents to maintain a consistent system state [CL99].

However, for CPSs these assumptions might not apply and solving consensus is fundamentally different from conventional consensus. Platoons, for example, are safety critical CPSs and the safety of a single passenger weights more than the decision of any majority. As a result, we cannot tolerate a single negative vote or failure for certain decisions.

In our consensus scheme, we therefore require *all* participants to agree on the same state transition. Note that the goal of our scheme is not to tolerate failures but to reliably detect if an unanimous decision was reached by consensus, or in case consensus could not be reached, detect which vehicles are responsible for the decision failure. These responsible vehicles could have failed to transmit their vote, voted against the decision, or tried to send a malicious message.

Agreement Execution: Promise vs. Reality

Considering a platoon in which vehicles want to perform a merge operation, then reaching consensus is only a virtual agreement of the vehicles to perform certain actions. Besides changing some virtual data structures, these actions could also involve a physical process that can be blocked, slowed down, or changed by external constraints.

Many physical processes within the platoon require each vehicle to participate and work correctly. Even in the case consensus could be reached, it might not be possible to execute the overall operation safely. In the moment of execution, a vehicle that promised to perform a certain action might be forced to postpone or stop the action, or even perform a completely different action to ensure the safety of its passengers. Despite external influences that are beyond the consensus decision, any vehicle could also be subject to a critical failure or even maliciously block operations within the platoon by its mere physical presence.

Therefore, we try to reliably detect any failures and then separate sub-platoons from the failed vehicle.

Since failed or malicious vehicles pose a severe risk to other vehicles, another important goal is to distinguish these vehicles from vehicles that were forced to deviate from the consensus but otherwise operate correctly and honestly.

The matter gets more complicated as we also need to consider that malicious vehicles may send false messages but not all vehicles might be able to sense the true nature of the deviation.

In our platoon scenario we therefore distinguish two types of consensus rounds:

1. consensus about a planned action, which requires all vehicles to agree.
2. consensus whether and which vehicle failed, which requires $f + 1$ vehicles to agree.

In the following sections, we will discuss how these challenges can be solved or mitigated for the two scenarios of *platoon coordination* and *intersection scheduling*.

2.3 Managing Vehicle Platoons with Consensus[2]

Platoons are energy efficient, communication efficient and increase the road throughput by allowing the vehicles to drive with a reduced inter-vehicle distance [Jia+16].

Most conventional platoons declare a leader that is responsible for managing the platoon behavior and communication [Amo+15; MWK17]. However, participating vehicles need to trust the leader, which offers many attack vectors on the security, safety, and performance of the platoon itself as well as third parties that might query information about the platoon from the leader.

We therefore aim for a consensus-based platoon management where decisions must be approved by all vehicles as shown in Figure 8. This distributed management would increase the robustness of the platoon by reducing the chance that failures are not detected. Since all vehicles are involved, each vehicle will observe and approve the messages of other vehicles. While this increased communication overhead is often seen as a drawback compared to a centralized management, we argue that the number of vehicles in one platoon is small enough to keep the overhead low.

Overall, platoons are highly relevant CPSs but the conventional leader-based management poses severe risks on the security and safety of all platoon vehicles because it inherits

[2] Major parts of this section have been published in [RS19].

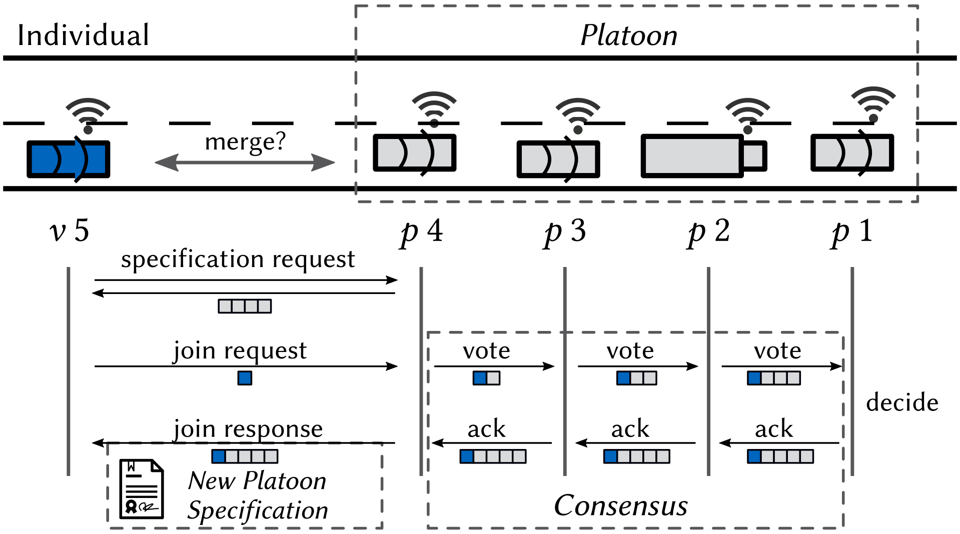

Figure 8: Our idea of a consensus-based join maneuver. An individual vehicle requests to join an existing platoon. The join request is forwarded to each platoon vehicle, which will vote on the request. If all vehicles agree, the new vehicle is accepted and the new specification is sent to all vehicles.

a single point of failure by design. We therefore propose a distributed, consensus-based management of the platoon, analyze the challenges that arise for this specific application, and provide a first solution to demonstrate feasibility.

2.3.1 Model of Vehicle Platoons

For describing and evaluating the platooning scenario, we consider a set of vehicles $\mathcal{V} = \{v_1, v_2, ...\}$ on a highway and some of them form a platoon $\mathcal{P} = \{p_1, p_2, ...\} \subseteq \mathcal{V}$. Vehicles v_x can join a platoon $\mathcal{P}$, two platoons $\mathcal{P}_1$ and $\mathcal{P}_2$ can merge to $\mathcal{P}_3 = P_1 \cup P_2$ and a vehicle v_x can leave a platoon $\mathcal{P}' = \mathcal{P} \setminus \{v_x\}$. As pointed out by [Amo+15], we assume a maximum platoon size of $|\mathcal{P}| = N = 20$. Each vehicle has specific properties, such as color, length, or maximum velocity.

Interaction Topologies

Due to the linear structure of a platoon, the communication and sensing capabilities of each vehicle can be described in general by the range in forward and backward direction within the platoon.

Following the notation of [Zhe+16], we consider three communication topologies (see Figure 9) that are suitable for consensus based platoon management:

- Predecessor – Successor (PS)
- Two-Predecessor – Two-Successor (2P2S)

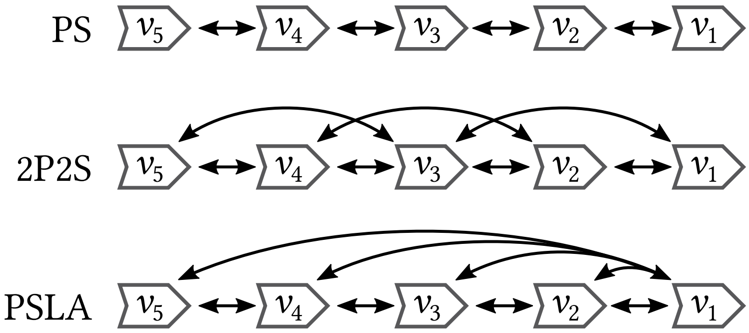

Figure 9: Considered communication topologies of neighboring vehicles. An arrow indicates that a vehicle can directly send messages using a wireless V2V VANET.

- Predecessor – Successor – Leader to All (PSLA)

Despite communication, we also need to consider the sensing topology of each vehicle. Sensing is important to verify the sent claims of vehicles, such as its physical presence and its identity. Since there are numerous vehicle properties that could be sensed in different ways, it is very difficult to model all sensing capabilities in a unified topology graph. We leave the general specification of heterogeneous sensing topologies as an open problem for future work. For simplicity, we only consider sensing the license plate of a vehicle and that each vehicle is able to read the license plate of its predecessor and successor in order to verify the authenticity of a vehicle. This sensing topology would correspond to the Predecessor-Successor (PS) communication topology.
We furthermore assume that it is possible to verify a vehicle specification according to a license plate using a trusted third-party certification service or distributed certification (e.g. using blockchain) [Row+17]. Further details on how Blochain can be used to certify, for example, the public key that belongs to a license plate will be discussed in Chapter 3.

Consensus Roles

In addition to different communication and sensing topologies, we also use seven distinct roles to describe the responsibilities of vehicles in our consensus system:

- *Requester:* vehicle that requests an operation
- *Receiver:* processes requests from a *Requester*
- *Responder:* responds to *Requester* when consensus is reached
- *Proposer:* proposes a new system state
- *Validator:* can validate a proposed state
- *Acceptor:* can vote for a proposed state
- *Learner:* will receive accepted state

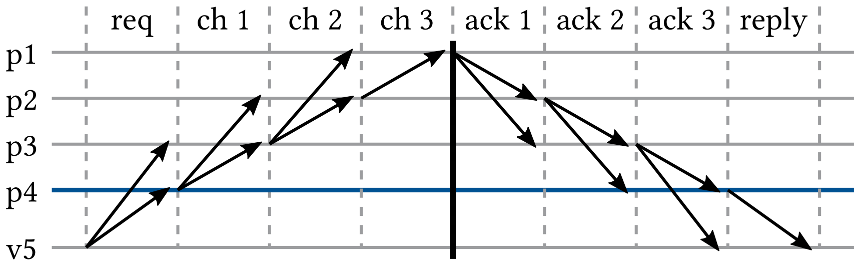

Figure 10: Join request, normal operation, 2P2S: Vehicle v_5 sends the request to the platoon vehicles p_4 and p_3. p_4 will start a consensus round and when p_1 receives a valid chain of accepting votes, it decides for the new platoon and sends an ACK back to v_5.

We will use these terms in the following to refer to vehicles with the respective role.

2.3.2 Our CUBA Protocol

We now introduce our Chained Unanimous Byzantine Agreement (CUBA) protocol, which is suitable for platoons and other distributed CPSs that want to reliably detect failures because they cannot afford to tolerate (ignore) any single failure. CUBA works by passing messages hop-by-hop instead of using broadcasts. Each hop confirms the messages sent by previous hops. This concept is similar to BChain [Dua+14], a general consensus protocol, which, however, is *not* feasible for platoons. For a detailed discussion, refer to the related work in Section 2.3.3.

In contrast to BChain, our protocol is designed to terminate successfully only if all involved acceptors agree on the same value. In case no consensus can be reached, our protocol will detect which *Acceptor* was responsible for the failing consensus, such that the correct nodes can take action. The maximum number of failures f that can reliably be detected depends on the underlying network topology.

Role Assignment

We apply the following roles:

- Any vehicle v_x (including platoon vehicles) is a Requester.
- Any platoon vehicle p_x is Acceptor, and Learner.
- The platoon vehicles at the top p_1 and tail p_N are Receivers, Responders, and Proposers.
- The direct neighbors of a vehicle v_x are Validators for that vehicle as they can physically sense v_x and read its license plate.

Normal Operation

In normal operation, all vehicles operate correctly and need to agree on the same proposal. The proposal could be any planned platoon operation. We use four message types: $\langle \textsc{Ch} \rangle$ (chain), $\langle \textsc{Ack} \rangle$ (acknowledgment), $\langle \textsc{Nak} \rangle$ (no acknowledgment), and $\langle \textsc{Spt} \rangle$ (suspect).
Each message is structured as $\langle T, s, (h), l_o, (l_n), (m), \sigma \rangle$ where fields in parentheses are optional. T is the type of the message ($\langle \textsc{Ch} \rangle$, $\langle \textsc{Ack} \rangle$, $\langle \textsc{Nak} \rangle$, $\langle \textsc{Spt} \rangle$), s is an integer indicating the sequence number of the current consensus round, h is the cryptographic hash of the previous message, l_o is the own license plate number of the sender, l_n is the license plate number of the next (succeeding) vehicle if present, m is the proposed message or state to vote upon, and σ the signature of the sending vehicle.
The protocol execution is illustrated in Figure 10.

1. A Proposer proposes a new state or planned operation and forwards the proposal in form of a $\langle \textsc{Ch} \rangle$ towards the other end of the platoon.
2. Each intermediate vehicle validates the proposal, and votes for it by appending its own $\langle \textsc{Ch} \rangle$ message to the proposed $\langle \textsc{Ch} \rangle$. If an intermediate vehicle receives several chained $\langle \textsc{Ch} \rangle$ messages, it will first validate each vote by verifying four predicates:
   a) the sequence number in each message matches the current sequence number,
   b) h of the current message matches the hash of the previous $\langle \textsc{Ch} \rangle$,
   c) l_o of the message matches l_n of the previous message,
   d) the signature σ is valid using the public key corresponding to l_o.
3. Once the last Acceptor (other Proposer) received the proposal and all votes, it decides. If all votes agree on the proposal, it decides for the proposal and sends an $\langle \textsc{Ack} \rangle$ including all signatures back to the other Acceptors (now in the role of Learners). If there is one vote against the proposal, it decides against the proposal and sends an $\langle \textsc{Nak} \rangle$ together with *all* signatures.
4. Each intermediate vehicle (now Learner) receives and validates the signatures of the $\langle \textsc{Ack} \rangle$ or $\langle \textsc{Nak} \rangle$ and decides accordingly until the last Learner (=Proposer) is reached.

To ensure that all vehicles will receive the $\langle \textsc{Ack} \rangle$ of a successful consensus round we make the critical assumptions that each vehicle can send messages to the next $f+1$ vehicles in one direction. Otherwise we could not guarantee that decisions for a proposal are propagated to all vehicles. This assumption limits the possible topologies to 2P2S.

Example: Platoon Formation

Platoon formation can happen between two single vehicles, between a single vehicle and an existing platoon or between two existing platoons. The Proposers of a platoon will also

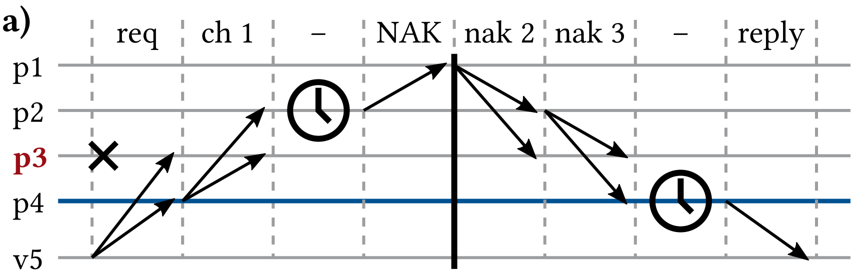

Figure 11: Join request, p_3 failure/timeout: When p_4 starts the consensus round, p_2 receives the message and starts the timeout timer. Since p_3 is unresponsive, p_2 will wait until the timeout and then forward a ⟨Nak⟩.

react to join requests from vehicles outside of the platoon. In the following, we consider a single vehicle v_5 that wants to join an existing platoon $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$ as illustrated in Figure 8. The single vehicle v_5 approaches the tail vehicle p_4 of the platoon until it can read its license plate and then runs the following protocol:

1. The Requester v_5 requests and receives the current platoon specification ⟨Spec⟩ from p_4.
2. The Requester v_5 verifies ⟨Spec⟩ and decides whether it wants to join.
3. The Requester v_5 sends a join request to the tail p_4. The request contains information about the vehicle v_5.
4. The Proposer p_4 starts a new consensus round by forwarding a ⟨Ch⟩ message including the join request.
5. When the consensus round was successful, the Proposer p_4 replies with the new platoon specification ⟨Spec⟩ including all ⟨Ch⟩.
6. The Requester v_5 verifies ⟨Spec⟩ and if valid, accepts it and becomes p_5. The next consensus decision now requires votes from 5 platoon vehicles.

Initial Formation The initial platoon formation between two individual vehicles v_1 and v_2 works similar to the described join maneuver. The difference is that the Requester v_2 will receive a ⟨Spec⟩ that includes only v_1. If v_2 sends the join request, v_1 can immediately decide as it represents a platoon with only one vehicle. Once v_1 appends a ⟨Ch⟩ with its signature to the request and sends the chain back to v_1, a new platoon $\mathcal{P} = \{p_1, p_2\}$ is formed.

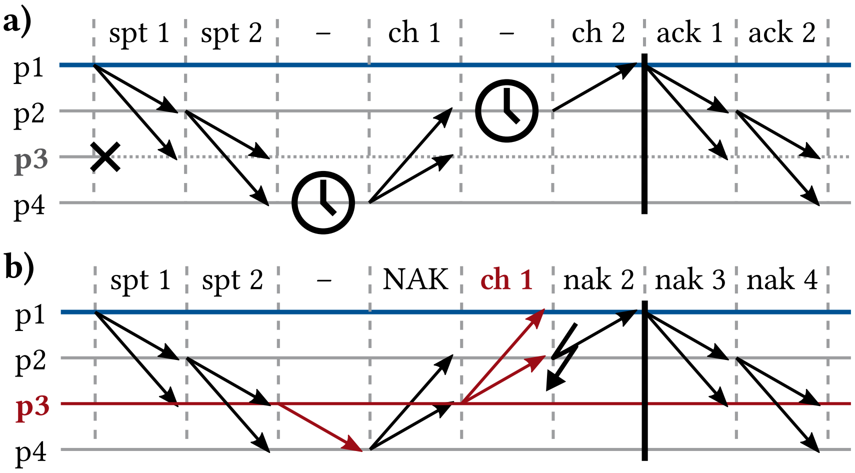

Figure 12: Suspect round to determine which vehicle failed proposed by p_1. a) p_3 was suspected by p_4 and really timed out. b) p_3 is malicious and suspected p_4, but p_4 is still running.

Failed Consensus

Each vehicle sets a timer with the time period that corresponds to the expected consensus execution time for the remaining set of vehicles. This time is calculated as

$$t_{\mathrm{TO}} = (N - i) \cdot \tau \tag{2.1}$$

where $(N - i)$ represents the number of remaining vehicles that need to vote and τ is a fixed and pre-defined timeout value for every vehicle. In case, the successor vehicle can provide a consensus proof (valid ⟨Ack⟩) before the timer runs out, the vehicle decides for the consensus value and forwards ⟨Ack⟩. In all other cases, the vehicle decides that the consensus failed and forwards a ⟨Nak⟩ in which case it *may* also include the ID of another vehicle it suspects to deviate from the protocol or to have timed out.

An example for a timeout of p_4 is shown in Figure 11. p_2 will wait until p_3 times out and then forwards a ⟨Nak⟩. When the ⟨Nak⟩ returns from p_1, p_2 will forward it to p_3, giving it a second chance to respond. Forwarding the messages to all vehicles despite an early ⟨Nak⟩ is important for determining the failed vehicle. p_4 will wait until its timeout timer runs out and decides that the consensus failed.

Under the assumption that it is not possible to forge messages, consensus is only reached if and only if *all* vehicles respond correctly and in time $< \tau$. Any non-correct response will result in a non-valid ⟨Ack⟩ or a timeout. Therefore, each vehicle is able to reliably detect a failed consensus.

Detection of Failing Vehicle

However, detection of the vehicle that is responsible for the failure is more complicated. For example, a malicious vehicle could falsely accuse a neighboring vehicle to have timed out.

After a failed consensus round, the last acceptor checks if the ⟨Nak⟩ includes a suspected vehicle and – when true – starts a suspect round. This special consensus round requires only $f + 1$ chain votes from neighbors in communication range of the suspected vehicle to be successful because with a maximum of f faults at least one vote will be correct. Furthermore, before chaining votes, ⟨Spt⟩ messages are forwarded to all vehicles. This serves two purposes: 1. each vehicle knows which vehicle is suspected, 2. each neighbor of the suspected vehicle has the chance to observe a timeout or conflicting messages of the suspected vehicle before deciding against it.

An example of a suspect round for a timeout of p_3 is illustrated in Figure 12a which would be triggered by p_1 after the timeout round in Figure 11. In the normal round, p_2 would suspect p_3 for the timeout and after the consensus failed, p_1 would start a suspect round suspecting p_3 by forwarding a ⟨Spt⟩ message. p_4 would witness another timeout by p_3 and voting with a ⟨Ch⟩ against it. Once the ⟨Ch⟩ reaches p_2 and p_3 does not respond in time, p_2 will also vote against p_3, reaching $f + 1$ votes. Thus, p_1 will decide that p_3 timed out and forwards the decision with the signatures of the $f + 1$ votes in an ⟨Ack⟩ message. Once the failed vehicle is identified, the platoon is split to ensure the safety of the remaining vehicles.

For the network topology, we need $f + 1$ communication hops in both directions to reliable identify f failing vehicles, which means a 2P2S topology for $f = 1$.

2.3.3 Evaluation: Releated Work and Discussion

In this section we describe related work and then discuss the advantages and drawbacks of CUBA. To illustrate the general feasibility of our protocol, we evaluate the performance of CUBA analytically. While an experimental validation would be interesting, CUBA and related approaches follow deterministic rules, which justify an analytical evaluation on certain metrics, such as message complexity.

Related Work

BFT-ARM Platooning The authors of [Weg+16] designed BFT-ARM, a consensus protocol for continuous sensor values in an asynchronous inter-vehicle network. The protocol uses median validity, where the decided value is only required to be close to the median of all correctly proposed values and claims to tolerate up to $f < \frac{N}{3}$ Byzantine nodes. This is

Protocol	Messages per Round
PBFT	$2N^2 - 2N$
BFT-ARM	$3N^2 - N - 2$
BChain	$2Nf + N - 4f^2 - 1$
CUBA	$2Nf + 2N - f^2 - 3f - 2$

Table 2.3: Number of messages per consensus round depending on the number of agents N and the maximum number of possible failures f.

achieved by calculating the median only over the sorted $(2f + 1)$ middle values in the full range of all $(3f + 1)$ proposed values, cutting off $f/2$ values at each end of the range.
The protocol is focused on *tolerating* some faulty measurements in order to agree on a common value and is not designed for safety critical decisions that require the agreement of all involved vehicles. Furthermore, the protocol relies on a trusted subsystem which provides unforgeable counter values. Overall, the application focus of BFT-ARM is different from our protocol and thus not suitable for platoon management.

BChain BChain [Dua+14], is a general consensus protocol that does not work for platoons in its original form but shares some concepts with our CUBA. Nodes are ordered within a chain, which is divided into two parts: The first $2f+1$ nodes within the chain are acceptors and the last f nodes are learners. A request is forwarded hop-by-hop from the head (1st node) towards the acceptor tail (node $2f + 1$) using $\langle \textsc{Ch} \rangle$ messages. Once the acceptor tail receives and accepts a $\langle \textsc{Ch} \rangle$, it will send 3 messages: 1) a reply to the client, 2) an $\langle \textsc{Ack} \rangle$ message that traverses backwards to the head, and 3) its $\langle \textsc{Ch} \rangle$ to the learners.
In order to handle failures, each node starts a timer after sending its $\langle \textsc{Ch} \rangle$ and in case a timer expires before receiving an $\langle \textsc{Ack} \rangle$, the node will issue a $\langle \textsc{Spt} \rangle$ to the head. The head is then responsible for re-ordering the chain, such that malicious or crashed nodes are moved towards the end.
While such a reordering is possible in a consensus overlay network where the underlying network topology allows several routes between nodes, it is not feasible in platoons where the routes and connections depend on the spacial location of nodes. It would require changing the position of vehicles by overtaking maneuvers. Furthermore, requests must be always sent to the head, which increases the risk of blocking or slowing down the protocol execution if the head is faulty. Compared to our protocol, BChain can not guarantee to terminate in the first round but would require $3q$ rounds in the worst case, where q is the number of faulty nodes [Dua+14].

Consensus from Control Theory In control theory, algorithms such as the Average Consensus are used to solve a distributed control problem. While this family of algorithms is well-studied and suitable for controlling, e.g. the distance between platoon vehicles [Wan+12], it does not consider faulty or malicious agents but rather assumes that every agent is running and responding within a certain time period.

Discussion: Low Communication Overhead

Consensus Safety CUBA offers a safe consensus based platoon management. In contrast to majority-based consensus protocols, CUBA requires all vehicles to agree on the same decision and this decision is verified by all vehicles. Therefore, failures can be reliably detected, which prevents any unintended interactions between the vehicles. The sequential execution of all messages in an ordered chain also helps to avoid collision and retransmission of messages because only one vehicle is sending. Furthermore, the fixed timeouts guarantee a deterministic consensus execution within a bounded period of time, where BChain would perform expensive re-chaining in the case of failure.
In contrast to BFT-ARM, our protocol does not require *view changes*, where the Proposer is changed when it is suspected, as any identified failure will lead to a splitting of the platoon. Therefore, we achieve a simpler protocol that only utilizes two types of consensus rounds. Due to the use of signatures, it is not possible to produce valid proofs for wrong decisions as long as one vehicle is correct. Reliable detection of failures is possible as long as the vehicles can reach $f+1$ neighboring vehicles in each direction.

Verifiable Platoon Specification Using signatures during consensus also enables us to generate a platoon specification that was signed by all involved vehicles. Vehicles outside of the platoon can then query the specification and verify the signatures to ensure that the specification is correct and corresponds to the agreement. Such a specification could also be uploaded to a decentralized database, such as a blockchain, in order to allow an easy distribution of the specification.

Communication Overhead While conventional consensus protocols often measure the throughput of processed requests for state machine replication, we are interested in the number of messages that need to be sent to run one consensus round because the bandwidth is limited for VANETs. We therefore derived the equations in Table 2.3 for the number of messages from the protocol description in the corresponding papers. Note that the number of messages does not change with the number of actual failures q but only with the maximum number f. The reliability of a consensus-based approach introduces some overhead to the leader-based approach, for which we assume $2N$ messages. In this case, the leader sends messages to each vehicle and waits for an ACK from each vehicle assuming a PSLA topology. Figure 13 illustrates that all decentralized approaches require more messages than the leader-based communication but in contrast to BFT-ARM, CUBA

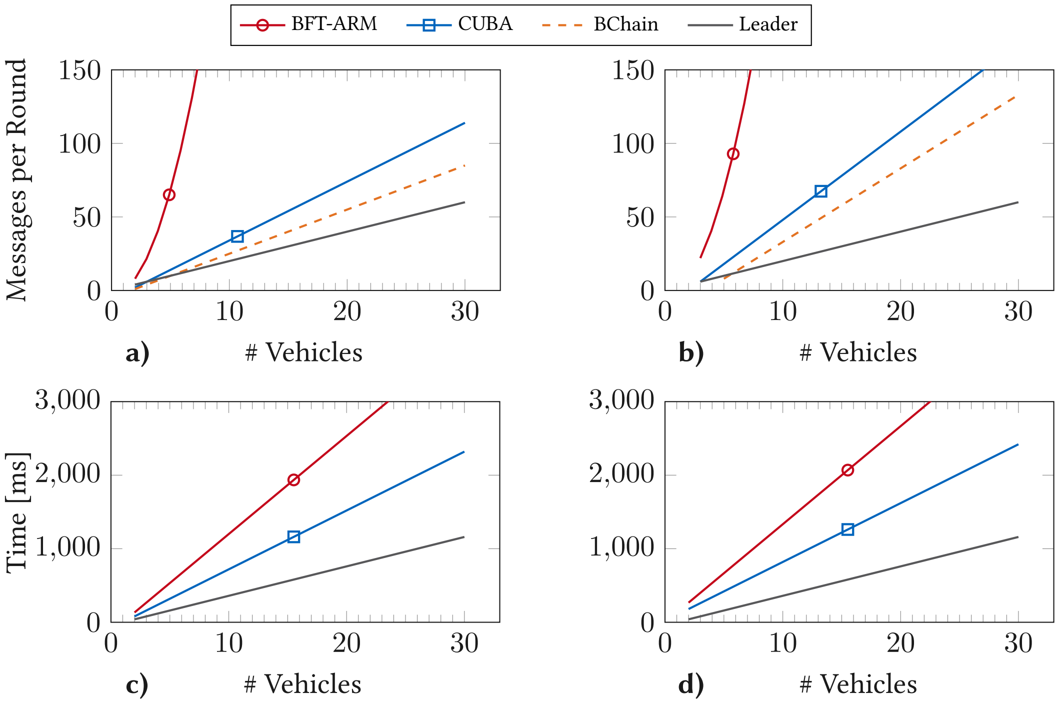

Figure 13: Calculated communication overhead based on Table 2.3. Top shows the required number of messages per round for different platoon sizes. a) assuming maximum possible failures $f = 1$ b) assuming maximum possible failures $f = 2$. Bottom shows the consensus time per round. c) in normal operation without failures ($f = 1, q = 0$). d) with one failure ($f = q = 1$).

also scales linear to the number of platoon vehicles. Furthermore, the messages are more equally distributed among the vehicles in CUBA compared to the leader-based approach, where the leader needs to process all messages. Note that BChain is not compared but only shown for reference because this protocol cannot be applied to platoons.

Another important metric is the overall time required to reach consensus, which is illustrated at the bottom in Figure 13. Here, we assume 40 ms latency between two vehicles for transmission and processing and 100 ms as the timeout timer value. For large platoons this results in consensus times around 2 s but CUBA always terminates faster than BFT-ARM due to the reduced rounds required to reach consensus. Timing for BChain is not shown as it would require re-ordering of vehicles for which do not have timing assumptions.

Summary of CUBA

We have illustrated the benefits of distributed, consensus based platoon management over conventional centralized and leader-based platoons and presented CUBA, a new consensus protocol for platoon management, which addresses the challenges of consensus in CPSs. CUBA focuses on failure detection and failing vehicle identification and guarantees to terminate in a fixed time window. For typical platoon sizes up to 20 vehicles, the communication overhead of CUBA is low compared to leader-based systems and significantly less than related consensus approaches for platoons.

The creation of a chain hop-by-hop ensures a linear scaling of the message overhead. However, the overall latency to reach a consensus increases faster compared to the leader-based approach because each message is forwarded along the platoon in both directions. This offers a trade-off in the design according to the assumed or allowed platoon size for a specific application.

We have analytically evaluated the performance without considering packet loss. While a certain probability of packet loss would affect all approaches, it would be interesting to evaluate the details in an experimental setup or simulation. Furthermore, our assumed constant latencies do not reflect the variance of real-world message delays. However, we still think that they are sufficient to reveal timing tendencies and provide an approximation of values that can be expected in a real implementation. Since our intention was to present a novel approach for an efficient consensus protocol and evaluate its general feasibility, we leave a detailed experimental evaluation for future work.

In the following section, we will extend the idea of decentralized vehicle cooperation from platoons on the same lane towards cooperation between different road segments at road intersections.

2.4 Cooperative Intersection Scheduling over VANET[3]

In this section, we propose a decentralized protocol called Consensus-based Intersection Scheduling for Connected Autonomous Vehicles (CISCAV) that requires no additional infrastructure and can tolerate timing deviations due to unpredictable but detectable events, such as pedestrian movement or ambulances. Vehicles cooperate via direct VANET communication to agree on a schedule that specifies the groups and the order in which approaching vehicles will cross the intersection.

Road intersections are not only a hot spot for accidents but also cause delays and congestions, especially in urban areas.

According to [INR20], drivers in the US spend an additional 100 hours per year in their vehicles due to congestion and these delays have caused additional costs of 88 billion USD in 2019. This includes the costs for higher fuel consumption, higher wear at certain components and additional costs due to longer delivery times for goods. Apart from the financial impact and delays, the congestions also contribute to environmental damage and diminish the life quality in urban areas due to pollution and smog.

One way to increase the efficiency and safety at intersections is the installation of an Intersection Manager (IM) unit at every intersection, which performs the scheduling of all autonomous vehicles. This approach may be suitable in areas with high traffic volume but it requires a high financial and logistical effort to install such a unit at every intersection in rural areas [CE16]. Furthermore, a centralized intersection management introduces a single point of failure, which would require a high amount of maintenance and redundancy to guarantee continuous operation.

Another approach could be a cloud-based management engine, which virtually maps every existing intersection and acts as a scheduling service to which all vehicles on the road connect. However, this idea requires a continuous Internet connection via cellular towers and is very sensitive to (temporary) communication failures such as packet collisions and network delays.

By contrast, a decentralized approach in which vehicles communicate directly with each other would be much more robust to individual failures and does not require any additional communication infrastructure.

2.4.1 Problem Description and Assumptions

We addresses the scheduling problem for intersections and the required communication on an architectural level. We assume that once the vehicles have agreed on a schedule for crossing the intersection, they can execute a safe crossing autonomously. Therefore, we

[3] Major parts of this section have been published in [RBS21].

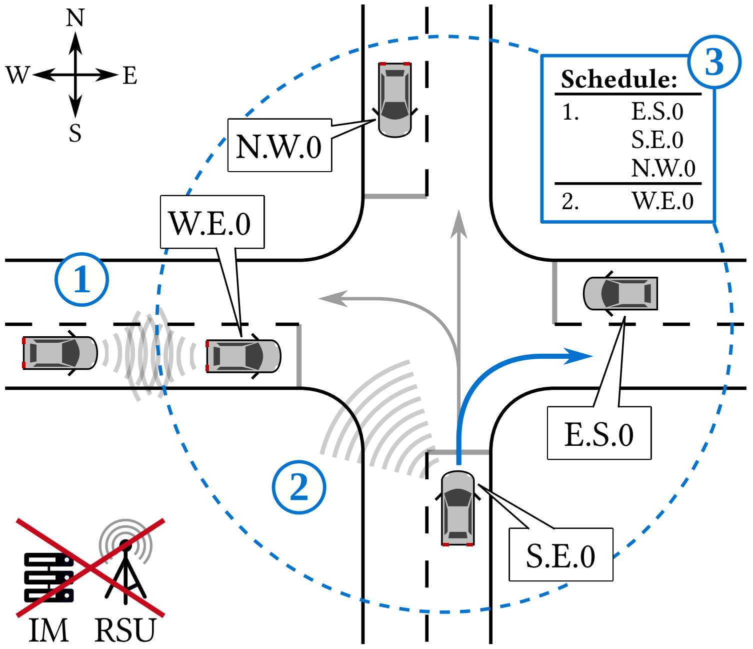

Figure 14: Vehicles agree on schedule groups using three communication phases: ① Intra-Lane Exchange, ② Inter-Lane Exchange, and ③ Schedule Group Consensus. No Road-Side Units (RSUs) or central Intersection Managers (IM) are required.

do not cover the details of physical driving maneuvers such as controller inputs, sensor uncertainties, or data processing.

Instead, we focus on the problem of establishing and agreeing on the order in which vehicles should cross the intersection under the following assumptions:

- Not all vehicles are autonomous but autonomous vehicles are able to detect human-driven vehicles via sensors.
- Autonomous vehicles can directly communicate with each other and there exist no RSUs.
- Environmental events can force vehicles to slow down or deviate in other ways from the schedule (semi-deterministic).
- Vehicles are cooperative but selfish, which means they could send fake data if they gain an advantage (semi-cooperative).

We introduce a complete decentralized and offline approach which only uses direct wireless communication between the participating vehicles. Our goal is an algorithm which works even in rural areas without the need for cellular coverage, central traffic management systems, or access to cloud infrastructure. In particular, we

- propose a distributed intersection management scheme which uses consensus to agree on schedule groups (Section 2.4.2),

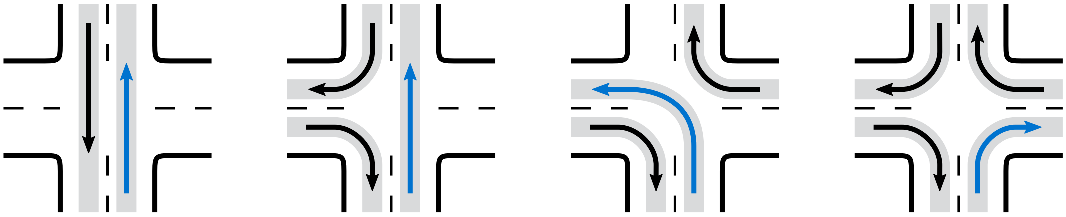

Figure 15: Possible groups of vehicles that can cross an intersection simultaneously. Rotated versions are not shown but are also valid.

- provide a `C++` implementation for the realistic simulation framework SUMO (Section 2.4.3),
- discuss related approaches from literature and why they are over-optimistic (Section 2.4.4),
- simulate and evaluate safety and delay (Section 2.4.4).

Intersection Model We consider an intersection with four roads $R = \{rN, rE, rS, rW\}$ where each road is labeled according to its cardinal direction: rN (north) for the top road, east, south, and west accordingly. Each road has two lanes, one for incoming traffic and one for outgoing traffic. There exists an obligation to drive on the right side of the road. Figure 14 depicts the intersection and the nomenclature of the roads. The intersection is embedded in the center of an area of 400×400 meters, thus each incoming and outgoing lane has a length of 200 meters.

Vehicle Model Vehicles are either 1. human-driven and non-communicating or 2. Connected Autonomous Vehicle. CAVs are able to communicate with each other over a direct, short-range VANET connection, which does not require any additional infrastructure such as RSUs or cellular radio towers. Furthermore, CAVs are able to detect human-driven vehicles by onboard sensors.

Problem Scenario A set $V = \{v_1, v_2, ...v_N\}$ of $N \in \mathbb{N}$ vehicles arrives at the intersection. Each vehicle is defined as $v_i = rX.rY.Z$, with the current incoming road $rX \in R$, the desired outgoing road $rY \in R$, and the index $Z \in \mathbb{N}_0$, which counts the vehicles on each incoming road. For example, $v_i =$ W.S.0 is the first vehicle on the west road that will arrive at the intersection desiring a right turn towards south. In our scenario, the four cardinal directions are used for an intuitive discussion but any number of roads with unique identifier mappings could be used.

Group	Members	Pattern	Schedule	Order of Crossings
0	car W.E.0 car N.W.0 car E.S.0 car S.E.0	SRLR	2111	1. car N.W.0 car E.S.0 car S.E.0 2. car W.E.0
1	car W.E.1	S–––	1–––	1. car W.E.1

Table 2.4: The two schedule groups that are formed based on the scenario from Figure 14. The direction pattern is **S**traight-**R**ight-**L**eft-**R**ight which results in a 2111 schedule (predefined look-up table). The first three vehicles of schedule group 0 can cross simultaneously.

2.4.2 Our CISCAV Protocol

The main idea of our protocol is the use of schedule groups specifying the sequence in which vehicles will cross the intersection without exact timing information. Excluding timing requirements from the schedule allows vehicles to focus only on the order and right of way independent of individual vehicle properties that are time-sensitive, such as agility, reaction time, or accuracy of location estimation. Furthermore, excluding time also allows the protocol to tolerate unexpected events that cause vehicles to slow down or stop such as pedestrians or ambulances.

Schedule Groups

A schedule group consists of a set of vehicles and assigns each vehicle an order value based on their desired direction, which would be the outgoing lane ID (= cardinal point for simplicity). The combination of directions corresponds to a predefined ordering, which is stored in a look-up table. Vehicles with the same order value can cross the intersection simultaneously. Once all vehicles of the same order have crossed the intersection and broadcast that, the next order vehicles start crossing until all vehicles of the schedule group have crossed the intersection. For our 4-leg intersection example, Figure 15 shows the possible combinations of vehicles that can cross together and Table 2.4 shows examples of concrete schedule groups.

Virtual MiniMap

While approaching the intersection, other detected vehicles will be added to the list of known vehicles. The state information of all known vehicles will be stored in a virtual MiniMap, which will be continuously updated over time. For example, license plate ID, velocity, desired direction and index position towards the intersection will be stored here. The index of each vehicle corresponds to the number of vehicles on the same lane that will cross the intersection before them. Therefore, the vehicle next to the intersection has

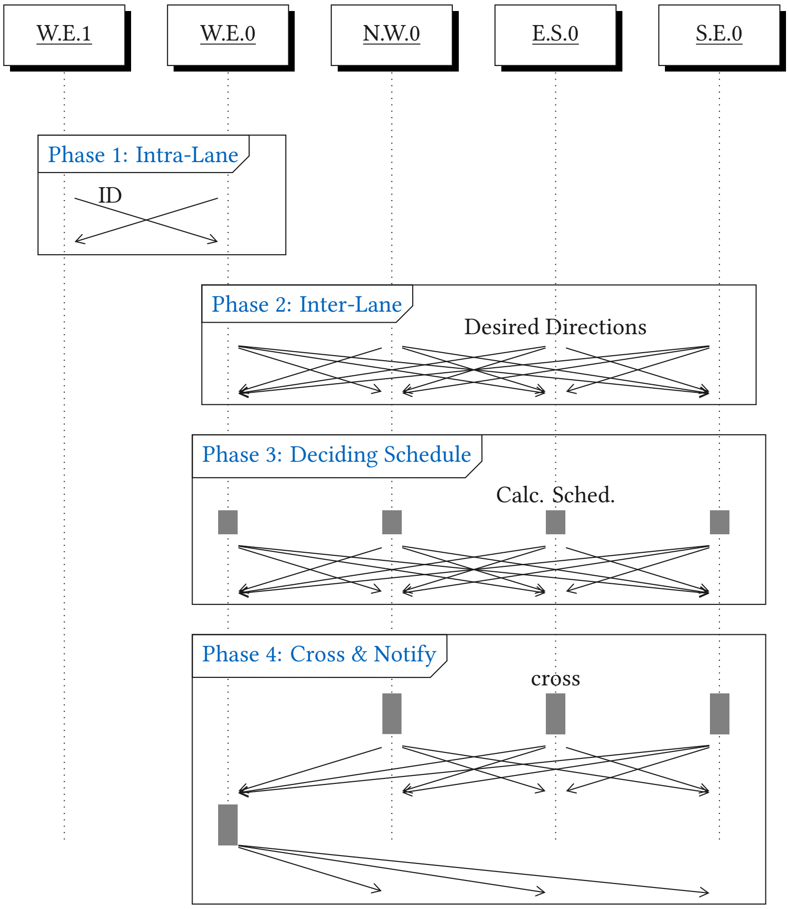

Figure 16: Sequence diagram of our consensus protocol. Arrows indicate messages sent over VANET. Gray bars indicate the time needed for computation or executing an operation.

index 0 and the vehicle behind it has index 1. Note that the indices only establish a relative order on each incoming road but are not synchronized with other roads. Once a larger gap occurs and the next vehicle is outside of the communication range of the previous vehicle, it will start with index 0 again.

