

POLITECNICO DI TORINO

Master's Degree in Computer Engineering



Master's Degree Thesis

Design of a Trustless Protocol for Secure Message Relay in Inter-Satellite Communication

Supervisors

Prof. Danilo Bazzanella
Dr. Andrea Gangemi

Candidate

Sara SIMONE

Company Tutors

Ing. Luca Simonini
Ing. Alberto Congedi

ACADEMIC YEAR 2024/2025

Abstract

In recent years, space communications have undergone a significant transformation, driven by the rapid growth of satellite constellations and the involvement of multiple independent operators. This increasing complexity calls for secure and interoperable solutions capable of enabling collaboration between networks that do not share mutual trust.

This thesis proposes a decentralized protocol for securely relaying encrypted messages between satellite constellations operated by different entities. The system addresses the challenge of cross-network message delivery, where the source and destination are covered by different satellite constellations that do not have direct communication paths. To bridge this gap, the protocol relies on intermediate nodes that act as relays and are economically incentivized to forward messages in a verifiable and secure manner.

Instead of relying on a central authority, the protocol uses blockchain technology to coordinate operations in a transparent and tamper-proof way. A smart contract deployed on the Ethereum network records the message handoff, verifies that delivery conditions are met, and automatically handles payments through an escrow system. Intermediate nodes are compensated only if they provide cryptographic proof of correct forwarding and if delivery to the final recipient is confirmed.

Security and authenticity are guaranteed through the combined use of public-key encryption, Zero-Knowledge Proofs (ZKPs), and digital signatures. The system has been implemented using Solidity for smart contract development, MetaMask for key and signature management, and ZoKrates for generating and verifying cryptographic proofs. Testing has been conducted on the Sepolia Ethereum testnet.

This work demonstrates how a trustless protocol—where correctness and fairness are enforced by code rather than institutional trust—can support secure, auditable, and autonomous message relay in space networks. The proposed approach constitutes a concrete step toward the deployment of decentralized and resilient satellite communication infrastructures, capable of enabling cooperative scenarios among independent entities. This thesis project was conducted in collaboration with **Thales Alenia Space**.

Contents

Abstract	1
1 Introduction	1
1.1 Background and Context	1
1.2 Objectives	2
2 State of Art	4
2.1 Satellite communication	4
2.2 Frequency Spectrum and Bands	6
2.3 Link Types in Satellite Communications	8
2.4 Security and Privacy	9
2.5 Blockchain and space applications	22
3 Architecture and Implementation of the Proposed Protocol	27
3.1 Scenario	27
3.2 Development Environments and Tools	28
3.3 Technical Implementation	33
4 Comparative Analysis	48
4.1 Spacecoin	48
4.2 Other Relevant Architectures	50
5 Conclusions	52
5.1 Achieved Results	52
5.2 Limitations and Future Perspectives	52

1 Introduction

1.1 Background and Context

In recent years, the satellite communications sector has undergone a profound transformation. The number of low Earth orbit (LEO) satellite constellations has grown rapidly, driven by large-scale deployments from initiatives such as Starlink, OneWeb, and Kuiper. This proliferation reflects a structural shift in the industry, raising concerns over coordination mechanisms [1]. At the same time, the private sector—comprising technology startups, pay-per-use commercial operators, and specialized ground segment providers—has taken on an increasingly central role in the development and operation of space infrastructure. [2]

The advent of the so-called Space Economy 4.0—characterized by increased commercialization, the proliferation of LEO mega-constellations, and the involvement of multiple independent actors—has significantly increased the complexity of space infrastructure management. This evolving landscape highlights the need for innovative solutions that can ensure interoperability, security, and reliability across heterogeneous and decentralized systems. [3]

In this context, one of the most critical challenges lies in enabling secure communication between heterogeneous satellite networks, especially in the absence of pre-established trust among the operators involved. Constellations owned by different entities do not necessarily share common protocols or mutual authentication mechanisms. This significantly complicates the implementation of a secure, verifiable, and incentivized data exchange process. In parallel, blockchain and zero-knowledge proof technologies have proven effective in domains where distributed security is critical, due to their ability to enable trustworthy interactions among untrusted parties without the need for a central authority. This paradigm is based on the cryptographic verification of code rather than user reputation, making authorization, payment, and auditing processes both automated and transparent. In this context, numerous initiatives are emerging that aim to leverage the potential of blockchain in the space domain, reflecting the sector’s growing interest in decentralized models of communication and cooperation.

Projects such as SpaceChain and SpaceCoin adopt different approaches to integrate blockchain, decentralized identities, and smart contracts into orbital infrastructures, contributing to the development of a more secure and transparent space ecosystem. These protocols aim to enhance the management of resources and communications among independent entities, reflecting a growing trend toward the adoption of cryptographic mechanisms as the foundation of trust, in contrast to traditional centralized models.

The work presented in this thesis contributes to this innovative line of research by proposing a secure relay protocol for the exchange of encrypted messages between independent satellite constellations. Two ground stations, served by distinct networks, communicate through intermediate satellites that are both economically incentivized and cryptographically authenticated. The proposed framework fosters

the development of an encrypted, resilient, and interoperable orbital infrastructure. This thesis has been carried out in collaboration with *Thales Alenia Space*, contributing to the company’s broader exploration of blockchain-based and trustless architectures for secure inter-satellite communication.

1.2 Objectives

The objective of this thesis is the design, implementation, and experimental validation of a decentralized and trustless protocol for the secure relay of encrypted messages between satellite nodes operating in independent contexts. In particular, the work explores how the integration of blockchain-based technologies, advanced cryptography, and zero-knowledge proofs can ensure the integrity, confidentiality, authentication, and traceability of communications—even in the absence of pre-existing trust among the entities involved. The proposed architecture is based on a hybrid on-chain/off-chain model, in which critical operations—such as verification, record-keeping, and the release of funds to the nodes performing the message relay—are handled by the blockchain, while the physical transmission of the encrypted data occurs through traditional channels between space nodes. The system is designed to operate in distributed scenarios characterized by the absence of centralized coordination, the presence of heterogeneous actors, and intermittent network conditions that are features typical of next-generation satellite infrastructures.

To achieve this goal, the protocol integrates the following key components:

- *End-to-end asymmetric encryption*, based on an ECIES (Elliptic Curve Integrated Encryption Scheme) model adapted to the Ethereum ecosystem. In this mechanism, the sender encrypts the message using the recipient’s public key, ensuring that only the intended recipient can decrypt it using their corresponding private key.

Starting from a shared secret generated at session initialization via asymmetric cryptography, the protocol automatically derives two ephemeral keys: one for symmetric encryption of the payload and one for data authentication (authenticated encryption). In this way, each message is protected using symmetric algorithms—more efficient for handling large volumes of data—while the underlying trust relationship between sender and recipient remains anchored to the initial asymmetric encryption.

On one hand, symmetric encryption ensures high performance and message integrity; on the other, asymmetric encryption enables a secure handshake without the need for pre-shared secret keys. The result is an end-to-end communication channel that provides confidentiality, integrity, authentication, and forward secrecy, while eliminating reliance on central authorities or prior agreements. This design allows the entire infrastructure to be immediately operational in heterogeneous, multi-operator environments.

- *Smart Contract Solidity* for the logical and economic management of the protocol. Smart contracts act as impartial “digital arbitrators,” responsible for recording key events in the message relay process, validating cryptographic proofs, and managing the conditional release of payments through an escrow mechanism. In this way, the correct behavior of participants is both incen-

tivized and verifiable on-chain, without the need for human intervention.

- *Zero-Knowledge Proof (ZKP)* are used to attest that the message has been correctly forwarded within a predefined time window, without revealing any sensitive details about the operation. The use of zk-SNARKs enables the generation of compact, efficient, and non-interactive cryptographic proofs, fully compatible with the Ethereum environment and verifiable within a few milliseconds. This module allows off-chain events to be reliably and transparently anchored to an on-chain timeline.
- *ECDSA digital signature*, following the standard used by Ethereum, is employed to unequivocally confirm that the message has been received by the intended recipient. The signature ensures sender authentication, message integrity, and non-repudiation, preventing any node from denying the successful receipt of the packet.
- *Deployment and full testing on Ethereum Sepolia*, a public testnet that faithfully replicates the behavior of the Ethereum mainnet while using tokens with no real economic value (sepoliaETH). Interaction with the blockchain was carried out using well-established tools such as MetaMask (for identity management and digital signatures), Remix IDE (for smart contract development and deployment), and ZoKrates (for modeling, generating, and verifying zk-SNARK proofs).

The resulting system is designed to operate without any pre-existing trust between the parties, integrating the entire decision-making and verification process into rules encoded in software and enforced through cryptographic guarantees. Each node involved in the relay is economically incentivized to behave correctly, as the reward is conditional upon the presentation of formal evidence of successful message forwarding and final receipt. This approach eliminates the need for off-chain agreements, centralized authorities, or trust-based assumptions, significantly reducing the vulnerabilities and limitations associated with centralized architectures. Together, these components form an innovative, scalable, and auditable protocol, designed to integrate seamlessly into a rapidly evolving space ecosystem—where the reliability of communications and cooperation among independent operators are becoming increasingly critical.

2 State of Art

2.1 Satellite communication

Satellite communication refers to the use of artificial satellites to provide connectivity between ground stations, mobile devices, and other infrastructure, enabling the transmission of data, voice, and video on a global scale. Satellite communications constitute a fundamental element of the modern telecommunications landscape, as they overcome the limitations of terrestrial networks, ensuring coverage even in remote or hard-to-reach areas. These systems rely on the interaction of three distinct segments -space, ground, and user (Figure 2.1)- and employ various orbital regimes to position satellites strategically according to operational requirements. [4]

2.1.1 Architecture

User Segment

The user segment consists of all ground-side components that let end users connect to the satellite network. It includes user terminals (e.g., VSAT antennas, satellite modems), local networking equipment (such as routers or switches interfacing with LAN/WLAN), and the software responsible for managing uplink/downlink communication, encryption, and authentication. These terminals typically handle the transmission and reception of user data directly with the satellite.

Ground Segment

The ground segment comprises Earth stations, gateways, network operation centers (NOCs), and telemetry, tracking, and command (TT&C) facilities. Its main functions include:

- Transmission of control signals and reception of satellite telemetry.
- Monitoring and control of the satellite's health and subsystems.

Space Segment

The space segment is represented by the satellites themselves and the satellite links for uplink and downlink. To ensure efficient signal transmission, modulation, coding, and multiple-access techniques are employed within the space segment. Each satellite consists of two main components [5]:

- **Payload:** contains the mission-specific instruments and equipment.
- **Platform:** encompasses and supports all the essential onboard subsystems, delivering the payload's required technical services—power, propulsion, thermal regulation and attitude control.

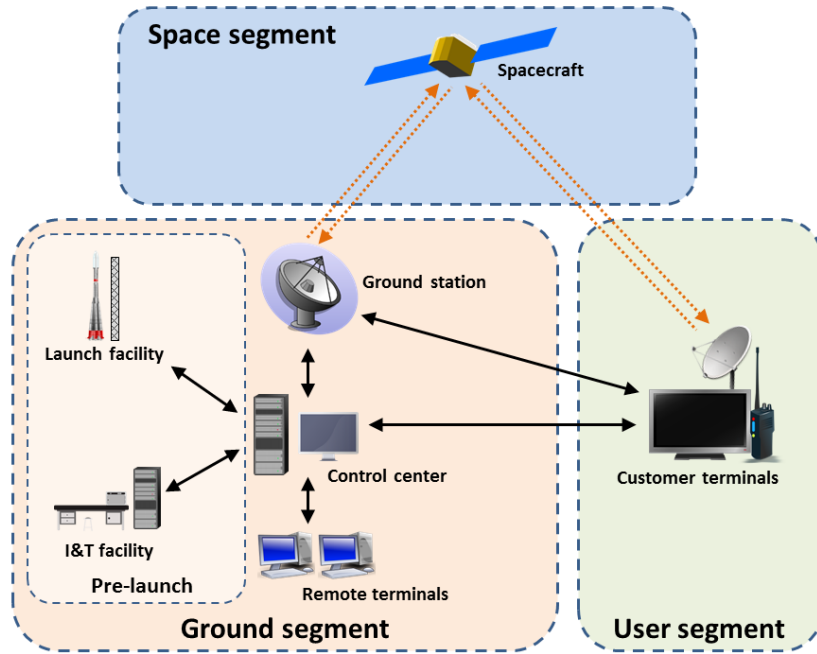


Figure 2.1: Ground Segment, User Segment and Space Segment

2.1.2 Satellite Orbits

Satellites can follow different orbital paths to suit specific operational requirements and ensure the necessary coverage depending on the mission type. Each path offers distinct advantages as well as limitations; therefore, choosing the appropriate orbit is a crucial decision in the design of space activities [6].