CISCAV Protocol Description

The CISCAV Protocol (**C**onsensus-based **I**ntersection **S**cheduling for **C**onnected **A**utonomous **V**ehicles) is a sequence of the following four phases. Any autonomous vehicle which approaches an intersection has to execute all four phases to reach a consensus on a schedule of crossings.

Phase One: Intra-Lane Information Exchange Before getting near the intersection, each vehicle will already exchange messages with surrounding vehicles on the same lane during driving. The acquired information such as desired direction, velocity, and position of vehicles is stored in the virtual MiniMap.

Phase Two: Inter-Lane Information Exchange Vehicles within 150 m to the intersection will broadcast messages to exchange information with vehicles on other incoming roads. This new information will also be stored in the MiniMap, which will incrementally create an ordered list of all vehicles for each incoming road.

Phase Three: Deciding Schedule An optimized schedule is created based on the concept of schedule groups. All vehicles which are the first ones on each incoming road that are not already part of an existing schedule group form a new schedule group. Based on the pattern of desired directions, a schedule order is calculated as shown in Table 2.4. Afterwards, every member of a schedule group repeatedly broadcasts the schedule group and updates the MiniMap with received broadcasts. Once all members have received a broadcast with the same schedule from *all* other members, they lock the schedule group.
CAVs which do not participate in this phase will not gain any advantage because they would block themselves as well.

Phase Four: Cross & Notify Once all vehicles from one schedule group are next at the intersection, they cross the intersection based on their agreed schedule. They continue to broadcast results and positions to inform new arriving vehicles on the agreed schedule and the states of other vehicles.
Figure 16 shows the message exchange during each phase. Algorithms 1 and 2 describe the pseudo code of our implementation that is discussed in Section 2.4.3.

Consensus Properties

CISCAV ensures that vehicles agree on the schedule group. In conventional consensus protocols, such as PBFT [CL99], participants can start in a conflicting state and the protocol must converge to agreement. In our situation, we already have a fixed set of schedules (look-up table) such that every correct vehicle will calculate the same schedule. As discussed in [RS19], consensus for traffic decisions cannot tolerate any failing vehicle. As soon as a conflict arises, the consensus fails and needs to be repeated until unanimous agreement is reached or otherwise resolved manually.
The only parameter that can cause disagreement is the set of vehicles in a schedule group because some vehicles could try to join when the agreement round has already started. In case a vehicle has decided on a schedule group, it will no longer accept any proposals or votes to extend or change a schedule group. In case a vehicle has not decided yet, the currently proposed schedule group can be extended (but never reduced). Extensions can happen until there is a vehicle from every incoming road in the schedule group. After

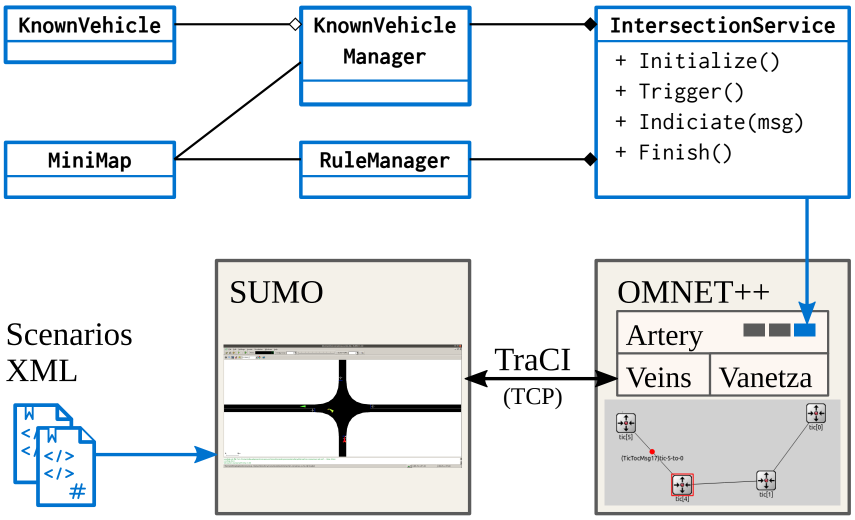

Figure 17: Illustration of our software architecture and its interaction with the simulation environment. The XML scenario files specify the random traffic flows for SUMO, which will spawn vehicles accordingly. Our protocol runs as service in Artery.

that there are no further updates possible and every correct vehicle will calculate the same schedule group.

Handling Human-Driven Vehicles

Since there is no synchronous timing requirement for the schedules, CISCAV can tolerate human-driven vehicles as well. We assume that human-driven cars can be detected in Phase 2 because they appear on sensors (e.g. Camera, LIDAR) but do not respond to messages. Because of this, it will not be possible for autonomous vehicles to form a schedule group and, as a result, they will fall back to conventional traffic policies. Only in case that all the vehicles next at the intersection are CAVs, a schedule group is formed. In this case, the vehicles deviate from the conventional traffic policies and perform a more efficient crossing. In this manner, CISCAV can tolerate any percentage of human-based vehicles and will “kick in” once only CAVs meet.

2.4.3 Implementation as SUMO/Artery Service

We have implemented CISCAV in C++ as a message handling service for the simulation frameworks Artery and SUMO, which aim for a realistic modeling on several abstraction layers. Our overall architecture is depicted in Figure 17.

Algorithm 1: `Trigger()`

```
Data: myData = {currRoadId, nextRoadId, frontVeh, backVeh, ScheduleGroup}, MiniMap
if not near intersection then
    if frontVeh or backVeh are unset then
        broadcast myData;
if ScheduleGroup is empty then
    create ScheduleGroup from MiniMap;
    broadcast myData
else if ScheduleGroup is non empty then
    broadcast ScheduleGroup
else if ScheduleGroup is final then
    while others before me in ScheduleGroup do
        wait for notifications
    cross intersection
```

Used Frameworks

In detail, we have build CISCAV on top of the following framework stack (from physical- to communication-level):

- **SUMO** or "Simulation of Urban MObility" simulates road traffic on various road network geometries. The geometries are either exported from open street map data or are created by hand. Our custom-designed traffic flow is defined in XML-files which are loaded by the simulator. The application either runs in combination with V2X communication system or it is able to run independently where traffic flow is simulated without communication among the created vehicles [Lop+18].
- **OMNeT++** is a discrete event simulator for network communication [VH08].
- **Vanetza** implements the ETSI[4] C-ITS protocol suite and supports the specified communication standards [Rie15].
- **Veins** simulates Inter-Vehicular Communication. It connects OMNeT++ and SUMO [SGD11] .
- **Artery** The V2X simulation framework enables communication based on the ETSI ITS-G5 specification. It is the highest abstraction of all underlying communication and vehicular simulations [Rie+15].

Software Details

Our main class `IntersectionService` inherits from Artery's `ItsG5Service`, which allows us to send messages and provides callbacks for the initialization of vehicles, receiving of

[4] The European Telecommunications Standards Institute (ETSI) specifies the message types and data structures for Intelligent Transportation System (ITS).

Algorithm 2: `Indicate(IntersectionMessage)`

```
Data: newVeh from recv. IntersectionMessage, MiniMap, myData
if not near intersection then
    if newVeh not in MiniMap then MiniMap.add(newVeh);
    if newVeh in front of me then
        set myData.frontVeh = newVeh
    else if newVeh behind me then
        set myData.backVeh = newVeh
else if near intersection then
    if my ScheduleGroup is final then return;;
    if newVeh.ScheduleGroup equals my ScheduleGroup then
        set newVeh as agreeing
        if all vehicles in my ScheduleGroup agree then
            mark ScheduleGroup as final;
    else if newVeh.ScheduleGroup extends my ScheduleGroup then
        update my ScheduleGroup
    else
        keep my ScheduleGroup
```

messages, and finalization (simulation end) of vehicles. We use or overwrite these functions to include our protocol. Every vehicle runs its own instance of our service class.

- `Initialize`: Called on vehicle creation. We initialize our internal data structures such as `roadID` and schedule groups with the vehicle itself as the only member.
- `Trigger` (Algorithm 1): Called periodically by the simulation. Here, we broadcast periodic beacons for discovering other approaching vehicles. Additionally, a schedule group is created with all first vehicles on the other incoming roads, which are in a certain range close to the intersection and are not already part of a schedule group. Each position must be verified by at least one other vehicle in order to be accepted in a schedule group. Updated information is broadcast at the end of the function call.
- `Indicate` (Algorithm 2): Called on message reception. Every received message is processed in this function, which updates our data structure and creates a virtual map of the intersection. The updated data is used in the next call of Trigger.
- `Finish`: This function is called when a vehicle exits the simulation. It collects all diagnostic data.

Algorithm			Intersection		Vehicle				Results
Name	**Type**	**Simulation**	**S/L**	**Legs/Near**	**L** [m]	**V** $\left[\frac{m}{s}\right]$	**Acc.** $\left[\frac{m}{s^2}\right]$	**L-S-R** [%]	**Avg. Delay**
AIM08 [DS08]	Cent./Res.	Custom	+1	125 m/−	N/A	25.0	N/A	0-100-0	0.2 s@0.5 v/s/ln
Prio14 [Qia+14]	Cent./Res.	SUMO/10 min	+3	290 m/50 m	N/A	12.0	−4 ... 2	20-70-10	15 s@0.5 v/s/rd
Delay17 [Zhe+17]	Cent./Res.	SUMO	+1	50 m/50 m	N/A	N/A	N/A... N/A	20-70-10	35 s@0.5 v/s/rd †
CSIP19 [AR19]	Cent./Res.	AutoSim	+2	500 m/N/A	2.6	13.4	N/A... N/A	0-33-66	≈0 s@0.5 v/s/rd
NoStopSign08 [VDS08]	Dec./Res.	Custom	+1	125 m/75 m	N/A	N/A	N/A	15-70-15	20 s@0.4 v/s/ln
MP-IP12 [Azi+12]	Dec./Res.	AutoSim/1000 veh.	+2	N/A	< 5	N/A	N/A ... 3	25-50-25	2 s@0.5 v/s/rd
MutEx14 [Wu+14]	Dec./Res.	NS-3	+2	50 m/50 m	N/A	N/A	N/A	N/A	19 s@0.5 v/s/rd
VTL15 [Shi+15]	Dec./Res.	SUMO+NS3	+4	200 m/N/A	N/A	16.7	−6 ... 3	N/A	35 s@N/A
DIMP18 [Lia+18]	Dec./Res.	SUMO	+(1/2)	500 m/358 m	5 m	13.41	−2 ... 1	N/A	17 s@0.5 v/s
CICAP17 [EMB17]	Dec./Res.	SUMO	+1	700 m/100 m	5	30.0	−1.5 ... 0.8	N/A	N/A
CISCAV (ours)	Dec./Res.	SUMO / 3600 s	+1	200 m/100 m	4.3	13.9	−7.5 ... 2.9	33-33-33	32 s@0.5 v/s/rd

Table 2.5: Overview of related work. **Type** is a combination of {Central/Decentral} and {Reservation/Control}. **S/L** indicates the shape of the intersection (⊢, +, ✳) and the number of *incoming* lanes for each direction. **Legs/Near** is the length of the intersection roads per leg and the distance from the center at which vehicles start broadcasting messages. **L, V, Acc.** states the vehicle *L*ength; max. *V*elocity; and maximum deceleration (negative) and *Acc*eleration considered. **L-S-R** are the probabilities for a vehicle to turn left (L), go straight (S), or turn right (R) in percent. Each related approach misses at least one of these parameters (N/A).

†: Authors just state 45s total travel time and we subtracted 10s to estimate the delay because 10s was the total travel time for the lowest spawn rate of 0.1v/s, for which we assume almost no interruptions.

2.4.4 Evaluation of Safety and Delay.

We evaluate our approach based on the most important metrics *safety* and *delay*. For us, the delay is the *additional time* for a vehicle to completely cross an intersection compared to a crossing where the vehicle has priority from the beginning and can cross without any interactions with other vehicles.

Related work sometimes measures *throughput*. However, this metric is difficult to define for low spawning rates because the exit rate is basically the entry rate. Based on our findings, throughput is only interesting if the spawn rate is maximum. Since intersections normally experience income rates below the maximum, we only measure delay.

Related Work

A good overview of cooperative IM approaches is presented in two surveys [CE16] and [RM16] from 2016 while more recent developments are covered in the introduction of [Mir+19].

We found that we can classify existing approaches in roughly two types: 1. Exclusive Reservation and 2. Control Optimization. Each type can be further distinguished into *centralized*, if it uses a central manager which handles the calculation and communication, or *decentralized*, if vehicles calculate individually and communicate directly with each other.

Exclusive Reservation: The intersection is divided into discrete *resources*, such as *tiles* or pre-defined *paths*. Vehicles then negotiate and agree on a conflict-free reservation schedule where each vehicle gets exclusive access to resources until it crossed the intersection. Exclusive reservation is safe but does not optimize throughput as vehicles often reserve more resources than actually needed. Furthermore, it is compatible with human-driven cars, which simply can get their own reservation slot based on existing traffic rules.

Control Optimization: Instead of discrete reservation, vehicles try to calculate smooth and collision-free trajectories in the continuous spectrum of possible trajectories, which can be followed by accelerating or braking. The trajectory search is formulated as a control problem with certain constraints (collision-free) and certain optimization objectives (e.g. latency, smooth acceleration). While this approach yields good results in simulations, its computation is expensive (state explosion), introduces hard real-time requirements, and often does not consider failures, such as a vehicle unable to apply acceleration or braking as claimed.

For this reason, we will cover only *Exclusive Reservation* and focus on *decentralized* approaches as these are most comparable to our work. Table 2.5 summarizes our findings. As it can be seen, the delay times for the same spawn rate of 0.5 v/s/rd vary significantly between almost zero seconds up to 35 s@0.5 v/s/rd. In general, centralized approaches report lower delay times, which can be explained by the fact that a central intersection manager with global knowledge can compute an optimal scheduling. However, all approaches differ in their parameter selection, which makes them difficult to compare. At least the traffic simulator SUMO is the most common choice, which should ensure realistic modeling of the vehicle dynamics.

Detailed descriptions of each listed approach can be found in Appendix A.2.2.

Experimental Setup

As discussed before, we have selected SUMO as simulator because it is well-established in literature and focuses on realistic traffic simulations. In SUMO, we have created random traffic flows for our 4-leg intersection. We simulate a 4-leg intersection because it is the most common type of intersection ($\approx 80\%$) [AW06] and commonly used in literature. Vehicles have equal probabilities ($33.\overline{3}\%$) for each direction: left, straight, or right. Using equal probabilities ensures that we are not optimistic about the percentage of straight and right turns as these are easier to handle. Although assuming a higher percentage of vehicles going straight might be more realistic, we decided to use equal probabilities as a conservative parameter selection.

We randomly spawn vehicles at the beginning of the roads (200 m from the center) with a given probability that is equal for all directions. For example, 0.5 v/s/rd means that on average 0.5 vehicles will be spawned per second at each road. We spawn vehicles over a period of 3600 s and run the simulation until every vehicle has crossed the intersection. We have performed this experiment for varying spawn probabilities over four crossing policies: Traffic light, priority road, left before right and CISCAV. This ensures that we cover the most common policies at intersections. To be able to compare our work against related approaches, which assume 100% autonomous vehicles, we have only spawned 100% CAVs. However, the design of CISCAV allows to also handle human-driven vehicles as mentioned earlier in Section 2.4.2.

CISCAV results

During the simulation, no accidents occurred. While we did not explicitly stress-test our design with injected failures, it illustrates that random traffic flow is handled safely under normal conditions. Figure 18 shows the delays for each traffic policy. For a spawn rate of 0.5, CISCAV has an average delay of about 32 s. This is a much higher value than the values

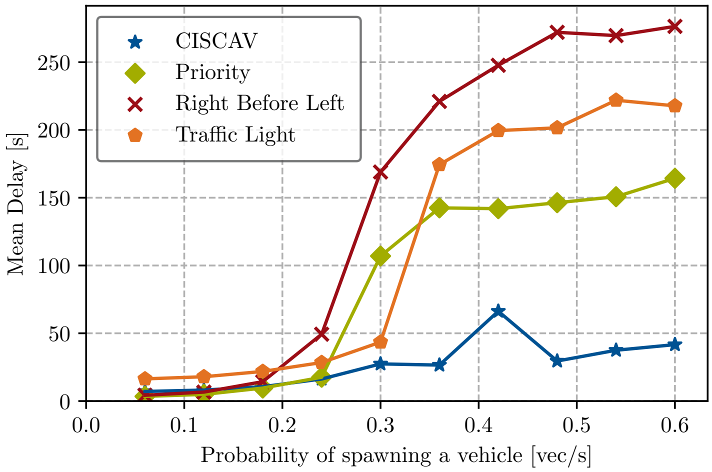

Figure 18: Average delay measurements for all four traffic policies over different spawn probabilities.

reported by most related work. One of the reasons is that CISCAV is currently creating schedule groups in onion-like layers and does not set a lasting priority like a green light for a certain road. Another reason could be that related approaches have different definitions of delay or use over-optimistic assumptions. However, CISCAV outperforms all conventional traffic policies by far and illustrates how much optimization is possible by autonomous intersection scheduling.

Reproducing AIM08 Results

Since the AIM08 [DS08] is such an influential paper, we decided to reproduce results for our intersection model using their open-source AIM simulator version 1.0.3. We have simulated a single +1 intersection for 1000 s at a spawn rate per road of $1800\,\mathrm{v/h/rd} = 0.5\,\mathrm{v/s/rd}$ using $v_{\max} = 15\,\mathrm{m/s}$ (closest). The simulator outputs entry and exit times of the vehicles in seconds. We have calculated the delay as

$$t_{\text{delay}} = T_{\text{exit}} - T_{\text{entry}} - \Delta T_{\text{free}} \tag{2.2}$$

with $\Delta T_{\text{free}} = 11$ s as the measured mean travel time from entry to exit for a vehicle without intersection.

To our surprise, the average delay is $\hat{t}_{\text{delay}} \approx 13.5$ s, which is far away from the 0.2 s reported in the original paper. We have contacted the authors of AIM08 and they assumed that the difference in delay time is a result of different parameter selection. Furthermore, they confirmed that our calculation of delay is correct and that they have used the same equation to evaluate their approach. Overall, this experiment illustrates the high sensitivity of the delay with regard to simulation parameters such as lane numbers and vehicle velocity.

2.4.5 Summary: High robustness at fair performance.

Performance of Related Work: While most related approaches report much lower delay times than CISCAV, they are often based on optimistic assumptions in the best case, and completely unrealistic assumptions in the worst case. For example, in some simulations of [AR19], vehicles approach the intersection equally spaced at a constant speed and keep minimum safety distances during the crossing. The tendency of some papers to optimize delays on idealized conditions has also been pointed out by [Lia+18], which has reproduced VTL15 in SUMO and found a delay of 45 s@0.5 v/s/rd, which is significantly higher than the original value. Although performance is important, CISCAV's primary focus lies on reaching a safe agreement on exclusive reservation under unreliable conditions before any actions are taken. This results in delays that cannot compete with delays under ideal conditions but are still much lower than existing policies, such as traffic lights.

Weaknesses of CISCAV: In this first version of CISCAV, we create schedule groups in onion-like layers, which means vehicles from the same lane cannot cross the intersection directly after each other if the other lanes have also incoming vehicles. In high traffic conditions, this is not the most efficient way to reduce average delay and leads to frequent start-stop maneuvers when several vehicles approach the intersection from the same lane. Currently, we do not handle different vehicle types and the fact that large trucks might prevent certain schedule combinations and need to cross smaller intersections alone. Furthermore, our current implementation of CISCAV cannot resolve any permanent communication failures after schedule groups have been confirmed. That is, once a schedule is agreed and the last vehicle of the previous schedule group has left the communication range, the current group will wait for the missing crossing notification forever. However, this is no limitation of the protocol itself and can be resolved easily using sensors.

Strengths of CISCAV: CISCAV is completely decentralized and does not depend on any additional infrastructure. The use of schedule groups focuses purely on the order in which vehicles cross and therefore allows to tolerate arbitrarily vehicle timings and delays. It is designed in a way that allows to be transient for human-operated vehicles. This means that schedules, which deviate from the existing traffic rules, will only be created once only CAVs approach an intersection. We have implemented and evaluated CISCAV in the realistic simulation tool SUMO using C-ITS messages standardized by the EU. In its first version, it performs well in low load scenarios and outperforms current traffic rules.

Conclusion: CISCAV can reach a distributed agreement on exclusive reservation by creating schedule groups before any actions are taken, allowing it to tolerate arbitrary timing deviations as they can occur in real-world conditions.

In comparison to the sometimes over-optimistic related work, CISCAV gives realistic estimations for the delay of safe autonomous intersection crossing. Our measured average delay of $\leq 32\,\mathrm{s}$ at a very busy intersection supports the hypothesis that communication-based intersection handling can outperform currently existing traffic rules. However, related work in general indicates that the average delay can be further reduced. As we have discussed before, CISCAV could be improved by considering multiple vehicles from the same lane together. It would be interesting to combine CISCAV with our CUBA protocol presented in Section 2.3 to schedule entire platoons instead of individual vehicles.

Overall, CISCAV does not require any additional infrastructure, makes conservative assumptions, and focuses on providing high safety guarantees, which is a fundamental requirement if we want to convince people in the near future to actually trust their lives on autonomous intersection management.

Chapter 3

Data Certification via Blockchains

Contents

Suppose you have just downloaded a binary file called `smarthome.app` from a website. Is this file save to execute? Traditionally, we would accept a new binary if we can get a Transport Layer Security (TLS) certificate from the company's website (green padlock in Web-browser) *and* if we trust the company. We usually trust a company because some majority of people we know (e.g. our friends) has already used the software and told us that the company and their software is trustworthy. We basically have reached consensus on the companies trust status and synchronized this state of trustworthiness among our friends. But how can we verify the trustworthiness of a company if none of our friends has experience with it? How could we extend the certification process to include the global consensus of all people, even when we do not know them?

In the last chapter, we have seen how consensus protocols can be used in CPS to decide and synchronize the current system state. However, the decision on the current state is only available to the devices that have actively participated in the decision making process. Since other non-participating devices – or devices that have been temporarily disconnected

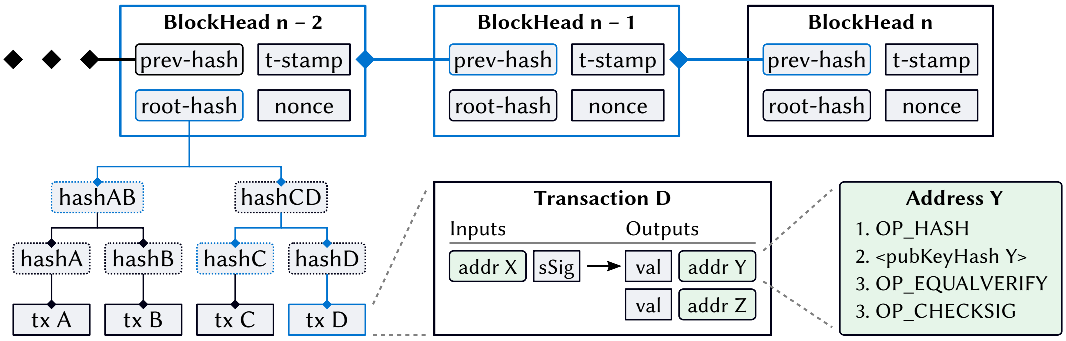

Figure 19: Structure of a blockchain with Unspent Transaction Output (UTXO), such as Bitcoin. The blocks are connected via the hash value of the previous block. Each block contains several transactions that are combined to single Merkle tree root hash. Each transaction contains multiple input and output addresses, where an address consists of a public key and operation code. The blue lines indicate that there is a chain of hashes that connect transaction D to the latest block n.

from the network – might also be interested in the current/previous system state, we need to find a mechanism to securely propagate this information. This propagation requires a certification of the consensus data, because devices cannot trust each other but need to verify the correctness of received data by themselves.

One of the first working example of consensus certification on a global scale was provided by Bitcoin [Nak08] in 2008 by its use of a novel mechanism called *blockchain*.

Blockchain for data certification: Cryptocurrencies such as Bitcoin [Nak08] reach consensus on global financial transactions in a large, trustless, and open peer-to-peer network without any central authority. The transactions are stored in a distributed chain of blocks (blockchain), each block securing the order and integrity of previous blocks. While cryptocurrencies focus on transactions of assets, it is possible to store any data in a blockchain [CD16] and there exist ideas to transfer this technology to the embedded domain, such as for secure peer-to-peer firmware updating and validation [LL17; Nik+17; Ste+17]. For this scenario, a manufacturer would publish a signed hash of the latest firmware on a blockchain, where the signature is verified by all full powered network participants and – if valid – included in the blockchain. An IoT node could then receive a new firmware from any possible peer and verify its validity and integrity by simply comparing its hash to the hash in the blockchain [CD16]. This solution is more robust as it avoids a single point of failure, more efficient because the end nodes do not need to perform public key cryptography, and more transparent as any new update is publicly verified on the blockchain.

3.1 Blockchain: History, Assumptions, and Model

This section provides an overview on blockchain, its working mechanisms, and related approaches. In general, the term *blockchain* is used to refer to a certain technological approach consisting of a data structure (ledger) *and* a consensus protocol.

3.1.1 Blockchain as Datastructure

A *blockchain* is a distributed data structure that stores a system state over time and is shared and replicated by all nodes within a network of participants [CD16]. Formally, a blockchain is an ordered chain

$$\mathcal{C} = \{B_i \,|\, i \in 1, \dots, n\}, \qquad B_i \prec B_{i+1} \tag{3.3}$$

of n blocks B_i where n is the height of $\mathcal{C}$ and i the height or index of B_i. Each block confirms and reinforces the data of its preceding block by including the hash of the previous block in its own block (Figure 20a). For all hashes, we assume a single cryptographic hash function $H(\cdot)$ that outputs a hash h of κ bits:

$$H(\cdot) : \{0,1\}^* \to \{0,1\}^\kappa. \tag{3.4}$$

Including the previous hash in each block secures the integrity and ordering of the blocks because any change to the data of an existing block would result in changing hashes of all consecutive blocks. A minimal block is represented by the tuple $B_i = \langle h_\mathrm{p}, h_\mathrm{d} \rangle$ with the prev-hash $h_\mathrm{p} = H(B_{i-1})$. The actual data D_i of a block can be of arbitrary structure and is bound to a block only by its hash $h_\mathrm{d} = H(D_i)$.

Depending on the consensus mechanism, additional information must be stored in each block. Note that this information can be stored in the block data D_i, which could then again be divided into consensus header information and another payload hash h_CP, leading to a hierarchical layer structure. However, using only two hashes as the block header is a minimal, yet sufficient specification to model any kind of blockchain application.

Blockchain Transactions

Transactions are broadcasted events that transfer the ownership of an asset. Transactions can be created manually by humans or automatically by *Smart Contracts*, which will be discussed later in Section 3.4.1. In principle, a transaction is valid if it is cryptographically signed with the key pair that belongs to the owner of the transferred asset. Nodes in the network will validate transactions and – if valid – include them in the blockchain, to confirm their validity. To keep track of the ownership of transferred assets in a blockchain, there are two common models:

1. the *account-based* model, where each node simply stores the balance of all assets that each account owns. Transactions simply reduce the amount of an asset on the sending account and increases the amount on the receiving account.
2. the *Unspent Transaction Outputs* (UTXO) model, where assets are stored on addresses that belong to cryptographic keys instead of accounts. Instead of adjusting balances for each address, a transaction fully "consumes" assets from a list of input addresses and reproduces them on a list of (new) output addresses.

The validity and the order in which transactions are added to the blockchain and applied is determined by the underlying consensus mechanism.

Bitcoin Transactions While we aim for a general application of blockchain and do not want to rely on one particular block data model, we will provide some insights on how Bitcoin transactions work as one specific example on how blockchain works as a public Distributed Ledger Technology (DLT).

Bitcoin uses the UTXO model, which is illustrated in Figure 19. The list of transactions is connected to the block header via a Merkle Tree. Each transaction corresponds to a leaf in the tree. The Merkle tree is then constructed by hashing the leafs and the resulting hash values in pairs of two until only one hash value is reached. For example, the intermediate hash `hashAB` (h_{AB}) is calculated as

$$h_{AB} = H(h_A || h_B)$$

where the operator $||$ means concatenation of both values.

Each transaction consists of a list of input addresses with signatures, a list of output addresses, and the value of coins that is transferred to each output address. Note that the value on the input address is completely consumed. It is not required to specify the value on the input address because it is known by every node due to the previous transactions.

Bitcoin addresses are based on a simple, stack-based, scripting system that describes how the next person wanting to spend the Bitcoins can do so. As shown in Figure 19, the standard transaction to a public key `Y`, describes the following operations:

- `OP_HASH`: hash the top item on the stack and put the result on the stack.
- `<pubKeyHash Y>`: add the hash of the public key Y, which was specified in transaction D to the stack.
- `OP_EQUALVERIFY`: compare the previous two items on the stack.
- `OP_CHECKSIG`: verify the signature of the input.

If a sender S wants to spend the value val on address Y, he will create a new transaction that uses <addr Y, sSig> as input. The signature sSig consists of <sig S>, <pubKey S> and both values are placed on the stack. Thus, the script of Y will hash the top item, which is <pubKey S> and compare the result against <pubKeyHash Y>. Therefore, the sender S can only spend the coins if the public key Y in transaction D is equal to his public key S and he provides a valid signature for it.

3.1.2 Blockchain as Consensus Protocol.

While a blockchain can be seen as pure data structure, it will not be usable for most applications (e.g. DLT) if not paired with a consensus algorithm for the block creation process.

While there are many different consensus protocols [CV17], currently the most common ones for blockchains are Proof-of-Work (PoW) [Nak08] and Proof-of-Stake (PoS) [Sal20]. The advantage of these two over traditional protocols, such as PBFT, is that they enable a trustless, open network where everyone can join without the need to keep a list of permissioned nodes allowed to vote for a certain system state.

Proof of Work (PoW)

Most blockchain implementations secure the construction of the chain by a cryptographic puzzle called Proof-of-Work (PoW). In a PoW blockchain, the block structure is extended by a nonce field $ctr \in [0, 2^{32}[$ such that $B_i = \langle h_\mathrm{p}, h_\mathrm{d}, ctr \rangle$.

Nodes with high computational power, called "Miners", verify new data D_{n+1} according to application-specific rules[1], pack them into a new unconfirmed block B_{n+1}, and iteratively try to find a ctr such that

$$B_{n+1} = \big\langle\, H(B_n),\, H(D_{n+1}),\, ctr \,\big\rangle \quad \wedge \quad H(B_{n+1}) < T \tag{3.5}$$

where T is the target value of the hash. Since the output of the hash is unpredictable, the only way to find such a low hash is to brute force it. T can be seen as the difficulty of the PoW puzzle: the lower T, the more tries are necessary on average.

The first miner who finds a valid hash, shares its block with the network and retrieves a reward in return. Each node in the network will validate the new block, append it to its local blockchain, and start the mining race on the next block.

Since a higher number of miners will find a valid hash in shorter time, the average time needed to find a valid hash would decrease. To keep the network stable, the difficulty is adjusted by consensus to keep the average blocktime constant.

[1] E.g. in Bitcoin the miners validate transactions but they could also validate the signature of a new embedded device firmware.

PoW is a consensus mechanism that enables a trustless, open network where everyone can participate. A node joining the network could receive several different blockchains but will always select the one representing the most PoW as the common consensus by verifying and summing up the PoW of each block. Another advantage of PoW is that by using a gossip protocol, where each block is forwarded to a fixed number of random peers, the message complexity scales linear to the number of participants and is in $O(N)$ [Yeo+17], where N is the number of participants. However, nodes cannot absolutely determine when the consensus is reached. At any time, a longer valid blockchain could appear, replacing blocks of the shorter one. The likelihood that such an event changes a certain block quickly approaches zero with the number of succeeding blocks. The blockchain is secure because of the assumption that an honest majority of processing power will on average generate PoW faster than any dishonest minority [Nak08; GKL15].

Proof of Stake (PoS)

In PoS [Sal20], the voting weight is correlated to the stake of a node within a cryptocurrency system to avoid the energy waste of PoW. Nodes need to bind a certain amount of money to their vote and if it is considered correct, they get paid back a higher amount. Otherwise, the money is lost. The first cryptocurrency that has implemented PoS was PeerCoin [KN12]. The security of the consensus is based on the assumption that nodes which possess a higher stake, have a higher incentive to participate honestly in the consensus protocol [Sal20]. Furthermore, stake in form of money is required to participate in the block validation.

3.1.3 Alternative Ledgers: Directed Acyclic Graphs

Instead of a blockchain, there exist also ideas to create a distributed ledger based on another datastructure called Directed Acyclic Graph (DAG).
In a DAG, transactions are not stored in a linear chain of blocks but within *messages* that are connected to two or more previous messages. These additional backlinks form a kind of *web* of blocks instead of a chain, but this web also grows in just one direction (acyclic = no loops).
Messages become final if they are *deep enough* in the DAG, where the exact conditions vary among implementations. Two prominent implementations of a DAG-based ledger are *IOTA* [Pop18] and *HashGraph* [Bai16].

IOTA

IOTA [Pop18] is DAG-based DLT and also a cryptocurrency. The DAG is called *Tangle* and stores all messages. In order to create a new message, a node needs to verify two or more previous messages. The ability to propose new messages is also bound to an asset called

Mana. *Mana* can be earned by verifying and endorsing correct messages. The *Mana* also controls the *rate* at which a node can create new messages and furthermore determines which messages are included first into the DAG when too many messages are created. Financial IOTA transactions are just one possible content for the messages but they could also include other types of data.

Within the testnet, IOTA uses a central coordinator to ensure that enough messages are validated and the DAG grows continuously. The coordinator creates special milestone messages and all messages below a milestone are considered final. As of 2022, IOTA is transitioning to protocol 2.0 where they want to remove the central coordinator and run the Tangle in a fully decentralized mode.

Hashgraph

The *Hashgraph* [Bai16; BL20] is a DAG-based DLT and also includes a patented consensus mechanism from the company *Swirlds*.

The two main concepts of the Hashgraph are called *gossip about gossip* and *virtual voting*. The first means that each node tells two other nodes about new messages (gossip) and also from which node it received this message (gossip about gossip). As a result, every node knows everything all the other nodes know and in which order they received new information.

With this knowledge, it is possible that each node can simply calculate how every other node would vote (virtual voting). Thus every node calculates votes according to the rules. One important property of the Hashgraph is that new message are accepted to be part of the history once they are received by a node and the consensus only needs to determine how the messages are ordered.

The ordering of messages is based on analyzing how many messages reference a message A before another message B. The goal is to ensure fairness in the sense that if a message A was received before message B by most nodes, then the message ordering algorithm will sort message A before B.

3.2 Efficient Verification of Blockchain Integrity[2]

Blockchain provides decentralized records of consensus decisions in large, open networks without a trusted authority, making it a promising solution for the IoT to distribute verifiable data, which could be a new firmware update, for example.

However, nodes that join a blockchain network, are required to receive the blockchain in full length to verify the consensus state.

Due to the constant growth of the blockchain, downloading and processing the entire chain requires more and more resources over time. Each additional block increases the communication overhead, memory allocation, processing time, and power consumption of each device. For example, to verify the current Bitcoin blockchain, an embedded device would need to download and hash approximately 58.4 MB of block headers[3], already using simplified verification [KMZ17].

For the majority of cheap IoT devices, the blockchain length will quickly exceed their resource capabilities, rendering verification impossible. In the case of firmware updates, this might introduce severe vulnerabilities or leave devices unable to continue their intended service. It is therefore necessary to develop a verification approach for the embedded domain, in order to efficiently and safely use blockchain technology in constrained IoT environments.

Contribution In this section, we propose a generic blockchain extension that enables highly constrained devices to verify the inclusion and integrity of any block within a blockchain. Instead of traversing block by block, we construct a *LeapChain* that reduces verification steps without weakening the integrity guarantees of the blockchain.

LeapChain is a generic blockchain data structure and applicable to any kind of blockchain technology. The concept reduces verification steps by additional backlinks as illustrated in Figure 20 and enables embedded devices to verify blockchain content using only a few kilobytes of Random Access Memory (RAM).

Applied to Proof-of-Work blockchains, our scheme can be used to verify consensus by proving a certain amount of work on top of a block.

Our analytical and experimental results show that, LeapChain requires less memory and computation compared to existing approaches and that only LeapChain provides deterministic and tight upper bounds on the memory requirements in the kilobyte range. This significantly extends the possibilities of blockchain application on embedded IoT devices.

[2] Major parts of this section have been published in [RS18a].

[3] Calculation based on 730 000 block-headers (reached Apr. 2022) of 80 bytes. We use $1\,\text{MB} = 1\,000\,000\,\text{B}$

3.2.1 LeapChain Extension

We will now explain our blockchain extension that inserts additional connections with a special backlink pattern to speed up traversing the chain without weakening its integrity guarantees. Note that to traverse a conventional blockchain backwards, a node needs to iteratively verify the direct predecessor of a block using the prev-hash h_p block by block. As shown in Figure 20, we extend the conventional block structure, such that each block header stores one additional back-linking leap-hash h_l that "points" further back than just the direct predecessor – leaping over several blocks in between.
The memory overhead of this extension is minimal compared to the size of the full blocks of a blockchain. Since this additional leap-hash is part of the header that is hashed, it provides the same integrity mechanism as the prev-hash. Overhead and integrity will be further analyzed in Sections 3.2.3 and 3.2.3.

Leap-Width The distance between the current block B_i and the block B_{i-w} that matches the leap-hash $h_\mathrm{l} = H(B_{i-w})$ is the leap-width w. With our extension, it is possible to traverse back the blockchain *either* step by step using the prev-hash h_p *or* in steps of width w using the leap-hash h_l. The first intuition would be to choose a constant w, which allows to reach any block within approximately $\frac{1}{w}$ of the steps required for the conventional blockchain. However, this would improve the amount of steps only by a linear factor, which would contradict our goal to reach any block with a sub-linear amount of steps. Since the blockchain is continuously growing in height, we need a flexible leap-width $w(i)$ depending on the current height i of a block.

Backlink Pattern In order to achieve a logarithmic scaling, we use several leap-widths $w(i) \in W$ based on different exponents to a constant base b. These leap-widths are calculated from the block height i according to

$$w(i) = \begin{cases} b^b & \text{if } i \bmod b = 0 \\ b^{(i \bmod b)} & \text{otherwise} \end{cases} \in W = \{b^1, b^2, \ldots, b^b\} \tag{3.6}$$

which ensures that 1) there are exactly $|W| = b$ different leap-widths, 2) all leap-widths have a single common divisor b and 3) each leap-width is b times the previous leap-width. Beginning with $i = 1$, we assign each block of height i a leap-hash that belongs to the $w(i)$-th previous block. If $w(i)$ points to a block index $i < 0$ we set the leap-hash to the hash of the genesis block ($i = 0$). This pattern has the following four properties:

𝔭1) Each block leaps back to a block with the same leap-width forming a continuous *leapline*.