Geostationary Orbits (GEO)

Satellites in Geostationary Earth Orbit (GEO) are positioned at approximately 35,786 km above the equator, and their orbital period is synchronized with the Earth's rotation. As a result, they appear stationary from the perspective of an observer on the ground. This apparent immobility makes them ideal for applications such as direct-to-home television broadcasting, global communications, and meteorological monitoring.

Advantages: GEO satellites provide extensive coverage of the planet. Each satellite can observe up to one-third of the Earth's surface; consequently, only three GEO satellites are required to cover most of the globe. Their fixed position relative to the ground makes them well suited to applications that demand a stable, continuously pointed antenna—such as television services and radio broadcasts.

Disadvantages: Due to their high altitude, communication latency can be significant, rendering GEO satellites less suitable for latency-sensitive applications like videoconferencing or real-time internet services. Additionally, they require higher transmission power and are more expensive to launch into orbit compared to satellites in lower orbital regimes.

Medium Earth Orbits (MEO)

Satellites in Medium Earth Orbit (MEO) follow trajectories at altitudes ranging roughly from 2,000 to 35,786 km with various inclinations relative to the equatorial plane. MEO is the chosen regime for global navigation satellite systems such as GPS, Galileo, and GLONASS.

Advantages: MEO represents a favorable trade-off between coverage area and communication latency. Satellites in this orbit experience lower round-trip delays (approximately 40–125 ms) compared to GEO (≈ 250 ms) and—when deployed in a properly designed constellation—can provide continuous coverage with a moderate number of spacecraft. These characteristics make MEO especially well suited for applications like global navigation and Earth monitoring.

Disadvantages: To achieve continuous global coverage with MEO satellites, a larger number of satellites is required compared to a GEO constellation. Additionally, the maintenance and operational costs for MEO satellites are higher than those for satellites in Low Earth Orbit (LEO).

Low Earth Orbits (LEO)

A LEO orbit is an orbit that is relatively close to the Earth’s surface. It generally lies at an altitude below 1000 kilometers, with the lowest orbits starting around 160 kilometers. While this is significantly closer than higher orbital paths, it still represents a considerable distance from the ground. Satellite constellations in Low Earth Orbit (LEO) represent a true revolution in space technologies providing rapid communications thanks to low latency and high transmission speeds. Among the main constellations are Starlink, OneWeb, and Kuiper.

Advantages: Among the benefits of a lower orbital altitude there are the reduction in launch costs, the reduced communication latency, and the lower power requirements for both transmitting and receiving signals.

Disadvantages: The primary drawback is that each LEO satellite, orbiting at a lower altitude, covers a smaller area. Consequently, to ensure global coverage, a large number of satellites is required.

2.2 Frequency Spectrum and Bands

The electromagnetic spectrum encompasses the full range of electromagnetic radiation, from low-frequency radio waves to high-frequency gamma rays.

Satellite communications rely on specific frequency bands within the electromagnetic spectrum, each allocated for particular applications based on their physical properties and regulatory constraints. Below is an overview of the most commonly used bands and their primary purposes: [7]

- VHF-band (30-300 MHz): these signals VHF feature relatively long wavelengths compared to higher-frequency bands. This characteristic makes them particularly well-suited for short-distance space communications, such as those involving Low Earth Orbit (LEO) missions. VHF also exhibits favorable propagation behavior, enabling reliable links both between satellites and with

ground stations.

- UHF-band (300 MHz- 3GHz): Slightly higher in frequency than VHF, UHF offers better data capacity and is widely used for communication between spacecraft and ground infrastructure. Its signals can penetrate the Earth's atmosphere more effectively, enabling dependable transmission over longer distances with minimal signal loss.
- L-band (1–2 GHz): This band is commonly used for mobile satellite services, navigation systems, such as GPS and telecommunication such as Iridium. It is valued for its strong atmospheric penetration and reliability in various weather conditions.
- S-band (2–4 GHz): Employed in telemetry, tracking and command (TT & C) weather radars and maritime radar systems. This band is also used by NASA for communication with crewed spacecraft such as the ISS and previously with the Space Shuttle. It offers a balance between range and bandwidth efficiency.
- C-band (4–8 GHz): Widely adopted for satellite television and data communications, especially in regions with heavy rainfall, due to its reduced susceptibility to signal degradation caused by atmospheric moisture. Its robustness against atmospheric attenuation makes it a reliable option for Earth-to-space links.
- X-band (8–12 GHz): Primarily reserved for military applications, the X-band supports advanced radar systems including continuous wave, pulsed, synthetic aperture, and phased array radars. It is also used for weather monitoring, maritime and air traffic control, and surveillance.
- Ku-band (12–18 GHz): A popular choice for commercial satellite communication, especially for direct-to-home television broadcasting and satellite broadband internet services. It provides higher data throughput compared to lower frequency bands but is more affected by adverse weather.
- Ka-band (26–40 GHz): Known for its ability to support high-capacity and high-resolution applications, this band offers wide bandwidth, making it well-suited for data-heavy applications such as high-speed Internet services and high-definition satellite television.

The choice of frequency band is a fundamental design decision, influenced by several factors including the type of communication link (uplink, downlink, or crosslink), the mission profile, environmental conditions, and spectrum availability. This decision directly affects the performance and efficiency of the communication system. In particular, the frequency band determines the achievable data rate and bandwidth. Higher frequencies typically enable faster data transmission due to their shorter wavelengths. Lower frequency bands, such as VHF and UHF, are often used for low-data-rate communication in Low Earth Orbit (LEO), where simplicity and robustness are prioritized over throughput. When greater data volumes need to be transmitted—such as for Earth observation or scientific missions—higher bands like S-band and X-band are preferred, offering a better balance between capacity and reliability. Ku-band and X-band are commonly adopted for deep-space or interplanetary missions

However, the benefits of higher frequencies come with increased sensitivity to atmospheric conditions. Electromagnetic waves at these frequencies are more prone to attenuation caused by absorption and scattering from atmospheric particles and moisture. Therefore, selecting the most appropriate frequency band involves careful trade-offs between bandwidth availability, signal robustness, system complexity, and mission-specific constraints.

2.3 Link Types in Satellite Communications

In satellite communications, there are mainly three types of links Figure 2.2:

- **Uplink:** the transmission link from the ground segment to the satellite. In this link, data or signals are sent from a ground station (e.g., gateway, user terminal) to the satellite in orbit. Uplinks require high transmission power to compensate for atmospheric attenuation and the large propagation distance.
- **Downlink:** the reception link from the satellite to Earth, employed to convey data, signals or multimedia content from orbiting satellites to ground stations. Given the inherently limited transmission power available on board, the downlink must be designed with optimal efficiency to ensure high-quality signal reception.
- **Crosslink or Inter-Satellite Link (ISL):** the direct connection between two or more satellites without passing through the ground segment. These links are essential for networked satellite systems—such as LEO constellations—because they enable data transfer among orbiting nodes, thereby enhancing coverage, redundancy, and latency performance.

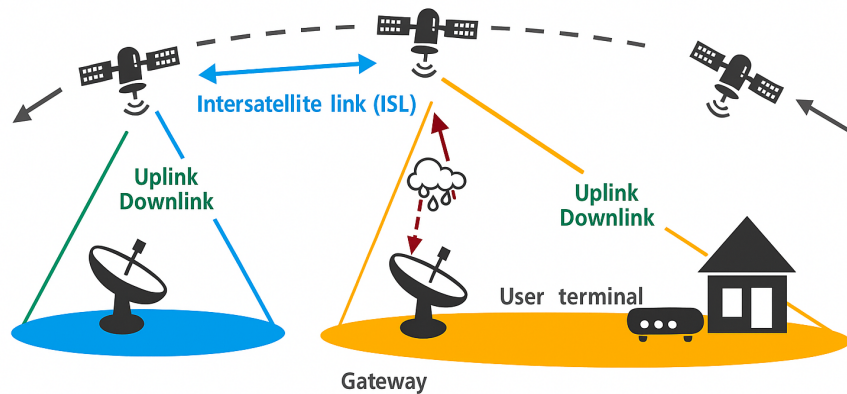


Figure 2.2: Uplink, Downlink, and Inter-Satellite Links (ISLs)

To establish these connections, two primary technologies are employed: radio-frequency links(RF) and optical links. [8]

- **Radiofrequency (RF):** RF links operate using electromagnetic waves across frequency bands such as L (1–2 GHz), C (4–8 GHz), and Ka (26–40 GHz). They are widely adopted due to their robustness against atmospheric interference and operational simplicity. The technological maturity of RF transmission is evident in the use of standardized components—such as parabolic antennas and beamforming systems—and in decades of accumulated experience within the field.

RF links are employed across all types of satellite communication—uplink, downlink, and crosslink—ensuring reliable connections even over long distances, despite the challenges posed by widely spaced or fast-moving orbital configurations. However, the RF spectrum is both regulated and limited, making it susceptible to congestion, particularly in heavily trafficked orbital regions.

- **Optical Links:** rely on the use of laser beams to transmit data at extremely high speeds. Although primarily employed for inter-satellite crosslinks, they can also be used in advanced and experimental scenarios for uplink and downlink communications, particularly between satellites and high-altitude optical ground stations.

These links offer higher bandwidth capacity, immunity to electromagnetic interference, and enhanced security due to the narrow focus of the laser beam. However, they require highly accurate Pointing, Acquisition, and Tracking (PAT) systems, which are difficult to maintain between satellites in relative motion. [9]

A key characteristic of optical links is the high directivity of the laser, which minimizes signal dispersion but makes the system extremely sensitive to even minor pointing errors. Additionally, optical links are affected by channel anisotropy, meaning non-uniform signal propagation caused by residual atmospheric turbulence or micro-vibrations of the satellite platform.

While this *anisotropy* complicates the maintenance of a stable connection, it also enhances physical-layer security, as any potential eavesdropper must be perfectly aligned with the laser beam in order to decode the signal. To fully exploit this property, techniques such as precoding and optimized constellation configurations are employed. [10]

In many systems, a hybrid strategy is adopted, in which RF links are used as a backup channel whenever optical links become degraded or misaligned.

2.4 Security and Privacy

Security is one of the greatest challenges in the space domain and cryptography plays a fundamental role in protecting confidentiality, integrity, and authenticity in space communications. In particular, asymmetric cryptography, digital signatures, and cryptographic hashes—widely leveraged by blockchain—unequivocally prove the originator of each message, thereby ensuring integrity and non-repudiation. Finally, Zero-Knowledge Proofs enable the validation of identities or access thresholds without exposing sensitive data, reducing both computational overhead and the risk of

profiling in M2M communications.

2.4.1 Cryptography Fundamentals

Cryptography is the discipline that encompasses methods for transforming data in order to conceal its informational content. The concept of *randomness* is fundamental to cryptography, since transformed data must appear random so that an attacker cannot derive any information from it. A cryptographic system is based on two main operations, as illustrated in Figure 2.3 [11]:

- **Encryption:** the process of converting a plaintext message P into a ciphertext C through an encryption algorithm E and a key K .
- **Decryption:** the inverse operation, which converts a ciphertext C back into plaintext P by means of a decryption algorithm D and a key K , which may differ from the one used for encryption.

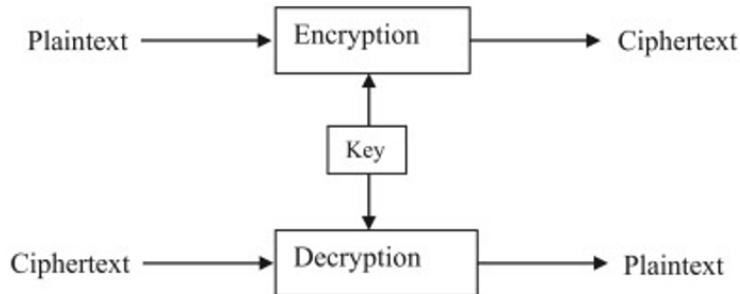


Figure 2.3: Encryption and Decryption

Cryptographic Algorithm Strength and the Principle of Kerckhoffs

A cryptographic algorithm is considered *strong* when it is computationally infeasible for an attacker to decrypt the ciphertext without the correct key. The strength of such an algorithm depends primarily on the key length, which must strike a balance between:

- **Complexity:** encryption and decryption operations should not be excessively time-consuming;
- **Security:** a brute-force attack must require a prohibitive amount of time for an attacker.