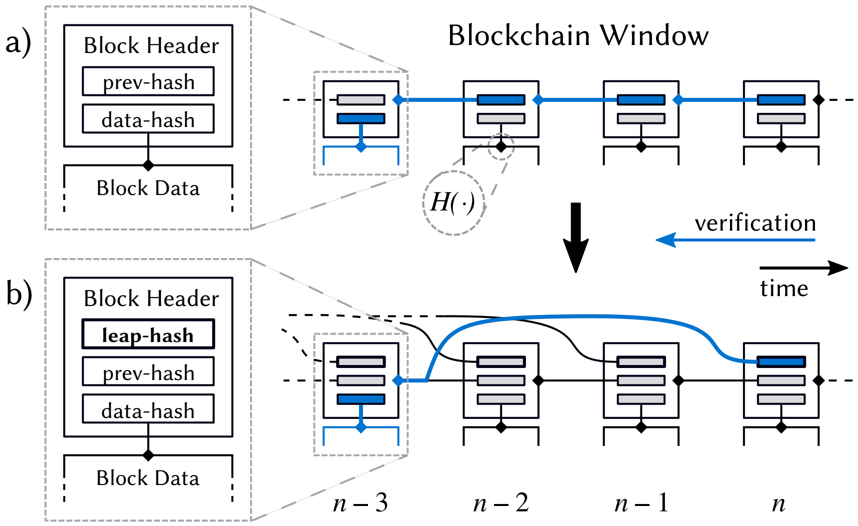

Figure 20: LeapChain Verification. We extend the conventional block structure a) that only connects a block to its direct predecessor, by a leap-hash b) allowing us to traverse the blockchain with a reduced amount of steps to verify the inclusion and integrity of block data.

$\mathfrak{p}$2) There are $\frac{w}{b}$ leaplines for each leap-width w.

$\mathfrak{p}$3) Each block belongs to exactly one particular leapline.

$\mathfrak{p}$4) Any leapline can be reached within b consecutive blocks.

If we need to jump b times on one leapline, we can also jump once on the next wider leapline instead, which leads to a logarithmic amount of steps based on the distance.

Example Consider the case where base $b = 4$, which would result in four leap-widths $\{4^1, 4^2, 4^3, 4^4\} = \{4, 16, 64, 256\}$. The resulting leap pattern is illustrated in Figure 21. All blocks with index $i \bmod 4 = 1$, which are colored in green[4], form a single leapline with leap-width $w = 4$ (see $\mathfrak{p}$1). The next leap-width is $w = 16$ (orange) and between two connected orange blocks, we have $4 - 1$ other orange blocks, each belonging to one of the $\frac{16}{4} = 4$ separate leaplines of width $w = 16$ (see $\mathfrak{p}$2). For $w = 64$, we have $\frac{64}{4}$ different leaplines and so on. The common base $b = 4$ ensures that all leaplines jump multiple of 4 and thus a block of one leapline will never hit a block of another leapline (see $\mathfrak{p}$3). As a result, block 2 can be reached from block 69 in 4 steps: 68, 67, 3, 2. Of course, the maximum leap-width $w_{\max} = 256$ is not sufficient to maintain a logarithmic scaling for large blockchains. In Section 3.2.3 we will compare different choices of the base b and its implications.

[4] References to the colors of leaplines are only made to support the reader but are not mandatory as leaplines can be identified by block index as well.

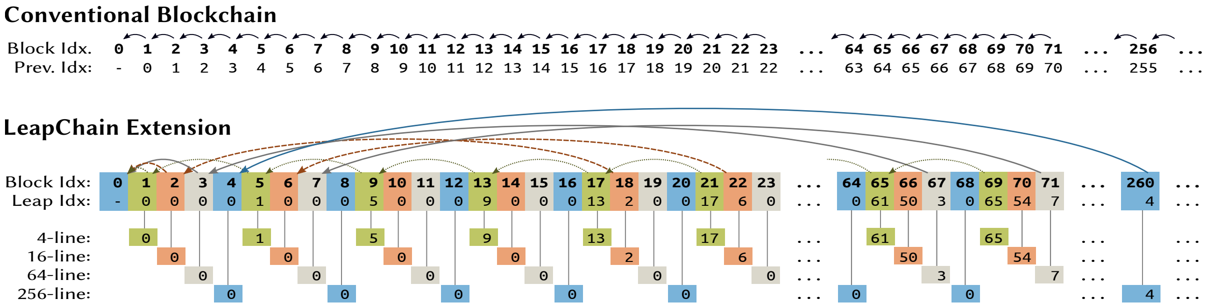
Conventional Blockchain
Block Idx. 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 ... 64 65 66 67 68 69 70 71 ... 256 ...
Prev. Idx: - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ... 63 64 65 66 67 68 69 70 ... 255 ...
LeapChain Extension
Block Idx: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 ... 64 65 66 67 68 69 70 71 ... 260 ...
Leap Idx: - 0 0 0 0 1 0 0 0 5 0 0 0 9 0 0 0 13 2 0 0 17 6 0 ... 0 61 50 3 0 65 54 7 ... 4 ...
4-line: 0 1 5 9 13 17 ... 61 65 ... ...
16-line: 0 0 0 0 2 6 ... 50 54 ... ...
64-line: 0 0 0 0 0 0 ... 3 7 ... ...
256-line: 0 0 0 0 0 ... 0 0 ... 4 ...

Figure 21: The LeapChain extension and its interlink pattern. In a conventional blockchain only the hash of the previous block is stored, which allows only to verify (jump to) the previous block. With the LeapChain extension, an additional leap hash is stored in each block that allows wider jumps. In this example we use a leap base of $b = 4$ resulting in 4 colored leaplines that allow us to jump back either 4, 16, 64, or 256 blocks. Each leapline can be reached within 4 consecutive blocks.

3.2.2 LeapChain Verification Methodologies

In this section, we discuss how the leap pattern can be used to verify that a certain block X and its data D_X, which could, e.g., be a firmware hash, are included in a blockchain $\mathcal{C}$ and were confirmed by consensus. For this verification, we first need to prove that X is indeed part of $\mathcal{C}$ by checking the integrity of the chain and second that the most recent blocks of $\mathcal{C}$ reflect the current consensus.

Note that these two verification steps are fundamentally different. For inclusion verification, we just need the hashes that are stored in the header and we will illustrate that, in this case, LeapChain is sufficient to provide a general solution with the same integrity guarantees as the full chain. For consensus verification, however, we need to consider the underlying consensus mechanism which we do not cover in general due to its diversity. We will only illustrate and analyze that LeapChain works for the common Proof-of-Work mechanism with sufficient security (detailed in Section 3.2.3) compared to the full chain.

In general, we can use the additional backlinks to construct a LeapChain that proves the integrity of the blockchain $\mathcal{C}$ between two blocks $X, Y \in \mathcal{C}$ with $X \prec Y$. This LeapChain efficiently traverses $\mathcal{C}$ from Y to X by a subset of blocks $\mathcal{L} \subseteq X, ..., Y$, providing evidence that X is indeed part of the same chain as Y.

In the following, we assume a network running a distributed blockchain application. We distinguish two types of nodes that run the exact same application but differ in the amount of blocks they can afford to store. First, we have a memory-constrained node – called verifier V – that wants to verify that a block X and its data D_X is part of the blockchain $\mathcal{C}$. V cannot afford to store $\mathcal{C}$ entirely. The second type is a full node – called prover P – that stores the entire blockchain $\mathcal{C}$. Verifier V therefore requests a “proof” $\mathcal{L}$ from a prover P in order to verify properties about the entire blockchain that it cannot verify solely from processing its partial local copy of the blockchain. Prover P constructs the proof $\mathcal{L}$ from

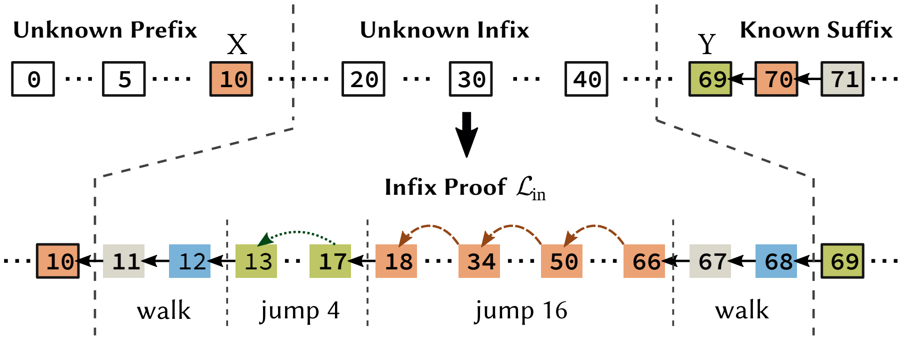

Figure 22: Infix-proof using $b = 4$. $\mathcal{L}_{\text{in}}$ traverses from $Y = 69$ to $X = 10$ (distance $\delta = 59$) using only 10 intermediate blocks. The first leap-width is $w = b^e = 4^2$ because $e = \lfloor \log_4(59) \rfloor = 2$.

its full local copy of $\mathcal{C}$ and sends it back to V, which will process $\mathcal{L}$ to decide whether the property about $\mathcal{C}$ holds or not.

In Section 3.2.2, we first discuss the case that V knows a valid, more recent block $Y \succ X$ that is part of the current consensus and needs to look up a previous block X that was pruned. This scenario applies to embedded nodes that keep a rolling window of the most recent blocks (suffix) and works for any kind of blockchain.

In Section 3.2.2 we discuss the case that V only knows the genesis block of the blockchain and needs to verify that block X is part of $\mathcal{C}$ *and* that it is accepted consensus of the network. This scenario applies to nodes that initially join a blockchain network and only works for PoW blockchains.

Verification of Inclusion with Infix Proofs

Verifier V can verify that any block X is part of $\mathcal{C}$ if it knows a more recent block $Y \in \mathcal{C}$ with $X \prec Y$. First, V requests block X including data D_X from the network. Any node that knows D_X and the corresponding block $X = \langle h_\text{p}, h_\text{l}, h_\text{d} \rangle$ may send it to V as a reply. The replying node does not need to know the full chain and does not need to be trusted. To verify that D_X belongs to X, V simply checks $h_\text{d} = H(D_X)$. Afterwards, V requests a LeapChain $\mathcal{L}_{\text{in}}$ from P.

The prover P, which knows the complete blockchain $\mathcal{C}$ or at least $X...Y \subseteq \mathcal{C}$, is able to construct a LeapChain $\mathcal{L}_{\text{in}} \subseteq X...Y$ as an infix proof. This proof $\mathcal{L}_{\text{in}}$ connects the two blocks X and Y using much less blocks than the full subchain $X...Y$ but provides the same integrity. Figure 22 shows an example how $\mathcal{L}$ connects X and Y. As a requirement, each block $L_j \in \mathcal{L}_{\text{in}}$ must keep the same order as in $\mathcal{C}$ and only link back to a previous block, such that each backlink is secured by the hash of its corresponding block.

Proof Construction The infix proof $\mathcal{L}_{\text{in}}$ will be constructed based on the distance δ between the known block Y at height i_h and the target block X at height i_t. The block

```
1  function leap_chain_idx($i_\mathrm{h}$, $i_\mathrm{t}$):
2    leapch = list($i_\mathrm{h}$)
3    while leapch.last() $> i_\mathrm{t}$ do
4      dist = leapch.last() - $i_\mathrm{t}$
5      $e = \lfloor \log_b(\texttt{dist}) \rfloor$
6
7      # walk to leap line
8      if ($e > 0$):
9        steps = $(\texttt{leapch.last()} - e) \bmod b$
10       foreach i in [ 1, steps ] do
11         leapch.append(leapch.last() - 1)
12       done
13     endif
14
15     # jump down leapline or directly walk (e = 0)
16     leaps = leapcount(dist, $e$)
17     foreach j in [ 1, leaps ] do
18       leapch.append(leapch.last() - $b^e$)
19     done
20   done
21
22   return leapch[1:-1]  # prune $i_\mathrm{h}$ and $i_\mathrm{t}$
```

Listing 3.1: Pseudocode for LeapChain construction.

indices of $\mathcal{L}_\mathrm{in}$ are determined by Algorithm 3.1. First, we initialize the LeapChain with $\mathcal{L}_\mathrm{in} = \{Y\}$ (line 2). We iteratively calculate the exponent e of the largest leapline that fits into the remaining distance `dist` (l. 3-5), how many steps we need to walk to this leapline (l. 9), and how often we can jump on this leapline (l. 16), which is expressed by

$$\texttt{leapcount}(\texttt{dist},\ e) = \lfloor \texttt{dist} / b^e \rfloor . \tag{3.7}$$

We append all involved blocks to $\mathcal{L}_\mathrm{in}$ (l. 10-12, 17-19) and start the next iteration until we reach block X. The last iteration is likely to be $e = 0$ and $w = 1$ which means we have a "walking" phase at the end. Since both hashes h_l and h_p are included in the blocks, this phase can be seen as an iteration of leaps with $w = 1$ and does not need to be considered a special case. At the end, we prune the blocks X and Y from the proof (l. 22) because they are already known by the verifier and do not need to be transmitted.

The resulting infix proof $\mathcal{L}_\mathrm{in}$ proves the integrity of $\mathcal{C}$ between any two blocks $X \prec Y$ and thus also proves consensus for X if Y is known to be part of the consensus. If we do not know whether Y is part of the consensus, we need to verify this with a suffix proof.

Verification of Consensus via Suffix Proofs

We will now illustrate that LeapChain is not only suitable for verifying the inclusion of blocks but can also improve the efficiency for verifying the consensus property. We will illustrate this only for the common Proof-of-Work mechanism but we are confident that our concept could be adapted to other mechanisms as well.

If we consider a PoW blockchain, V can request two LeapChains in order to verify that a block X is part of $\mathcal{C}$ *and* that X is common consensus with a certain probability. In this case, *V* does not need to know any recent block $Y \succ X$ of the blockchain but only the hash $H(B_0)$ of the genesis block B_0 or $H(B_C)$ of another commonly known checkpoint block B_C where $B_0 \preceq B_C \prec X$.

The PoW consensus requires that each block hash needs to be below the target value T, which is determined by a certain difficulty. This difficulty is usually calculated from previous blocks using timestamps. Since we will leap blocks, *V* cannot determine the difficulty for the blocks of a LeapChain because it does not know the difficulty of the intermediate blocks. However, this problem could be solved by including the current difficulty in the block data such that miners can extract and verify the difficulty from each single block.

In the following, we choose another option and assume a constant difficulty, which means that each block hash needs to be below the same target value T that will never change. Even though practical blockchain applications based on PoW with a flexible number of Miners require a variable difficulty, we use this simplification because it is used by [GKL15], the main theoretical framework for PoW, and thus also used by [KLS16; KMZ17], to which we later want to compare our work using similar assumptions. As pointed out by [GKL15] and [KMZ17], analyzing a constant difficulty is sufficient, because accounting for variable difficulty can be easily achieved by counting blocks proportional to their difficulty. We will further discuss this matter in Section 3.2.3.

We store the additional fields, required for the PoW consensus, in the block data, such that the size of our block header does not change and we do not need to consider any differences between our minimal model for a general blockchain and the special PoW case.

First, *V* verifies that X is connected to B_C and thus part of the same blockchain as B_C by requesting an infix proof $\mathcal{L}_{\text{in}}$ from a prover *P* using the method described in Section 3.2.2. However, till now $\mathcal{L}_{\text{in}}$ is not sufficiently secure to proof that X is part of the current consensus. As shown in Figure 23, an adversary could reuse the existing prefix $B_0 \ldots B_{i-1}$ and only mine a block X' that includes some fake data and valid backlinks to the existing prefix. Therefore, *V* requests a second proof $\mathcal{L}_{\text{su}}$ that puts a certain amount of cumulative PoW on top of X as evidence that X was confirmed by several succeeding blocks and is indeed part of the consensus.

Since each block needs to satisfy the same target value T, each block contributes the same amount of required PoW. We therefore construct a proof $\mathcal{L}_{\text{su}}$ of certain length m confirming block X by m succeeding blocks which inherit m times the PoW of a single block.

An adversary would now need to mine a fake block X' and all m blocks of $\mathcal{L}_{\text{su}}$ to convince *V* that X' is part of the consensus, which is infeasible.

In order to calculate $\mathcal{L}_{\text{su}}$, we adjust the `leapcount` function of Algorithm 3.1 such that we only leap blocks if there are enough blocks left that will increase the cumulative PoW. Therefore, after each block we add to $\mathcal{L}_{\text{su}}$, we need to have $m - |\mathcal{L}_{\text{su}}|$ blocks left that we

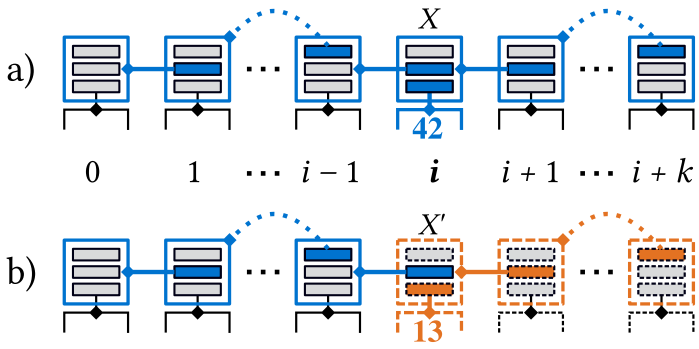

Figure 23: a) Valid prefix and suffix proofs of the block X (blue) with a data value of 42. b) In case no more recent block $Y \succ X$ is known, an adversary could try to provide proofs for a fake block X' with a different value 13. The adversary could reuse the prefix but needs to mine a suffix proof from block i to $i + k$ (dashed orange).

could "walk" block by block to produce the required length. More specifically, the amount of leaps on each leapline b^e with $e > 0$ should satisfy

$$\texttt{dist} - \texttt{leaps} \cdot b^e > (m - |\mathcal{L}_{\text{su}}|) - \texttt{leaps} \tag{3.8}$$

which can be solved for `leaps` and leads to

$$\texttt{leapcount}(\texttt{dist},\ e) = \begin{cases} \left\lfloor \frac{\texttt{dist} - m + |\texttt{leapch}|}{b^e - 1} \right\rfloor & e > 0 \\ \texttt{dist} & e = 0 \end{cases} \tag{3.9}$$

for Algorithm 3.1. If $m > \delta$, the algorithm now adds all δ indices to `leapch` and at least m indices otherwise.

If *V* receives two competing PoW LeapChains $\mathcal{L}_{\text{su}}1$ and $\mathcal{L}_{\text{su}}2$ with different blocks (see Figure 24), we have two possibilities: 1) take the one with more cumulative PoW or 2) challenge the provers by requesting another specific LeapChain. Both cases together allow us to select an arbitrary level of security.

Challenge-Response As mentioned by [Bac+14] as a preferable property, our scheme allows *V* to challenge *P* if the *index* of a block Y is known with $X \prec Y \preceq B_{\text{last}}$. *V* can then calculate one out of several valid LeapChains and request this specific chain as proof from *P*, which makes it more difficult for *P* to create a fake chain. The number of possible LeapChains increases with distance δ, however, we leave determining the exact value as an open question for future research.

Note that a block does not need to store its index because a verifier can always determine all indices as long as one block index (e.g. 0 for genesis) of the LeapChain is known. This property results from the deterministic leap-line assignment based on the block index.

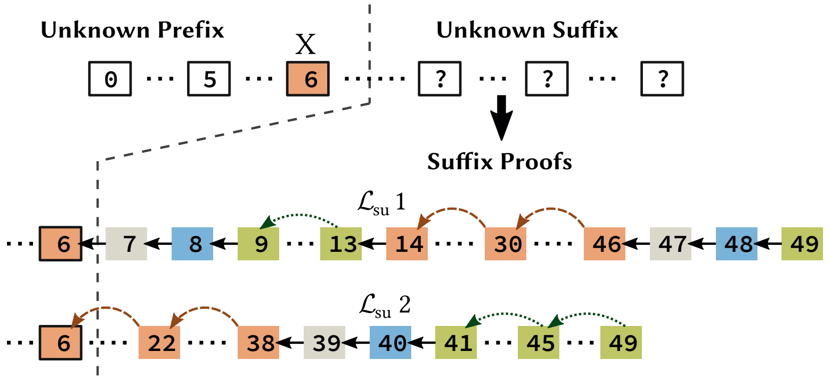

Figure 24: Suffix-proof for consensus verification of block $X = 6$. Here, two valid proofs $\mathcal{L}_{\text{su}}1$ and $\mathcal{L}_{\text{su}}2$ are shown that connect X to a recent block $B_n = 49$, putting 43 additional blocks on top of X. However, $\mathcal{L}_{\text{su}}1$ is longer and proves more cumulative PoW.

3.2.3 Evaluation: Secure Verification with few Kilobytes

For our approach, we first introduce upper bounds to resource requirements, which enable to select an appropriate embedded hardware platform. Afterwards, we experimentally compare LeapChain against related work in a simulation illustrating overall performance gains, and on embedded hardware to underpin LeapChain's feasibility for real-world applications.

Analytical Discussion

Maximum Proof Size As shown in Figure 22, a LeapChain proof $\mathcal{L}_{\text{in}}$ can be divided into 3 parts: 1) an initial walk part to reach the first leapline, 2) a jump part using several leaplines until we 3) walk again to reach the target block. We estimate the maximum size of $\mathcal{L}_{\text{in}}$ based on the worst-case length for each of these 3 parts.

1) The desired leapline can be reached in a maximum of $b - 1$ steps, because the pattern repeats after b blocks. 2) Each leapline is used $b - 1$ times in the worst case, because if we need to jump b times, we could jump once using the next higher leapline. The highest exponent of a leapline we need to consider is $e = \lfloor \log_b(\delta) \rfloor$. Thus, each of the e leaplines

b	4	6	8	12	16
δ_{log}	1029	279945	134 e6	107 e12	295 e18
$w_{\text{max}} = \alpha^{-1}$	256	46656	16.8 e6	8.92 e12	18.4 e18
$\lvert\mathcal{L}_{\text{in}}(\delta_{\text{log}})\rvert$	20	44	76	164	284
size$(\mathcal{L}_{\text{in}}(\delta_{\text{log}}))$	1.92 kB	4.22 kB	7.30 kB	15.7 kB	27.3 kB

Table 3.6: LeapChain parameters for several bases b. The maximum size of $\mathcal{L}_{\text{in}}$ applies when using a hash of 32 bytes (e.g. SHA-256).

adds $b-1$ blocks and between the leaplines we need one additional step block to reach the next lower leapline. This results in $\lfloor \log_b(\delta) \rfloor \cdot (b-1+1)-1$ maximum blocks for the jump part. 3) When using all leaplines, the last leap-width is $w=b$. Thus, we need at most $b-1$ blocks to walk to the target block, but since the target block hash is already included in the second last block, we only need $b-2$ blocks. Combining these results, the proof length is bounded by

$$|\mathcal{L}_{\text{in}}(\delta)| \;\leq\; b \cdot \lfloor \log_b(\delta) \rfloor + 2b - 4, \quad \delta \leq \delta_{\log} \tag{3.10}$$

with the corresponding proof size of $\text{sizeof}(\mathcal{L}_{\text{in}}) = \text{sizeof}(B) \cdot |\mathcal{L}_{\text{in}}|$. When all b leaplines are used, we reach $\left|\mathcal{L}_{\text{in}}(\delta_{\log})\right| \leq b^2 + 2b - 4$ at the maximum logarithmic distance $\delta_{\log}$.

Logarithmic Distance Since the maximum leap-width $w_{\max} = b^b$ is a finite constant, the proof length will only scale logarithmic until a distance

$$\delta_{\log} = b^{b+1} + 2b - 3 \tag{3.11}$$

and scale linearly with a very low slope $\alpha = b^{-b}$ afterwards. $\delta_{\log}$ is derived from the worst cases for each leapline (see previous paragraph) with the difference that the distance includes the target block, resulting in $b-1$ instead of $b-2$ for the last walking phase.
However, b and thus $\delta_{\log}$ can be chosen according to application requirements and for $b=8$ we get already $\delta_{\log} \approx 134 \times 10^6$. If we would be really running out of memory, an embedded device could perform several proof-rounds, each time remembering only the last hash as a checkpoint for the next round.
Table 3.6 provides several parameter sets to illustrate which maximum distance $\delta_{\log}$ can be verified in $\log_b$ scaling and the corresponding bound of steps $\left|\mathcal{L}_{\text{in}}(\delta_{\log})\right|$. Note that these are upper bounds and, in practice, a more efficient proof can be found for $\delta \approx \delta_{\log}$. After $\delta_{\log}$ is reached, $|\mathcal{L}_{\text{in}}|$ increases linearly by 1 every $\alpha^{-1} = b^b$ additional blocks.

Overhead Since we store a single additional hash in each block, the memory overhead over a conventional blockchain corresponds to the size of the hash generated by the used hash function. For Bitcoin, which uses SHA-256 and a full block size of 1 MB [PD16], the memory overhead for nodes storing the full chain would be $32\,\text{B}/1\,\text{MB} = 0.0032\%$. Considering only block headers, the overhead would be $32\,\text{B}/80\,\text{B} = 40\%$, which is compensated as soon as $\delta \geq 1.40 \cdot |\mathcal{L}_{\text{in}}|$.
The computational overhead for full nodes is negligible as each leap-hash belongs to a block for which the hash is already known.

Security

We analyze security based on the difficulty for an adversary to provide a fake proof. Our verification method relies on two different mechanisms, the infix proof for proving the inclusion of a block in the blockchain and a suffix proof for proving that a block is accepted by consensus. The security of the infix proof relies on the integrity of the chain of hashes, while the security of the suffix proof relies on the security of the underlying consensus mechanism, which we will discuss for Proof of Work.

Integrity Guarantees The security of an infix proof $\mathcal{L}_{\text{in}}$ relies solely on the preimage resistance of the hash function H. In order to change any block B_i that existed before a valid and known block B_n, an adversary would need to successfully run a preimage attack on the hash function H. Note that a preimage attack is much more difficult than a collision attack, which only requires to find any two identical hash values and not a specific one. Since the hash of B_i is included in the next block B_{i+1}, the adversary would need to find an alternative block B'_i with the exact same hash as B_i, thus satisfying $H(B_i) = H(B_i)$. For an ideal hash function of κ bits, this requires 2^κ tries on average, which for 256 bit is infeasible. For every block in $\mathcal{L}_{\text{in}}$ either the prev-hash or the leap-hash stores the hash of the previous block and both hash values are stored in the block header which also gets hashed to obtain the current block hash. The working principle of the prev-hash and the leap-hash is the same and thus the integrity of every block, whose hash is included by one of the two hash fields within the infix proof, is guaranteed. As a result, any valid infix proof provides the same security as the full chain regarding its integrity guarantees.

Consensus Guarantees In the case of a PoW blockchain, the security of the suffix proof $\mathcal{L}_{\text{su}}$ relies on its cumulative PoW that an adversary would need to spend to construct a fake proof inheriting the same PoW. The cumulative PoW is expressed as multiple of the *minimum required* PoW ($= 1$ PoW) to mine a single valid block with $H(B_i) < T$. In our scenario, T is assumed to be constant and therefore every block inherits the same PoW on average. As a result, the cumulative PoW of $\mathcal{L}_{\text{su}}$ can be estimated as the length $|\mathcal{L}_{\text{su}}|$ times the average PoW of any block $B_i \in \mathcal{C}$.
Since we place the required *nonce* field in the block-data, a miner would need to calculate two hashes – the data-hash and the block header-hash – for each attempt to solve the PoW, which will increase the computational effort. However, the purpose of PoW is to perform a certain amount of work involving billions of hash operations, so the double hashing can simply be adjusted by the difficulty. As already mentioned, our approach can be easily adapted to variable difficulty by storing the difficulty together with the nonce in the block data. As shown by [GKL15] and [KMZ17], the consensus is then determined by counting the PoW of a block proportional to its difficulty.

Although $\mathcal{L}_{\text{su}}$ provides in principle less security compared to the full chain due to a shorter chain length, LeapChain provides a sufficiently high and more constant security. First, the distribution of the consensus guarantees in the full chain is not constant but increases from the most recent block to the genesis block, making recent blocks less secure than older blocks. Second, the overall security of every block infinitely increases with each new block that is appended to the blockchain, securing already sufficiently secured blocks by an increasingly superfluous amount of cumulative PoW on top of them. Therefore, we use the fact that LeapChain is more efficient than the full chain, in the sense that it allows to freely choose a flexible security parameter $m = |\mathcal{L}_{\text{su}}|$, in order to ensure a constant security level for the suffix proof. For the m most recent blocks, LeapChain provides the same security as the full chain because, in this case, the suffix proof $\mathcal{L}_{\text{su}}$ equals the suffix of the full chain. For blocks that are older than the m most recent blocks in the full chain, the parameter m can be chosen between the shortest possible proof length $|\mathcal{L}_{\text{in}}|$ and the distance δ.

As an example, we assume an attacker *P′* with 10% of all hashing power that wants to convince a verifier *V* of a fake block X' and we set $m = 50$. If the latest block is within the first 50 blocks after X', the attacker would need to mine all blocks from X' to the latest block faster than the honest majority and the likelihood of success can be expressed by the equation from [Nak08]:

$$\text{Succ}(z, q) = 1 - \sum_{k=0}^{z} \frac{(\lambda)^k e^{\lambda}}{k!} \left(1 - \left(\frac{q}{1-q}\right)^{z-k}\right) \tag{3.12}$$

Here, z is the amount of blocks to mine and $\lambda = z\frac{q}{1-q}$ where q equals the hashing power of the attacker. For 50 blocks, this results in $\text{Succ}(50, 0.1) = 7.3 \times 10^{-17}$.

For comparison, most Bitcoin applications require only 6 most recent blocks to trust the consensus as settled [Vuk16; KMZ17], which results in an attack success probability of $\text{Succ}(6, 0.1) = 2.4 \times 10^{-4}$. Note that each attack attempt demands high computational effort, so even if this probability seems relatively high, an attacker would need to spend a great amount of money for every single attack attempt. The security of a blockchain in general relies on the fact that attacks cause huge financial damage to an attacker with overwhelming probability. However, by requiring 50 blocks, $\mathcal{L}_{\text{su}}$ exceeds this basic security and approaches the very small success probability of $\text{Succ}(50, 0.1)$. For blocks older than the first 50 blocks an attacker has more time to mine fake blocks but still needs to continuously mine blocks as every suffix proofs contains recent blocks and the blocks chosen for the suffix proof are changing. As an improvement, one could also require that each suffix proof always contains the b most recent blocks in a row to ensure at very least a security of $\text{Succ}(8, 0.1) = 1.7 \times 10^{-5}$.

Even in the unlikely case that an attacker would manage to provide a fake proof, the verifier *V* which would then receive proofs with different versions of block X, could still challenge

the provers, which would require the attacker to find another fake proof within a few seconds, which is infeasible.
Overall, LeapChain offers a controllable, constant, and high security and we will evaluate its embedded performance for an already very high security level of $m = 50$.

Related Work

Most of the related work on verification of block inclusion is focused on Simplified Payment Verification (SPV) in cryptocurrencies which was already mentioned in the original Bitcoin whitepaper [Nak08]. A light node that wants to verify that a certain transaction is accepted, only keeps the block headers without block data of the entire blockchain and verifies that the transaction belongs to a certain block header. While block headers are much smaller than the full blocks, this "naive SPV" approach scales linear with the length of the blockchain and hence is not feasible on highly constrained devices.

Sidechain SPV Another idea sketched in [Bac+14] suggests that each block creates additional backlinks to *every* previous block using a Merkle tree and including the root hash in the block header. Skipping back is only allowed if the actual PoW of the current block exceeds the cumulative PoW of all blocks in between, resulting in an average proof length of $\log_2(\delta)$. However, [Bac+14] does not evaluate the proof size including the huge amount of additional hashes of the backlinks which would exceed the memory capabilities of embedded nodes.

Skipchains The approach in [Nik+17] proposes a Skipchain $\mathcal{S}_k^l$ where each block stores l backlinks to the k^i-th previous blocks for $0 \leq i \leq l-1$. If $k < 0$, this corresponds to a probabilistic scheme where the number of skipped blocks equals the number of successful Bernoulli trials with probability k. Considering blocks with high PoW, this could be modeled as a $\mathcal{S}_2^l$ for ideal PoW distribution. However, any chosen l is fixed and finite, resulting either in a very limited logarithmic scaling or a large amount of backlinks, increasing the proof size. In contrast to Skipchain, our LeapChain approach only requires a single backlink to achieve logarithmic scaling via its special pattern, which significantly reduces the memory overhead of each block in a proof.

Proofs of Proof of Work (PoPoW) The most optimized state-of-the-art approach is the PoPoW scheme [KLS16; KMZ17], which we use as a benchmark. In this scheme, each block stores a vector of backlinks only to those blocks that randomly happen to inherit a higher (or "deeper") PoW than required. The approach determines a "depth" μ for each hash $H(B)$, such that $H(B) < 2^{-\mu} \cdot T$ and the j-th element of the vector stores the hash of the nearest previous block that satisfies $j \leq \mu$. This means that on average, the j-th backlink points to every 2^j-th previous block. A verifier can request a proof-chain connected by the backlinks of depth j, such that each block of the proof represents at least 2^j times the minimal PoW required for a block.

While PoPoW relies on probabilistic assumption on how often low hashes occur, LeapChain is fully deterministic. The proof size of PoPoW [KMZ17] scales with $\log_2 \log_2 |\mathcal{C}| \cdot \log_2(\delta)$, while LeapChain scales independent of the chain length $|\mathcal{C}|$ with only $b \cdot \log_b(\delta)$ by using only a single backlink.

Simulation

We simulated our LeapChain approach in Python on the block headers of randomly generated blockchains using our block structure. We included random data-hashes as the block data is not relevant for our proof construction. For the hash function $H(\cdot)$ we have chosen SHA-256, which outputs a hash of 32 bytes. As shown by [GP16], the computation time of SHA-256 is linear to its input size. Since hashing is the computationally most intensive operation in the verification, the overall computational effort is proportional to the total amount of data bytes that are hashed.

For comparison, we implemented the Proofs-of-Proof-of-Work (PoPoW) scheme [KLS16] with the interlink vector stored directly in the block header using our block structure. We have not spend the extra effort to implemented SkipChain as well because PoPoW is more optimized than SkipChain due to the selection of specific blocks with high PoW. Therefore, PoPoW should be sufficient for an objective evaluation.

For PoPoW, we also applied suggested optimizations such as storing the vector in a Merkle tree, which requires only $\log \log |\mathcal{C}|$ hashes to be included in the proof for each block. When the prev-hash is used, the interlink vector was omitted to further reduce the proof size. For proof of inclusion, we iteratively selected the interlink with the longest jump that approaches the target block. For the PoW verification, we constructed proofs for all possible interlink depths and then selected the shortest proof that provides the required cumulative PoW.

We measured the following three metrics that are relevant for the verification on an embedded device:

1. The size of the proof in bytes that needs to be transmitted and processed
2. The computational effort for verifying the proof in hashed bytes
3. The security of the suffix proof as its cumulative PoW

Results For embedded devices, the proof size is most important because it corresponds to the amount of data that needs to be received and processed. Figure 25 shows the measured

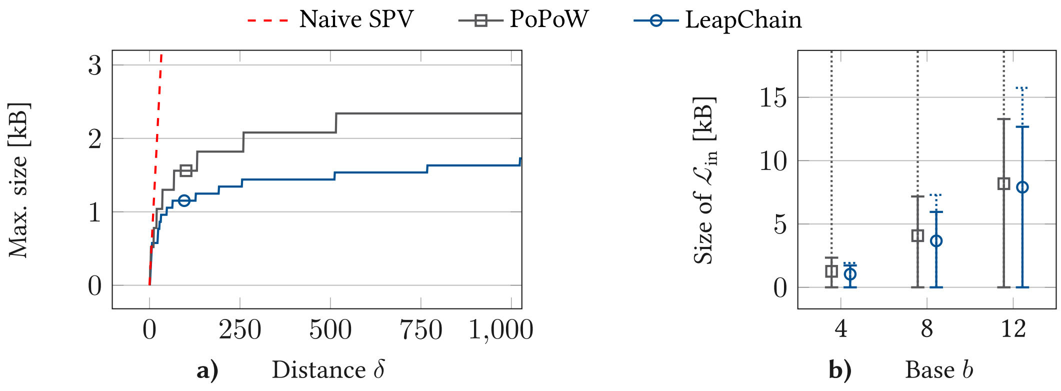

Figure 25: Simulated infix proof sizes when using ideal PoW distribution. a) Maximum proof size for $b = 4$ and $|\mathcal{C}| = 1029$ over δ. PoPoW and LeapChain scale logarithmic compared to the naive SPV. b) Average (marked with □/○), min./max. (—), and analytical upper bound (┄) of proof size over all $\delta = [1, \delta_{\log}]$ depending on b.

size and analytical bounds of infix proofs for several bases b. In contrast to the naive SPV, the maximum proof size of PoPoW and LeapChain scales logarithmic, leading to small proof sizes.

For this measurement, we used ideal PoW distribution for PoPoW, where exactly every 2^{μ}-th block has a hash $< T/2^{\mu}$ and is included in the interlink. Even in this best case for PoPoW, LeapChain provides 11% smaller proof sizes on average.

When using a random PoW distribution, as it occurs in real blockchains, the performance gain of LeapChain is even higher for infix proofs (Figure 26a). Since PoPoW relies on probabilistic assumptions about how often low hashes will occur, proof sizes are subject to a high variance, which leads to peaks of large proof sizes. In the absolute worst case, PoPoW would need to included every block leading to the same proof size as the naive SPV. Regarding the computational effort, LeapChain also significantly outperforms PoPoW. For LeapChain, the hashed data is equal to the proof size but PoPoW requires additional hash operations for looking up the interlink in the Merkle tree.

In the case of suffix proofs (Figure 26b), which require a minimum cumulative PoW, PoPoW comes closer to the size of LeapChain on average but still produces a high amount of peaks that are two to three times larger. Although PoPoW links to blocks that inherit a higher PoW, only the *required* PoW to convince a verifier increases the proof security. A higher cumulative PoW is only of advantage when several competing proofs need to be compared, in which case LeapChain can also challenge the provers. We have chosen a high required cumulative PoW of $\sum \text{PoW} \geq 50$, although Bitcoin already considers a block as part of the consensus if there is a suffix of 6 recent blocks with $\sum \text{PoW} \geq 6$ [Vuk16; KMZ17].

Overall, the high variance of PoPoW proofs poses a critical uncertainty for embedded devices which cannot afford to provide large resource reserves [Sen16]. By contrast, LeapChain is fully deterministic and guarantees tight upper bounds of proof size and computational

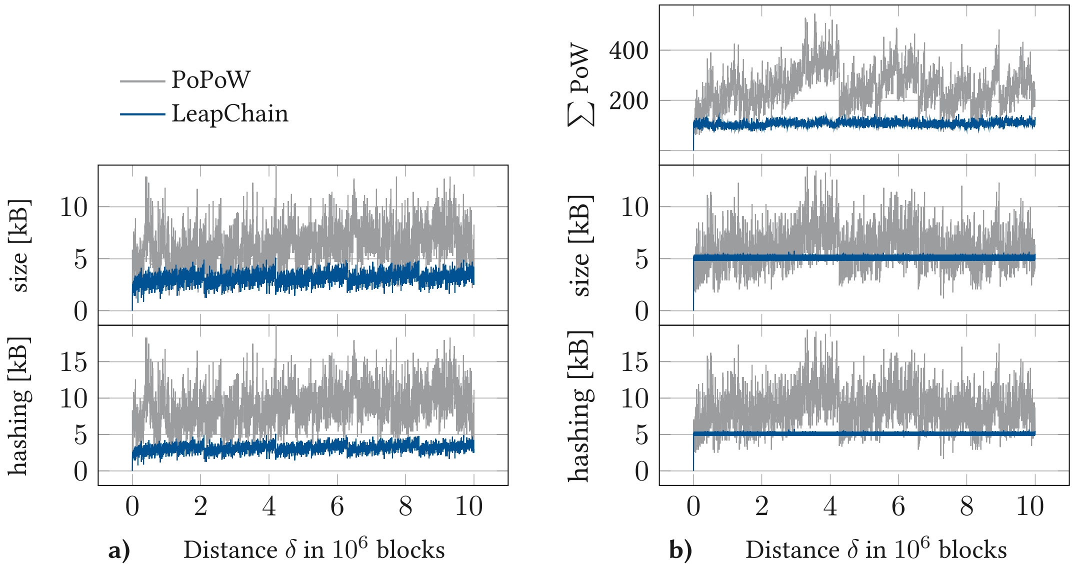

Figure 26: Simulated proofs using $|\mathcal{C}| = 10 \times 10^6$, $b = 8$ and random PoW distribution. a) Infix proofs. b) Suffix proofs with required cumulative PoW $\sum \text{PoW} \geq 50$. LeapChain outperforms PoPoW in both scenarios by achieving lower average and maximum values and a smaller variance for proof size and computational effort.

effort, which are close to measured maximum values.

Running on an ESP32 Chipset

In order to compare the approaches on a real embedded IoT platform, we tested them on the ESP32 chipset (2×240 MHz) using MicroPython v1.8.6 with 57 kB available SRAM on the WiPy 2.0 board. For the implementation we used SHA-256 of the `uhashlib`. We tested 3602 infix proofs with $|\mathcal{C}| = 10 \times 10^6$ and $\delta \in [1, |\mathcal{C}|]$ in steps of $\Delta\delta = 2777$. Our LeapChain approach with $b = 8$ verified all proofs within an average time of 196 ms and a maximum time of 275 ms. PoPoW with ideal PoW distribution (best case) was about 14% slower on average (224 ms) and 29% slower on the longest proof (355 ms), which is in line with our simulation results. Considering its worst-case behavior on random PoW distribution, PoPoW ran out of memory and crashed on the first proof after 133 blocks. Although an optimized C implementation would further reduce hardware requirements, our implementation already demonstrates LeapChain's feasibility on constrained IoT platforms and shows that PoPoW inherits the risk to fail completely.