The robustness of an algorithm is not achieved by relying on the principle of *security through obscurity* (STO), which attempts to enhance or preserve system security by keeping the underlying algorithms secret. This approach is considered weak because it depends solely on secrecy: once the algorithm is discovered, the system can be broken. Although deprecated today, it has been historically used (e.g., in the case of the Enigma machine).

In contrast, the **Kerckhoffs's principle** advocates that the strength of a cryptographic system should not depend on the secrecy of the algorithm itself, but rather on the secrecy of the keys. According to this principle, algorithms should be publicly known and subject to scrutiny by the cryptographic research community.

Therefore, the security of an algorithm relies on the following conditions being met:

- Keys are kept confidential;
- Keys are managed only by trusted systems;
- Keys have sufficient length.

Under these circumstances, the secrecy of the encryption and decryption algorithms themselves becomes irrelevant.

2.4.2 Symmetric or Secret-Key Cryptography

In *symmetric cryptography*, there is a single key—denoted by K —that performs both encryption and decryption. The key is shared between sender and receiver through the use of key-exchange algorithms.

$$\begin{aligned}K_1 &= K_2 = K, \\C &= \text{Enc}(K, P), \\P &= \text{Dec}(K, C).\end{aligned}$$

Symmetric cryptography requires a low computational load; consequently, it is well-suited for encrypting large amounts of data. The disadvantage is that each pair of users must share a different key, so for private communication among N people $N(N - 1)/2$ keys are needed.

Stream and Block Ciphers

Symmetric encryption algorithms can be classified into two main categories: *stream ciphers* and *block ciphers*.

Stream Ciphers Algorithms

These algorithms operate on a continuous stream of data, typically one bit or byte at a time, without dividing the plaintext into blocks. For each bit of plaintext, a corresponding bit from a pseudorandom keystream is generated and combined using the XOR operator to produce the ciphertext. The same keystream must be used during decryption to recover the original data.(Figure 2.4)

The ideal stream cipher is the *one-time pad*, which uses a key as long as the message and guarantees perfect secrecy. However, it is impractical due to the length and management of the key. Real-world stream ciphers rely on synchronized pseudorandom key generators shared between sender and receiver. Only the seed of the generator is securely exchanged.

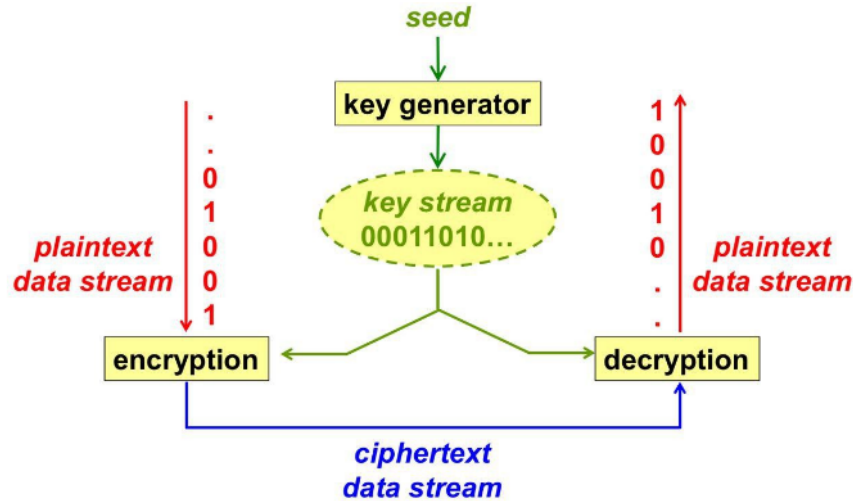


Figure 2.4: Stream Cipher

According to Kerckhoffs’s principle, the generator must be public, while the seed must be secret and unpredictable. Modern stream ciphers include *Salsa20* (now considered outdated) and *ChaCha20*, a faster and more secure variant. Both support 128 or 256-bit keys and perform 20 rounds of XOR-based mixing to ensure strong diffusion.

To prevent keystream reuse and ensure semantic security, modern stream ciphers also require a nonce (number used once) as input. The nonce is a unique, public value that is combined with the secret seed to generate a distinct keystream for each encryption operation. While the seed remains fixed for multiple messages, the nonce must vary every time to prevent the reuse of keystream segments, which would compromise confidentiality by revealing patterns in the plaintext. In particular, *XSalsa20*—an extended version of the *Salsa20* cipher—adopts a 192-bit nonce, offering enhanced security and allowing randomly generated nonces to be safely used with the same seed without risking collisions. [12]

To ensure message authenticity and integrity, stream ciphers are often combined with a Message Authentication Code (MAC). A MAC is a short tag generated from the ciphertext and a shared secret, which allows the recipient to verify that the message has not been altered. [13]

In this context, *Poly1305* is a fast and secure MAC algorithm commonly paired with stream ciphers like *XSalsa20*. It computes an authentication tag using a one-time key derived from the encryption process. This tag ensures that any tampering with the encrypted message is immediately detectable by the recipient, providing an essential layer of protection in authenticated encryption schemes. [14]

Block Cipher Algorithms

Block cipher algorithms divide plaintext into fixed-size blocks and use the same key to transform each block into a ciphertext block of equal size. Modern block ciphers typically operate on 128-bit blocks. Since the encryption operates on entire blocks, these algorithms are generally slower than *stream ciphers*.

A widely used example of block cipher is the *Advanced Encryption Standard (AES)*,

standardized by NIST in 2001. AES operates on 128-bit blocks and supports key sizes of 128, 192, or 256 bits.

When the length of the data to be encrypted does not exactly match the block size required by the algorithm, specific strategies must be employed to ensure proper processing. If the data exceeds the block size, operational modes such as *ECB* (*Electronic Code Book*) or *CBC* (*Cipher Block Chaining*) are used to divide and handle the plaintext across multiple blocks. If the data is shorter than the block size, *padding* techniques are applied to extend the final block to the required length.

- **Electronic Code Book (ECB):** ECB is the simplest block cipher mode: each block is encrypted independently from the others:

$$C_i = \text{Enc}(K, P_i) \quad \text{and} \quad P_i = \text{Dec}(K, C_i)$$

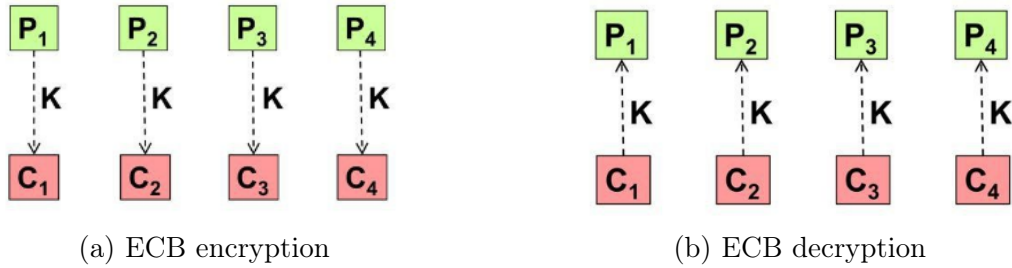


Figure 2.5: Electronic Code Book (ECB) mode

Advantages: errors affect only one block; parallel processing is possible.

Disadvantages: identical plaintext blocks produce identical ciphertexts, revealing patterns. Block order can be altered without detection. Therefore, ECB offers poor security guarantees and should be avoided.

- **Cipher Block Chaining (CBC):** CBC improves security by XORing each plaintext block with the previous ciphertext block before encryption:

$$C_i = \text{Enc}(K, P_i \oplus C_{i-1}) \quad \text{with } C_0 = IV$$

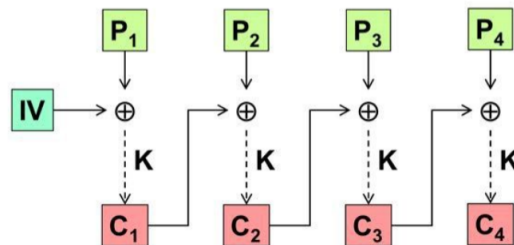


Figure 2.6: CBC Encryption

Decryption is performed as:

$$P_i = \text{Dec}(K, C_i) \oplus C_{i-1}$$

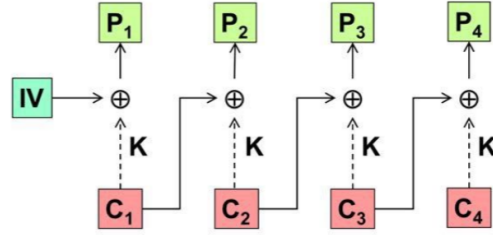


Figure 2.7: CBC Decryption

The initialization vector (IV) ensures uniqueness for the first block and must be public but never reused.

Advantages: avoids swapping and known-plaintext attacks.

Disadvantages: errors propagate to subsequent blocks; encryption is sequential and not parallelizable.

- **Padding:** To align data with the block size, padding is applied to the final block:

$$D + L = N \times B \quad \text{with } 1 \leq L \leq B$$

Even if the plaintext length is an exact multiple of the block size, a full padding block is added to avoid ambiguity. Common padding schemes include a 1-bit followed by zeros (e.g., in DES) or a sequence from 1 to L (e.g., in IPsec).

- **Galois/Counter Mode (GCM):** Galois/Counter Mode (GCM) is a mode of operation for block cipher algorithms, such as AES, that provides both data confidentiality and integrity. Unlike modes like ECB or CBC, GCM belongs to the class of *Authenticated Encryption with Associated Data* (AEAD), as it combines encryption with a message authentication mechanism.

Specifically, GCM integrates the CTR (Counter) mode, which encrypts blocks using a counter, with an authentication code computed over a Galois field, ensuring that any modifications to the data are detected. It is widely used in security protocols such as TLS (Transport Layer Security), IPsec (Internet Protocol Security), and SSH (Secure Shell), due to its efficiency and strong security guarantees.

2.4.3 Asymmetric or Public-Key Cryptography

In *asymmetric cryptography*, two different keys—a public key and a private key—are used to encrypt and decrypt data. [11]

- **Private key** (K_{pri}): known only to the owner.
- **Public key** (K_{pub}): distributed as widely as possible.

Moreover, the keys have the following characteristics:

- **Complementary Roles:** neither key is intrinsically labeled “encryption only” or “decryption only”; each pair of keys is generated together and serves both functions depending on how it is used.

- **Reciprocal Functionality:** data encrypted with one key can only be decrypted with the other key in the same pair.

Therefore, the keys are not random; instead, they share a mathematical relationship. This means that one key provides information about the other in the pair. To maintain an acceptable level of security, the key length must be at least 2048 bits. With public-key cryptography, it is possible to create a confidential message for a specific receiver and thus guarantee confidentiality without sharing any secret, as is required in symmetric cryptography. Specifically, the sender encrypts the message with the receiver's public key, and the receiver decrypts it using their private key. The decryption output will be correct only if the two keys belong to the same pair.

RSA and DSA. Among the most widely adopted asymmetric algorithms are RSA (Rivest–Shamir–Adleman) and DSA (Digital Signature Algorithm), both introduced in the 1970s and 1990s respectively. These algorithms are based on different hard mathematical problems.

RSA is built upon the *Integer Factorization Problem* (IFP): given a number $n = pq$ that is the product of two large prime numbers p and q , it is computationally infeasible to recover the original factors p and q . This one-way function enables secure key generation, encryption, and digital signatures. RSA supports both confidentiality (by encrypting with the receiver's public key) and authentication (by signing with the sender's private key).

DSA, on the other hand, is based on the *Discrete Logarithm Problem* (DLP) in finite fields. Specifically, given a large prime p , a generator g , and an output $y = g^x \bmod p$, it is hard to determine the exponent x . DSA is designed exclusively for digital signature generation and verification, making it unsuitable for encryption. The security of DSA relies on strong randomness and ephemeral keys, which must be unique for each signature to avoid vulnerabilities.

Although RSA and DSA remain widely used, they are increasingly being replaced by elliptic curve variants such as ECDSA (Elliptic Curve Digital Signature Algorithm), which offer the same level of security with smaller key sizes and improved efficiency.