Summary of LeapChain

Our approach enables embedded IoT devices to verify data integrity and consensus on a blockchain within milliseconds. The LeapChain proof size scales logarithmically with $b \log_b(\delta)$ to the block distance δ while maintaining the same integrity guarantees as the full chain. Although, the security for verifying consensus decisions is not the same as for the full chain, it is very high and adjustable.

For consensus verification of PoW blockchains, LeapChain provides less security on average than the full chain. Nevertheless, LeapChain provides a dynamically adjustable security parameter m and already our selection of $m = 50$ significantly exceeds the security requirements of most existing blockchain applications.

The implementation of LeapChain into an existing blockchain would probably be difficult as it requires changes to the block structure. Therefore, LeapChain should be considered when creating a new blockchain network.

Setting $b = 8$, we could verify the inclusion of any block out of 134 million blocks using at most 76 block headers and outperform existing approaches, such as PoPoW [KMZ17], by at least 11% regarding average proof size. While existing approaches could exceed several megabytes of proof size in the worst case, LeapChain guarantees a deterministic and tight upper bound of 7.3 kB, enabling efficient and safe blockchain applications on embedded IoT devices.

3.3 Secure Time Synchronization via Blockchain[5]

Distributed IoT devices use time synchronization to a global reference time, such as Coordinated Universal Time (UTC), to agree on communication periods, schedule actions and establish an order of registered events. Many applications such as traffic signaling systems [Tra08] or logging events from sensor measurements, require accuracy from a few hundred milliseconds to a second.

Existing time synchronization methods, such as the Network Time Protocol (NTP) [Mil91], rely on the questionable assumption that certain servers can be trusted. In order to authenticate these servers, digital certificates need to be verified, which requires more computational power than many embedded devices can afford. Furthermore, in case of malicious NTP servers, the time to synchronize is severely affected [Mar+10]. This is further aggravated if the gateway is malicious since fallacious time is received independent of the chosen server. Since communication over the Internet involves malicious and faulty nodes, the NTP synchronization can not provide robust and secure timing information for embedded devices.

Consequently, a mechanism is needed that 1. does not rely on individual servers, 2. provides verifiable timing data, and 3. uses only light-weight cryptography that can be computed by resource-constrained devices. We thus aim to achieve secure datetime synchronization for a decentralized network in the presence of malicious nodes.

Blockchain provides a *trustless* system for agreement on a global scale. Each accepted block in the chain is immutable and reinforced by every new block appended to the chain. In case of Proof-of-Work (PoW) chains, even the content can be verified for integrity due to inherent properties of its hash value as we have discussed in Section 3.2. This allows secure verification of any block with a minimum amount of hash operations, which is feasible on most embedded devices [RS18a].

We can therefore extract the timestamps (datetime information) from block headers of the Ethereum blockchain to estimate the current time. Note that we only need to read the headers of *already mined blocks* and are not required to participate in the mining process in any sense. This passive listening approach enables any device with Internet connection to access and verify the information from any public PoW blockchain.

As shown in Figure 27, upon reception of a valid block in a typical blockchain system, each miner will create a new unconfirmed block and starts mining. If the mining is successful, the miner will distribute its new block, which contains a timestamp corresponding to the creation time. The objective is to enable each device to synchronize to a UTC clock independently using the timestamp information in the block headers.

Our evaluation in Section 3.3.1 shows that the timestamps σ and the elapsed time between

[5] Major parts of this section have been published in [Reg+20].

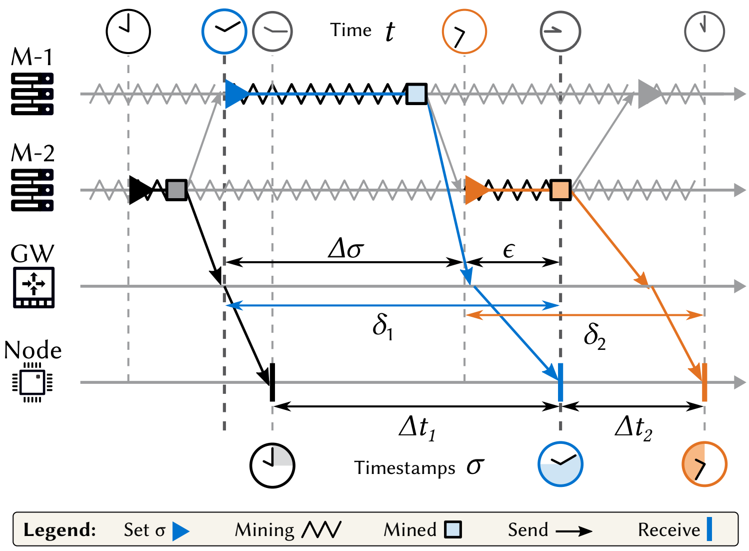

Figure 27: Synchronization scenario. A node receives blocks with timestamps σ_i from two miners M-1 and M-2 via a gateway GW. Upon reception, $\sigma_i = t_i - \delta_i$ because each miner sets the timestamp when it receives the *previous* block. The goal is to estimate t based on σ_i and the observable Δt between receptions.

block receptions Δt are sufficient to estimate the current UTC to an accuracy of one second. However, this is non-trivial due to issues such as timestamp resets, missing blocks, and forks in the blockchain, which will be discussed in this section.

3.3.1 Blockchain Timing Model

In this section, we will measure and evaluate the timestamp information in the Ethereum blockchain. We derive assumptions from our observations, which has been used to create our model of timestamp distribution.

Time Notation

It is important to understand that we distinguish several types of time variables of a block B_i sent by any miner M to our IoT node:

1. The true, absolute, and unknown time $t \in \mathbb{R}$.
2. The elapsed time $\Delta t = t_i - t_{i-1}$ in seconds between the reception of blocks B_{i-1} and B_i that is, in general, observable by the internal clock of any node.
3. The estimated time $\hat{t}_i$ of the true time t_i in epoch seconds.
4. The observed timestamp σ_i in epoch seconds, which is stored in block B_i.
5. The timestamp delay $\delta_i = \sigma_i - t_i$ consisting of the mining time of miner M and the transmission delay to our node.

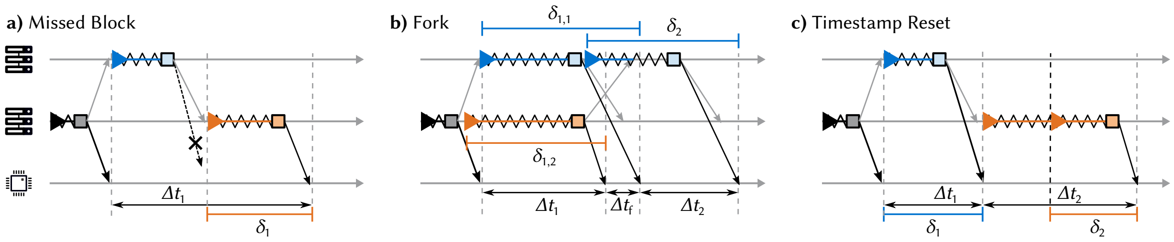

Figure 28: Special cases when syncing via blockchain: a) A missed block occurs when a received block is further in the future than the next block. b) A fork occurs when two valid blocks with the same index i are received. c) A timestamp reset occurs when miner M-2 updates the block timestamp during mining which leads to the condition $\delta_i < \Delta t_i$.

Observation of Block Timestamps

First, we recorded a total of 5100 blocks from Ethereum and analyzed the statistical distribution. We used the `geth` client version 1.8.23 and executed the following capture script.

```
eth.filter('latest', function(error, hash){
  t_localtime = Date.now() / 1000;
  block = eth.getBlock(eth.blockNumber)
  if(block.hash == hash) {
  console.log([eth.blockNumber,
      t_localtime, block.timestamp].join(", "));
  }});
```

We have performed 8 capture rounds of blockchain data from the Ethereum MainNet at four geographical locations: Munich-Germany, Singapore-Singapore, Bangalore-India and Dublin-Ireland. As the map in Figure 29 shows, the connected peers are not locally clustered but distributed among the large cities around the globe.

Challenges when reading Blockchain Timestamps

While the block headers allow global time to be extracted and used, there are numerous challenges associated with the timestamp delays that need to be addressed and resolved.

Gaps A gap occurs when a client misses one or several blocks due to the timeout or disconnection of a peer. As shown in Figure 28a, the IoT device does not receive the block from Miner 1 but only the succeeding block from Miner 2. This results in a time difference between two consecutive blocks (Δt_1) greater than the block time (δ_1) of the recent block leading to a positive offset. Since Δt_1 does not reflect the true delay, it should not be used for estimation. Instead, the IoT device needs to extrapolate the current time estimate from its previous estimation.

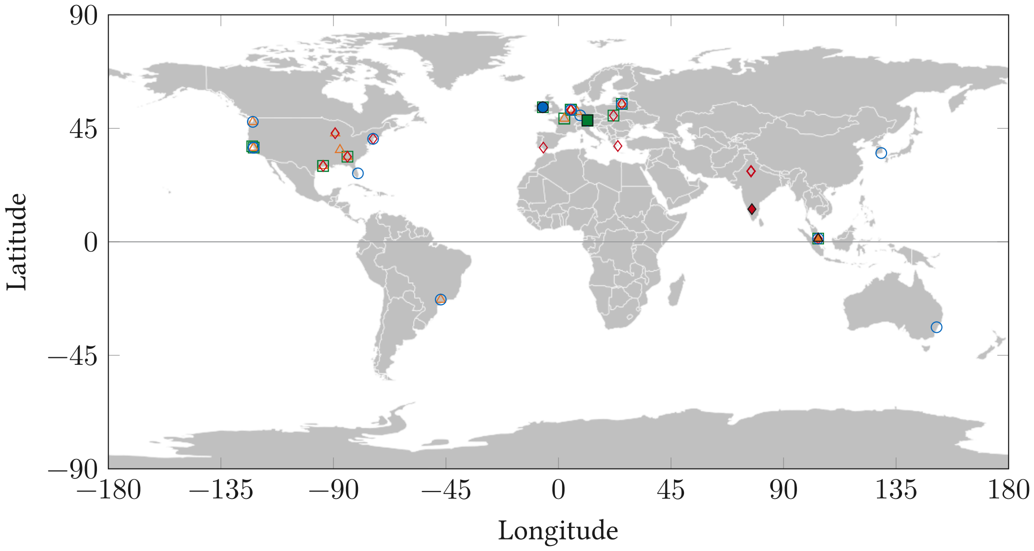

Figure 29: True locations (filled) of our geth clients and estimated locations of connected peers based on their IP address (outline).

Forks A fork happens when two *different* blocks are mined on top of the *same* previous block. In this case, there exist two different versions of the blockchain. The network is initially unsure of which block will be included in the final blockchain and will, therefore, buffer a number of blocks to ascertain which chain will be accepted and continued. An example of a Fork is shown in Figure 28b, where both miners 1 and 2 create a valid block. However, the peers accept only one of the blocks and include it in the blockchain. The other block is discarded. Depending on which block is accepted, the elapsed time between the forked blocks Δt_f needs to be added either to Δt_1 or Δt_2.

Timestamp Reset The assumption that the timestamp σ_i of a block $\mathcal{B}_i$ corresponds to the time when the miner received the previous block $\mathcal{B}_{i-1}$ might not hold. Each miner may, during mining, reset the timestamp to the current time, which is illustrated in Figure 28c. Miner 2 starts mining the block after receiving the block from miner 1. However, the timestamp is reset during the mining process by miner 2 and this is received by the IoT device when the mined block is propagated. This phenomenon results in timestamps being faster than the block reception time similar to gaps, i.e., $\delta_2 < \Delta t_2$, yielding a positive offset.

Observation Results

Our results are given in Figure 30. We found that on average $\Delta t \approx 14.757$ s and $\Delta\sigma \approx 14.743$ s. These values are close to the theoretical blocktime of 15 s. The small difference confirms the fact that miners set the timestamp of a new unmined block immediately after they receive the previous mined block. The true block delay δ, which we measured on

Name	Blocktime	Height	Hashrate (hash/s)	Difficulty
Bitcoin	10 min	5.96 k	90 E	12.75 T
Ethereum	15 sec	8.63 M	180 T	2.75 P
Litecoin	2 min	1.71 M	324 T	11 M

Table 3.7: Common blockchains and their parameters (End 2019)

.

reception against the NTP-synced system time, is on average larger than $\Delta\sigma$. This indicates that miners (who set timestamps on reception) receive the blocks faster than our geth client. This is reasonable given the high-speed networks, as miners choose reliable and low-latency peers in large mining pools to gain an advantage over other miners in the mining race.

As expected, we found the timestamps of the blocks to be monotonously increasing except when forks occur. A key observation from the timestamp delays (δ_i) shown in Figure 30, is that they follow an exponential distribution, aligning with prior observations in literature [DW13]. Among all 5100 measured blocks, we observed 93 forks, 327 gaps and an estimated number of 225 timestamp resets. While forks and gaps are directly measurable from the block index, the timestamp resets are more difficult to identify because they could occur any time and a positive offset could also be caused by receiving the previous block before the miner. Since this is unlikely for large offsets, we found 0.5 s to be a reasonable decision threshold before we consider a positive offset to indicate a reset.

Modelling Timestamp Distribution

Based on our observations, we assume that

- Miners will normally set the timestamp of a new block to be mined at the moment they receive a valid block.
- Miners will reset the timestamp during mining in roughly 4% of blocks.
- Miners are better inter-connected and will receive most new blocks earlier than light-clients. Thus $\Delta t < \delta$ for most blocks.

As observed from our initial experiments, the probability density function (pdf) of the observed timestamp delays of PoW blockchain follows an exponential distribution that is expressed as

$$p(\delta, \beta, \tau) = \begin{cases} 0 & \delta < \tau, \\ \frac{1}{\beta} \exp\left(-\frac{\delta - \tau}{\beta}\right) & \delta \geq \tau \end{cases} \tag{3.13}$$

with the scale parameter β as the average block time in seconds and the shift parameter

	mean [s]	stdev
Δt	14.757	16.82
$\Delta\sigma$	14.743	16.82
δ	15.028	12.88

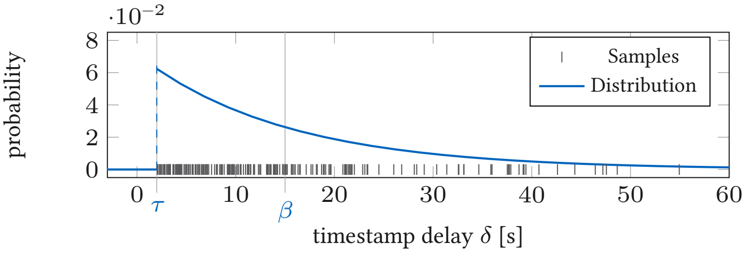

Figure 30: Left: Results of the time stamp analysis given in seconds. Right: Fitted probability density for the timestamp delay of the Ethereum blockchain with $\beta = 15$ s.

τ as the minimum timestamp delay. For example, for the Ethereum blockchain we found $\beta \approx 15$ s and $\tau \approx 1$ s which is plotted in Figure 30.

The distribution of the delays is exponential because the PoW to create a block is a repeated Bernoulli trial. Each attempt to find a nonce that results in a hash value with enough leading zero bits is one Bernoulli trial.

The exponential distribution describes the probability distribution of the time between events in a Poisson point process. The block creation events follow a Poisson distribution because they are of sum of n independent Bernoulli distributed variables, where n is the number of participants in the network. Due to the delay of messages after a block is found, the distribution is shifted. Therefore, we assume a shifted exponential distribution for the overall timestamp delay δ.

3.3.2 Our Time Estimators

In this section, we introduce two different estimation methods that offer minimal drift in observation even in the presence of block uncertainties. Each method has a naive approach, which we will use to illustrate the idea and an improved variant that can be used for the actual synchronization resulting in a total of four estimators. With these four estimators, we create a basic range of diversity for this initial work on blockchain-based time synchronization.

Maximum Likelihood Estimator (MLE)

The MLE is a general approach to estimate the parameters of probability distribution [RP01]. We have chosen the MLE due to its well established mathematical foundation. It can be used in an exponential distribution to estimate timestamp delay only based on timestamp observations. The likelihood function of the timestamp delay distribution is

$$L(\tau, \beta) = \frac{1}{\beta^n} \cdot \exp\left(-\frac{n\left(\bar{\delta} - \tau\right)}{\beta}\right) \quad \text{with } \bar{\delta} = \frac{1}{n}\sum_{i=1}^{n} \delta_i$$

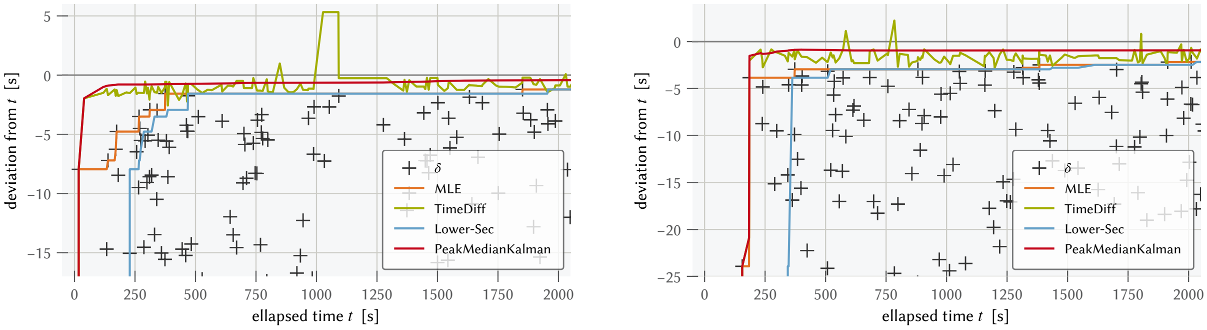

Figure 31: Comparison of all estimators for two capture rounds in Ireland (left) and Singapore (right). For simplicity, we plot the deviation from the true time t such that t corresponds to the x-axis. δ are the true timestamp delays and only shown for reference. The MLE and its variant LowerSec serve as a very secure lower bound. The TimeDiff estimator is closer to t but severely overshoots when a miner resets a timestamp, which can be seen at $t \approx 1000$ s on the left. Due to filtering, the PeakMedianKalman estimator converges fast and without overshooting. As seen on the right plot, there can be an initial delay if the node cannot find enough peers from the beginning.

where $\overline{\delta}$ is the mean of the timestamp observations δ_i and n the total number of the observations. With respect to τ, the function increases until the minimum observed offset δ_{min} and thus, we obtain $\hat{\tau}_{\text{MLE}} = \delta_{\text{min}}$. We can use this result to formulate a simple decision rule for estimating the current time based on the received timestamps. Whenever we receive a timestamp σ_i that is further in the future than our prediction $\hat{t}_i = \hat{t}_{i-1} + \Delta t$, based on previous timestamps, we will switch to this new time $\hat{t}_i = \sigma_i$. This estimation works because the timestamps are always delayed but from time to time we will receive a timestamp with a lower delay than all previous timestamps.

This estimator will not overshoot the true time t and therefore serves as a lower bound for our estimation $\hat{t}$. However, the MLE has two drawbacks, which are visible in Figure 31. First, it converges slowly towards the true time and it will not get closer than τ. Second, since we accept the timestamp from the latest block, an attacker *could* mine one fake block with a timestamp in the future, which would result in a permanently wrong estimation.

Secure Lower Bound Estimator – LowerSec

We now introduce our first proper estimator *LowerSec*, which is the improved variant of the MLE. To overcome the second limitation of the MLE, we accept a timestamp from a block B_i only if m additional blocks have been mined on top, which reinforces B_i. This is the same technique used to consider a transaction settled and provides increasing security with the number of additional blocks m. For our secure lower bound estimator *LowerSec*, we have chosen $m = 10$.

Time Difference Estimator – TimeDiff

While MLE-based approaches only provide for a lower bound, we introduce *TimeDiff* estimator to improve upon the accuracy of time estimation. Remember that the correct time t could be obtained as $t_i = \sigma_i + \delta_i$.

However, we cannot observe δ_i directly and need to estimate it based on the available information. Basically, $\delta_i = \delta_{m,i} + \tau_i + \epsilon_i$ where $\delta_{m,i}$ is the mining time, τ_i the propagation delay to the next miner, and ϵ_i the additional propagation delay to our node. Since mining is a race between miners, we assume that miners use a fast hardware and low latency connection, such that the time between receiving a block and setting the timestamp of the next pending block is negligible. This means that $\Delta\sigma_i = \sigma_i - \sigma_{i-1} = \delta_{m,i} + \tau_i$, which allows us to observe the timestamp delay between miners.

Furthermore, we assume that miners receive blocks earlier than our node, which means $\epsilon_i > 0$. Again, ϵ_i cannot be observed directly, but since $\Delta t_i = \epsilon_{i-1} + \delta_i$ and from our observations we know that $\sum \Delta t_i \approx \sum \delta_i$, we can conclude that ϵ_i is usually small.

Therefore, we can estimate the true time t with

$$\hat{t}_i = \sigma_i + \Delta t_i + Q_{\mathrm{e}} \tag{3.14}$$

where $Q_{\mathrm{e}} = 0.5\,\mathrm{s}$ is the quantization error for timestamps. However, *TimeDiff* produces a lot of peaks due to the issues explained in Section 3.3.1 and overshoots the true time.

Peak Median Kalman Filter – PMK

To overcome the peak problem of TimeDiff, we propose a Kalman filter estimator with the previous highest peaks which runs the median of them. The median of the peaks is fed to the Kalman Filter for the estimation. This ensures that we get a secure and accurate estimation near zero. Using the median among the peaks ensures that a few overshooting peaks are filtered out. Even if there was an accepted overshot peak, the estimation shifts back to the correct value when peaks stabilize. The PeakMedianKalman will converge to the mean of the peaks of the TimeDiff estimations. It not only provides a fast convergence similar to TimeDiff, but also ensures a stable and secure drift.

Kalman Filter In order to smoothen spikes, we use a Kalman Filter [Kal60] because it is a well known and effective estimator for processes with Gaussian noise. The general system equations are given as

$$\hat{t}_n = \hat{t}_{n-1} + \Delta t + \mathcal{W}_{n-1} \qquad s_n = \hat{t}_n + \mathcal{V}_n \tag{3.15}$$

with the time estimate $\hat{t}$, the observed timestamp s, the elapsed time $\Delta t = t_n - t_{n-1}$, the Gaussian process noise $\mathcal{W}_n$, the Gaussian measurement noise $\mathcal{V}_n$ with the following probability distribution:

$$p(w_n) \sim \mathcal{N}(0, Q_n) \qquad p(v_n) \sim \mathcal{N}(0, R_n) \tag{3.16}$$

We assume $Q_n \approx 1.5 \times 10^{-3}$ as the variance of the process noise $\mathcal{V}$, since crystal clocks with at least 100ppm (1×10^{-4}) [TA19] should be very accurate to around 1.5 ms over the period between two blocks. While the measurement noise is not directly gaussian, the standard deviation would be β, and so we set $R_n = \beta^2 = 225$. Overall, the exact values are less important than their ratio, which ensures that the Kalman filter will trust its prediction more than the measurement update.

3.3.3 Evaluation: Synchronization Accuracy within One Second

In this section, we will discuss related approaches and then evaluate our own approach.

Related Synchronization Methods

Time synchronization has been well established in the literature with various goals including energy efficiency, accuracy, speed of convergence and reduction in re-synchronizations. Most of the notable works in the Wireless Sensor Network (WSN) domain [EGE02; LT16] used for embedded IoT devices provide accurate and efficient local clock synchronization without any global datetime information. However, these protocols assume trustworthy data from the nodes which may not be necessarily the case. To handle the presence of malicious devices, [Gan+05] uses message authentication codes and secret keys between two pairs of nodes for verification. The approach discussed in [HPS08] models the temporal variation of messages from neighbors and classifies any deviation as an attack. While the issue of authentication is addressed, the security definitions and the codes have to be regularly updated.

Blockchain has gained traction as the decentralized scheme to provide a trustless and secure means of communication. Open Timestamps [Ope] use timestamp data from Bitcoin to only timestamp and validate documents to prove the authenticity of a document. Relying on its data structure, [Fan+19a] used their data to transmit and store the clock data into a ledger verified by a consensus node. It necessitates the presence of a computationally-capable consensus node for verification. The authors in [Fan+19b] extended the architecture to use Proof-of-Stake (PoS) and a custom blockchain whose length can be controlled. However, the consensus mechanism consumes significant time due to computational complexity and the device of the highest stake has to be re-elected in the event of a failure. By contrast, our approach only requires to passively use specific fields of block headers from freely available public blockchain to achieve datetime synchronization on resource-constrained nodes.

NTP [Mil91] is the most prevalent datetime synchronization protocol in use for network synchronization. NTP relies on a multi-hop client-server synchronization mechanism, with each level called *stratum* synchronizing to the level above it. GPS or atomic clocks generate pulses at stratum 0 to which stratum 1 synchronizes to a few microseconds. As the stratums increase, the synchronization error grows higher with a longer time to achieve synchronization. Despite its popularity, NTP does not provide a way to securely synchronize the clock. While most time servers available for synchronization are at stratum 3, time servers could synchronize devices from even lower stratums. Since response times vary across these servers, the synchronization time could suffer from delays. Additionally, failure of any such server leads to a routing change to a new server, exacerbating the delay. With package/security updates released once a few months, NTP servers are vulnerable to attacks due to obsolete packages. NTP servers operate on a limited bandwidth and resource, requiring them to limit the sync requests to avoid overload issues [NTP16], leading to longer delays and/or failed sync at clients.

Our approach overcomes these drawbacks of security and cryptographic complexity because the blockchain offers peer-to-peer validation and efficient hash verification. As the blockchain network is inherently decentralized, it can also handle a higher number of requests without any overload issues.

Evaluation of Our Approach

We discuss and evaluate our estimators according to convergence time, accuracy, stability, drift, and security. We experimentally tested and analyzed our estimators by feeding them with the σ and Δt of the previously captured data, which we obtained with the geth client.

Convergence, Accuracy, and Stability Figure 31 visualizes these metrics for all estimators for two geographical locations. Table 3.8 shows the results when running the estimation over all 8 captured data sets at 4 locations with a total of 5100 blocks.

MLE and *LowerSec* are slowly converging to the current time but are very stable since the error can only decrease. The estimation will never overshoot the true time, and over time these estimators can achieve a fair accuracy of a few seconds.

The *TimeDiff* estimator provides a faster convergence but is also unstable because each estimation is only based on the last two blocks. The *PeakMedianKalman (PMK)* achieves the best synchronization with an average error of -0.36 s and a standard deviation of only 0.89 s.

Drift Since all estimators use Δt to estimate the time between timestamp inputs, they are equally affected by clock drift. Most internal RC clocks are accurate to 3%, and therefore, the clock drift for measuring Δt between two Ethereum blocks would be around $0.03 \cdot 15\,\mathrm{s} = 0.45\,\mathrm{s}$ which is still reasonable for achieving an overall estimation accuracy within one second. However, if a blockchain with a high blocktime is used, measuring Δt with RC oscillators will significantly reduce accuracy. Since most devices have crystal oscillators with an accuracy of around 20ppm to 100ppm (20×10^{-6} to 100×10^{-6}) [TA19], we have used these deviations for estimating our accuracy and therefore the clock drift is negligible between consecutive blocks.

Security The security of each estimator depends on the impact of forged blocks on the estimation. To forge even one block, an attacker would require exorbitant computing power and would still have a low success probability.

While MLE accepts a valid block immediately and could be tricked persistently by one forged block, *LowerSec* is highly secure since it uses only blocks that are confirmed by 10 additional blocks on top. Forging 10 consecutive blocks of any of the blockchains listed in Table 3.7 faster than the rest of the network is infeasible and therefore it is infeasible for an attacker to convince *LowerSec* of a fake timestamp.

TimeDiff and *PMK* also accept each block immediately and *TimeDiff* could be temporally tricked by a single forged block. Due to the median filtering, PMK cannot be attacked by individual forged blocks at a low rate. In case an attacker could forge blocks at a constant high rate, PMK would slowly converge to the fake timestamps. However, forging at a high rate is infeasible. A detailed discussion on the probability of forging blocks and why it is infeasible, was already presented at Section 3.2.3.

While it is infeasible to create fake blocks faster than the blockchain network, a man-in-the-middle attack could delay existing blocks and therefore convince a node to accept a datetime in the past. However, if the IoT node can persistently store its current time and only accepts more recent timestamps, the attack is mostly limited to delays between sleep cycles. Furthermore, as soon as a few recent blocks are received by the IoT node, its time will be synchronized again to the timestamps of the recent blocks.

	med	avg	stdev	max	tt1s	tt2s
MLE	-1.26	-1.72	1.54	-0.3	12830.0	3846.0
LowerSec	-1.29	-136.65	2740.9	-0.3	13050.0	3966.0
TimeDiff	-0.91	-0.93	1.05	5.3	98.0	35.0
PMK	-0.28	-0.36	0.89	0.6	128.0	32.0

Table 3.8: Estimation errors in seconds given as median, average, standard deviation and maximum. The "Time to 1 s" (tt1s) is the average synchronization time in seconds until the error is within ± 1 s. For tt2s accordingly.

Summary of Blockchain Time Synchronization

We have proposed a novel synchronization method that leverages the public datetime information from the timestamps in block headers. These timestamps are validated and confirmed by a decentralized blockchain network which removes issues such as a trusted relationship and single point of failure found in centralized approaches such as NTP.

We presented four different estimators which use the time difference data between the block reception time and timestamps of consecutive blocks.

The timestamps of confirmed blocks alone serve as a very secure lower bound for the estimation of the datetime with a usual accuracy of several tens of seconds. Under the realistic assumption that a node receives blocks not later than 1 s after the other miners on average, our advanced estimator PMK, which uses the Kalman-filtered peaks of $\sigma + \Delta t$, can improve the accuracy to about 1 second. Overall, our PMK estimator provides the best trade-off between accuracy, convergence, and security.

We evaluated our approach analytically and experimentally to demonstrate the feasibility of using public blockchains for time-synchronization. For all approaches presented, we observe that there is almost no computation and power overhead (due to absence of mining) and negligible memory overhead (storage of 10 timestamps is less than one kilobyte).

These results were obtained by using the Ethereum blockchain, which has an average block time of 15 s. When using blockchains with longer block times, the accuracy of our approach will decrease. Furthermore, our approach can also not mitigate man-in-the middle attacks, which would allow an adversary to forward blocks with old timestamps to the IoT node. However, this attack can be limited by regularly storing the current time in a non-volatile memory. In general, such an attack can only offset the estimated datetime into the past but not into the future.

Overall, our approach only requires a node to passively listen to the stream of block headers to provide a secure and robust synchronization with fair accuracy, which makes it suitable for embedded IoT devices.

3.4 Smart Contracts in Natural Language[6]

Blockchain does not only offer the agreement and storage of data, such as timestamps, but also the agreement and storage of entire programs that should be executed by all participants. These programs are called *Smart Contracts* and are basically scripts on a blockchain that allow to securely automate multi-step trading of digital tokens in a decentralized manner [CD16]. While cryptocurrencies focus on representing money, the traded tokens can represent any piece of information in a complex decentralized process, such as items in a supply chain that are transferred between multiple parties, or a license key for an embedded firmware update that unlocks new capabilities.

Once a smart contract is deployed on the blockchain, its code cannot be altered and its behavior will be enforced by all participants in the network. While this enforcement offers security in the sense that parties can rely on the terms and conditions specified in the contract code, it also inherits a great risk, since any unintended execution path cannot be undone.

In 2016, an unknown attacker exploited an unintended behavior in one of the biggest smart contracts on the Ethereum platform called "The DAO" [Fin16]. This contract was meant to securely automate the fund raising of a Startup but the attacker obtained $50 million worth of Ether currency. The smart contract was vulnerable to recursive calls using an overwritten default function, which allowed the attacker to withdraw money several times from *The DAO* account before it could update its balance correctly [Dai16]. Some people argue that the exploit was part of the specification of the contract and thus legal. However, since the attack involved 15% of all available Ether at the time, the core developers decided to hard-fork the blockchain, which means to create a completely new blockchain that restarts from a block before the hack.

This example illustrates that the correctness of smart contracts according to their intended behavior is crucial for all involved parties. Even if a smart contract is written bug-free, the question remains whether it actually does what people think it will do. A common approach is to formally verify the program code against a specification that is considered to cover the intention. It detects errors because the specification is often more abstract and easier to understand than the code itself. However, verification only shifts the problem to the specification because then the specification needs to be correct and complete by human reasoning. Thus, the behavior is eventually determined by some sort of code and we need to investigate how intention can be expressed clearly in code, such that also non-software developers can reason about it. The IoT aims for nothing less than connecting the entire world but not everyone can understand program code or abstract specifications. To increase the adoption of this promising technology and to enable non-experts to deploy

[6] Major parts of this section have been published in [RS18b].

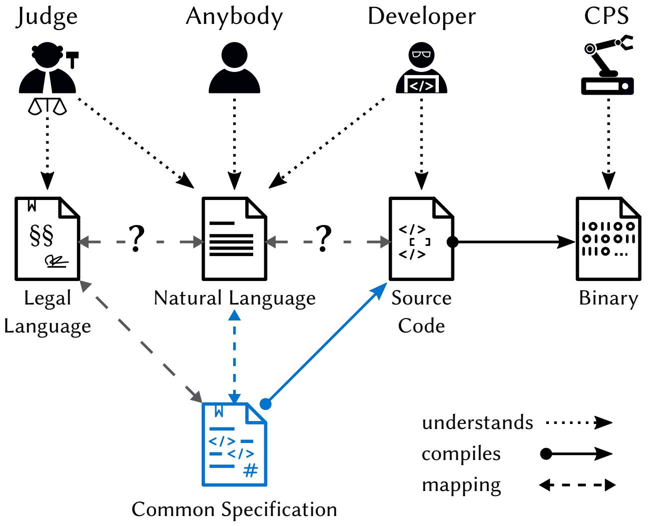

Figure 32: Our vision: A natural language specification that can be compiled to smart contract source code and is legally enforceable in court. To achieve this goal we need an unambiguous mapping between natural language and smart contract instructions.

smart-contracts safely, we need to increase the user friendliness of smart contracts and limit the possibilities for unintended consequences. Human intention is mostly specified in natural language, which is easy to understand for most humans but often highly ambiguous and subject to interpretation.

Most programming languages are unambiguous [Sch65] by providing only a small set of data types and operations from which complex behavior can be specified. However, most programming code introduces an arbitrary mapping to natural language by allowing the developer to freely choose names for functions and data. We simply cannot trust a custom defined function `sum(a,b)` to correctly add `a` and `b` until we break it down to operations that are predefined by the language itself where we have a common understanding of their behavior.

Some domains, such as law, managed to create specifications that express conditions and intentions in a type of natural language that is less prone to misinterpretation *and* provides a common understanding that is continuously reinforced by the decisions made in court. In this section, we want to investigate how natural language concepts can be used to create a smart contract specification language that is human readable, compilable to executable code, and legally enforceable, as it is illustrated in Figure 32.

3.4.1 Smart Contracts: History, Implementations, Requirements

First, we present the working principles of smart contracts, their current implementations and theoretical models, and derive a set of requirements.

The term *Smart Contract* was coined by Nick Szabo as “a set of promises, specified in digital

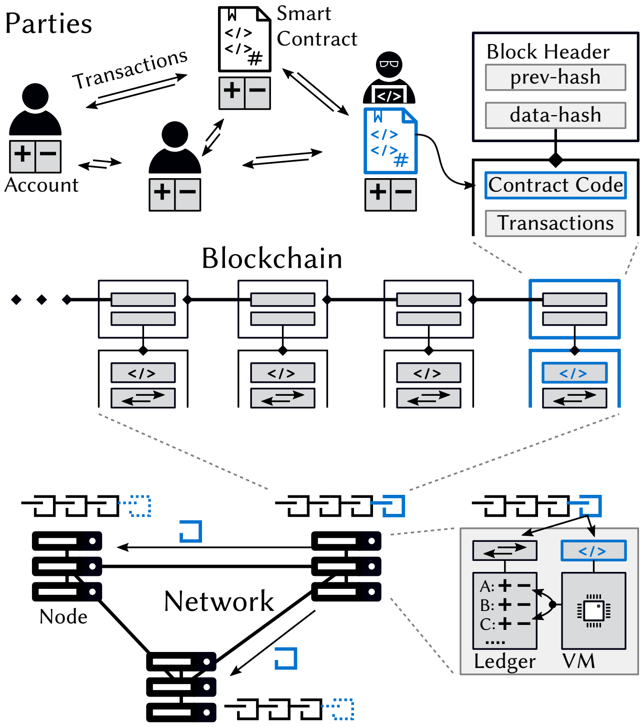

Figure 33: Architecture of a smart contract platform. Contract code and transactions between accounts are stored in a blockchain which is replicated and processed by network nodes.

form" [Sza96] and a transaction protocol that enforces these promises. In general, smart contracts provide a marketplace of services concerning the "exchange and tracking of a digital asset" [CD16]. These digital marketplaces are proposed to be used for automation in many scenarios such as supply-chain tracking, energy trading in smart grids, property renting, or embedded firmware updates [CD16] and might be important for future decentralized CPS architectures.

In its modern implementation, which is shown in Figure 33, a smart contract is a program code that is stored and executed by the network of participating nodes [Zhe+20]. The nodes keep track of the ownership of all existing assets. The assets belong to accounts which in turn belong to the trading parties. A party could be either a human or a smart contract itself.

In its basic form, a smart contract specifies conditions on incoming transactions which will automatically trigger further transactions if those conditions are met. The accounts, which own the assets, are independent from the network nodes, since a network node is just the computer that runs the platform and not necessarily an entity that owns assets. However, in most scenarios nodes get paid for the computation in digital assets and thus need to hold an account as well.

The network nodes are responsible for validating and applying transactions as well as exe-

Platform	Ledger, Consensus	OP Codes / Language	Features
Bitcoin [Nak08]	UTXO, PoW	Script† / Ivy	Linear execution conditions, no loops
Ethereum [But+15]	Accounts, PoW→PoS	EVM / Solidity	General purpose computing
Neo [NEO]	Accounts, D-BFT	NeoVM / C#, Java, ...	Many compilers for high-level languages
NXT [com14]	Accounts, PoS	Templates† / Website Forms	Just parameters, no coding
Corda [Hea16]	UTXO, Raft	JVM / Java, Kotlin	stateless functions, links legal prose
Cardano [Car]	UTXO, PoS	IELE / Plutus	functional programming
Tezos [Pse14]	Accounts, PoS	Michelson / Liquidity	formal verification

Table 3.9: Different platforms that implement smart contracts.
†: language limited and *not* Turing-complete.

cuting the instructions of a smart contract, which in turn could generate new transactions. In most cases, the consensus about the ownership of assets is reached by using a blockchain that keeps track of all transactions of assets that are ever made since the start of the network.

Existing Implementations

In order to identify implementation constraints, we will evaluate several platforms that support smart contracts. Our findings are summarized in Table 3.9.

Details of the mentioned platforms and implementations are described in Appendix A.2.3. We found that most existing implementations either execute low-level byte code in a Turing-complete virtual machine (VM), or restrict contract capabilities to fixed templates that offer simple conditional execution of transactions. Since byte-code languages are difficult to write and read, some offer compilers from high-level programming languages.

The high diversity of the languages used to program smart contracts illustrates the problem that there is no suitable known language yet and the platforms often attempt to create such a language by their own.

Most implementations also identified the need to limit language constructs to achieve a deterministically terminating program which achieves some safety on the execution level. However, on the semantic level, there is almost no effort to provide safety by providing a common understanding for all involved parties. All considered programming languages offer infinite aliasing of operations and data structures.

All smart contract models use the business concept of a specific predefined currency and only allow additional tokens to be issued and traded. However, currencies and tokens could both be derived from the model of a generic asset that is traded between parties.

What is also missing is the concept of permissions. Accounts need to authenticate via a signature to transfer their assets but beyond that there are no restrictions for transactions of assets. However, we think that permissions could also be considered for the trading and issuing of assets.

The alternative approaches from literature separate the executed code from the legal prose and link these two together via parameters. This enables a natural description of the intention in the legal prose while the code could be more standardized. However, there is no way to determine whether the legal prose actually matches the contract code or how any dispute can be resolved in case the code does not completely behave the way it is stated in the prose. Overall, we think that a separation of code and prose is just a temporary solution that introduces new types of problems.

General Requirements

We use the results from our survey and discussion to identify requirements that are elementary for smart contract applications and constrain the design space for a smart contract language.

As we have seen from existing implementations, the execution of transactions including validation of signatures will be handled by the underlying DLT, such as blockchain and PoW, and thus does not necessarily need to be expressed by the contract language. On the other hand, application specific semantics about currencies, sensor values and commands in the domain of IoT are probably too broad to be covered in a single language. Within the IoT, we therefore focus only on the trading aspect of CPSs that exchange data by treating this data as an asset that can be owned and transfered without considering its actual meaning. We see trading as the first and fundamental layer of behavior that a smart contract specifies and enforces. For example, a CPS could trade a license key for a firmware update, unlocking new capabilities. While the exchange of the correct data can be enforced by the contract, the validation of the key and the installation of the firmware is domain specific and needs to enforced by the involved parties. However, further layers could subsequently extend a contract's expressiveness to the application domain in the future.

Trading Ontology: Smart contracts are about trading digital assets with different properties and therefore the contract language should provide keywords and operations in natural language terms that are already used and understood in the real world domain of trading to ease human reasoning about the semantic.

Ownership Management: The type system of the language needs to be able to model owning parties and owned assets. All existing assets need to be globally identified and assigned to a specific owner. If assets can also be created by parties, the type system should also represent which assets were created by which party.

Trading Logic: Smart contracts need to express the logic for trading assets. In its basic form, it needs to specify conditions on a received transaction which will trigger other transactions. This logic could be formalized in general as

$$\begin{gathered}\text{When } A \text{ transfers } x \text{ with properties } P_x \text{ to } B,\\ \text{then } B \text{ transfers } y \text{ with properties } P_y \text{ to } C\end{gathered}$$

where A, B, C are accounts and x, y are assets. A and C could also refer to the same account, to express a kind of exchange contract. To express conditions on properties, the language also needs to express mathematical relations for the comparison of properties.

3.4.2 Our SmaCoNat Language

In this section, we illustrate concepts that can help to design a natural language-oriented specification for smart contracts that allow human reasoning on a high abstraction layer. Afterwards, we provide details about our implementation of such a language, called *SmaCoNat*.

Concepts for a natural Smart Contract Language

Our overall vision is a language that is human readable, safe to use, legally binding, and executable. While this goal seems very ambitious, we want to evaluate concepts that can help to approach it. We focus on two main problems that we think are crucial: readability and safety.

First, current programming languages are hard to read for humans because they are designed to be parsed by compilers and therefore enforce a syntax that contradicts many aspects of natural language. While natural language is context-sensitive and ambiguous, it provides a common understanding by all humans and is much easier to read.

We therefore propose that we should shift programming language syntax towards natural language sentences as long as they can be compiled deterministically to executable machine instructions. For example, giving names to variables instead of directly using memory addresses was a huge step to improve human readability and it did not restrict the ability to compile such a language to machine instructions.

Second, current programming languages are unsafe in the sense that it is easy to write code that expresses a behavior that is not intended. One reason is that only a few operations are defined by the language itself and that a programmer is allowed to create new functions with

Key-Value	Natural Sentence	Hybrid

entity: {
  "type": "Account"
  "name": "Bob"
  "issuer": "Alice"
  "year": "2018"
  "fund": "42 BTC"
}
	Account "Bob"
	issued
	by "Alice"
	on "2018"
	owns "42 BTC".
		Account "Bob"
		issued by "Alice"
		with
		{
		 year: "2018",
		 fund: "42 BTC"
		}.

Table 3.10: Different approaches for specifying properties. A list of key-value pairs could also be specified in a natural sentence using prepositions.

arbitrary names. We therefore propose that we can improve language safety by reducing the possibility to repetitively alias logic and data structures by custom names.

In the following, we explain our specific concepts in more detail.

Limit Custom Naming One source of ambiguity is the possibility to choose own function names. While the sectioning of code into custom functions is fundamental to most programming languages in order to handle complexity, it also allows to alias operations with arbitrary names. Humans and parsers are required to resolve each function name until there are only predefined operations such as mathematical arithmetic.

We should limit aliasing, such that a human only needs to resolve a few names before reaching predefined operations that are built upon common understanding. Finding a suitable balance between the amount of predefined operations and the amount of allowed aliasing would be a task for future studies.

Limit Nesting Another concept that is heavily used in programming languages but not in natural language sentences is nesting of statements. For example, `if`-statements are often nested in a programming language but in natural language we would rather define a list of conditions concatenated using *"and"* or *"or"*.

We should limit nesting of logical structures and should aim for a more sequential specification as it occurs in natural sentences.

Sectioning the Code Structure Most documents group the text into sections. For example, within the first pages, special terms and acronyms are defined and later used in the text. In most programming languages, there is no structure enforced, hence allowing the programmer to declare and define data types any time.

Consequently, we propose a strict separation of data declaration and operational statements. The entire contract code should be sectioned allowing only certain language constructs in each section.

Predefined Type System Most programming languages use a type system that provides only some base types of data, such as Integers, Floats, and Strings and then allow the programmer to define new types derived from them.
Weakly typed languages, such as Python, are most ambiguous because variables can change the type of data they represent by implicit type conversions. Conventional typed languages, such as C, assign each variable an explicit type but variables of the same type may be mixed even they represent different quantities. Strongly typed languages such as Ada, allow to derive distinct types from the same base type, which are incompatible to each other and may only be mixed by explicit type conversion.
For natural languages, we can observe that many types are already implicitly defined. For example, a "Temperature" is completely incompatible to "Velocity", even though both could be represented by real numbers. However, synonyms, such as "Velocity" and "Speed", lead to confusion about compatibility and should be avoided.
For smart contracts, we therefore suggest that each data type or data-structure should be predefined. This is possible because we only focus on the trading logic between CPSs and do not try to cover all possible application scenarios of data structures. Programmers should be only allowed to assign values to these predefined types, which could then be evaluated on a higher application layer. This way, the value of an asset or token could encode a complex data structure as string using, e.g., the JSON format, but this string value would remain meaningless for the semantics of the contract.

Natural Language Syntax Programming languages define special keywords and symbols that are often not or only partially related to a natural language meaning. A statement in a programming language would be easier to understand if it reads like a natural sentence. To achieve this, all keywords and all identifiers in a smart contract should be meaningful words. These keywords should provide one context-insensitive meaning and should be easily distinguishable from each other. Table 3.10 illustrates how prepositions could be used to specify properties of a data structure in a natural sentence.
From analyzing the smart contract platforms and theoretical frameworks, we identified the following list of terms that could provide a trading ontology by answering the questions about *Who*, *What* and *How*:

- *Who:* Entity, Party, Account, Agent, User, Actor
- *What:* Data, Object, State, Message, Asset, Item, Token, Quantity, Currency, Value
- *How:* Transaction, Event, Action, Transition

Another step towards a natural language syntax is a reduced use of symbols. For example, in `C`, the symbol `&` is used as a boolean operation, for accessing a memory address and

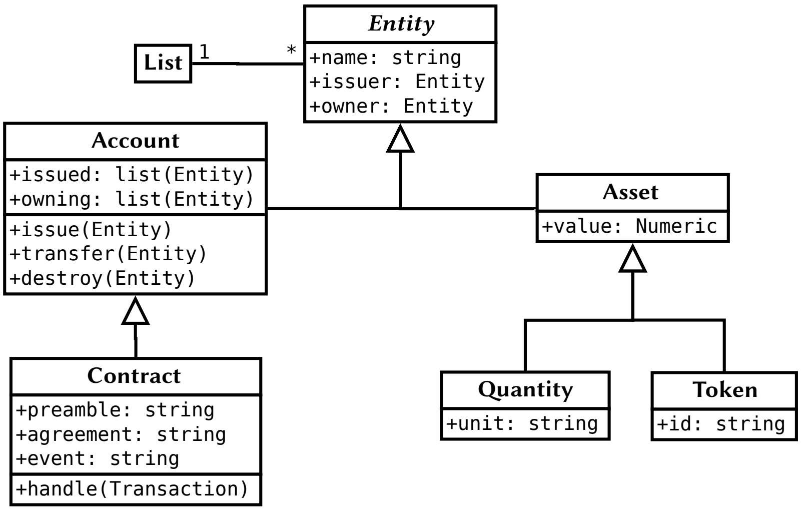

Figure 34: UML diagram of our proposed data structures.

for declaring a reference. These different use cases make it hard to understand `C` code, especially for beginners, since there is almost no correlation to the natural meaning of the `&` symbol. We should also limit the need for parentheses or other delimiters to group several statements or expressions, because nesting of delimiters is a typical source of confusion. For example, instead of using `{` and `}` to mark the body of an `if`-statement, we should use `then` and `end if`, as it is already done by some languages.

Human-readable Global Identifiers Since smart contracts allow everyone to globally register accounts and globally trade assets, we need global identifiers. When it comes to globally identifying data, e.g. specific transactions, cryptographic hashes are the common choice for smart contract platforms because they provide enough entropy to be unique. However, hashes are not human readable and could be easily confused. We should aim for natural language identifiers instead. A successful implementation of this idea are domain names, which are more readable than IP addresses.

We therefore suggest to use a scope system similar to URLs, but based on the issuer of an account. An account would then be identified by its name and the name of its issuer, which could result in an identifier such as `licensekey.alice.company` to identify a license key issued by the account “Alice” which in turn was issued by a globally known “Company”. This would allow any account to issue its own version of an asset called `licensekey` that could be traded.

Implementation of SmaCoNat as DSL

In this section, we use the previously described methods to propose a new Domain-Specific Language (DSL), we call *SmaCoNat* (*Sma*rt *Co*ntracts *Nat*ural), suitable to express smart contract behavior in a natural language syntax. We do not aim for a full-featured language but rather illustrate how to implement our concepts for a small set of types and operations. We implemented SmaCoNat with *Xtext* [ES], a framework for developing DSLs that is part of the Eclipse Modeling Framework. All code examples in the remainder of this section, on how we implemented the language are given in simplified EBNF syntax instead of full Xtext syntax.

Type System and Trading Ontology As discussed before, all data types will be predefined. We model a smart contract by using the standard primitive types, such as Integer and Strings, and a tree-structured hierarchy of a few composite types, which are shown in Figure 34.

Since smart contracts automate trading, we use a minimal ontology for our data model that captures the very nature of trading: Ownership. Therefore, the common abstract class is an *Entity*, which represents everything that can be created and possessed.

From an *Entity*, we derive the concrete types *Accounts* and *Assets*. *Accounts* are the actors/agents in the system that transfer *Assets*. *Assets* are any information that can be traded such as currencies, tokens, or sensor data. *Assets* are *issued* or *revoked* by an *Account*, and can be *transfered* from one *Account* to another.

Enforced Structure In contrast to languages such as `C`, where a valid program is simply an unstructured list of statements, we enforce a certain structure on the first level of the code. Thus, we first define the whole contract code as an ordered sequence of five distinct rules:

```
Contract = Heading, AccountSection, AssetSection, AgreementSection,
    ↪ EventSection;
```

The *Heading* states the contract language and the version of the language. After the heading, all involved accounts and assets must be declared. The agreement section specifies the behavior that will be executed once the contract is signed by all involved accounts. The events specify the behavior of the signed contract when an asset is transfered to the contract. Agreements and events are only allowed to refer to previously declared accounts and assets.

Global Identifiers Identifiers for entities must explicitly name the type followed by the Account names of the chain of all involved issuers until a known Account alias is reached. For example, an account identifier is defined as

```
AccountId = 'Account', NAME, ('by' NAME)*, 'by', AccountAlias;
```

where `AccountAlias` refers to a list of globally known special accounts or a previously defined account alias and `NAME` is a terminal rule that matches strings enclosed in single

quotation marks. We defined the three special account aliases `Self, Genesis, Anyone` and one special asset alias `Input`.
`Self` matches the Account belonging to the contract. `Genesis` is the Account that issued and owns all entities in the initial state of the distributed ledger and has no issuer/owner itself. `Anyone` matches any Account and has no issuer. The asset `Input` refers to the asset that was sent to the contract.

Single Aliasing To avoid repetition of long and unreadable global identifiers, it is allowed to alias accounts and assets during their declaration. For example, the `AccountSection` rule is defined as

```
AccountSection =
  '§ Involved Accounts:',
  (AccountId, ('alias', NAME)?, '.')*
;
```

Logic Operations: A contract may perform basic arithmetic operations on the primitive types. For asset types we define only three fundamental operations:

```
ASSETOP = 'issue' | 'transfer' | 'revoke';
```

Logic Conditions: The contract may also contain non-nested conditional statements on boolean expressions. Boolean expressions consist of the equality relation (`equal to`) for all types and the additional relations `smaller than` and `larger than` for numeric primitive types. All relations may be negated by perpending the keyword `not`.

3.4.3 Evaluation: High Readability with Medium Expressiveness

In general, it is difficult to evaluate a programming language for its safety and expressiveness. In this section we give an example for a valid SmaCoNat contract showing its feasibility to express a typical contract behavior and finally compare other languages regarding our language concepts. The results are summarized in Table 3.11 and will be explained in the following.

SmaCoNat CPS Example

Listing 3.2 shows an example of a smart contract fully written in SmaCoNat. It specifies the behavior of a CPS that manages 42 parking lots by selling parking tickets and controlling the parking barrier. In this scenario involved are the controller and two barriers from a globally known company *AComp* as well as any vehicle approaching the barriers. One

```
Contract in SmaCoNat version 0.1.

§ Involved Accounts:
Account 'BarrierIn' by 'AComp' by Genesis alias 'BarrierIn'.
Account 'BarrierOut' by 'AComp' by Genesis alias 'BarrierOut'.

§ Involved Assets:
Asset 'TheCoin' by Genesis alias 'TheCoin'.
Asset 'ParkTicket' by Self alias 'Ticket'.
Asset 'OpenBarrier' by Self alias 'Open'.

§ Agreement:
Self issues 'Ticket' with value 42.
Self issues 'Open' with value 1.

§ Input Event:
if Input is equal to 'TheCoin' from Anyone
and if value of Input is equal to 0.3
then
  Self transfers 'Ticket' with value 1 to owner of Input.
  Self transfers 'Open' with value 1 to 'BarrierIn'.
  Self issues 'Open' with value 1.
endif

if Input is equal to 'Ticket' from Anyone then
  Self transfers 'Open' with value 1 to 'BarrierOut'.
  Self issues 'Open' with value 1.
endif
```

Listing 3.2: Example program written in SmaCoNat.

ticket costs 0.3 units of the globally known currency *TheCoin* which was issued by the Genesis block.

While this example is very simplified, it illustrates how a system functionality can be mapped to a smart contract. Furthermore, we do not need to perform a lot of checks, such as checking the remaining tickets or validity of tickets because this will be handled on the lower transaction layer by the network. Also in case the central controller fails, any vehicle can still leave the parking lot by transferring its ticket directly to the barrier.

Comparison

Enforced Structure: Enforcing a sectioned code structure is a step towards code safety. By contrast, most languages such as Solidity [Etha] just group statements to functions but allow any structure within a group. Its grammar [Ethb] on the top level is defined as

```
SourceUnit = (PragmaDirective | ImportDirective | ContractDefinition)*
```

and within the body of `ContractDefinition` any order of statements is allowed. Only Liquidity [OCa18] enforces a code structure on the top level such that type declarations are only allowed before the entry point. By contrast, SmaCoNat strictly enforces a structure that separates different language aspects, making it easier to analyze the code.

	SmaCoNat	**Solidity**	**Plutus**	**Liquidity**	**Rholang**
Structure	*section*	function	function	*section*	function
Typing	*predefined*	strong	strong	strong	behavioral
Aliasing	*single*	infinite	infinite	infinite	infinite
Ontology	*trading*	general/ *trading*	general	general/ *trading*	general
Global IDs	*names*	hash	hash	hash	hash
Special Symbols	*few*	some	some	many	many

Table 3.11: Evaluation and comparison of our DSL against other contract languages specified in [Etha; IOH21; OCa18; Luc19].

Type System: Almost all languages use a static and strong type system and allow the programmer to create own types. This does not only introduce aliasing of data instances but also aliasing of data structures. Rholang uses behavioral types, which are worse in the sense that a programmer is allowed to specify custom behavior for the types making types another source of custom named behavior. SmaCoNat only uses predefined types which provide a common understanding.

Expressiveness: Some concepts that make SmaCoNat safer and more readable also pose limitations to the expressiveness of the language. For example, we did not consider loops, which could be enabled to some extent by allowing iterations over lists of entities. While the other languages are considered Turing-complete, most smart contracts do not require loops and Turing-completeness [But+15]. Overall, we believe that SmaCoNat can already express a wide range of behaviors despite the restrictions we put on the language.

Trading Ontology and Identifiers: Solidity predefines some operations on their `address`-type, such as `balance` and `transfer` that are specific for trading. Liquidity also defines trading-specific functions such as `Account.create()` and `Current.balance()`.
All other languages only use a general-purpose computing ontology which makes it impossible to reason about the semantic on a higher abstraction layer. For identifying transactions and accounts, it seems that all considered languages use hash digests that can be assigned to variables with custom names, which makes them more readable but also introduces the aforementioned issues. All considered languages use symbols instead of words for all delimiters and partially for operations. While the general usage in Solidity [Etha] and Plutus [IOH21] can be considered moderate, Liquidity [OCa18] and Rholang [Luc19] make heavy use of symbols to encode semantics. Rholang even overloads symbols with different

meanings depending on the context. SmaCoNat allows symbols only for arithmetic operations and punctuation of natural language such as the period `'.'` to mark the end of a statement.

Summary of SmaCoNat

Smart contracts are promising for secure automation in IoT environments. However, for broad acceptance, we need a readable and safe contract specification that can be directly compiled to executable instructions. Existing implementations lack a well-defined mapping to natural language, prohibiting human reasoning on higher abstraction layers.

Therefore, we derived several language design concepts that can be used to narrow the gap between conventional source code and natural language descriptions to approach a unified contract language. We implemented a DSL called *SmaCoNat* by predefining a small set of operations and data types that allow to directly express the trading logic with predefined operations and limits custom naming of identifiers.

On the downside, *SmaCoNat* is not Turing complete and has only limited features, which makes it less expressive than other smart contract languages, such as Solidity [Ethb]. For example, loops are currently not possible and thus existing smart contracts that rely on loops can probably not be expressed in *SmaCoNat*. Furthermore, we have not implemented a compiler for the language but only specified the syntax rules in the framework `Xtext`. However, our example code with verified syntax, which is shown in Listing 3.2, has demonstrated how simple tasks in the domain of CPS can be specified.

In contrast to existing smart contract languages, *SmaCoNat* enforces a clear code structure, limits aliasing, and builds purely on natural language identifiers, hence enabling a common understanding of code semantics on higher abstraction layers.

This helps to prevent unintended behavior in smart contracts, which can cause severe damage as we have discussed in the introduction (see Section 3.4).

Chapter 4

Lightweight Message Authentication

Contents

Early in the morning, Alice receives an email that says: “*A heavy storm approaches our town, please stay at home today – police department*”. Now these are quite important news. How can Alice verify that this message is authentic and comes indeed from the police and not from someone who is trying to fool her?

Currently, we verify most news by trusting certain publishers of information, e.g. the website of a journal or a public TV channel. In order to remove the trust from publishers, we need to find a mechanism to verify the authenticity of the original message from an authorized entity, such as the police. Since deep fake videos that are generated by neural networks can already be utilized by the average person [Aga+19], we cannot rely on video or audio recordings to be more secure than text messages.

One solution to this problem is cryptographic message authentication where an entity (e.g. the police) attaches some mathematical proof based on secrets to a message that allows the receiver to verify the integrity and authenticity of the message.

In the previous chapters, we came to the conclusion that we can provide verifiable truth by agreement using consensus protocols. We certify that agreement by storing the data in a blockchain, such that other agents can verify its integrity and correctness. However, the correctness of most consensus protocols depends on the ability of agents to authenticate the messages of other agents. In the previous chapters, we have simply assumed that such a mechanism for authenticating messages exists.

Furthermore, message authentication is not only important for consensus protocols, but for communication networks in general. Not all application data can and should be verified by a distributed consensus network and stored in a blockchain. Especially, real-time critical data that is common for control systems often relies on direct communication to satisfy the low latency requirements [CSB19]. In this scenario, some authorized entity (e.g. a sensor) provides the necessary input data to the controller and the controller verifies the authenticity of the received data using message authentication.

Since message authentication is a fundamental requirement – not only for consensus protocols but for communication networks in general – we will investigate different types of message authentication schemes. We will also analyze which authentication schemes will probably remain secure in the future, and how we can use those schemes on resource-constrained embedded devices.

4.1 Overview on Message Authentication

Message authentication is a cryptographic mechanism to guarantee *integrity* and *authenticity* of a message by allowing the receiver of a message to verify that the message was not altered (*integrity*) and sent by a specific sender (*authenticity*). Message authentication can be categorized into two groups [LN04]:

- **Symmetric Message Authentication** uses a shared secret key k to generate a secure random number from a message, which serves as a kind of "digital fingerprint". Only if the receiver knows the same secret key as the sender, it can compute the same random number from the message and compare both values.
- **Asymmetric Message Authentication** uses a key pair consisting of a public key k_{pub} and a private key k_{sec}. The sender uses the private key to sign the message or the hash of it. The receiver uses the public key of the sender to verify the signature.

In the following, we will provide a short overview of these two variants and their suitability for decentralized systems.

4.1.1 Symmetric Message Authentication Codes

A Message Authentication Code (MAC) is a short cryptographic checksum or tag that is appended to a message. For this scheme two parties, Alice and Bob, have exchanged a shared secret key k. In order to construct the MAC μ_m for a message m, the sender Alice will generate a cryptographic random number from the message based on a shared key k. A common choice for the keyed Pseudo Random Number Generator (PRNG) is a hash function H. The construction of the keyed-hash MAC (HMAC) [BCK96] is the following:

$$\mu_m = \text{HMAC}(m, k) = H\left(k \oplus opad || H(k \oplus ipad || m)\right)$$

where $opad$ and $ipad$ are two fixed constants of same length as k. Alice sends the pair m, μ_m to Bob who can now authenticate the message m by constructing the MAC from m and k himself and then comparing it against the received μ_m:

$$\text{HMAC}(m, k) \overset{?}{=} \mu_m$$

Besides this common construction, there exist other hash constructions and constructions from symmetric block ciphers such as *Poly1305* [Ber05], which uses 128 bit Advanced Encryption Standard (AES).

The advantage of MACs lies in their relatively simple design and their fast computation. The most important drawback of MACs is that they require a distinct secret key for every communication link between two agents. Any two agents that want to start communicating, need to exchange a shared secret. In a centralized system, we have a star topology where every client has exactly one communication link to the server. Thus, the server has to create and store a shared secret for every connected client, but each client only has to store one key. However, in a decentralized network, where each agent connects to many peers, the key exchange and the handling of all those shared keys would introduce a high overhead in communication and storage for *every* agent [DPP08]. For this reason, most decentralized applications in larger networks prefer digital signatures over MACs.

4.1.2 Asymmetric Message Authentication with Digital Signatures

Asymmetric message authentication relies on Public Key Cryptography (PKC), which consists of a public key and a private key to construct a digital signature. Digital signatures try to replicate some properties of a conventional handwritten signature but with cryptographic security guarantees. In addition to the guarantees of a MAC, a digital signature also guarantees *non-repudiation*, which means that the signer of a message cannot deny having signed the message under the assumption that the private key is still secret. As of 2022, the two most common types of PKC in use are

1. **RSA** [RSA78]: Invented in 1977, it is one of the oldest schemes and relies on the factorization problem of the product $n = pq$, computed from the multiplication of two large prime numbers p and q. Typical sizes for the prime numbers are e.g. 2048 bits or 617 decimal digits.

2. **ECDSA** [JMV01]: A signature based on ECC over finite fields, which relies on the hardness of the discrete logarithm problem in elliptic curves. Typical sizes for public key and private key are 256 bits, which offers 128 bit of security.

In both cases, each agent generates a key pair, consisting of a distinct public key k_{pub} and a private key k_{sec}, and uses two functions S and V for signing and verification with these keys.

Signatures with PKC In order to construct a signature with PKC, we need a key pair $(k_{\text{sec}}, k_{\text{pub}})$ and a hash function H. The signature σ_m for a message m is computed by signing (e.g. encrypting) the hash of the message with the private key of the sender:

$$\sigma_m = S\left(m, k_{\text{sec}}\right) \overset{e.g.}{=} \text{enc}\left(H(m), k_{\text{sec}}\right)$$

In order to verify the message authenticity, the receiver needs the triple $(m, \sigma_m, k_{\text{pub}})$, where k_{pub} is the public key of the sender. The verification function outputs either true or false for the validity of the signature. For example, we could decrypt the signature with the public key and compare it against the hash of the received message.

$$V\left(m, \sigma_m, k_{\text{pub}}\right) \overset{e.g.}{=} \left[H(m) \overset{?}{==} \text{dec}\left(\sigma_m, k_{\text{pub}}\right)\right]$$

Only the correct sender, which is in possession of its private key, is able to generate σ_m. On the other hand, everyone in possession of the public key can verify the validity of the signature.

Example of RSA Signatures

In order to understand the construction of a conventional signature, we provide a toy example of RSA-based encryption and decryption. The parameters used here are artificially small but the process is the one described in the original paper [RSA78].

Generate Key pair In order to generate the key pair $(k_{\text{sec}}, k_{\text{pub}})$, we

1. Choose two distinct prime numbers, such as $p = 11$ and $q = 7$
2. Compute $n = pq$ giving $n = 11 \cdot 7 = 77$
3. Compute *Euler's totient* function $\phi(n) = (p-1)(q-1) = 10 \cdot 6 = 60$
4. Pick an integer $d < \phi(n)$ that is relatively prime to $\phi(n)$, thus d must satisfy $\gcd\left(d, \phi(n)\right) = 1$. By choosing a prime, we only need to verify that d is not a divisor of $\phi(n)$, for example $d = 47$

5. Compute e as the modular multiplicative inverse of $d \bmod \phi(n)$ yielding $e = 23$, because $d \times e = 1 \bmod \phi(n)$ is solved by $47 \cdot 23 = 1081 \equiv 1 \bmod 60$

One important property of the key pair is that the encryption with one of those keys, e.g. $k_1 = k_{\text{sec}}$, can only be reversed by decrypting with the other key, e.g. $k_2 = k_{\text{pub}}$. The private key is $k_{\text{sec}} = (n = 77, d = 47)$. Decryption function is $m = \text{dec}(C, k_{\text{sec}}) = c^d \bmod n$. The resulting public key is $k_{\text{pub}} = (n = 77, e = 23)$. Encryption function is $C = \text{enc}(m, k_{\text{pub}}) = m^e \bmod n$.

Signing a message: In order to sign the message, we "decrypt" its hash value using the private key $k_{\text{sec}} = (77, 47)$. As noted in the original paper, "deciphering an unenciphered message 'makes sense' [...][because] each message is the ciphertext for some other message" [RSA78]. Due to the property of the key pair, encryption and decryption are interchangeable and the notation is chosen for consistency with the paper.
For example, if we would like to sign the message "Hi from Emanuel", and we assume that the hash of the message is $h = H(m) = 42$, we compute

$$\sigma_m = S(m, k_{\text{sec}}) = \text{dec}(h, k_{\text{sec}}) = h^d \bmod n = 42^{47} \bmod 77 \equiv 70.$$

The final signed message is then the tuple (m, σ_m). While the decimal representation of the intermediate value 42^{47} would have 76 digits, the final value $42^{47} \bmod 77$ can be efficiently computed using repeated squaring and multiplication. Therefore, the computation is also feasible for values in the range of 2048 bit.

Verifying a signature: Anyone who is in possession of the public key $k_{\text{pub}} = (77, 23)$ and receives $(m, \sigma_m) = (\text{"Hi from Emanuel"}, 70)$ can verify the validity of the signature by "encrypting" (in reality decrypting) the signature and comparing it against the hash of the message.

$$V(m, \sigma_m, k_{\text{pub}}) = \left[H(m) \overset{?}{==} \text{enc}\left(\sigma_m, k_{\text{pub}}\right) = 70^{23} \bmod 77 = 42\right]$$

Since 42 is the hash of the message, the receiver will accept the message as authentic.

Applications of Signatures

Digital signatures are used in a variety of applications and especially the modern web, where we purchase and pay for products with online services, would not be possible without them.
For example, the Transport Layer Security (TLS) [Res18] in its version TLS 1.3 from 2018, is used to authenticate communication partners via certificates and establish secure HTTPS

connections, which is the basis for secure web browsing. There are estimates that roughly 60% of all Internet connections are established with TLS to achieve security and that this number is further increasing [SKD20]. TLS offers RSA as well as Elliptic Curve Digital Signature Algorithm (ECDSA) as options for PKC in general and signatures in particular. Digital certificates based on signatures can not only authenticate devices but have also been proposed to secure IoT microservices [PD18; PD19]. Certificates with a short life time of a few minutes are used to sign executable files and their metadata, which allows authentication of service executables even when their metadata (e.g. access rights) changes at runtime.

Another example are cryptocurrencies. As described in Section 3.1.1, Bitcoin uses signatures to verify the authenticity and validity of transactions. In order to spend the Bitcoins that are "stored" on a certain address, the transaction issuer needs to create a valid signature with the private key that belongs to the public key of that address. In fact, most cryptocurrencies rely on ECDSA as their default signature scheme [HC21] and probably most IoT applications do, because ECDSA offers smaller signature sizes compared to RSA for the same security.

4.1.3 The Quantum Computer Threat

For the authentication of messages, PKC has slowly evolved to become usable on embedded devices due to decreased costs for general processing power and hardware accelerators. For example, a TLS connection using RSA-1024 could be established in around 250 ms on a *Raspberry 3B+* [Ham+21].

However, in 1994, *Peter Shor* [Sho94] published an algorithm for quantum computers that allows to efficiently solve integer factorization and discrete logarithm – the two hard problems on which the security of current PKC depends.

By using quantum Fourier transformations, the algorithm can factor large numbers in polynomial time, while the best classical algorithm needs an exponential amount of time. Algorithms for quantum computers exploit the fact that manipulations of a quantum state, which is called *qubit*, "proceeds down all possible paths simultaneously [and] each of these paths has a complex probability amplitude determined by the physics of the experiment. [...] An equivalent way of looking at this process is to imagine that the machine is in some superposition of states at every step of the computation." [Sho94].

While current quantum computers are not sufficiently powerful to perform Shor's algorithm for large numbers with hundreds of digits as used in cryptography, the science of quantum computers is already well understood and the industry is pushing their development and manufacturing.

A first real implementation of Shor's algorithm has already been presented in 2019, when the numbers 15, 21, and 35 (equivalent of 5-bit RSA) were factored on an IBM quantum

Algorithm		Size [Byte] of $k_{\text{pub}}, k_{\text{sec}}, \sigma$			Ex. Time [ms]		Security [bit]	
Name - Variant	**Problem Type**	**Pub.**	**Priv.**	**Sig.**	**Sign**	**Verify**	**Classic**	**PQ**
RSA – 3072	Int. Factorization	387	384	384	3.19	0.06	128	≈ 0
ECDSA – 320	EC Discrete Log	40	20	40	1.32	1.05	160	≈ 0
Dilithium – II	Lattice	1184	2800	2044	0.82	0.16	100	91
Falcon – 512	Lattice (NTRU)	897	1281	690	5.22	0.05	114	103
(qTESAL – pI)	Lattice (RLWE)	14880	5184	2592	–	–	95	95
GeMSS – 128	Multivariate	352190	13440	32	–	–	128	75
Rainbow – Ia	Multivariate	58144	92960	64	0.34	0.83	143	106
(MQDSS – 48)	Multivariate	46	13	20854	10.30	7.25	160	99
Picnic – L1FS	Hash-Based	33	49	34036	4.09	3.25	128	64
SPHINCS+ – SHA256	Hash-Based	32	64	16976	93.37	3.92	128	64
XMSS – SHA256_10	Hash-Based	68	64	2500	–	–	128	64
LMS – SHA256_10	Hash-Based	56	48	2512	–	–	128	64

Table 4.12: Overview of NIST candidates in round 3 in comparison to RSA and ECDSA. Finalists are marked in bold font, alternatives normal font, dismissed candidates from round 2 in parenthesis. Execution times (Ex. time) are measured on an Intel `i5-8350U` processor with 16 GB RAM [SKD20]. Combined values from [SKD20; FF20; Raa+21; Cam+20].

computer with five, six and seven superconducting qubits respectively [ASK19].

Furthermore, quantum computers with 50 qubits (IBM) and 72 qubits (Google) are already in use [Vil+19] and are approaching the point at which they will be able to completely break currently used PKC such as RSA and ECDSA [Mav+18].

Although 1000 qubits would be required to break 160 bit ECDSA [Mav+18][1], it is unclear how fast quantum computers will evolve in the near future and thus, new crypto-systems need to be developed, thoroughly tested, and broadly adopted *before* we reach that threshold. With 41 billion expected IoT devices by the year 2025 [Lue20a], we need to find new solutions that are quantum-resistant and at the same time compatible with the resource constraints of these devices.

Therefore, the National Institute of Standards and Technology (NIST) has started a *Post-Quantum Cryptography Standardization* competition in 2016 with the goal of finding new schemes for PKC that can resist quantum computer attacks [Ala+20]. From the initial 82 submissions, 69 candidates satisfied the minimum requirements and 26 of them moved forward to the second round.

In 2020, NIST announced round three and has selected 3 finalists for digital signature schemes and some alternative candidates for later standardization from round two. An overview of important candidates is shown in Table 4.12.

[1] A more recent study from 2021 came to the conclusion that the number might be higher for practical attacks and around 1465 qubits would be required for 160 bit ECDSA [HC21].

NIST categorizes the schemes according to the underlying hard problem used to construct the key pair. One category consists of hash-based signatures that – as the name suggests – use hash functions to derive the public key from a random secret key.

Hash-Based Signatures Hash-Based Signatures (HBSs) are promising candidates for quantum-secure signatures on embedded IoT devices. Hash functions are very fast because no floating point operations are required and they are often accelerated in hardware. The security of Hash-Based Signatures (HBSs) only relies on the well-studied security properties of the underlying hash function, which makes HBSs very flexible and allows to replace the hash function in case its security becomes compromised. In most signature schemes, hashing of the message is already required, which would allow to reuse the binary code and build a quantum-secure signature based on a single, well-studied cryptographic primitive. However, HBSs produce large signatures and can only sign a limited amount of messages because all signing keys have to be pre-generated before the first use. While the amount of possible signatures can be chosen, the size of the signatures significantly increases the communication overhead, which is a huge challenge in IoT environments where bandwidth is limited.

The category of HBSs is further divided into *stateful* and *stateless* schemes. A stateful scheme needs to remember which hash values have already been used to sign messages and they must not be reused afterwards in order to ensure security. Since careful state management is required, NIST does not recommend to use stateful schemes as a general signature scheme.

However, *stateful* schemes need less computation and have smaller key sizes compared to *stateless* schemes because they involve less hash values overall. For this reason, stateful schemes are especially interesting for IoT applications and NIST has published recommendations that stateful schemes may be used for resource-constrained devices [Coo+20].

For example, the IETF also considers stateful HBS for secure firmware updates of IoT devices [Mor+19].

4.2 Hash-Based-Signatures: History and Approaches

We will now introduce the core elements and working principles of Hash-Based Signatures (HBSs), which are necessary to understand our contributions. Hash-based signature schemes generate key pairs by using cryptographic hash functions. Most schemes first generate several hash-based Few- or One-Time Signatures (OTSs) and later combine these with a Merkle Tree to create a Many-Time Signature (MTS), which can be used for a large number of signatures. For an OTSs, a random value (private key) is used to generate a set of secrets, which are then individually hashed. All these individual hash digests constitute the public key of the OTS. Since the secrets correspond to the preimages of the hash function, it is infeasible to guess the secrets from the output image (public key). In order to sign a message, a certain set of the secrets is revealed and transmitted together with the message. The receiver then needs to verify that this set of secrets (the signature)

- uniquely encodes the message digest and
- belongs to the public key.

Revealing a unique combination of secrets can only be done once. While some schemes allow to reveal secrets from the same public key a few times, it will always lower its security, and for most schemes (e.g. OTSs) using them more than once already means broken security.

In order to create an MTS based on OTSs, the public keys of several OTSs are combined using a Merkle Tree, which is a binary tree of hashes. The root hash of the Merkle tree is then the overall public key of the scheme and each message is signed by an unused OTS at the leafs.

In the remainder of this chapter, we assume a single cryptographic hash function $H(\cdot)$ that outputs a hash of n bytes or $N = 8n$ bits:

$$H(\cdot) : \{0,1\}^* \to \{0,1\}^{8n} \tag{4.17}$$

4.2.1 Lamport One-Time Signature

In 1979, L. Lamport proposed the first hash-based OTS by using $2N$ hashed secrets in pairs of two in order to encode a message digest of N bits [Lam79]. The hashes of $2N$ secrets are distributed as public key and each pair of secrets is used to encode one bit of the message digest d. Depending on the bit value, one of the two secrets is revealed in the signature while the other one is kept secret. Figure 35 illustrates the concept for a 6-bit hash function.

4.2.2 Winternitz One-Time Signature (WOTS) and WOTS+

The core idea of the Winternitz One-Time Signature (WOTS) [Mer89] is to sign multiple bits of the message digest using only one secret of the private key. This is achieved by iteratively hashing each secret w times instead of only once, resulting in several chains of hashes. For signing, we group $\log_2(w)$ bits of the message digest d together, where w is the so called Winternitz parameter. The grouped bits then encode the position of a hash within the chain that will be revealed.

Since revealing any hash within a chain also indirectly reveals all succeeding hashes in the chain, an attacker could easily forge a signature. For messages where the digest bits are larger or equal to the original message in every group, the attacker could simply reveal a

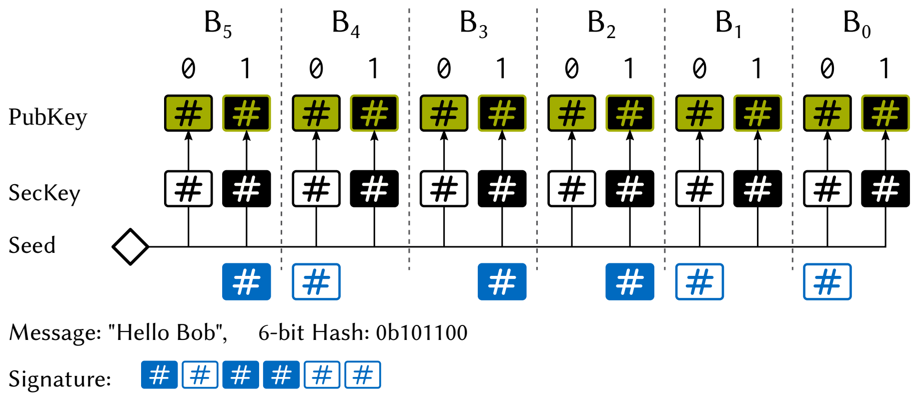

Figure 35: Concept of the Lamport OTS for $N = 6$. A set of $2N$ secrets are hashed and the hashed values are the public key. The message "Hello Bob", which should be signed, has a hash of `101100`. Each bit of the message hash determines whether the left or the right secret of each group of two secret hashes is revealed and attached to the message.

hash at an increased position in the hash chain. In order to prevent this attack, a checksum C is appended to the message that encodes the sum of all indirectly revealed hashes, ensuring that an attacker who wants to increase digest bits must also decrease bits of the checksum C at the same time. This is practically infeasible since in either case the attacker would now need to find the preimage of a given hash.

The checksum is calculated as

$$\begin{aligned} C &= C_{\text{max}} - d = (w-1) \cdot \ell_1 - d = \\ &= \sum_{i=1}^{\ell_1} ((w{-}1) - \text{BASE}_w(d_i)) \end{aligned} \tag{4.18}$$

where d_i is the i-th group of w bits of the message digest, w the Winternitz parameter and ℓ_1 the number of hash chains needed to encode the message digest d.

In total, a WOTS requires $\ell = \ell_1 + \ell_2$ separate hash chains of length w. The first ℓ_1 chains are used to encode the message digest d and the last ℓ_2 chains are used to encode the checksum. Both values depend on the chosen w and are calculated as

$$\ell_1 = \left\lceil \frac{8n}{\log_2(w)} \right\rceil \qquad \ell_2 = \left\lceil \frac{\log_2(\ell_1(w-1))}{\log_2(w)} \right\rceil \tag{4.19}$$

The public key consists of a single hash, which is calculated as the hash of the last hashes of all hash chains concatenated together. Note that the related approach XMSS [Hül+18] uses a binary tree instead of concatenation to obtain a single hash.