The most common use cases for asymmetric cryptography are:

- *Digital signatures:* are one of the most common applications of asymmetric cryptography used also in the blockchain. In this case, asymmetric cryptography is used to encrypt a message's hash $h(M)$ with the sender's private key K_{pri} . Specifically, the flow is as follows:

1. The signer S computes

$$H = \text{Enc}(S_{K_{\text{pri}}}, h(M)).$$

2. Transmit $M \parallel H$.
3. The verifier V computes

$$X = \text{Dec}(S_{K_{\text{pub}}}, H).$$

4. If $X = h(M)$, then the signature is valid.

The security properties provided by digital signatures include:

- *Integrity*: any modification to the data will produce a different digest than the one received.
- *Authentication*: only the sender, possessing the private key, can generate the signature on the message hash.
- *Non-repudiation*: if the digest verification succeeds, the signer cannot deny having generated the message.
- *Key exchange*: One of the fundamental applications of asymmetric cryptography is key exchange, i.e., the process of establishing a shared symmetric key between two parties over an insecure channel. This symmetric key will then be used for actual encryption and decryption using fast symmetric algorithms like AES.

A prominent example of this approach is the *Diffie–Hellman* key exchange protocol, which allows two parties to derive a common secret without transmitting it directly. This is achieved through mathematical operations that rely on the *Discrete Logarithm Problem*, ensuring that an attacker who intercepts the communication cannot compute the shared key.

2.4.4 Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) is a class of public-key cryptography based on the mathematical properties of elliptic curves over finite fields. An elliptic curve is defined by an equation of the form $y^2 = x^3 + ax + b$, and consists of a set of points that satisfy this equation. In cryptography, special mathematical rules are used to perform operations on these points, making it possible to build secure cryptographic systems. [15]

Its security relies on the hardness of the *Elliptic Curve Discrete Logarithm Problem (ECDLP)*: given two points P and $Q = kP$ on the curve, it is computationally infeasible to determine the scalar k .

Compared to traditional systems like RSA or DSA, ECC offers equivalent security with significantly smaller key sizes, resulting in improved performance, reduced memory usage, and lower energy consumption. For example, a 256-bit ECC key provides comparable security to a 3072-bit RSA key.

ECC is employed in a variety of cryptographic primitives and protocols:

- **ECDSA (Elliptic Curve Digital Signature Algorithm)** is widely adopted in blockchain systems like Ethereum and Bitcoin. It enables transaction authentication and data integrity through digital signatures composed of a tuple (v, r, s) generated from the signer’s private key. [16]
- **ECIES (Elliptic Curve Integrated Encryption Scheme)** is a hybrid encryption scheme that combines ECC with symmetric encryption and message authentication to ensure both confidentiality and integrity. [17]
- **ECDH (Elliptic Curve Diffie–Hellman)** is the elliptic curve variant of the classic Diffie–Hellman key exchange protocol. It enables two parties to establish a shared symmetric key over an insecure channel without directly

transmitting the key. ECDH leverages the mathematical properties of elliptic curves, offering the same functionality as traditional DH but with significantly smaller key sizes and improved computational efficiency. The derived shared secret can then be used in symmetric encryption schemes, making ECDH a common component in secure messaging protocols and encrypted communications. [18]

- In the field of **Zero-Knowledge Proofs (ZKPs)**, elliptic curves—particularly *pairing-friendly* curves such as BN254—are fundamental. These curves enable efficient zkSNARK verifier implementations (e.g., *verifier.sol*) as used in frameworks like *ZoKrates*, allowing for privacy-preserving on-chain verification.

Thanks to their balance of security, efficiency, and scalability, elliptic curves are now essential in modern cryptographic protocols, especially within blockchain environments, IoT systems, and privacy-enhancing technologies.

2.4.5 Hashing

Hashing is the process of generating a fixed-size output called *hash* or *digest* from an input of variable size using the mathematical formulas known as *hash function*. A well-designed cryptographic hash function must exhibit the following essential properties: [19]

- *Determinism*: The same input must always produce the same output. This guarantees consistency and reliability across different computations and systems.
- *Pre-image resistance*: Computing the hash of a given input is fast, however reversing that process is impossible. The only way to discover an input that matches a particular hash is to try different candidates until one produces the desired result.
- *Avalanche effect*: Inputs that differ by a single bit produce hashes that differ by half of their bits on average, so small tweaks can't be used to predictably guide the hash.
- *Collision resistance*: Since hashes output a fixed-size value, distinct inputs can in theory collide, but a well-designed hash makes locating any two that do so require an astronomically large number of trials—so many that finding a collision is effectively impossible.

Most hash functions process input data in fixed-size blocks. The message is divided and, if necessary, padded to align with the block size required by the algorithm. Each block is processed sequentially, with each round updating an internal state that influences the final result. The output is typically presented as a hexadecimal string. Due to the *avalanche effect*, even a one-bit change in the input completely alters the output hash, ensuring high sensitivity to input variations. This process is shown in Figure 2.8.

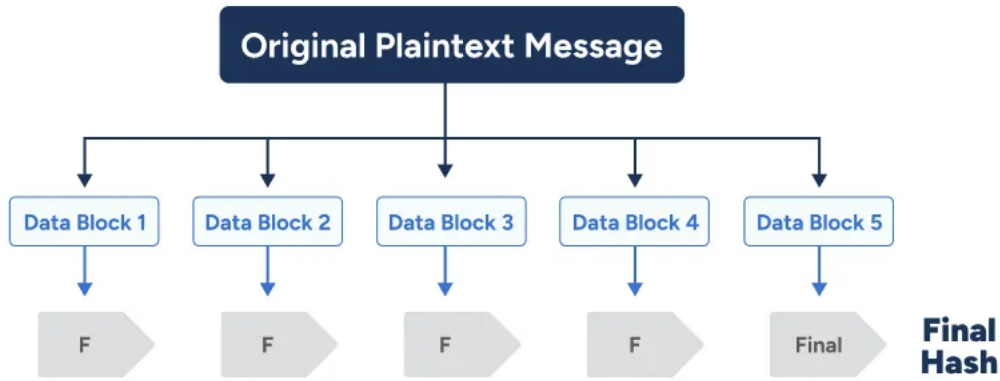


Figure 2.8: General structure of a hash function processing an input message.