Figure 37 illustrates an example WOTS where we use $n = 16$ and $w = 4$, which results in $\ell_1 = 8$ and $\ell_2 = 3$. With such an (insecure!) scheme we could sign each $\log_2(w) = 2$ bits of the message digest d by one chain. The maximum checksum would be $C_{\text{max}} = (4-1) \cdot 8 = 24$ and so we need $\ell_2 = 3$ chains c_i for the checksum. In our example, the

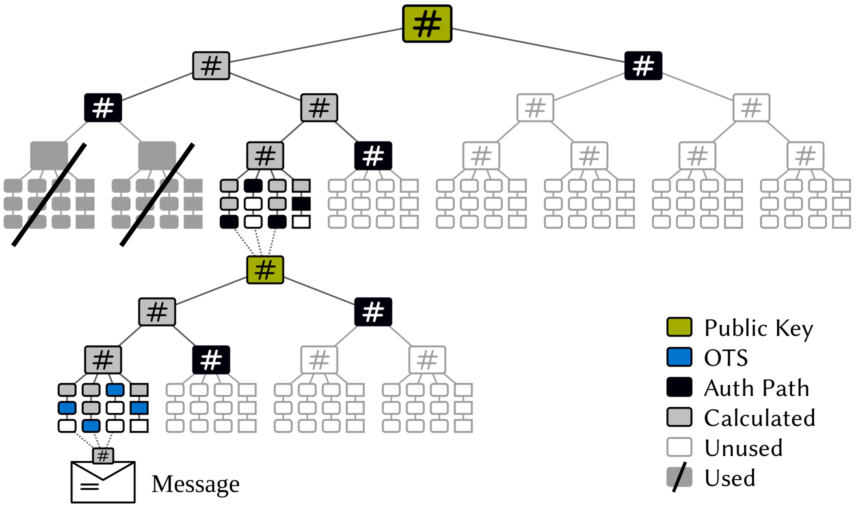

Figure 36: Hash-based MTS with two levels. The hash of the message is signed by a Merkle signature, whose root hash is again signed by a second Merkle signature.

checksum is $C = 24 - (2 + 3 + 0 + 2 + 1 + 3 + 3 + 1) = 24 - 15 = 9 \equiv 021_4$. We will explain in Section 4.3.1 how we can reduce ℓ_2 to 2.

WOTS+

One problem with WOTS is that the security of the OTS is lower than the security of the used hash function because an attacker just needs to find a preimage for *any* known hash to change the signature. This reduces the number of expected trials until a preimage is found from 2^n to $2^{n-\ell}$.

WOTS+ [Hül17] mitigates the problem by using random values $\vec{r}$ that are XORed with every intermediate hash *before* hashing it again. XORing individualizes the hash calls and ensures that each trial in a brute-force attack is only valid for a single target hash. However, the random values are part of the public key, increasing its size by $n(w - 1)$ bits, which is w times larger compared to the conventional WOTS.

4.2.3 Merkle Tree

A Merkle Tree is a binary tree entirely made of hash values. Every node corresponds to the hash value of the concatenation of its two child hashes: $h = H(h_{\text{left}}||h_{\text{right}})$. The Merkle tree is used to link several WOTSs together and create a single public key which is the root hash of the Merkle tree.

As the name suggests, a WOTS can only be used to sign a single message because the signature contains parts of the secret key. In order to sign several messages with one key pair, the secret key is used as a seed to generate l distinct WOTSs. The l public hashes of

n	w	ℓ_1	ℓ_2	$\|\sigma_{\mathrm{W}}\|$	#H
10 B	4	40	4	440 B	176
10 B	16	20	3	230 B	368
10 B	32	16	2	180 B	576
16 B	4	64	4	1088 B	272
16 B	16	32	3	560 B	560
16 B	256	16	2	288 B	4608
20 B	4	80	4	1680 B	336
20 B	16	40	3	860 B	688
20 B	32	32	2	680 B	1088
32 B	4	128	5	4256 B	532
32 B	16	64	3	2144 B	1072
32 B	256	32	2	1088 B	8704
40 B	4	160	5	6600 B	660
40 B	16	80	3	3320 B	1328
40 B	32	64	3	2680 B	2144

n	h	|auth|	|leafs|
10 B	8	80 B	3 kB
10 B	10	100 B	10 kB
10 B	16	160 B	655 kB
16 B	8	128 B	4 kB
16 B	10	160 B	16 kB
16 B	16	256 B	1049 kB
20 B	8	160 B	5 kB
20 B	10	200 B	20 kB
20 B	16	320 B	1311 kB
32 B	8	256 B	8 kB
32 B	10	320 B	33 kB
32 B	16	512 B	2097 kB
40 B	8	320 B	10 kB
40 B	10	400 B	41 kB
40 B	16	640 B	2621 kB

Table 4.13: Typical parameters for WOTS (left) and Merkle trees (right) depending on the underlying hash size n. $|\sigma_{\mathrm{W}}|$ is the size of the WOTS and #H the number of hash operations to generate it.

these l WOTSs are then used as leaf hashes to build a Merkle tree of height $h = \lceil \log_2(l) \rceil$.

4.2.4 Existing Hash-based Signatures: LMS and XMSS

There are two other important implementations of stateful hash-based signature schemes (LMS and XMSS) [Cam+20] which we will describe in this section. We modified their code to measure the number of hash calls.

Name	GitHub Repository	Version
LMS [MCF19]	github.com/cisco/hash-sigs	d2db1b2
XMSS [Hül+18]	github.com/joostrijneveld/xmss-reference	bb2d285

Leighton-Micali Signature (LMS)

LMS is a stateful HBS and actively developed as RFC 8554 [MCF19]. LMS uses a security string that is prepended to the input of every hash invocation to mitigate preimage attacks when multiple images of the same hash function are known. The security string is distinct for every hash invocation within and between signature trees, such that any given hash image needs to be attacked with its individual security string.

The security string is up to $21{+}n$ bytes long and consists of 6 parameters $(I, [r|q], D, [\varepsilon|j|C])$ [MCF19]. I is a random 16 B identifier for the key pair, r or q are the 4 B index of either the node in an authentication path call or the leaf index in an OTS hash call, D is a 2 B identifier for the context in which the hash function is invoked, j is a 1 B iteration number for the private key, and C is a n byte random number only used when the message is hashed.

LMS also supports a hierarchical tree structure with several layers of LMS subtrees.

The keys are stored as $privkey = (type, I, seed)$ and $pubkey = (type, I, h_{\text{root}})$, where seed and h_{root} are both n byte and type is four byte. The signature consists of $sig_i = (i, \sigma_i, type, \text{auth}_i)$ for the i-th leaf.

The adjustable parameters are $w \in \{2, 4, 16, 256\}$ and $h \in \{5, 10, 15, 20, 25\}$. The only hash function for all combinations is SHA-256.

XMSS and XMSS^{MT}

The eXtended Merkle Signature Scheme (XMSS) [Hül+18] consists of a binary hash tree of height h and the 2^h leaf hashes are the root hashes of WOTS+. In the multi-tree variant XMSS^{MT}, several layers of trees are used to increase the total number of signatures.

In contrast to LMS, XMSS only requires a second-preimage resistant hash function because it uses additional bit masks to enhance security [HRS16]. Instead of a security string, each node of the tree is XOR-ed with random bit masks.

The signature therefore contains the 32-bit leaf index i, the n-byte random seed r for the masks, a WOTS+ signature, and an authentication path, summing up to $|\sigma| = (4 + n + (\ell + h) \cdot n)$ byte [Hül+18]. The public key consists of typestring, Merkle root, and the seed r, so $(4 + n + n)$ bytes.

While the XOR masks allow to prove security in the standard model, they also increase the amount of data needed to verify the validity of a signature and therefore contradict our goal of an IoT-suitable solution.

Furthermore, XMSS uses so called L-Trees to calculate the root hash of each WOTS+, which increases the number of hash operations compared to concatenating all top hashes in one call.

4.3 Adaptive Merkle Signature Architecture[2]

We propose a new MTS, which we call Adaptive Merkle Signature Architecture (AMSA), based on the Winternitz One-Time Signature. One of the main differences is a reduced size of the signature and more parameter choices. Our implementation is published on GitHub [TUM19] for review and further research. This section will describe our optimization mechanisms and implementation details.

4.3.1 Improved MinWOTS

We now introduce our efficient variant called MinWOTS, which reduces the signature size while retaining the full security of the original WOTS. Basically, we reduce the signature size by using a separate and higher w_c for the checksum bits, which is based on the idea discussed in [EGM90].

A typical parameter choice for WOTS is $n = 16, w = 16$ which results in $\ell_1 = 32, \ell_2 = 3$ for the conventional WOTS. Here, we can encode $w^{\ell_2} = 4096$ checksum values. However, the maximum possible checksum value is $w \cdot \ell_1 = 512$, which means that 3584 encodings are not used. Choosing $\ell_2 = 2$ does not work as it can only encode $w^2 = 256$ values. We therefore allow a different hash-chain length w_c only for the checksum bits to encode these bits more efficiently. In our example we would choose $w_c = 23$ to encode up to $23^2 = 529$ values, leaving only $529 - 512 = 17$ encodings unused. More general, w_c is calculated as

$$w_c = \left\lceil \sqrt[\ell_2 - 1]{\ell_1 \cdot w} \right\rceil \tag{4.20}$$

with ℓ_2 from the conventional WOTS as stated in Equation 4.19. The found w_c allows us to reduce the number of required hash chains for the checksum by one, meaning $\ell_2' = \ell_2 - 1$. For many usable parameter sets this will reduce $\ell_2 = 3$ to $\ell_2' = 2$, saving n bytes of signature size.

Note that the maximum amount of hash operations for the checksum also decreases from $3w = 48$ to $2w_c = 46$, since the full chain needs to be hashed during signing and verification.

Full Rootkey Hashing Another improvement we adopt from [MCF19] is to generate the WOTS-rootkey by hashing all WOTS-pubkeys at once, without using a tree. This reduces the amount of hash operations from $n \cdot (2\ell - 2)$ to $n \cdot \ell$ bytes. For $n = 10, \ell = 23$ this would reduce hashed bytes from 440 to 230.

[2] Major parts of this section have been published in [RS20].

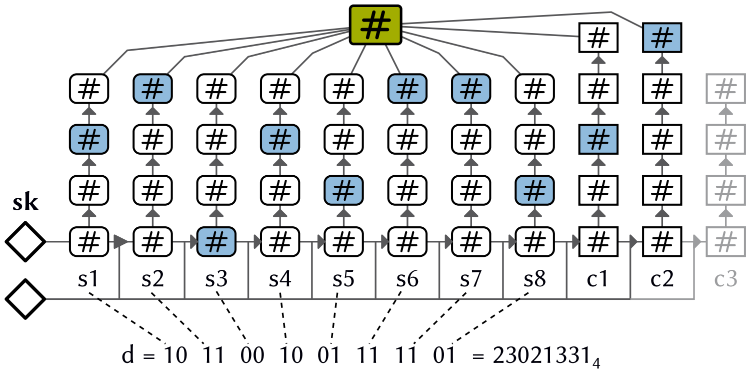

Figure 37: Concept of WOTS illustrated for $n = 2$ and $w = 4$. The digest d is split into equally sized chunks of $\log_2(w) = 2$ bits and each chunk is encoded by one hash-chain s_i. In our variant we allow checksum-chains with different lengths which reduces the total number of chains by one (c_3 not needed).

4.3.2 Further Tree-Construction Improvements

Security String

In order to avoid lowering the security for parallelized brute-force attacks, we use a security string to individualize hash calls. However, we simplify and reduce our security string to a fixed size of 16 random bytes I, which is similar to the identifier I from LMS.
Therefore, we adjust the first 5 bytes of I by setting them to index values in the following way:

- $I[0]$: the WOTS chain index $i_c \in 0..\ell - 1$
- $I[1]$: the hash index within a WOTS chain $i_h \in 0..w - 1$
- $I[2..4]$: the Merkle node index $i_m \in 0..2^{h+1}$

The remaining 11 bytes are unique for the entire key pair and avoid attacks on several keys at once. When calculating the message hash, WOTS public key, or during Merkle hashing, $I[0] = I[1] = 255$. This way, we ensure that no two revealed hash values, for which the preimage is unknown, can be targeted by the same preimage guess. Note that for each WOTS leaf, the hash calls during WOTS chaining, WOTS public key generation, and Merkle traversing, require already different preimages by design since they have different input lengths: n, $n \cdot \ell$, and $2n$ respectively.

Typecode

The specific parameter choices for n, w, and h are stored in a typecode, which will be part of the public key. While XMSS and LMS use 4 bytes, we use a 2 byte encoding (bits 0 till 15) that is shown in the following table:

Bits	Param	Values
`0..1`	H	`00`: SHA-256, `01`: BLAKE, `1x`: reserved
`2..5`	n	`00`: 10, `01`: 16, `10`: 20, `11`: 32, `1xxx`: reserved
`6..7`	w	`00`: 4, `01`: 16, `10`: 32, `11`: 256
`8..11`	h	$4 + x$ (values from 4 to 20)
`12..15`	–	reserved

Note that we allow any height h between 4 and 20 to enable a better optimization of the scheme to the available resources. In contrast for LMS, where $h \in \{5, 10, 15, 20, 25\}$, choosing $h = 15$ could be already too large for embedded devices, while $h = 10$ only allows to sign 1024 messages.

Multi-Layer Merkle Tree

We construct the Merkle Tree similar to LMS and XMSS and allow several layers of trees. A two layer scheme is illustrated in Figure 36.

Each Merkle tree in each layer has its own security string and typecode. In contrast to XMSS, where each tree in each layer needs to have the same typecode, LMS allows different typecodes between layers but not between trees. AMSA goes even further and allows choosing any valid parameter set for each tree.

While several layers will increase the signature size, they will significantly reduce the key pair generation time and allow to offload a larger fraction of the signature to a gateway, which will be discussed in the next section.

Trade-off: Private Key Compression

There are several possibilities which data is stored as the private key. The fastest signing process can be achieved by storing all hashes of all OTSs and all Tree hashes as the private key. During signing, all required hashes can be picked from memory without any re-computation. While for $(n, w, h) = (32, 16, 10)$ this would mean storing 35 MB as private key, for $(n, w, h) = (32, 256, 16)$ it scales to 18 GB.

On the other side of the spectrum, only the initial seed for generating all WOTS leafs could be stored but then the entire tree needs to be recomputed for each signature [MCF19].

Therefore, we use our own variant of the Merkle tree traversal algorithm [BDS08], which caches all right nodes in the Merkle tree and only h left nodes. While the left nodes of the tree are calculated when traversing the leafs for each new signature, right nodes are the computationally most expensive nodes to recompute.

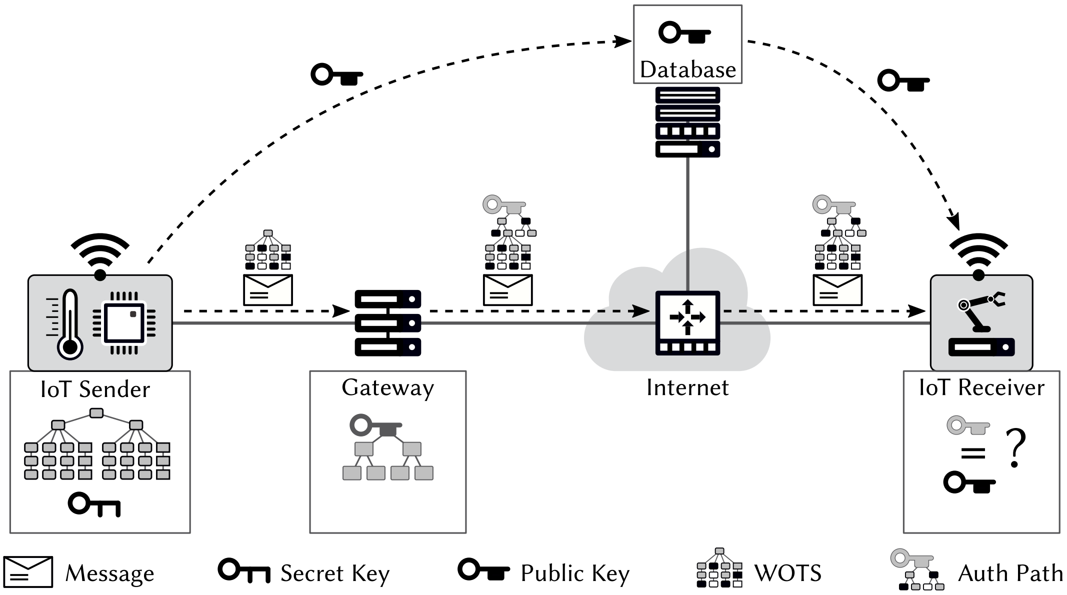

Figure 38: Overview of our signature architecture AMSA. An IoT sender creates several WOTSs from a private key and offloads the authentication path – which connects the WOTS to the public key – to a gateway. Afterwards the IoT sender only needs to create the WOTS to sign a message, reducing the communication overhead for the sender. If a message is sent, the gateway will append the authentication path to complete the signature.

The caching is achieved by storing all right nodes (odd index starting from 0) of each level down to level $h-1$ and all h leftmost nodes during the generation of the key pair.

Whenever we use our key to sign a message, we only need to recompute a single WOTS root (leaf hash on level h). The other leaf hash will be cached from the previous round, such that we always know the first hash of the authentication path. For example, if we sign a left leaf, we will recompute the right leaf hash. If we sign a right leaf, we have cached the left leaf hash from the previous signature.

The remaining hashes of the authentication path can be directly read from the cached tree hashes. Only if a left subtree is exhausted, the cache of left hashes needs to be updated with the root of that subtree.

In total, we cache $2^{h-1}-1$ right node hashes, h left node hashes and 2 WOTS private keys summing up to $n\left(h+2^{h-1}-1\right)+2n\ell$ bytes.

4.3.3 Auxiliary Authentication Gateway

We now describe our idea to reduce the effective signature size by offloading the authentication path of a signature to a gateway.

In a typical IoT scenario, which is shown in Figure 38, we assume a resource-constrained IoT node, which signs messages using an AMSA key pair and sends them to a receiving node via a more powerful gateway.

This gateway could now be utilized to provide the authentication paths of the AMSA signature. This would release an IoT node from the burden of transmitting the full authentication path of the WOTS each time it signs a message.

After generating the AMSA tree, the IoT node sends all leaf hashes to the gateway. Note that the gateway can not use the leaf hashes to create signatures and therefore there is no trusted relationship between node and gateway.

When the node signs a message using a WOTS, it only sends the WOTS and the leaf index to the gateway. The gateway constructs and appends the authentication path to the message before forwarding the message to the receiver. Offloading the authentication path has several advantages:

1. The effective signature size for the IoT node is reduced.
2. The IoT node does not need to construct the authentication path and thus only needs to store the current WOTS and the next seed.
3. In a multi layer signature, all parts but the bottom most WOTS could be offloaded.

Example If the gateway constructs the authentication paths, then the IoT node needs to transmit 2^h leaf hashes during key generation and afterwards only the OTS. The effective signature size for the node over all 2^h signatures is:

$$\frac{n \cdot 2^h + n \cdot 2^h \cdot \frac{8n^h}{\log_2(w)}}{2^h} = n + n \cdot \frac{8n^h}{\log_2(w)} \tag{4.21}$$

Basically, the transmission overhead for the authentication paths per signature is reduced from $h \cdot n$ to only n. For $h = 10, n = 20, w = 16$ this would result in $(840 + 20) = 860\,\mathrm{B}$ instead of $(840 + 200) = 1040\,\mathrm{B}$, which is a signature reduction by 17.3%. With a higher n or lower h, this percentage becomes smaller, which is why we state 17.3% as the possible reduction.

The efficiency for multi layer signatures is almost the same. In case we use two layers with $h_0 = 10, h_1 = 10$ to get the same number of signatures for the first bottom tree, we need to also transmit the WOTS and the auth. path of the top tree which would be additional $1040\,\mathrm{B}$ over 1024 signatures. In summary, this would result in $(840 + 20 + \frac{1040}{1024}) = 861.02\,\mathrm{B}$ per signature.

Overall, utilizing the gateway for providing the authentication path can significantly reduce the computational effort and transmitted data of the signed messages for the IoT nodes connected to the gateway.

Algorithm	**XMSS**	**LMS**	**AMSA**	ECC
$\lvert k_{\text{pub}}\rvert$	68 B	56 B	50 B	64 B
$\lvert k_{\text{sec}}\rvert$ (min)	64 B	48 B	50 B	32 B
$\lvert \sigma_m \rvert$	2500 B	2512 B	* 2434 B	64 B
$\#H_{\text{gen}}$	1166345	1098761	1082377	N/A
$\#H_{\text{sign}}$	579	512	521	N/A
$\#H_{\text{verify}}$	613	546	554	N/A

Table 4.14: Theoretical comparison of stateful HBSs schemes for $n = 32, h = 10, w = 16$ and ECDSA (ECC) for $n = 32$ as reference. While all HBSs have similar key sizes and number of hashed calls, AMSA provides the smallest signature. *: For normal operation. If the auxiliary gateway is used, the signature size for the sender is 2146 B (11.8% reduction).

4.3.4 Evaluation: Signature Size around 2kB for 128 bit Security

In this section we will evaluate our implementation, which is written in C, and compare it to related implementations regarding the following metrics.

- transmitted data (Size of k_{pub}, k_{sec}, σ_m)
- performance (hash calls, hashed data)
- required memory (Binary Size, RAM)
- readability of the code (Lines of Code)

The results are summarized in Table 4.14 and Table 4.15, and will be explained in the following.

Performance

The speed of the scheme clearly depends on the number of hash operations. From analyzing the call graph, we found that 94% of the CPU time of the `AMSA_sign` function is spent in the hash compression function. This means that a huge performance improvement is possible if the hash function is hardware accelerated. The callgraph of the entire test program is shown in Appendix A.1.2.

The hash function is called with four different input lengths: 1) the length of the message, 2) n for generating WOTS chains, 3) $n \cdot \ell$ when calculating the WOTS root, 4) $2n$ when calculating Merkle nodes. Since the execution time is proportional to the input length, we compare the approaches based on the total amount of data x that is hashed. In addition we provide execution times for two different processors: A `Cortex M0` with 48 MHz from the Arduino Zero and an `Intel i7-7600U` with 2.8 GHz. While the numbers for the Cortex M0 without operating system should be deterministic, the timings for the Intel i7 were measured on a Ubuntu operating system and have some variance. Therefore, we run each sign and

Algorithm	**XMSS**	**LMS**	**AMSA**	uECC
$\lvert x_{\text{gen}}\rvert$	331.6 MB	60.8 MB	37.8 MB	N/A
$\lvert x_{\text{sign}}\rvert$	468.4 kB	152.1 kB	55.3 kB	N/A
$\lvert x_{\text{verify}}\rvert$	182.9 kB	27.3 kB	19.1 kB	N/A
t_{sign} on Intel i7	2.2 ms	1.3 ms	0.79 ms	0.61 ms
t_{verify} on Intel i7	0.81 ms	0.31 ms	0.24 ms	0.69 ms
t_{sign} on Cortex M0	3004 ms	1373 ms	431 ms	841 ms
t_{verify} on Cortex M0	808 ms	143 ms	152 ms	438 ms
|Binary|	151.1 kB	107.3 kB	34.5 kB	36.6 kB
LOC	1.9 k	3.9 k	1.1 k	33.6 k

Table 4.15: Experimental comparison of stateful HBSs schemes and ECC as a reference. We state computational effort as the total amount of input $|x|$ (in bytes) to the hash function H and as specific timings t. The timing values are averaged over 1024 calls. We compiled each HBS for `SHA256_W16_H10` and uECC [Mac19] for `secp256r1` using `-O3`. The Lines of Code (LOC) were counted using the tool `cloc` and skipped implementations of hash functions.

verify method 1024 times and calculate the average. For the sign operation of our AMSA implementation, we have measured a minimum of 0.64 ms, an average of 0.79 ms, and a maximum of 0.97 ms. For the verify operation, we have measured a minimum of 0.21 ms, an average of 0.24 ms, and a maximum of 0.38 ms. This indicates that our measurements on the Intel i7 are consistent within an average error margin[3] of 28% around the average execution times.

Security

The security of a signature is based on the difficulty for an attacker to forge a valid signature/message pair. For HBSs, this difficulty relies on the security of the underlying hash function, which is discussed by three resistances:

1. First-preimage: difficulty to find a preimage x of one known image $h = H(x)$.
2. Second-preimage: difficulty to find a second preimage y with $H(y) = h$ of one known preimage-image pair x, h with $h = H(x)$.
3. Collision: difficulty to find any two values a, b that result in the same image $H(a) = H(b)$.

Conventional Preimage Attacks Hash-based Signatures can be forged if an attacker can find a preimage for one of the revealed hash values. However, not all hash values are equally important.

[3] Averaged over $\{18.9, 22.7, 12.5, 58.3\}$, where $1 - \frac{0.64}{0.79} = 18.9\%$ is the maximum lower error for signing.

For example, if an attacker can find the preimage of a hash in the WOTS chain, he/she can change the signature by only a single bit. If the checksum chain was attacked, the attacker can change ℓ_1 bits. By contrast, another preimage for the message hash or the WOTS root hash would allow to sign a completely different message. Attacking the Merkle tree is most profitable, because in the case of success, an attacker could forge up to 2^{h-1} arbitrary messages. To do so, the attacker would construct a new $h-1$ AMSA tree with a public key k'_{pub} and then tries to find any n-byte value x such that $k_{\text{pub}} = H(k'_{\text{pub}}||x)$ completing the tree to its full height.

However, already for a 128 bit hash, it is very difficult to find another preimage. Even if we assume the *entire* Bitcoin network with a current hash rate of $\approx 80 \times 10^{18}$ hashes/s focused on one 128 bit hash preimage attack, it would still take an expected time of $2^{127}/80 \times 10^{18} = 67$ billion years.

Preimage Attacks on Quantum Computers The overall performance of a quantum computer relies on several factors including number of qubits, coherence time, and error rates [Mav+18] and has been continuously growing over the last years.

While the security of ECDSA and RSA would be completely broken by Shor's algorithm, the security of hash functions is only reduced to half by Grover's algorithm.

However, results from [Amy+16] suggest that a real QC preimage attack of SHA-3-256 would require $\approx 2^{166}$ operations instead of the theoretical optimum of 2^{128}. In general, current research suggests that hash functions provide at least the same security against QC attacks compared to classical attacks when their number of digest bits is doubled. Therefore, choosing $n = 32$ (256 bit) for HBSs provides at least 128 bit security against quantum computers. Since early quantum computers will probably be expensive to construct and to operate, which discourages long-term bruteforce attacks, this security level should be sufficient for the foreseeable future.

Summary of our Adaptive Merkle Signature Architecture

Quantum-secure schemes increase the size of signatures and keys compared to classical schemes such as ECDSA based on ECC. While stateless Hash-Based Signature (HBS) have especially large signatures, stateful HBS are more efficient and flexible alternatives with signature sizes around 2.5 kB for 128bit security.

However, the existing schemes XMSS and LMS, which were standardized by NIST, allow only a very narrow set of parameters, which diminishes the flexibility of HBSs.

Our AMSA leverages the large variety of parameters to enable adaption of the scheme to the available resources of devices. This adaption is crucial to overcome the security challenges for constrained devices in an efficient manner.

Furthermore, AMSA provides two improvements to reduce the signature size without lowering security. First, we propose a more efficient encoding for the WOTS by using different chain lengths. Second, we utilize an auxiliary gateway to append the authentication path for the Merkle-tree, which is non-critical information that does not require confidentiality.

When identical security parameters are compared to state-of-the-art HBSs, AMSA provides 2.6% smaller signatures in general and 17.3% smaller signatures for the sender if an auxiliary gateway is used.

However, some desired properties are still missing. Similar to XMSS and LMS, our scheme can only sign a limited amount of messages, which needs to be specified when generating the key pair. Furthermore, the general signature size reduction of 2.6% is still not sufficient to get into the range of conventional signature sizes with tens of bytes.

Chapter 5

Final Discussion

Contents

At this point, we have presented several system-level designs, protocols, and algorithms to approach the research challenges that we have outlined in Section 1.3. In this chapter we will summarize the key findings of this dissertation, discuss advantages and disadvantages of a decentralized IoT, and provide suggestions for future research directions.

The big picture. Currently, most automated and intelligent systems are individual devices with a specific task. Even the most astonishing projects with artificial intelligence focus on solving specific problems and then presenting a solution to humans. In this sense, a chess computer plays chess, an assembly line produces a product, and a neural network to detect faces "only" searches a given database for the best match.

The *Internet of Things* is changing this isolated automation paradigm by distributing billions of smart and connected devices and enabling them to interfere with the state of our digital and physical environment. Spanning a global network, the state changes of these devices will eventually have global consequences.

The question we are facing is whether these global networks should be governed and operated through centralized coordinators or whether the control should be distributed among all participants forming a Decentralized Autonomous Organization (DAO). The majority of existing and planned IoT applications is clearly headed towards centralized cloud services that are operated by a few big companies such as Amazon and Microsoft. In this dissertation we have looked into the alternative approach of decentralizing and democratizing automated processes by providing secure mechanisms to synchronize and certify state changes, which is one of the major challenges when there is no central coordinator.

5.1 Key Findings of this Dissertation

From the results of the individual approaches, we will formulate some distilled *key findings*, that should have more far-reaching implications on the design of decentralized IoT than the individual results.

5.1.1 The Internet of Things will grow.

In Chapter 1, we have presented several statistics from reputable sources that highlight the visions, expectations, and estimated growth rates of IoT projects. The number of devices that communicate only among each other has already surpassed the connections of smartphones and computers operated by humans. None of the analyzed sources indicates that the process of digitalization, automation, and increasing connectivity will slow down in the foreseeable future.

While approximately 78% of IoT projects are planned with centralized cloud architectures [IT15], the idea of DAOs is also accelerating. For example, the number of cryptocurrencies increased from around 500 in 2014 to 10 000 in 2022 [Bes22]. Many of these cryptocurrencies incorporate smart contracts to allow fully decentralized automation of trading processes.

Decentralized automation requires a simple common language: When devices should interact with each other, they need to speak a common language. This is achieved by writing complex program code that specifies rules, protocols, and data structures for machines. However, the complexity of current programming languages has lead to many errors, unintended behavior, and in general incompatible interfaces between systems. Since we want to create a global network of billions heterogeneous devices that interact with each other and with humans, we need a common language to ensure interoperability between many ontological domains. At the same time this language needs to be simple enough to provide confidence in the expected outcome before processes, which could be irreversible, are executed. In Section 3.4, we have identified concepts that would be required in order to specify complex automation tasks via smart contracts in a language that is deterministic for machines and allows easier human reasoning than conventional programming languages.

5.1.2 Consensus in CPS faces heterogeneous capabilities and safety constraints.

Distributed databases need consensus to achieve consistency between redundant data storages and therefore a majority is still good enough to retrieve the correct dataset. By contrast, in many CPS we have unique and heterogeneous agents that need consensus to orchestrate a sequence of actions.

In Chapter 2, we have discussed consensus protocols in the context of Intelligent Transportation Systems consisting of Connected Autonomous Vehicles. We found several aspects that require a different mindset for transferring consensus protocols to this domain and some should apply to CPS in general.

Unanimous Agreement for Safety: While most consensus protocols focus on majority voting, CPS can often not tolerate any faults to ensure safety. This results from the fact that a CPS consist of heterogeneous agents with special roles, capabilities, and physical locations. We cannot treat every agent as another redundant data storage.

In the example of CAVs, we cannot rely on majority agreement because non-synchronized vehicles could block execution or take actions that lead to damage or injuries. In the case of intersection scheduling, we concluded that vehicles should not be allowed to enter the intersection until *all* approaching vehicles next to the intersection have agreed on the same crossing schedule to avoid collisions.

Local Validation Capabilities: Many IoT projects require physical sensor data as input for their application logic before they can take any actions. However, sensors that measure physically quantities are always deployed locally and so they will measure data with local validity. Cross-validation by other sensors could increase the trustworthiness of the data but currently there seems to be no unified framework on how to achieve that. It seems fair to say that fully decentralized and independent validation of sensor data by a majority of participants is infeasible.

By contrast, blockchains are considered secure because a majority of nodes validates every transaction in the underlying consensus protocol. This is only possible because the balance of crypto-assets is not bound to physical measurements but globally verifiable by any node with a copy of the blockchain.

We therefore conclude that the high security of decentralized verification cannot be simply transferred to physical measurements because only a very limited amount of devices can validate those measurements. Although blockchain and smart contracts are still valuable tools to secure data, they currently cannot provide the same security for physically-correlated data as they do for pure digital data.

5.1.3 Blockchain participation is limited but certification is promising.

We have seen that a major part of the success of blockchains stems from the PoW consensus mechanism. PoW is the first consensus protocol that allows open participation by any node without a registration process, which is one important aspect to enable true DAOs. However, PoW consumes huge amounts of energy to mine blocks, which makes it also infeasible for resource-constrained devices to participate in the block creation.

PoW allows cheap verification. Nevertheless, one interesting characteristic of PoW consensus is its easy verifiability with a single hash operation per block. Once a block is mined, every device can verify its integrity and basic validity by running a single hash operation over a few bytes of block header. This cheap verification is highly suitable for resource-constrained devices and around two magnitudes less computational expensive than conventional authentication methods, such as cryptographic signatures. Furthermore, the verification cost remains constant and is independent from the number of participants, which means it has perfect scalability even for very large global networks.
Since blocks are only created in regular intervals, which vary between tens of seconds and several minutes, this type of verification is not suitable for application data that must be available within short time bounds. As a result, processes within CPS that have real-time constraints should not use blockchain for validation and certification of data.
Overall, blockchain certification with PoW is very promising for data that either remains valid for long period of time or can tolerate the delay of block creation. For the first case, we have presented *LeapChain* in Section 3.2 to retrieve and verify any type of long-term data from a blockchain efficiently. For the second case, we have presented a novel method in Section 3.3 to verify the timestamps from block headers in order to synchronize nodes to a common reference time with one second of accuracy. Another application would be to store the public keys of vehicles that can be retrieved by looking up the license plate number, as we have suggested in Section 2.3.

Blockchains are Quantum-Resistant Since the data structure of Blockchains is created by using only hash functions, the transactions that are stored in the Blockchain remain verifiable even when quantum computers will break ECDSA.
By contrast, digital certificates for websites, firmwares, etc. that are issued today can no longer be *exchanged* securely once quantum computers can forge them because they could origin from an adversary. This is also true for Blockchain transactions that rely on conventional PKC such as ECDSA. Cryptocurrencies would need to stop accepting new transactions that are signed with ECDSA because otherwise an adversary with a quantum computer could spend assets from any address.

However, all transactions, which are stored in the Blockchain until quantum supremacy is achieved, remain verifiable because their integrity is guaranteed in the chain.
This offers another interesting perspective for hash-based Blockchain structures to certify data with conventional PKC that must remain verifiable *after* conventional PKC becomes insecure. For example, firmware signatures could remain verifiable for a long period of time when the hash of the firmware is stored in a public blockchain which would be interesting for devices with long life-cycles.

5.1.4 Hash-functions are efficient and versatile primitives for the IoT.

During our exploration for efficient and robust solutions we have also seen that cryptographic hash-functions can be utilized in many different ways and offer more functionality than just serving as a digital fingerprint.
Hash-functions are well-understood, use fast integer math and can be accelerated in hardware with little area overhead [FW07], making them well-suited to run on even highly resource-constrained devices, such as RFID tags. Hash functions are also quantum computer resistant, which means that this type of primitive is expected to remain secure for a long period of time in the foreseeable future.

Hash functions guarantee integrity. By using a *Merkle*-tree, a large amount of data records, such as financial transactions, can be combined into a single root hash that represents a digital and verifiable fingerprint for all data records. Verifying the integrity of any one out of N data records is achieved efficiently by using only $\log N$ additional hash values.

Hash functions can authenticate messages. Conventional symmetric MACs are often based on hash functions but they require a shared secret for each communication link. For this reason, decentralized networks, which have many links, often rely on asymmetric PKC. While quantum computers will probably break current signature schemes in the next decades, there are already several quantum-resistant schemes that are ready for standardization. Especially hash-based signatures would be highly flexible and efficient.
In Section 4.3, we have demonstrated how we can construct a signature scheme by using hash functions only. Since the main drawback is the large signature size, we have proposed a solution to reduce the signature size without sacrificing security.
Overall, it seems that a single hash function could be used to provide all cryptographic tools required to implement a distributed ledger, such as a cryptocurrency. In combination with hardware acceleration, this would enable even highly resource-constrained devices to participate in the system securely. In case confidentiality is also required, a symmetric cipher could be used as the basic primitive instead, since a symmetric block cipher can also efficiently instantiate a hash function [LM92].

5.1.5 Centralized vs. Decentralized IoT

Finally, we try to answer the question whether decentralized architectures are superior to centralized architectures in general or if they are only beneficial in certain domains.

We have highlighted several advantages of decentralized architectures in this dissertation and proposed solutions how to implement them. However, centralized architectures also have their benefits. They are easier to implement, well-understood, and allow a direct control over the entire system.

Furthermore, for tasks that involve many computations but provide a single result, such as outputs of a large neural network or searches in databases, the cloud does offer a huge and dense amount of resources. In this case, the cloud can perform the computation quickly and since only a single result is provided as response, the longer communication distance does not decrease performance.

By contrast, offloading such tasks to a highly dynamic distributed system, where individual nodes can have large response times or disconnect completely, would delay the result and might overall be less efficient due to the increased communication overhead. We can conclude that offloading computations to the cloud makes sense when a few inputs require many computational resources to produce a single small result.

However, tasks that involve the coordination of many devices are well suited for decentralized architectures because centralized cloud architectures would experience the disadvantages that we have outlined in Section 1.2.1.

Especially, CPSs, which consist of individual devices that are already physically distributed, are a perfect match for decentralized coordination. In Section 2.4, we have demonstrated how vehicles can agree on intersection scheduling using distributed consensus. One of the major advantages we saw is that connecting vehicles directly over VANET does not require any additional infrastructure, which is very cost efficient. The difficult cross-validation of physical measurements that we have discussed in Section 5.1.2, is a general problem that is present in centralized *and* decentralized architectures. However, decentralized architectures might still have a higher chance of solving the problem efficiently because the extra information required would be most likely available in close distance. For example, in Section 2.3, we have proposed the combination of radio transmission and sensor data to verify the presence of vehicles. Our results for ITSs can probably be transferred to other CPSs that consist of individual agents because these agents can establish short and direct communication links to their neighbors and the local information they process and exchange is mostly relevant within a local area and less relevant for a central and global database.

Overall, we think that the IoT domains Intelligent Transportation System, smart grids, smart cities, and digital marketplaces will highly benefit from a globally decentralized architecture because they either consist of physically distributed devices or the data and services they provide is relevant for a large network of users.

The domains industry 4.0 and smart home automation consist of rather isolated networks. For example, the smart devices in one home, do not necessarily need to interact with the smart devices of another house or apartment. For these domains, a tree-like structure could make sense, where most data is processed in individual small local networks but some data is transferred to gateways and cloud servers for remote control and monitoring.

For the domain of smart health, it is more difficult to decide which approach would be suitable. One the one hand, we have sensitive medical data that we want to keep locally protected and the data is mostly relevant for the single person to which it belongs. On the other hand, the data is only valuable when compared to complex models to draw any conclusion and make predictions about potential health concerns. These models could be large neural networks that will process the data of billions of people in the cloud to improve its accuracy and incorporate the latest results from scientific discoveries. Since we are not able to draw a conclusion here, we hope that an increasing number of use-cases will clarify the situation in the future and reveal to which extend the IoT paradigm makes sense in this area.

5.2 Further Directions

In this section, we will discuss which related research questions are still unanswered and provide some ideas on how to approach them. Furthermore, we critically discuss the possible directions in which the idea of an automated Internet of Things should or should not proceed to evolve.

5.2.1 Open Research Questions

While we presented and evaluated several approaches, algorithms, and implementations that help to solve three of the core challenges of decentralization, there are further aspects to consider when looking at decentralization from an even wider perspective.

As we have also mentioned in Section 1.3.3 from the introduction, some IoT challenges were assumed solved to focus on the decentralization of communication architectures. While there exist suitable approaches for most of these challenges (e.g. 5G cellular networks will ensure connectivity), the exact details of how they work in decentralized networks might not be sufficiently evaluated for all of them.

We have yet to see how we can put together all pieces of this huge IoT puzzle that will enable full automation using smart devices on a global scale. In the following, we will outline some topics that are actively researched and deserve further attention when transitioning to a decentralized architecture.

Decentralized Software Updates. Software is not static but changes and evolves over time. Especially for devices with long life-cycles and several years (or even decades) of expected operation time, we need an efficient mechanism to keep their software up to date. Not only to bring new features to the system but also to ensure compatibility with other changing components and to close security vulnerabilities that were discovered after the first release. Again, we face the decision whether a central authority should release and enforce these updates or whether they can and should be approved as part of some democratic consensus. Actually performing software updates in a decentralized system is further challenging because the new software has to be accepted by each participant and thus there needs to be backwards compatibility and an incentive to perform the update. Otherwise, no one would go ahead and be among the first users to install the update.

Ensuring Privacy and Verifiability. The idea that every data record and every action within a network is publicly verifiable is partly responsible for the success of decentralized cryptocurrencies, such as Bitcoin, and provides security guarantees in general. However, for some use-cases, such as smart healthcare, which process sensitive user data, we need a mechanism to protect the data and restrict access to authorized parties only. For example, a physician could be authorized by a patient to access sensitive medical data that is generated by wearable devices.

One promising idea in this direction are non-interactive *Zero-Knowledge Proofs*, which use computation on encrypted data to verify certain properties without revealing the data itself. Within our research group, we have also looked into the possibility of providing anonymous authorization by proving accumulator membership in zero-knowledge [Lau+21]. This allows to authorize (or revoke) parties by including them in a publicly known accumulator, which is a large number of constant size. The accumulator has the property that a zero-knowledge proof can be generated to certify witness membership in the accumulator, where the proof does not reveal the identity of the witness owner.

Open global consensus without Proof-of-Work. While PoW has enabled open DLTs, where everyone can participate, it is computational expensive and consumes a lot of energy. According to the *Cambridge Bitcoin Electricity Consumption Index* at cbeci.org, Bitcoin alone consumes over 130 TW h per year (as of 2022), which is at least 0.5% of the global electricity consumption. Since electricity is not completely generated from renewable energy sources, Bitcoin has a direct impact on global CO_2 emissions. As a remedy, we should look for alternative consensus mechanisms that allow open participation and can be easily verified, such as Proof-of-Stake [Sal20]. In case, PoW turns out to be the only mechanism that offers these properties, we should either not use it or try to reduce the incentive to invest more hardware into mining. For example, Bitcoin and other PoW-chains could be used less for financial transactions and more for certifying important data and files, such as timestamps, software, authorization tokens, etc.

Single-Hash Authentication. While we have demonstrated how Hash-Based Signatures (HBSs) can be used for message authentication and how the size of the signature can be reduced, the overall size of the signature is still quite large. Other quantum-resistant signature schemes have the same problem that either the signature or the keys are very large and require a few kilobytes to be stored or sent, which is several magnitudes larger than conventional signatures using ECDSA.

In order to keep the message overhead similarly low to current ECDSA, we need to further improve post-quantum signature schemes or search for alternative approaches in this domain.

One less-known approach, which is already two decades old, is using protocols such as TESLA and μ-TESLA [Per+00; LN04]. These protocols can be used to asymmetrically authenticate messages via a single hash-based commitment, which is quite fascinating. The scheme works by revealing the next preimage within a hash chain after some "secure" delay time and the revealed preimage functions as the missing piece to complete the verification of a conventional MAC. Accurate time synchronization and strict enforcement of the delay time is crucial to guarantee security, which is why the scheme is not considered suitable for many applications.

However, many consensus protocols proceed in rounds, exchanging data that is often not real-time critical. When the data is stored in blocks that include timestamps, a mechanism for time synchronization is already present. As such, hash-based commitments might be suitable candidates for blockchain applications and could instantiate one of the most efficient broadcast authentication schemes in terms of computation time and message overhead.

Revoking unintended actions. Smart contracts guarantee that transactions will be executed according to specified rules. However, there is currently no mechanism to "undo" unintended or wrong executions that appear to be bugs in the code. While the term *unintended* might be difficult to define, decentralized blockchains cannot undo transactions, even when a majority would agree to do so. This problem has manifested in the DAO-hack, which we have presented in Section 3.4, where the community decided that it was necessary to hard-fork the entire blockchain in order to revert the changes of the hack. This is clearly not a suitable solution for operating a global cryptocurrency and better mechanisms have to be found.

5.2.2 Author's Opinion on the Future of Automated IoT

While the big picture is quite clear on how an automated IoT should look like, there seems to be little discussion about the potential impact on society and humankind from researchers proposing these technical solutions. Of course, a thorough discussion on this topic would deserve its own dissertation and is outside my research focus. However, I would like to discuss some risks and benefits for society as my own opinion in this section.

The Dark Side

A fully automated IoT will make decisions with global consequences in a fraction of the time it will take us humans to even start a discussion on action plans. Furthermore, it is unclear how much intervention capabilities humans will still possess in a fully decentralized and automated future.

For example, if an adversary can convince 51% of a Blockchain network that he/she possesses a billion tokens (e.g. Bitcoin), that would be the new global truth in a few minutes [Pse14; CV17]. Devices would accept his/her payments and many smart-contracts would accept his/her instructions. The impact such an adversary could have on the world from this moment on would be mostly limited by the coverage of monetized automation and the pace at which he/she is able to specify instructions.

The time and effort it takes for other humans to detect this error/attack, communicate the problem to all relevant stakeholders and decide on a plan to revert the changes *on a global scale* will be immense in comparison [Fin16; Dai16].

The speed of agreeing and spreading the agreement is where computers will outperform humans by magnitudes [Sum+21] and – in combination with the digitalization and automation of the world – will enable computers to control what we consider the current information state, therefore what we consider to be true, and finally on which basis we will make decisions; if we still make decisions at all.

In the fiction "Qualityland 2.0" [Kli20], the author describes the outbreak of World War 3 in a highly automated society. The war is started by unknown "triggers", fought by autonomous and connected weapon systems and is over in a few hours. After victory is (autonomously) declared by one country, the humans spend days trying to retrace why the war was started, how many people died, and what actually happened.

While this scenario seems quite extreme, we should not estimate its possibility as zero. If we project the current pace of technological development and automation into the near future and also take into account the various vulnerabilities that have already been found in IoT systems, it would be rather naive to ignore these potential risks.

In the worst case, the automated future might leave most humans as overwhelmed spectators that can only wonder about the rapid changes that happen in the world. Fully parallized, we stand in front of our terminals, unable to decide on any instruction because the network – billions of intelligent autonomous agents – change our situation and understanding of it faster than we are able to converge to any agreement. This network could balance us at the tipping point of any discussion about whether the changes are good or bad, trapping us in continuous undecidability until we either surrender our autonomy silently or fall back to our ancient conflict resolution strategy of fighting each other violently.

The Bright Side

Humans have a successful history of utilizing technologies that probably appeared dangerous in the beginning. Whether it was the discovery of fire that can burn down entire cities if used carelessly, or the development of nuclear power plants, where the process of nuclear fission could lead to a reactor meltdown if not balanced carefully, we have managed to control them. Both fire and nuclear fission are very safe forms of energy today [Rit20] and have enabled many innovations and discoveries that improved our standard of living. In this sense, we should maybe treat IoT automation with the same care as fire or nuclear energy.

If we carefully design global IoT systems, with safety and security as top priorities and also monitor how it impacts society, the benefits can be huge.

ITS will not only reduce travel time and emissions for human passengers [CE16] but also for all products and goods that are autonomously delivered to their customers. For this use-case, decentralized approaches are especially promising because no additional infrastructure that requires installation and maintenance is needed (see Section 2.4). Vehicles communicate directly with each other on demand making full use of short, low-latency connections.

As we have also mentioned in Chapter 1, smart devices can help us to use resources, such as water and energy, much more efficient and avoid excessive waste. The distribution of electricity and water can be regulated by smart grids that monitor supply and demand of resources. For example, these smart grids might perform the complex task of handling fluctuations of generated energy from solar panels and wind turbines and fluctuations of consumed energy when people start cooking or switch on lights in the evening [Ene21; CD16].

Citizens of a country could possess digital and self-sovereign identities that are stored locally on their smartphones and shared only on demand to access global services. Therefore, each person has control over his/her personal data and can decide which institutions are authorized to access personal information. By using zero-knowledge proofs, certain attributes of a person can even be proven without revealing the full identity [Lau+21]. This could enable anonymous and cryptographically secure voting in democratic elections and reduce the possibilities of corruption and frauds.

In fact, many bureaucratic tasks could be automated and verified by using smart contracts.

As soon as data is initially validated and available in a common format, it can be further processed by a decentralized network, which enforces the rules specified in a smart contract [RS18b]. This could be used to automatically open doors to rental apartments once the payment is received [CD16].

We will also have a decentralized and global payment system with instant transactions that is used by machines and humans alike. With no (or almost no) transaction fees, micropayments of tiny amounts become possible, which incentivizes more people to provide open Application Programming Interfaces (APIs) and services [CD16]. For example, property owners could maintain weather sensors in their garden and receive a tiny payment every time a service accesses the data. This could increase the accuracy of weather predictions and make weather stations operated by publicly funded institutions obsolete.

Overall, the increased automation enabled by the IoT might allow us to spend less time on unfulfilling mechanical or bureaucratic tasks. Instead, we might have more free time or additional capacity for solving other important problems, such as social inequality or global warming.

While it is difficult to predict how exactly the future will unfold, we are optimistic that the IoT will be one of the driving forces behind further improvements in the quality of life for everyone.

5.3 Concluding Remarks

The Internet of Things is already being implemented and the increasing automation will impact the daily lives of every one of us. We need to decide now, whether we want to continue with the current trend towards centralized cloud platforms operated and owned by large tech companies, or whether we want to have open and decentralized networks.

For several IoT use-cases, such as ITS, smart grids, and digital marketplaces, a decentralized architecture offers huge benefits in terms of efficiency, scalability, low latency, and robustness compared to centralized solutions. Centralized cloud servers with their huge storage capacity and computational power should instead be used to offload tasks that cannot be performed efficiently on edge devices, such as database searches or computations of large neural networks.

In the past, we have seen how projects, such as Bitcoin, Wikipedia, and the Internet itself, have truly disrupted entire businesses and became the gold standard in their domain because they have presented the first decentralized solution that works on a global scale. It is therefore likely that other domains are just waiting to be decentralized to reveal their full potential.

We have presented several novel approaches to solve the three major challenges that arise when using a decentralized architecture: *synchronization*, *certification*, and *authentication*. In addition, we have demonstrated how these approaches can be applied to Cyber-Physical Systems and confirmed that the required computation and communication can be performed efficiently even by highly constrained devices. With our contributions, we are therefore further closing the gaps between the individual puzzle pieces that will fit together to construct a fully autonomous and global Internet of Things orchestrating smart Cyber-Physical Systems and connecting humans in unprecedented ways.

Appendix A

Appendix

Contents

The Appendix chapter contains additional details that were not necessary to follow the narrative of the thesis but are included here to provide further clarifications and directions for the interested reader.

A.1 Additional Information

This section provides further details on topics that very only mentioned briefly in the main part of this dissertation.

A.1.1 Variants of the Consensus Problem

The consensus problem is just one way to look at the general problem of agreement. Over the last four decades, many variants have been analyzed and three common types should be shortly mentioned and compared:

- **Byzantine Agreement (BA) [LSP82]:** One "commander" agent a_p proposes a value v_p, the other agents agree to accept or reject v_p.

- **Distributed Consensus (C) [FLP82]:** Each agent a_i has its own initial value v_i; all agents agree on a single value v_c. Consensus is further classified according to the type of the value v_i:
  - Binary Consensus (bC): $v \in 0, 1$

  - ▷ Multi-Valued Consensus [TC84]: $v \in V$ (any set)
  - ▷ k-vector Consensus: $\vec{v} \in V^k, k \leq n$
  - ▷ Vector Consensus: $\vec{v} \in V^{n-f}$ (similar to IC)
- **Interactive Consistency (IC) [PSL80]:** Each agent a_i has its own initial value v_i; all agents must synchronize this vector of initial values $\vec{v} = [v_1, .., v_n] \in V$

All these problems are similar in the sense that if one is solved, the others are also easily solved in the synchronous case. For example,

- We can derive IC from BA by running BA n times, once for each process with that process acting as commander.
- We solve C by IC without overhead. IC produces a vector of all v_i and each node can apply a function on it to determine a single value v.
- We solve BA by IC without overhead. IC produces a vector of all v_i, which is already BA for each node.

However, in an asynchronous system, IC cannot be reduced to BA because BA cannot guarantee termination in non-synchronous systems (no termination) [Bra87].

A.1.2 Callgraph of AMSA

Figure 39 shows the callgraph of the AMSA code which was created with the tool `Valgrind`.

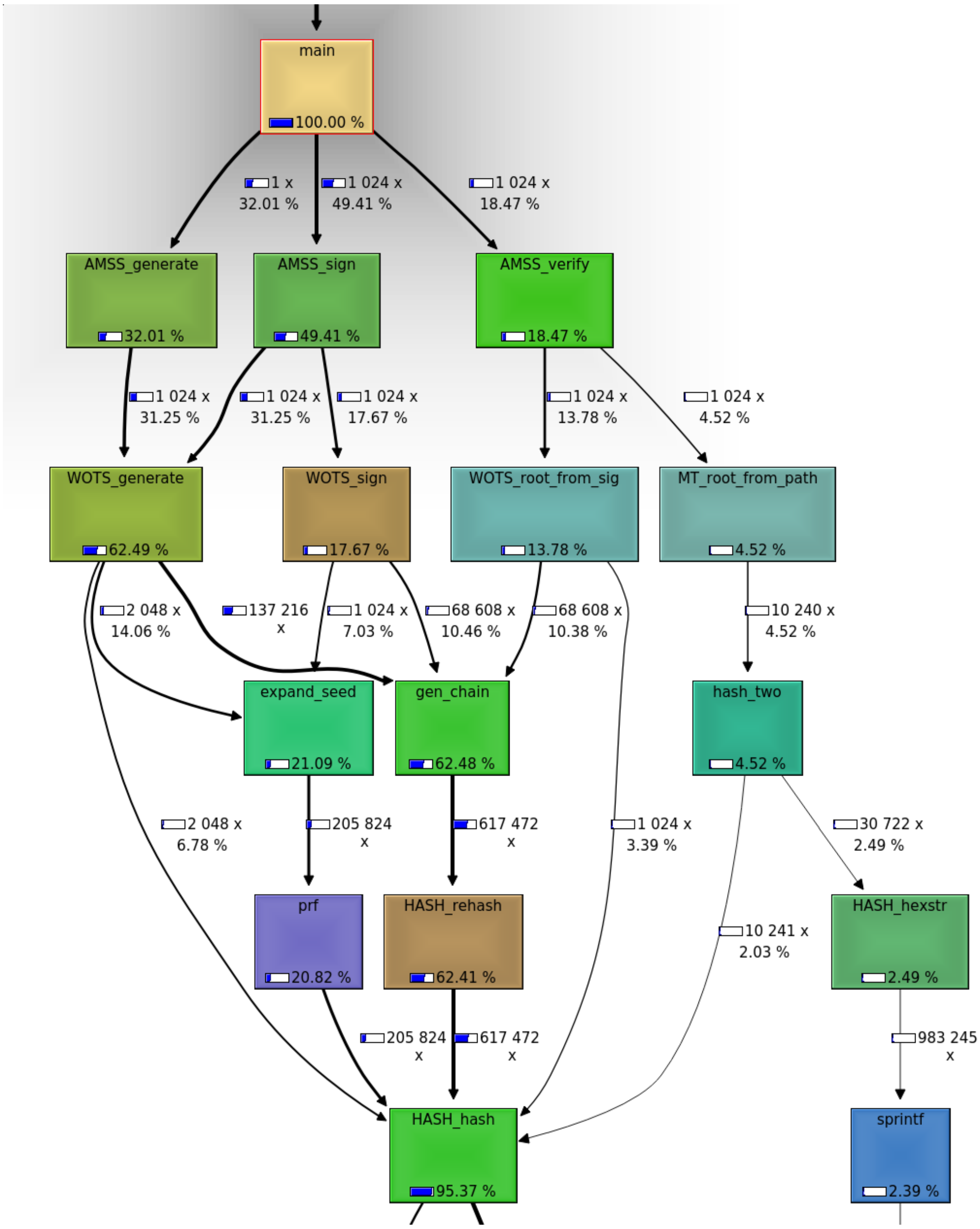

Figure 39: Callgraph of the AMSA code. The key pair is generated once in AMSS_generate(). Afterwards, we have 1024 calls of AMSS_sign and AMSS_verify. Overall, the program spends 95.37% of the time in HASH_hash(), which is a wrapper for the hash function such as SHA-256.

A.2 Details on Related Work

This section provides further details on related work that might be interesting.

A.2.1 Related Consensus Protocols and Variants

Bracha's Reliable Broadcast

Another important building block that is often found in consensus protocols is Bracha's Reliable Broadcast protocol [Bra87]. It ensures that if an honest agent sends a value v it will be either accepted eventually by all honest agents or none.

1. Initial: Send $Init(v)$
2. Echo: Send $Echo(v)$ if
   - receive one $Init(v)$ or
   - receive $(N + f)/2$ $Echo(v)$ messages or
   - receive $f + 1$ $Ready(v)$ messages.
3. Ready: Send $Ready(v)$ if
   - receive $(N + f)/2$ $Echo(v)$ messages or
   - receive $f + 1$ $Ready(v)$ messages.
4. Accept v if received $2f + 1$ $Ready(v)$ messages.

HoneyBadger Protocol

HoneyBadger [Mil+16] is a consensus protocol that is build from a set of several consensus primitives.

- Erasure Coding: Split data into n equal blocks and only require a subset of k blocks to restore it.
- Merkle Trees: Quickly validate the integrity of the original data from n blocks
- Reliable Broadcast: sender splits message m into n blocks and calculates Merkle root from them. Send each other node one block with branch hashes to Merkle root. Each node forwards its received block via ECHO message. Each node that receives k blocks, restores the original message m, valdiates its hash and sends a READY message. Each node that receives $2f + 1$ READY messages, will wait for k ECHO messages and decodes m.
- Cryptographic Common Coin: Threshold signatures are used to create a random common coin if $f + 1$ nodes contribute their share of the signature. An adversary cannot know or influence the outcome of the random coin.

BEAT Protocol

BEAT [DRZ18] is a family of asynchronous, BFT protocols for SMR. It improves over HoneyBadger by using a more efficient reliable broadcast with a message complexity of $O(|m|)$ instead of $O(N|m|)$. The idea is that hash branches do not need to be sent.

A.2.2 Related Approaches on Intersection Management

This section provides details on the working mechanisms of related intersection management approaches.

Centralized Exclusive Reservation

AIM08 [DS08] from 2008, divides the intersection into a grid of $n \times n$ tiles. Vehicles request a slot in space-time for crossing from a central intersection manager, which will then assign tiles to vehicles. Each vehicle sends a request including time-of-arrival (TOA), velocity, and size to the IM. The IM calculates a motion profile for the vehicle according to a policy, e.g. "First come, first serve" (FCFS), and sends it back to the vehicle. If no collision-free path can be found, the reservation request is rejected and the vehicle has to slow down, possibly resting until a trajectory becomes available.
Prio14 [Qia+14] provides dedicated lanes for left-turn, straight, and right-turn and maximizes vehicle speed. Instead of a custom simulator, Prio14 was implemented in SUMO. Each vehicle controls its speed and only decelerates if there is a risk of collision. An intersection controller collects all vehicle requests and assigns crossing slots which maximizes vehicle speed.
Delay17 [Zhe+17] provides a delay-tolerant centralized protocol, which considers communication delays and packet losses. Since it was also implemented in SUMO and focuses on real-world conditions, it is one of the few related works for which we consider results comparable to our work. The result stated in the table is for normal message delay (OMNeT). One of the main differences is their central IM, with all the previously mentioned drawbacks.
CSIP19 [AR19] tries to maximize throughput with a minimum gap between vehicles that is still comfortable for human passengers. Vehicles that approach the intersection are equally spaced and must keep a constant velocity to ensure that no vehicle needs to slow down while crossing the intersection.

Decentralized Exclusive Reservations

In **NoStopSign08** [VDS08], each vehicle generates and continually broadcasts a CLAIM (includes direction, arrival time, and exit time), which "reserves" the next free slot that does not conflict with already received claims. If conflicts occur, vehicles can send CANCEL messages and/or updated CLAIM messages. Simulation reaches near-zero delay for spawn rates of < 0.2 v/s. However, due to the lack of agreement confirmations from other vehicles, the authors reported collisions when the packet loss exceeds 40%.

In **MP-IP12** [Azi+12], the intersection has two lanes per direction and is divided into 4×4 tiles. Each vehicle continuously broadcast its state (ENTER, CROSS, EXIT) and the tiles it will occupy updated over time. Vehicle receiving ENTER/CROSS messages drive as far as possible into the intersection and only stop in front of conflicting tiles. Once crossed, vehicles broadcast EXIT to free the tiles. In the published videos of the updated algorithm [AR19], the vehicles are spawned in a *periodic* pattern with a distance that allows vehicles to cross alternatingly with minimal (hardly visible) speed variations. It remains unclear whether the protocol would work with random spawn patterns.

In **MutEx14** [Wu+14], vehicles broadcast their estimated arrival time at the intersection via *REQUEST* messages. Vehicles with shorter arrival times will respond with a *REJECT* message, blocking the vehicle from crossing until all vehicles with shorter arrival times have crossed. If no *REJECT* is received (timeout) or only *PERMIT* messages from vehicles with higher arrival times are received, the vehicle will cross the intersection. Evaluation is purely based on the network simulator NS-3 in combination with static time counting for vehicle movements (e.g. 4 s to turn left).

Virtual Traffic Light (**VTL15**), is a 2015 patented idea first described in [Fer+10] and improved in [Shi+15]. The first vehicle arriving at an intersection becomes a temporary IM and assigns itself a virtual red light. It stops and assigns virtual green lights to other vehicles. This simple and elegant idea provides reasonable delays but it is not robust as it does not consider selfish incentives, e.g. vehicles slowing down to *not* become the leader with the red light.

CICAP17 [EMB17] uses a real-time database implemented in SQLite3 for each vehicle, which will send and receive location and speed from neighboring vehicles. Each vehicle decides to accelerate or decelerate based on FCFS priority. If two vehicles arrive almost at the same time, the vehicle identifier is used to resolve the conflict but is unclear when this condition is triggered and how it is synchronized between vehicles. Furthermore, they did not measure crossing delay but only database transactions.

DIMP18 [Lia+18] uses clusters and the vehicle closest to intersection will reach an agreement on the schedule. They also limit deceleration to reduce passenger discomfort. However, they measure “waiting time” for vehicles without defining how it is calculated and it is unclear whether their spawn rates are per lane, per road, or per intersection in total.

A.2.3 Related Approaches on Smart Contracts

This section provides details on the exisitng platforms and ideas that support smart contracts.

Blockchains with Smart Contracts

Bitcoin [Nak08] started 2009 as the first pure digital cryptocurrency and established the *Blockchain* as tamper-proof DLT on which most other implementations are built upon. The Bitcoin Blockchain uses the UTXO model to keep track of the balances for each address. Bitcoin transactions also embed *Script*, a simple stack-based byte-code-language that specifies which conditions must be met (e.g. providing the correct signature) in order to spend the Bitcoins that were transfered by the corresponding transaction. The Script is a list of instructions that are linearly executed without backward jumps which leads to Turing-incomplete programs that will always terminate [STM16].

Ethereum is an account-based DLT with focus on decentralized general purpose computing. Accounts can optionally store contract code, which will be executed each time a triggering transaction is made to the corresponding account. This way, contract-controlled accounts can autonomously interact with each other, modeling complex multi-step processes. This feature, however, inherits the risk of creating an infinite loop between two accounts [But+15]. To solve this issue, contracts can only execute a certain amount of operations which is determined by the paid transaction fees of the triggering transaction. Contracts are written as stack-based byte-code for the Ethereum Virtual Machine (EVM). The EVM is Turing-complete and can access the storage of an account which is an infinite byte array. For convenient contract creation, Ethereum offers Solidity [Etha], an object-oriented programming language based on JavaScript that can be compiled to EVM code. Ethereum also defined the ERC-20 Token Standard [VB15] which defines an unified API for tokens.

Neo Ecosystem is similar to Ethereum as it trades digital assets between parties using a smart contract framework that is executed on their own stack-based NeoVM [NEO]. However, the NeoVM allows only certain operations and NEO provides compilers from several well-known languages (e.g. C#, Java, Python) to NeoVM instructions [NEO].

NXT is a DLT that offers several *transaction templates* designed as basic communication mechanisms for the creation and trading of tokens [com14]. These templates can be seen as fixed conditional contracts and the available features include, for example, asset trading, decentralized DNS, public polls, and encrypted messaging. A user can set up such a contract template by setting parameters in a web-interface and finally issue the contract as transaction to the NXT network. When another user transfers a certain token (e.g. money as NXT tokens) to the contract, it can automatically respond with another transaction when certain conditions are met.

Corda is a UTXO-based DLT for financial trading. It uses contract code that is linked to a legal prose to achieve automation and legal enforceability [Bro+16]. The smart contracts transfer *state-objects* between communicating parties. The state-objects can hold arbitrary business information and are processed by the contract code of the receiving party.
They identified *ownable* states as the fundamental building blocks for distributed ledgers from which they derive *fungible* assets. Fungible assets can – unlike unique tokens – be combined to represent a balance.
Contracts are executed as byte-code in a deterministic Java Virtual Machine (JVM) that allows only white-listed language constructs [Hea16]. For example, contracts are limited to "pure"-functions that can only consume or append data on the state-object that was transacted to the contract function. Storing persistent state variables outside the state-object as well as using any random or time-based function is not possible.
The legal prose consists of a template text that is filled with parseable constant parameters and the hash of the legal prose is attached to the contract as reference in the case of a dispute.

Cardano is a UTXO-based, Proof-of-Stake DLT and uses its own contract language *Plutus*, which is inspired by Haskell [Car]. Plutus therefore provides strongly typed, functional, general purpose programming [Car17]. However, Plutus does also allow arbitrary naming and thus does not provide any mechanism to link code to trading ontology.

Tezos is a self-amending, account-based smart contract platform that uses delegated PoS [Pse14]. It focuses on formal verification of contract code.

Theoretical Smart Contract Models

Despite specifying smart contracts with a programming language, we also found alternative approaches in literature, which we summarize shortly in the following. These approaches are mostly of theoretical nature and try to formalize concepts and requirements for smart contracts.

The Ricardian Contract is a model for digital traded assets in which assets are described as "contracts" between an issuer and a holder [Gri04]. This method allows each participant in a trading system to issue own (competing) assets with its own set of trading rules, representing any type of value. These contracts consist of *legal text*, *parameters*, and a *signature chain* which all is digitally signed by the issuer.

By including the signing-key of the issuer in the contract itself, it contains its own PKI and only this top-level signing-key needs to be authenticated to belong to the real issuer in the beginning.

Any transaction in this system includes the hash of the contract that issued the transfered asset to secure the claims and prevent changes in the contract claims. The same contract should be readable by people and parsable by programs.

This contract concept is used by some systems, such as CommonAccord [HH17].

Smart Contract Templates which are described in [CBB16b; CBB16a], extend the concepts of the Ricardian contracts and link legal agreements to executable code to achieve enforceability – either by law or by tamper-proof software execution.

Their contracts consist of two separate parts, the legal contract prose and the executable contract code. The legal prose is written in natural language which also includes parsable parameters. These parameters are used as configuration for a standardized, fixed executable code, whose behavior is only controlled by the provided parameters.

The parameters are key-value pairs that have an *identity* (key), a *type*, and a *value*. Parameters might be defined, assigned and referenced in different locations of the legal prose and could hold complex data structures. Using powerful parameters is necessary to use standardized code, which could be thoroughly tested and certified, in contrast to custom code that could lead to unintended behavior. [CBB16b]

The authors further sketch the idea that parameter values could also be expressions based on other parameter values or that a structured language could allow to directly write the expression into the legal prose [CBB16b]. They also expect that long-term research will lead to a language that can be compiled to executable code and is legally binding at the same time.

To enable this vision, the template system needs to satisfy several requirements, such as a common ontology that allows reasoning about the semantics of the contract, as well as a structured separation of large agreements into logical parts, such as definitions, obligations and schedules [CBB16a].

Overall, the presented template system uses a separation of code and prose. Increasing the parameter complexity for standardized code decreases the verification effort for instructions but increases the complexity for verifying that the ranges and combinations of parameter values are valid. At the end, a human still needs to understand what might happen, before the code is executed.

Bibliography

[AEH75] Eralp A. Akkoyunlu, Kattamuri Ekanadham, and Richard V. Huber. “Some constraints and tradeoffs in the design of network communications”. In: *Proceedings of the fifth ACM symposium on Operating systems principles*. Nov. 1975, pp. 67–74. DOI: 10.1145/800213.806523.

[Aga+19] Shruti Agarwal, Hany Farid, Yuming Gu, Mingming He, Koki Nagano, and Hao Li. “Protecting World Leaders Against Deep Fakes.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. Dec. 2019, pp. 38–45.

[Al-+15] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”. In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376. DOI: 10.1109/COMST.2015.2444095.

[Ala+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. Tech. rep. NIST Computer Security Resource Center, July 2020. DOI: 10.6028/NIST.IR.8309.

[AM13] Pierre-louis Aublin and Sonia Ben Mokhtar. “RBFT: Redundant Byzantine Fault Tolerance”. In: *IEEE 33rd International Conference on Distributed Computing Systems (ICDCS)*. July 2013, pp. 297–306. DOI: 10.1109/ICDCS.2013.53.

[Amo+15] Mani Amoozadeh, Hui Deng, Chen-Nee Chuah, H. Michael Zhang, and Dipak Ghosal. “Platoon Management with Cooperative Adaptive Cruise Control Enabled by VANET”. In: *Vehicular Communications* 2.2 (Apr. 2015), pp. 110–123. ISSN: 2214-2096. DOI: 10.1016/j.vehcom.2015.03.004.

[Amy+16] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. “Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3”. In: *International Conference on Selected Areas in Cryptography*. Springer International Publishing, 2016, pp. 317–337. URL: https://arxiv.org/abs/1603.09383.

[AOL19] Ehsan Ahvar, Anne-Cécile Orgerie, and Adrien Lebre. “Estimating energy consumption of cloud, fog and edge computing infrastructures”. In: *IEEE Transactions on Sustainable Computing* (2019). DOI: 10.1109/TSUSC.2019.2905900.

[AR19] Shunsuke Aoki and Ragunathan (Raj) Rajkumar. “CSIP: A Synchronous Protocol for Automated Vehicles at Road Intersections”. In: *ACM Trans. Cyber-Phys. Syst.* 3.3 (Aug. 2019). ISSN: 2378-962X. DOI: 10.1145/3226032.

[ASK19] Mirko Amico, Zain H. Saleem, and Muir Kumph. “Experimental study of Shor’s factoring algorithm using the IBM Q Experience”. In: *Physical Review A* 100.1 (July 2019). DOI: 10.1103/PhysRevA.100.012305.

[Ast+20] Vittorio Astarita, Vincenzo Pasquale Giofré, Demetrio Carmine Festa, Giuseppe Guido, and Alessandro Vitale. “Floating Car Data Adaptive Traffic Signals: A Description of the First Real-Time Experiment with “Connected” Vehicles”. In: *Electronics* 9.1 (Jan. 2020), p. 114. DOI: 10.3390/electronics9010114.

[AW06] Mohamed Abdel-Aty and Xuesong Wang. “Crash estimation at signalized intersections along corridors: analyzing spatial effect and identifying significant factors”. In: *Transportation Research Record* 1953.1 (Jan. 2006), pp. 98–111. DOI: 10.1177/0361198106195300112.

[Azi+12] Reza Azimi, Gaurav Bhatia, Raj Rajkumar, and Priyantha Mudalige. “Intersection Management using Vehicular Networks”. In: *SAE 2012 World Congress & Exhibition*. Apr. 2012. DOI: 10.4271/2012-01-0292.

[Bac+14] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. “Enabling Blockchain Innovations with Pegged Sidechains”. Whitepaper. Oct. 2014. URL: https://www.blockstream.com/sidechains.pdf.

[Bai16] Leemon Baird. *The swirlds hashgraph consensus algorithm: fair, fast, byzantine fault tolerance*. Tech. rep. SWIRLDS-TR-2016-01. 2016.

[Bar18] Asha Barbaschow. “Gemalto reports 4.6 billion record breaches in the first half of 2018”. In: *ZDNet* (Oct. 2018). URL: https://www.zdnet.com/article/gemalto-reports-4-6-billion-record-breaches-in-the-first-half-of-2018/.

[BBC21] BBC. “The Lazarus heist: How North Korea almost pulled off a billion-dollar hack”. In: *BBC* (June 2021). URL: https://www.bbc.com/news/stories-57520169.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. “Keying hash functions for message authentication”. In: *Proceedings of the 16th Conference on Advances in Cryptology (CRYPTO)*. Springer. Aug. 1996, pp. 1–15. DOI: 10.1007/3-540-68697-5_1.

[BDS08] Johannes Buchmann, Erik Dahmen, and Michael Schneider. “Merkle tree traversal revisited”. In: *International Workshop on Post-Quantum Cryptography*. Springer. 2008, pp. 63–78. DOI: 10.1007/978-3-540-88403-3_5.

[BEK21] Carsten Bormann, Mehmet Ersue, and Ari Keränen. *Terminology for Constrained-Node Networks*. Internet Draft of RFC 7228. IRTF, Oct. 2021. URL: https://www.ietf.org/archive/id/draft-bormann-lwig-7228bis-07.txt.

[Ben83] Michael Ben-Or. “Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols”. In: *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*. PODC ’83. Montreal, Quebec, Canada: ACM, 1983, pp. 27–30. ISBN: 0-89791-110-5. DOI: 10.1145/800221.806707.

[Ber05] Daniel J. Bernstein. “The Poly1305-AES message-authentication code”. In: *12th International Workshop on Fast Software Encryption*. Springer. Feb. 2005, pp. 32–49. DOI: 10.1007/11502760_3.

[Bes22] Raynor de Best. *Number of cryptocurrencies worldwide from 2013 to February 2022*. Tech. rep. Statista, Feb. 2022. URL: https://www.statista.com/statistics/863917/number-crypto-coins-tokens/.

[BL20] Leemon Baird and Atul Luykx. “The Hashgraph protocol: Efficient asynchronous BFT for high-throughput distributed ledgers”. In: *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE. Sept. 2020, pp. 1–7. DOI: 10.1109/COINS49042.2020.9191430.

[Bod+20] Umesh Bodkhe, Dhyey Mehta, Sudeep Tanwar, Pronaya Bhattacharya, Pradeep Kumar Singh, and Wei-Chiang Hong. "A Survey on Decentralized Consensus Mechanisms for Cyber Physical Systems". In: *IEEE Access* 8 (Mar. 2020), pp. 54371–54401. DOI: 10.1109/ACCESS.2020.2981415.

[Bra87] Gabriel Bracha. "Asynchronous Byzantine Agreement Protocols". In: *Information and Computation* 143 (1987), pp. 130–143. ISSN: 0890-5401. DOI: 10.1016/0890-5401(87)90054-X.

[Bro+16] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. "Corda: An Introduction". Whitepaper. 2016. URL: https://www.corda.net/.

[Buc+20] Heather Buckberry et al. *Smart Technologies Enable Homes to Be Efficient and Interactive with the Grid*. Tech. rep. Oak Ridge National Lab. (ORNL), Apr. 2020. URL: https://info.ornl.gov/sites/publications/Files/Pub139277.pdf.

[Buc16] Ethan Buchman. "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains". Ph.D. Dissertation. University of Guelph, June 2016. URL: http://hdl.handle.net/10214/9769.

[But+15] Vitalik Buterin et al. "A Next-Generation Smart Contract and Decentralized Application Platform". In: *GitHub* (Jan. 2015). URL: https://github.com/ethereum/wiki/wiki/White-Paper/a8a13f538f94e3d199c0e879a74c1309e645b3d8.

[Cam+20] Fabio Campos, Tim Kohlstadt, Steffen Reith, and Marc Stöttinger. "LMS vs XMSS: Comparison of Stateful Hash-Based Signature Schemes on ARM Cortex-M4". In: *Progress in Cryptology - AFRICACRYPT 2020*. Springer, 2020, pp. 258–277. DOI: 10.1007/978-3-030-51938-4_13.

[Car] Cardano. "Why we are building Cardano". Accessed 2018-03-17. URL: https://whycardano.com.

[Car17] Cardano. "Cardano Settlement Layer Documentation". Archived at https://github.com/input-output-hk/cardanodocs.com-archived/blob/master/_docs/2017-01-01-index.md. 2017. URL: https://cardanodocs.com/.

[CBB16a] Christopher D Clack, Vikram A Bakshi, and Lee Braine. "Smart Contract Templates: essential requirements and design options". In: *arXiv preprint arXiv:1612.04496* (Dec. 2016).

[CBB16b] Christopher D Clack, Vikram A Bakshi, and Lee Braine. "Smart Contract Templates: Foundations, Design Landscape and Research Directions". In: *arXiv preprint arXiv:1608.00771* (Aug. 2016). URL: http://arxiv.org/abs/1608.00771.

[CD16] Konstantinos Christidis and Michael Devetsikiotis. "Blockchains and Smart Contracts for the Internet of Things". In: *IEEE Access* 4 (2016), pp. 2292–2303. ISSN: 21693536. DOI: 10.1109/ACCESS.2016.2566339.

[CE16] Lei Chen and Cristofer Englund. "Cooperative Intersection Management: A Survey". In: *IEEE Transactions on Intelligent Transportation Systems* 17.2 (Feb. 2016), pp. 570–586. DOI: 10.1109/TITS.2015.2471812.

[Che+19] Wen-Liang Chen, Yi-Bing Lin, Yun-Wei Lin, Robert Chen, Jyun-Kai Liao, Fung-Ling Ng, Yuan-Yao Chan, You-Cheng Liu, Chin-Cheng Wang, Cheng-Hsun Chiu, and Tai-Hsiang Yen. "AgriTalk: IoT for Precision Soil Farming of Turmeric Cultivation". In: *IEEE Internet of Things Journal* 6.3 (June 2019), pp. 5209–5223. DOI: 10.1109/JIOT.2019.2899128.

[Chi+19] Supriya Chinthavali, Varisara Tansakul, Sangkeun Lee, Anika Tabassum, Jeff Munk, Jan Jakowski, Michael Starke, Teja Kuruganti, Heather Buckberry, and Jim Leverette. "Quantification of Energy Cost Savings through Optimization and Control of Appliances within Smart Neighborhood Homes". In: *Proceedings of the 1st ACM International Workshop on Urban Building Energy Sensing, Controls, Big Data Analysis, and Visualization*. 2019, pp. 59–68.

[Cis16] Cisco. *Visual Networking Index: 2021 Forecast Highlights*. Tech. rep. Cisco, 2016. URL: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2021_Forecast_Highlights.pdf.

[Cis20] Cisco. *Annual Internet Report*. Tech. rep. Cisco, Mar. 2020. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography (Extended Abstract)". In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '00. Portland, Oregon, USA: ACM, 2000, pp. 123–132. ISBN: 1-58113-183-6. DOI: 10.1145/343477.343531.

[CL99] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. Vol. 99. Feb. 1999, pp. 173–186. URL: https://pmg.csail.mit.edu/papers/osdi99.pdf.

[Cla21] Mitchell Clark. "What is BGP, and what role did it play in Facebook's massive outage". In: *The Verge* (Oct. 2021). URL: https://www.theverge.com/2021/10/4/22709260/what-is-bgp-border-gateway-protocol-explainer-internet-facebook-outage.

[Cle+09] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. "Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults". In: *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*. Boston, Massachusetts: USENIX, Apr. 2009, pp. 153–168. URL: https://dl.acm.org/citation.cfm?id=1558977.1558988.

[Coh09] Noam Cohen. "Microsoft Encarta Dies After Long Battle With Wikipedia". In: *The New York Times* (Mar. 2009). URL: https://bits.blogs.nytimes.com/2009/03/30/microsoft-encarta-dies-after-long-battle-with-wikipedia.

[com14] NXT community. "Nxt Whitepaper". Revision 4, July 2014. URL: https://whitepaper.io/document/62/nxt-whitepaper.

[Coo+20] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. *Recommendation for Stateful Hash-Based Signature Schemes*. Tech. rep. 800-208. NIST Computer Security Division Information Technology Laboratory, Oct. 2020. DOI: 10.6028/NIST.SP.800-208.

[Cor+11] Miguel Correia, Giuliana Santos Veronese, Nuno Ferreira Neves, and Paulo Verissimo. "Byzantine Consensus in Asynchronous Message-Passing Systems: a Survey". In: *International Journal of Critical Computer-Based Systems* 2.2 (July 2011), pp. 141–161. ISSN: 1757-8779. DOI: 10.1504/IJCCBS.2011.041257.

[Cos21] Katie Costello. *Gartner Predicts the Future of Cloud and Edge Infrastructure*. Article. Gartner, Feb. 2021. URL: https://www.gartner.com/smarterwithgartner/gartner-predicts-the-future-of-cloud-and-edge-infrastructure.

[CSB19] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. *Internet of Things (IoT) and New Computing Paradigms*. Vol. 6. Wiley Online Library, Jan. 2019, pp. 1–23. DOI: 10.1002/9781119525080.ch1.

[CV17] Christian Cachin and Marko Vukolic. "Blockchain Consensus Protocols in the Wild (Keynote Talk)". In: *31st International Symposium on Distributed Computing (DISC 2017)*. Vol. 91. 2017, pp. 1–16. ISBN: 978-3-95977-053-8. DOI: 10.4230/LIPIcs.DISC.2017.1.

[Dai16] Phil Daian. "Analysis of the DAO exploit". In: *Hacking, Distributed* (June 2016). URL: http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the presence of partial synchrony". In: *Journal of the ACM (JACM)* 35.2 (Apr. 1988), pp. 288–323. DOI: 10.1145/42282.42283.

[DPP08] Benedikt Driessen, Axel Poschmann, and Christof Paar. "Comparison of Innovative Signature Algorithms for WSNs". In: *Proceedings of the first ACM conference on Wireless network security*. Mar. 2008, pp. 30–35. DOI: 10.1145/1352533.1352539.

[DRZ18] Sisi Duan, Michael K. Reiter, and Haibin Zhang. "BEAT: Asynchronous BFT made practical". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 2028–2041.

[DS08] Kurt Dresner and Peter Stone. "A Multiagent Approach to Autonomous Intersection Management". In: *Journal of Artificial Intelligence Research* 31 (Mar. 2008), pp. 591–656. DOI: 10.1613/jair.2502.

[Dua+14] Sisi Duan, Hein Meling, Sean Peisert, and Haibin Zhang. "BChain: Byzantine Replication with High Throughput and Embedded Reconfiguration". In: *Principles of Distributed Systems*. Dec. 2014, pp. 91–106. ISBN: 9783319144719. DOI: 10.1007/978-3-319-14472-6_7.

[DW13] Christian Decker and Roger Wattenhofer. "Information Propagation in the Bitcoin Network". In: *13th IEEE International Conference on Peer-to-Peer Computing*. Sept. 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688704.

[EGE02] Jeremy Elson, Lewis Girod, and Deborah Estrin. "Fine-grained Network Time Synchronization Using Reference Broadcasts". In: *ACM SIGOPS Operating Systems Review* 36 (Dec. 2002). DOI: 10.1145/844128.844143.

[EGM90] Shimon Even, Oded Goldreich, and Silvio Micali. "On-Line/Off-Line Digital Signatures". In: *Advances in Cryptology (CRYPTO 89)*. Springer, 1990, pp. 263–275. DOI: 10.1007/0-387-34805-0_24.

[EMB17] Islam Elleuch, Achraf Makni, and Rafik Bouaziz. "Cooperative Intersection Collision Avoidance Persistent System Based on V2V Communication and Real-Time Databases". In: *IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. IEEE. Nov. 2017, pp. 1082–1089. DOI: 10.1109/AICCSA.2017.20.

[Ene21] U.S. Department of Energy. *DOE Invests $61 Million for Smart Buildings that Accelerate Renewable Energy Adoption and Grid Resilience*. Tech. rep. U.S. Department of Energy, Oct. 2021. URL: https://www.energy.gov/articles/doe-invests-61-million-smart-buildings-accelerate-renewable-energy-adoption-and-grid.

[ES] Sven Efftinge and Miro Spoenemann. “Xtext – Language Engineering Made Easy!” Official website. https://www.eclipse.org/Xtext, accessed 2018-04-03.

[ESS20] Jide S. Edu, Jose M. Such, and Guillermo Suarez-Tangil. “Smart Home Personal Assistants: A Security and Privacy Review”. In: *ACM Computing Surveys (CSUR)* 53.6 (Nov. 2020), pp. 1–36. DOI: 10.1145/3412383.

[Etha] Ethereum Foundation. *The Solidity Contract-Oriented Programming Language*. Accessed 2018-03-21. URL: https://github.com/ethereum/solidity.

[Ethb] Ethereum Foundation. *The Solidity Language Grammer*. Accessed 2018-03-21. URL: https://docs.soliditylang.org/en/latest/grammar.html?highlight=grammar.

[Eva11] Dave Evans. *The Internet of Things: How the next evolution of the internet is changing everything*. Tech. rep. Cisco, Apr. 2011. URL: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[Fac21] Facebook. *Facebook Data Centers*. Tech. rep. accessed on 2021-11-09. Facebook, 2021. URL: https://datacenters.fb.com/.

[Fan+19a] K. Fan, S. Wang, Y. Ren, K. Yang, Z. Yan, H. Li, and Y. Yang. “Blockchain-Based Secure Time Protection Scheme in IoT”. In: *IEEE Internet of Things Journal* 6.3 (June 2019), pp. 4671–4679. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2874222.

[Fan+19b] Kai Fan, Shili Sun, Zheng Yan, Qiang Pan, Hui Li, and Yintang Yang. “A blockchain-based clock synchronization Scheme in IoT”. In: *Future Generation Computer Systems* 101 (2019), pp. 524–533. ISSN: 0167-739X. DOI: 10.1016/j.future.2019.06.007.

[Far+19] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Kamran Abid, and Muhammad Azhar Naeem. “A Survey on the Role of IoT in Agriculture for the Implementation of Smart Farming”. In: *IEEE Access* 7 (Oct. 2019), pp. 156237–156271. DOI: 10.1109/ACCESS.2019.2949703.

[Fer+10] Michel Ferreira, Ricardo Fernandes, Hugo Conceição, Wantanee Viriyasitavat, and Ozan K. Tonguz. “Self-organized traffic control”. In: *Proceedings of the seventh ACM international workshop on VehiculAr InterNETworking*. Chicago, Illinois, USA: ACM, Sept. 2010, pp. 85–90. ISBN: 9781450301459. DOI: 10.1145/1860058.1860077.

[FF20] Tiago M. Fernandez-Carames and Paula Fraga-Lamas. “Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks”. In: *IEEE Access* 8 (Feb. 2020), pp. 21091–21116. DOI: 10.1109/ACCESS.2020.2968985.

[Fin16] Klint Finley. “A $50 Million Hack Just Showed That the DAO Was All Too Human”. In: *Wired Business* (June 2016). URL: https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human.

[FLB09] Andrea Forte, Vanessa Larco, and Amy Bruckman. “Decentralization in Wikipedia Governance”. In: *Journal of Management Information Systems* 26.1 (2009). URL: https://www.jstor.org/stable/40398966.

[FLP82] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. *Impossibility of distributed consensus with one faulty process*. Tech. rep. Massachusetts Institute of Technology, Sept.

1982. URL: http://publications.csail.mit.edu/publications/pubs/pdf/MIT-LCS-TR-282.pdf.

[FW07] Martin Feldhofer and Johannes Wolkerstorfer. "Strong crypto for RFID tags-a comparison of low-power hardware implementations". In: *2007 IEEE International Symposium on Circuits and Systems*. New Orleans, LA, USA: IEEE, May 2007, pp. 1839–1842. DOI: 10.1109/ISCAS.2007.378272.

[Gąg+19] Adam Gągol, Damian Leśniak, Damian Straszak, and Michał Świętek. "Aleph: Efficient Atomic Broadcast in Asynchronous Networks with Byzantine Nodes". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. Zurich, Switzerland: ACM, 2019, pp. 214–228. ISBN: 978-1-4503-6732-5. DOI: 10.1145/3318041.3355467.

[Gan+05] Saurabh Ganeriwal, Srdjan Čapkun, Chih-Chieh Han, and Mani B. Srivastava. "Secure Time Synchronization Service for Sensor Networks". In: *Proc. of the 4th ACM Workshop on Wireless Security*. Cologne, Germany: ACM, 2005, pp. 97–106. ISBN: 1-59593-142-2. DOI: 10.1145/1080793.1080809.

[Gar21] Gartner. *Gartner Says Worldwide IaaS Public Cloud Services Market Grew 40.7% in 2020*. Press Release. Gartner, June 2021. URL: https://www.gartner.com/en/newsroom/press-releases/2021-06-28-gartner-says-worldwide-iaas-public-cloud-services-market-grew-40-7-percent-in-2020.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: *EUROCRYPT (2)*. Springer, 2015, pp. 281–310. ISBN: 978-3-662-46803-6. DOI: 10.1007/978-3-662-46803-6_10.

[Gmb19] Siemens Mobility GmbH. *Siemens Mobility's pilot project shows how to reduce air pollution hotspots in Munich*. Tech. rep. Siemens Mobility GmbH, Feb. 2019. URL: https://press.siemens.com/global/en/pressrelease/siemens-mobilitys-pilot-project-shows-how-reduce-air-pollution-hotspots-munich.

[GP16] Sanat Ghoshal and Goutam Paul. "Exploiting Block-Chain Data Structure for Auditorless Auditing on Cloud Data". In: *Information Systems Security: 12th International Conference, ICISS 2016, Jaipur, India, December 16-20, 2016, Proceedings*. Ed. by Indrajit Ray, Manoj Singh Gaur, Mauro Conti, Dheeraj Sanghi, and V. Kamakoti. Springer, 2016, pp. 359–371. ISBN: 978-3-319-49806-5. DOI: 10.1007/978-3-319-49806-5_19.

[Gre+19] Christopher Greer, Martin Burns, David Wollman, and Edward Griffor. "Cyber-Physical Systems and Internet of Things". In: *NIST Special Publication* 1900 (Mar. 2019), p. 202. DOI: 10.6028/NIST.SP.1900-202.

[Gri04] Ian Grigg. "The Ricardian Contract". In: *Proceedings of the First International Workshop on Electronic Contracting*. IEEE, July 2004. DOI: 10.1109/WEC.2004.1319505.

[Ham+21] Mohammad Hamad, Emanuel Regnath, Jan Lauinger, Vassilis Prevelakis, and Sebastian Steinhorst. "SPPS: Secure Policy-based Publish/Subscribe System for V2C Communication". In: *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2021)*. Grenoble, France, Feb. 2021, pp. 529–534. DOI: 10.23919/DATE51398.2021.9474070.

[HC21] Stephen Holmes and Liqun Chen. "Assessment of Quantum Threat To Bitcoin and Derived Cryptocurrencies". In: *Cryptology ePrint Archive* (2021). URL: https://ia.cr/2021/967.

[Hea16] Mike Hearn. *Corda: A distributed ledger.* Tech. rep. Version 0.5. Nov. 2016. URL: https://www.corda.net/content/corda-technical-whitepaper.pdf.

[HH17] James Hazard and Helena Haapio. "Wise Contracts: Smart Contracts that Work for People and Machines". In: *Proceedings of the 20th International Legal Informatics Symposium.* Feb. 2017. DOI: 10.2139/ssrn.2925871.

[HPS08] X. Hu, T. Park, and K. G. Shin. "Attack-Tolerant Time-Synchronization in Wireless Sensor Networks". In: *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications.* Apr. 2008, pp. 41–45. DOI: 10.1109/INFOCOM.2008.17.

[HRS16] Andreas Hülsing, Joost Rijneveld, and Fang Song. "Mitigating Multi-target Attacks in Hash-Based Signatures". In: *Public-Key Cryptography – PKC 2016.* Berlin, Heidelberg: Springer, 2016, pp. 387–416. DOI: 10.1007/978-3-662-49384-7_15.

[Hül+18] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. *XMSS: eXtended Merkle Signature Scheme.* RFC 8391. IRTF, May 2018. 74 pp. DOI: 10.17487/RFC8391.

[Hül17] Andreas Hülsing. "WOTS+ – Shorter Signatures for Hash-Based Signature Schemes". In: *Cryptology ePrint Archive* 965 (2017). URL: https://eprint.iacr.org/2017/965.

[IBM12] IBM. *Making Markets: Smarter Planet.* Tech. rep. IBM, May 2012. URL: https://www.ibm.com/investor/att/pdf/investor0512/presentation/05_Smarter_Planet.pdf.

[IBM13] IBM. *Sensors remind doctors to wash up.* Tech. rep. IBM Research Blog. IBM, Nov. 2013. URL: https://www.ibm.com/blogs/research/2013/11/sensors-remind-doctors-to-wash-up/.

[IF21] Mike Isaac and Sheera Frenkel. "Gone in Minutes, Out for Hours: Outage Shakes Facebook". In: *The New York Times* (Oct. 2021). URL: https://www.nytimes.com/2021/10/04/technology/facebook-down.html.

[INR20] INRIX. *Congestion Costs Each American Nearly 100 hours, $1,400 A Year.* Mar. 2020. URL: https://inrix.com/press-releases/2019-traffic-scorecard-us/.

[IOH21] IOHK. "Formal Specification of the Plutus Core Language (version 2.1)". Apr. 2021. URL: https://github.com/input-output-hk/plutus.

[IT15] IBM and Telecoms. *IoT Outlook 2015.* Tech. rep. Telecoms.com Intelligence, July 2015. URL: https://telecoms.com/intelligence/iot-outlook-2015/.

[ITU16] ITU. *Unleashing the potential of the Internet of Things.* Tech. rep. ITU, 2016. URL: http://handle.itu.int/11.1002/pub/811983d5-en.

[Jia+16] Dongyao Jia, Kejie Lu, Jianping Wang, Xiang Zhang, and Xuemin Shen. "A survey on platoon-based vehicular cyber-physical systems". In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 263–284. DOI: 10.1109/COMST.2015.2410831.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. "The elliptic curve digital signature algorithm (ECDSA)". In: *International journal of information security* 1.1 (July 2001), pp. 36–63. DOI: 10.1007/s102070100002.

[Jon18] Nicola Jones. "How to stop data centres from gobbling up the world's electricity". In: *Nature* 561.7722 (2018), pp. 163–167. DOI: 10.1038/d41586-018-06610-y.

[Kal60] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* (Mar. 1960). DOI: 10.1115/1.3662552.

[Kli20] Marc-Uwe Kling. *QualityLand 2.0*. Ullstein Buchverlage, 2020. ISBN: 9783843723336.

[KLS16] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-panagiota Stouka. “Proofs of Proofs of Work with Sublinear Complexity”. In: *Financial Cryptography and Data Security*. Springer, Feb. 2016, pp. 61–78. DOI: 10.1007/978-3-662-53357-4_5.

[KMZ17] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. “Non-interactive proofs of proof-of-work”. In: *Cryptology ePrint Archive*. 2017. URL: https://eprint.iacr.org/2017/963.pdf.

[KN12] Sunny King and Scott Nadal. “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake”. Aug. 2012. URL: https://bitcoin.peryaudo.org/vendor/peercoin-paper.pdf.

[KP17] Jennifer King and Christopher Perry. *Smart buildings: Using smart technology to save energy in existing buildings*. Tech. rep. Amercian Council for an Energy-Efficient Economy, Feb. 2017. URL: https://www.aceee.org/research-report/a1701.

[Lam78] Leslie Lamport. “Time, Clocks, and the Ordering of Events in a Distributed System”. In: *Communications of the ACM* 21 (July 1978). DOI: 10.1145/359545.359563.

[Lam79] Leslie Lamport. *Constructing digital signatures from a one-way function*. Tech. rep. CSL-98, SRI International Palo Alto, Oct. 1979. URL: https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/.

[Lam98] Leslie Lamport. “The Part-Time Parliament”. In: *ACM Transactions on Computer Systems* 16.2 (1998), pp. 133–169. DOI: 10.1145/279227.279229.

[Lau+21] Jan Lauinger, Jens Ernstberger, Emanuel Regnath, Mohammad Hamad, and Sebastian Steinhorst. “A-PoA: Anonymous Proof of Authorization for Decentralized Identity Management”. In: *IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2021)*. Barcelona, Spain, May 2021. DOI: 10.1109/ICBC51069.2021.9461082.

[Li+14] Shancang Li, George Oikonomou, Theo Tryfonas, Thomas M. Chen, and Li Da Xu. “A Distributed Consensus Algorithm for Decision Making in Service-Oriented Internet of Things”. In: *IEEE Transactions on Industrial Informatics* 10.2 (Feb. 2014), pp. 1461–1468. DOI: 10.1109/TII.2014.2306331.

[Lia+18] Xiaoyuan Liang, Tan Yan, Joyoung Lee, and Guiling Wang. “A Distributed Intersection Management Protocol for Safety, Efficiency, and Driver’s Comfort”. In: *IEEE Internet of Things Journal* 5.3 (2018), pp. 1924–1935. DOI: 10.1109/JIOT.2018.2817459.

[LL17] Boohyung Lee and Jong-Hyouk Lee. “Blockchain-based secure firmware update for embedded devices in an Internet of Things environment”. In: *The Journal of Supercomputing* 73.3 (2017), pp. 1152–1167. DOI: 10.1007/s11227-016-1870-0.

[LM92] Xucjia Lai and James L. Massey. “Hash functions based on block ciphers”. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1992, pp. 55–70. DOI: 10.1007/3-540-47555-9_5.

[LN04] Donggang Liu and Peng Ning. “Multilevel μTESLA: Broadcast authentication for distributed sensor networks”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 3.4 (2004), pp. 800–836. DOI: 10.1145/1027794.1027800.

[Lop+18] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and

Evamarie Wießner. “Microscopic Traffic Simulation using SUMO”. In: *21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, Nov. 2018. DOI: 10.1109/ITSC.2018.8569938.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine generals problem”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (July 1982), pp. 382–401.

[LT16] M. Lévesque and D. Tipper. “A Survey of Clock Synchronization Over Packet-Switched Networks”. In: *IEEE Communications Surveys Tutorials* 18.4 (July 2016), pp. 2926–2947. DOI: 10.1109/COMST.2016.2590438.

[Luc19] Lucius Gregory Meredith. *Rholang V 1.1.* June 2019. URL: https://rchain-community.github.io/docs/rholang/.

[Lue20a] Knud Lasse Lueth. *State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time.* Tech. rep. IoT-Analytics, Nov. 2020. URL: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/.

[Lue20b] Knud Lasse Lueth. *Top 10 IoT applications in 2020.* Tech. rep. IoT-Analytics, July 2020. URL: https://iot-analytics.com/top-10-iot-applications-in-2020/.

[Mac19] Ken MacKay. *Micro-ECC.* Commit 867e40b. Sept. 2019. URL: https://github.com/kmackay/micro-ecc.

[Mar+10] Jim Martin, Jack Burbank, William Kasch, and David L. Mills. *Network time protocol version 4: Protocol and algorithms specification.* RFC 5905. RFC Editor, June 2010. DOI: 10.17487/RFC5905.

[Mas+20] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. “Recalibrating global data center energy-use estimates”. In: *Science* 367.6481 (2020), pp. 984–986. DOI: 10.1126/science.aba3758.

[Mav+18] Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, and Audun Jøsang. “The Impact of Quantum Computing on Present Cryptography”. In: *arXiv preprint arXiv:1804.00200* (2018). DOI: 10.14569/IJACSA.2018.090354.

[MCF19] David McGrew, Michael Curcio, and Scott Fluhrer. *Leighton-Micali Hash-Based Signatures.* RFC 8554. IRTF, Apr. 2019. 61 pp. DOI: 10.17487/RFC8554.

[McK18] McKinsey. *The Internet of Things: How to capture the value of IoT.* Tech. rep. McKinsey, May 2018. URL: https://www.mckinsey.com/featured-insights/internet-of-things/our-insights/the-internet-of-things-how-to-capture-the-value-of-iot.

[MD13] Ewen MacAskill and Gabriel Dance. “NSA files decoded: Edward Snowden’s surveillance revelations explained”. In: *The Guardian* (Nov. 1, 2013). URL: https://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded.

[Mer89] Ralph C. Merkle. “A Certified Digital Signature”. In: *Proceedings on Advances in Cryptology.* CRYPTO ’89. Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1989, pp. 218–238. DOI: 10.1007/0-387-34805-0_21.

[MGW09] David Meisner, Brian T. Gold, and Thomas F. Wenisch. “Powernap: eliminating server idle power”. In: *ACM SIGARCH Computer Architecture News* 37.1 (2009), pp. 205–216. DOI: 10.1145/2528521.1508269.

[Mil+16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. “The Honey Badger of BFT Protocols”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: ACM, 2016, pp. 31–42. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978399.

[Mil85] David L. Mills. *Network Time Protocol (NTP)*. RFC 958. IRTF, Sept. 1985. DOI: 10.17487/RFC0958.

[Mil91] David L. Mills. “Internet time synchronization: the Network Time Protocol”. In: *IEEE Trans. on Communications* 39.10 (Oct. 1991). ISSN: 0090-6778. DOI: 10.1109/26.103043.

[Mir+19] Amir Mirheli, Mehrdad Tajalli, Leila Hajibabai, and Ali Hajbabaie. “A consensus-based distributed trajectory control in a signal-free intersection”. In: *Transportation Research Part C: Emerging Technologies* 100 (2019), pp. 161–176. DOI: 10.1016/j.trc.2019.01.004.

[Mod+13] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. “A survey of intrusion detection techniques in cloud”. In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 42–57. DOI: 10.1016/j.jnca.2012.05.003.

[Mor+19] Brendan Moran, Milosch Meriac, Hannes Tschofenig, and David Brown. *A Firmware Update Architecture for Internet of Things Devices*. Internet-Draft. Internet Engineering Task Force, Apr. 2019. URL: https://datatracker.ietf.org/doc/html/draft-ietf-suit-architecture-05.

[MSO17] Daniel Minoli, Kazem Sohraby, and Benedict Occhiogrosso. “IoT considerations, requirements, and architectures for smart buildings—Energy optimization and next-generation building management systems”. In: *IEEE Internet of Things Journal* 4.1 (2017), pp. 269–283. DOI: 10.1109/JIOT.2017.2647881.

[MSW18] Mithun Mukherjee, Lei Shu, and Di Wang. “Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges”. In: *IEEE Communications Surveys & Tutorials* 20.3 (Mar. 2018), pp. 1826–1857. DOI: 10.1109/COMST.2018.2814571.

[MWK17] Santa Maiti, Stephan Winter, and Lars Kulik. “A conceptualization of vehicle platoons and platoon operations”. In: *Transportation Research Part C: Emerging Technologies* 80 (Apr. 2017), pp. 1–19. DOI: 10.1016/j.trc.2017.04.005.

[Nak08] Satoshi Nakomoto. “Bitcoin: A peer-to-peer electronic cash system”. Whitepaper originally posted on http://www.metzdowd.com/pipermail/cryptography/2008-October/014810.html. Oct. 2008. URL: https://www.bitcoin.org/bitcoin.pdf.

[NCP20] Emerson Navarro, Nuno Costa, and António Pereira. “A systematic review of IoT solutions for smart farming”. In: *Sensors* 20.15 (2020), p. 4231. DOI: 10.3390/s20154231.

[NEO] NEO. “NEO White Paper”. Accessed 2018-03-17. URL: http://docs.neo.org/en-us/index.html.

[Nik+17] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. “CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds”. In: *26th*

USENIX Security Symposium (USENIX Security 17). Aug. 2017, pp. 1271–1287. ISBN: 978-1-931971-40-9. URL: https://eprint.iacr.org/2017/648.

[NL20] Jeff Nijsse and Alan Litchfield. “A Taxonomy of Blockchain Consensus Methods”. In: *Cryptography* 4.4 (2020), p. 32. DOI: 10.3390/cryptography4040032.

[NTP16] NTP Pool News. *Excessive load on NTP servers*. Dec. 2016. URL: https://news.ntppool.org/2016/12/load/.

[OCa18] OCamlPro. “Liquidity: a Smart Contract Language for Tezos”. Version 0.16. Mar. 2018. URL: https://github.com/OCamlPro/liquidity.

[OO14] Diego Ongaro and John Ousterhout. “In search of an understandable consensus algorithm”. In: *2014 USENIX Annual Technical Conference (Usenix ATC 14)*. June 2014, pp. 305–319. ISBN: 978-1-931971-10-2.

[Ope] OpenTimestamps. *A timestamping proof standard*. https://opentimestamps.org. Accessed 2019-04-02. URL: https://opentimestamps.org.

[Pay78] J.A. Payne. *ARPANET Host to Host Access and Disengagement Measurements*. Tech. rep. NTIA-REPORT-78-3. U.S. Department of Commerce, May 1978. URL: https://www.its.bldrdoc.gov/publications/download/78-3.pdf.

[PD16] Joseph Poon and Thaddeus Dryja. “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments”. DRAFT Version 0.5.9.2. Jan. 2016. URL: https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf.

[PD18] Marc-Oliver Pahl and Lorenzo Donini. “Securing IoT microservices with certificates”. In: *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. Apr. 2018, pp. 1–5. DOI: 10.1109/NOMS.2018.8406189.

[PD19] Marc-Oliver Pahl and Lorenzo Donini. “Giving IoT Services an Identity and Changeable Attributes”. In: *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Apr. 2019, pp. 455–461. ISBN: 978-3-903176-15-7.

[Per+00] Adrian Perrig, Ran Canetti, J. Doug Tygar, and Dawn Song. “Efficient authentication and signing of multicast streams over lossy channels”. In: *Proceeding 2000 IEEE symposium on security and privacy*. IEEE, 2000, pp. 56–73. DOI: 10.1109/SECPRI.2000.848446.

[Pop18] Serguei Popov. *The Tangle*. Whitepaper, Version 1.4.3. 2018. URL: https://api.semanticscholar.org/CorpusID:4958428.

[Pse14] L.M. Goodman (Pseudonym). “Tezos — a self-amending crypto-ledger”. Sept. 2014. URL: https://tezos.com/whitepaper.pdf.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. “Reaching agreement in the presence of faults”. In: *Journal of the ACM (JACM)* 27.2 (1980), pp. 228–234. DOI: 10.1145/322186.322188.

[Qia+14] Xiangjun Qian, Jean Gregoire, Fabien Moutarde, and Arnaud De La Fortelle. “Priority-based coordination of autonomous and legacy vehicles at intersection”. In: *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2014, pp. 1166–1171. DOI: 10.1109/ITSC.2014.6957845.

[Que+19] Jonas Queiroz, Paulo Leitão, José Barbosa, and Eugénio Oliveira. “Distributing intelligence among cloud, fog and edge in industrial cyber-physical systems”. In: *16th*

International Conference on Informatics in Control, Automation and Robotics (ICINCO). 2019, pp. 447–454. DOI: 10.5220/0007979404470454.

[Raa+21] Manohar Raavi, Simeon Wuthier, Pranav Chandramouli, Yaroslav Balytskyi, Xiaobo Zhou, and Sang-Yoon Chang. "Security Comparisons and Performance Analyses of Post-quantum Signature Algorithms". In: *International Conference on Applied Cryptography and Network Security*. Springer. June 2021, pp. 424–447. DOI: 10.1007/978-3-030-78375-4_17.

[Rab83] Michael O. Rabin. "Randomized byzantine generals". In: *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE, 1983, pp. 403–409. DOI: 10.1109/SFCS.1983.48.

[RBS21] Emanuel Regnath, Markus Birkner, and Sebastian Steinhorst. "CISCAV: Consensus-based Intersection Scheduling for Connected Autonomous Vehicles". In: *IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*. Barcelona, Spain: IEEE, Aug. 2021. DOI: 10.1109/COINS51742.2021.9524266.

[Reg+20] Emanuel Regnath, Nitin Shivaraman, Shanker Shreejith, Arvind Easwaran, and Sebastian Steinhorst. "Blockchain, what time is it? Trustless Datetime Synchronization for IoT". In: *IEEE International Conference on Omni-layer Intelligent Systems (COINS 2020)*. Barcelona, Spain, Sept. 2020. ISBN: 978-1-7281-6371-0. DOI: 10.1109/COINS49042.2020.9191420.

[Res18] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. IRTF, Aug. 2018. DOI: 10.17487/RFC8446. URL: https://tools.ietf.org/html/rfc8446.

[Rie+15] Raphael Riebl, Hendrik-Jörn Günther, Christian Facchi, and Lars Wolf. "Artery: Extending Veins for VANET applications". In: *International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. June 2015, pp. 450–456. DOI: 10.1109/MTITS.2015.7223293.

[Rie15] Raphael Riebl. *Vanetza*. Accessed: 2020-05-06. June 2015. URL: https://github.com/riebl/vanetza.

[Rit20] Hannah Ritchie. *What are the safest and cleanest sources of energy?* Tech. rep. Our world in data, Feb. 2020. URL: https://ourworldindata.org/safest-sources-of-energy.

[RM16] Jackeline Rios-Torres and Andreas A. Malikopoulos. "A Survey on the Coordination of Connected and Automated Vehicles at Intersections and Merging at Highway On-Ramps". In: *IEEE Transactions on Intelligent Transportation Systems* 18.5 (Sept. 2016), pp. 1066–1077. DOI: 10.1109/TITS.2016.2600504.

[Row+17] Sean Rowan, Michael Clear, Mario Gerla, Meriel Huggard, and Ciarán Mc Goldrick. "Securing Vehicle to Vehicle Communications using Blockchain through Visible Light and Acoustic Side-Channels". In: *arXiv* abs/1704.02553 (2017). URL: http://arxiv.org/abs/1704.02553.

[RP01] Mezbahur Rahman and Larry M Pearson. "Estimation in two-parameter exponential distributions". In: *Journal of Statistical Computation and Simulation* 70.4 (2001), pp. 371–386. DOI: 10.1080/00949650108812128.

[RS18a] Emanuel Regnath and Sebastian Steinhorst. "LeapChain: Efficient Blockchain Verification for Embedded IoT". In: *Proceedings of the 2018 IEEE/ACM International Con-*

ference on Computer-Aided Design (ICCAD). San Diego, CA, USA, Nov. 2018. DOI: 10.1145/3240765.3240820.

[RS18b] Emanuel Regnath and Sebastian Steinhorst. "SmaCoNat: Smart Contracts in Natural Language". In: *2018 Forum on specification and Design Languages (FDL)*. Munich, Germany, Sept. 2018. ISBN: 978-1-5386-6418-6. DOI: 10.1109/FDL.2018.8524068.

[RS19] Emanuel Regnath and Sebastian Steinhorst. "CUBA: Chained Unanimous Byzantine Agreement for Decentralized Platoon Management". In: *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2019)*. Florence, Italy, Mar. 2019, pp. 426–431. DOI: 10.23919/DATE.2019.8715047.

[RS20] Emanuel Regnath and Sebastian Steinhorst. "AMSA: Adaptive Merkle Signature Architecture". In: *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2020)*. Grenoble, France, Mar. 2020. DOI: 10.23919/DATE48585.2020.9116517.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/357980.358017.

[Sal20] Fahad Saleh. "Blockchain Without Waste: Proof-of-Stake". In: *The Review of Financial Studies* 34.3 (July 2020), pp. 1156–1190. DOI: 10.1093/rfs/hhaa075.

[SB12] João Sousa and Alysson Bessani. "From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation". In: *2012 Ninth European Dependable Computing Conference*. May 2012, pp. 37–48. ISBN: 9780769546711. DOI: 10.1109/EDCC.2012.32.

[Sch65] Val Schorre. *A Necessary and Sufficient Condition for a Context-Free Grammar to be Unambiguous*. Tech. rep. System Development Corporation, July 1965. URL: https://apps.dtic.mil/sti/pdfs/AD0620657.pdf.

[Sch90] Fred B. Schneider. "Implementing fault-tolerant services using the state machine approach: A tutorial". In: *ACM Computing Surveys (CSUR)* 22.4 (Dec. 1990), pp. 299–319. DOI: 10.1145/98163.98167.

[Sen16] Shreyas Sen. "Invited: Context-aware energy-efficient communication for IoT sensor nodes". In: *Proceedings of the 53rd Annual Design Automation Conference*. DAC '16. Austin, Texas: ACM, 2016. ISBN: 978-1-4503-4236-0. DOI: 10.1145/2897937.2905005.

[SGD11] Christoph Sommer, Reinhard German, and Falko Dressler. "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis". In: *IEEE Transactions on Mobile Computing (TMC)* 10.1 (Jan. 2011), pp. 3–15. DOI: 10.1109/TMC.2010.133.

[She+16] Arman Shehabi, Sarah Josephine Smith, Dale A. Sartor, Richard E. Brown, Magnus Herrlin, Jonathan G. Koomey, Eric R. Masanet, Nathaniel Horner, Inês Lima Azevedo, and William Lintner. *United States Data Center Energy Usage Report*. Tech. rep. Lawrence Berkeley National Laboratory, June 2016. URL: https://eta.lbl.gov/publications/united-states-data-center-energy.

[Shi+15] Jingmin Shi, Chao Peng, Qin Zhu, Pengfei Duan, Yu Bao, and Mengjun Xie. "There is a Will, There is a Way: A New Mechanism for Traffic Control Based on VTL and VANET". In: *IEEE 16th International Symposium on High Assurance Systems Engineering*. IEEE. Jan. 2015, pp. 240–246. DOI: 10.1109/HASE.2015.42.

[Sho94] Peter W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. IEEE. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[SKD20] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. "Post-Quantum Authentication in TLS 1.3: A Performance Study". In: *Cryptology ePrint Archive* (2020). DOI: 10.14722/ndss.2020.24203. URL: https://ia.cr/2020/071.

[Sta19] Nick Statt. "Amazon says fully automated shipping warehouses are at least a decade away". In: *The Verge* (May 2019). URL: https://www.theverge.com/2019/5/1/18526092/amazon-warehouse-robotics-automation-ai-10-years-away.

[Sta21] Statista. *IoT and non-IoT connections worldwide 2010-2025*. Tech. rep. Statista, Mar. 2021. URL: https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/.

[Ste+17] Marco Steger, Ali Dorri, Salil S. Kanhere, Kay Römer, Raja Jurdak, and Michael Karner. "Secure Wireless Automotive Software Updates Using Blockchains: A Proof of Concept". In: *Advanced Microsystems for Automotive Applications 2017*. Springer, Aug. 2017, pp. 137–149. ISBN: 978-3-319-66972-4. DOI: 10.1007/978-3-319-66972-4_12.

[Ste20] Greg Sterling. *More than 200 million smart speakers have been sold, why aren't they a marketing channel?* Online Article. MARTECH, Feb. 2020. URL: https://martech.org/more-than-200-million-smart-speakers-have-been-sold-why-arent-they-a-marketing-channel/.

[STM16] Pablo Lamela Seijas, Simon J Thompson, and Darryl McAdams. "Scripting smart contracts for distributed ledger technology." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 1156. DOI: 10.1007/978-3-319-70278-0.

[Sum+21] Christoph Summerer, Emanuel Regnath, Hans Ehm, and Sebastian Steinhorst. "Human-based Consensus for Trust Installation in Ontologies". In: *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. Sydney, Australia: IEEE, May 2021, pp. 1–3. DOI: 10.1109/ICBC51069.2021.9461140.

[Sza96] Nick Szabo. "Smart Contracts: Building Blocks for Digital Markets". In: *EXTROPY: The Journal of Transhumanist Thought,(16)* 18.2 (1996), p. 28.

[TA19] Francisco Tirado-Andrés and Alvaro Araujo. "Performance of clock sources and their influence on time synchronization in wireless sensor networks". In: *International Journal of Distributed Sensor Networks* 15.9 (Sept. 2019). DOI: 10.1177/1550147719879372.

[TC84] Russell Turpin and Brian A. Coan. "Extending binary Byzantine agreement to multivalued Byzantine agreement". In: *Information Processing Letters* 18.2 (1984), pp. 73–76. DOI: 10.1016/0020-0190(84)90027-9.

[Tec20] TechVision. *Inside Amazon's Smart Warehouse*. YouTube Video. TechVision, Nov. 2020. URL: https://www.youtube.com/watch?v=IMPbKVb8y8s.

[Tha21] Thames Water. *Thames Water hits half a million smart meter milestone*. Tech. rep. Thames Water, Apr. 2021. URL: https://www.thameswater.co.uk/about-us/newsroom/latest-news/2021/apr/smart-water-meter-milestone.

[Tra08] U.S. Department of Transportation. *Traffic Signal Timing Manual*. FHWA-HOP-08-024. June 2008. URL: https://ops.fhwa.dot.gov/publications/fhwahop08024/index.htm.

[TUM19] TUM-EI-ESI. *Source code of the Adaptive Merkle Signature Architecture (AMSA)*. https://github.com/tum-esi/AMSA. 2019.

[Vai21] Lionel Sujay Vailshery. *Prognosis of worldwide spending on the Internet of Things (IoT) from 2018 to 2023*. Tech. rep. Statista, Jan. 2021. URL: https://www.statista.com/statistics/668996/worldwide-expenditures-for-the-internet-of-things/.

[VB15] Fabian Vogelsteller and Vitalik Buterin. "ERC-20 Token Standard". Nov. 2015. URL: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md.

[VDS08] Mark VanMiddlesworth, Kurt Dresner, and Peter Stone. "Replacing the stop sign: Unmanaged intersection control for autonomous vehicles". In: *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*. Vol. 3. May 2008, pp. 1413–1416.

[VH08] András Varga and Rudolf Hornig. "An overview of the OMNeT++ simulation environment". In: Jan. 2008, p. 60. DOI: 10.1145/1416222.1416290.

[Vil+19] Benjamin Villalonga, Dmitry Lyakh, Sergio Boixo, Hartmut Neven, Travis S. Humble, Rupak Biswas, Eleanor G. Rieffel, Alan Ho, and Salvatore Mandrà. "Establishing the Quantum Supremacy Frontier with a 281 Pflop/s Simulation". In: *arXiv preprint arXiv:1905.00444* (2019). URL: https://arxiv.org/pdf/1905.00444.

[Vuk16] Marko Vukolić. "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication". In: *Open Problems in Network Security (iNetSec)*. Ed. by Jan Camenisch and Doğan Kesdoğan. Springer, May 2016, pp. 112–125. ISBN: 978-3-319-39028-4. DOI: 10.1007/978-3-319-39028-4_9.

[Wan+12] Le Yi Wang, Ali Syed, George Yin, Abhilash Pandya, and Hongwei Zhang. "Coordinated vehicle platoon control: Weighted and constrained consensus and communication network topologies". In: *51st IEEE Conference on Decision and Control (CDC)*. Dec. 2012, pp. 4057–4062. DOI: 10.1109/CDC.2012.6427034.

[Weg+16] Martin Wegner, Wenbo Xu, Rüdiger Kapitza, and Lars Wolf. *Byzantine Consensus in Vehicle Platooning via Inter-Vehicle Communication*. Tech. rep. Berlin, Germany: Humboldt University, Mar. 2016. URL: https://www.ibr.cs.tu-bs.de/papers/wegner-kuvs2016.pdf.

[Weg21] Philipp Wegner. *Global IoT spending to grow 24% in 2021, led by investments in IoT software and IoT security*. Tech. rep. IoT-Analytics, June 2021. URL: https://iot-analytics.com/2021-global-iot-spending-grow-24-percent/.

[Wu+14] Weigang Wu, Jiebin Zhang, Aoxue Luo, and Jiannong Cao. "Distributed mutual exclusion algorithms for intersection traffic control". In: *IEEE Transactions on Parallel and Distributed Systems* 26.1 (2014), pp. 65–74. DOI: 10.1109/TPDS.2013.2297097.

[Xu+17] Wenbo Xu, Martin Wegner, Lars Wolf, and Rüdiger Kapitza. "Byzantine Agreement Service for Cooperative Wireless Embedded Systems". In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE. June 2017, pp. 10–15. DOI: 10.1109/DSN-W.2017.45.

[Yeh01] Y.C. Yeh. "Safety critical avionics for the 777 primary flight controls system". In: *20th Digital Avionics Systems Conference (DASC)*. Vol. 1. IEEE. 2001. DOI: 10.1109/DASC.2001.963311.

[Yeo+17] Kimchai Yeow, Abdullah Gani, Raja Wasim Ahmad, Joel JPC Rodrigues, and Kwangman Ko. “Decentralized Consensus for Edge-Centric Internet of Things: A Review, Taxonomy, and Research Issues”. In: *IEEE Access* 6 (Dec. 2017), pp. 1513–1524. DOI: 10.1109/ACCESS.2017.2779263.

[You17] Tracy You. “Wifi-equipped robots triple work efficiency at the warehouse of the world’s largest online retailer”. In: *Daily Mail* (Aug. 2017). URL: https://www.dailymail.co.uk/news/article-4754078/China-s-largest-smart-warehouse-manned-60-robots.html.

[Zet16] Kim Zetter. “That Insane, $81M Bangladesh Bank Heist? Here’s What We Know”. In: *WIRED* (May 2016). URL: https://www.wired.com/2016/05/insane-81m-bangladesh-bank-heist-heres-know/.

[Zhe+16] Yang Zheng, Shengbo Eben Li, Jianqiang Wang, Dongpu Cao, and Keqiang Li. “Stability and Scalability of Homogeneous Vehicular Platoon: Study on the Influence of Information Flow Topologies”. In: *IEEE Transactions on Intelligent Transportation Systems* 17.1 (Jan. 2016), pp. 14–26. DOI: 10.1109/TITS.2015.2402153.

[Zhe+17] Bowen Zheng, Chung-Wei Lin, Hengyi Liang, Shinichi Shiraishi, Wenchao Li, and Qi Zhu. “Delay-aware design, analysis and verification of intelligent intersection management”. In: *IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE. May 2017, pp. 1–8. DOI: 10.1109/SMARTCOMP.2017.7946999.

[Zhe+20] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. “An overview on smart contracts: Challenges, advances and platforms”. In: *Future Generation Computer Systems* 105 (Apr. 2020), pp. 475–491. DOI: 10.1016/j.future.2019.12.019.

[Zho+17] Ray Y. Zhong, Xun Xu, Eberhard Klotz, and Stephen T. Newman. “Intelligent Manufacturing in the Context of Industry 4.0: A Review”. In: *Engineering* 3.5 (Oct. 2017), pp. 616–630. DOI: 10.1016/J.ENG.2017.05.015.