Hashing plays a central role in various field of cybersecurity. Its main applications include [20]:

- *Data integrity verification:* Hashes are used to ensure that data has not been altered during transmission or storage. Any change in the original data results in a different hash, making tampering easily detectable.
- *Digital signatures:* Hashes enable compact, secure representations of data to be signed and verified efficiently.
- *Password storage:* Instead of storing passwords in plaintext, systems store hashed (and salted) versions, which enhances security against data breaches.
- *Blockchain technology:* In systems like Bitcoin, hashes link blocks together, verify transactions, and power consensus mechanisms such as proof-of-work.
- *Efficient data retrieval:* Hashing is used in hash tables and caching systems to optimize lookup speed in databases and memory structures.

Over the years, various cryptographic hash functions have been developed to ensure data integrity, authenticity, and resistance against cryptographic attacks. Among the earliest widely adopted algorithms is *MD5 (Message Digest 5)*, which produces a 128-bit hash. Although once popular, MD5 is now considered broken due to its vulnerability to collision attacks and is unsuitable for secure applications.

To address *MD5*'s limitations, the *SHA (Secure Hash Algorithm)* family was introduced by the U.S. National Security Agency (NSA) and standardized by NIST (National Institute of Standards and Technology). The first version, *SHA-1*, outputs a 160-bit digest and was long used in digital signatures and certificate verification. However, due to demonstrated collision vulnerabilities, *SHA-1* is now deprecated in favor of more secure alternatives.

The *SHA-2* family represents a significant improvement in terms of cryptographic strength. It includes a set of algorithms—*SHA-224*, *SHA-256*, *SHA-384*, and *SHA-512*—named after the number of bits in their output. [21]

Among them, *SHA-256* has become one of the most widely used hash functions in modern cryptography. It is employed in digital signatures, secure communication

protocols (such as TLS/SSL), password hashing schemes, and blockchain technologies. For instance, *Bitcoin* relies heavily on *SHA-256* for block hashing and proof-of-work consensus. To diversify cryptographic design, *SHA-3* was introduced as an alternative to *SHA-2*.

In particular *Keccak-256* is a cryptographic hash function that belongs to the Keccak family, which was selected as the winner of the *SHA-3* competition organized by the NIST. It produces a fixed output of 256 bits (32 bytes) and is known for its distinctive architecture and strong security properties. [22]

Unlike traditional hash functions, **Keccak-256** employs a *sponge construction*, a flexible and secure design paradigm that processes the input in two distinct phases: *absorption* and *squeezing*. This mechanism relies on a fixed-size internal state of *1600 bits*, which evolves through iterative transformations. The overall process can be summarized as follows:

- *Input Preparation*: The input message is padded with a specific delimiter to ensure it aligns with the sponge's internal structure and block size.
- *State Initialization*: A 1600-bit state is initialized to zero.
- *Absorption Phase*: The padded input is absorbed into the internal state in successive blocks. Each block influences the state, which is transformed at every step using a permutation function to ensure diffusion and security.
- *Permutation Function*: The state undergoes a series of nonlinear transformations designed to ensure diffusion and resistance to cryptographic attacks.
- *Squeezing Phase*: Once all input data has been absorbed, the output is extracted from the state. For Keccak-256, the algorithm outputs the first 256 bits, which constitute the final hash.

Key features of Keccak-256 include:

- **Variable-length output**: Although Keccak-256 itself produces a 256-bit hash, the Keccak framework supports outputs of arbitrary lengths, making it adaptable for a wide range of cryptographic needs.
- **High resistance** to pre-image, second pre-image, and collision attacks;
- **Efficient implementation** on both hardware and software, including constrained devices;
- **Parallel processing capability**, due to its sponge structure.

Keccak-256 is widely used in the Ethereum blockchain, where it serves as the primary hash function for a variety of core operations. It is used to compute unique identifiers for transactions and blocks, derive smart contract addresses, and ensure the immutability of stored data. Beyond its role in Ethereum, Keccak-256 is also employed in digital signature schemes, where it helps guarantee the authenticity and integrity of messages by hashing the data prior to signature generation and verification. Furthermore, like other cryptographic hash functions, Keccak-256 is essential in contexts where data integrity is critical, such as in secure storage systems and communication protocols. While SHA-3 does not replace the SHA-2 family, it com-

plements it by offering an alternative design paradigm based on sponge construction, with comparable security guarantees and enhanced structural flexibility.

Key Derivation Functions (KDF)

Hash functions, in addition to ensuring data integrity, can also be employed to derive cryptographic keys from a human-memorable secret, such as a password or passphrase. This allows users to work with easy-to-remember inputs rather than random binary sequences. However, since passwords lack sufficient entropy and are not truly random, they cannot be used directly as encryption keys without compromising security. [11]

A *Key Derivation Function (KDF)* is a mechanism designed to transform a secret passphrase into one or more cryptographic keys, typically using a hash-based process:

$$K = \text{KDF}(P, S, I)$$

where:

- K is the derived cryptographic key or key set.
- P is the user-provided password or passphrase.
- S is a salt: a unique random value added to ensure that the derived key is different for each user or session.
- I is the number of iterations used to slow down the computation and make brute-force attacks more difficult.

KDFs are essential components in password-based encryption schemes, as they harden weak secrets against attacks. A widely adopted example is **PBKDF2 (Password-Based Key Derivation Function 2)**, which employs a pseudorandom function such as HMAC-SHA1, combined with a salt of at least 64 bits and a recommended minimum of 1000 iterations, to produce secure cryptographic keys from passwords.

2.4.6 Zero-Knowledge Proof

A Zero-Knowledge Proof (ZKP) is a cryptographic method that allows one party (the prover) to demonstrate to another party (the verifier) that a statement is true without revealing any information beyond the validity of that statement itself. Usually the prover and the verifier exchange messages over several rounds to lower the chances of either side guessing or submitting false data. [23]

Features of ZKP

Zero-knowledge proof build on sophisticated cryptographic techniques and mathematical concept. They use tools like hash functions to create unpredictable challenges for the verifier, helping to establish trust between prover and verifier without revealing secrets.

Three properties that characterized a ZKP are:

- **Completeness:** if the claim is true, an honest prover can convince the verifier without difficulty.
- **Soundness:** If the claim is false, no dishonest prover can fool the verifier.

- **Zero-knowledge:** Neither side learns anything beyond the fact that the statement is true—no extra private information is disclosed.

Types of Zero-Knowledge Proofs

There are two types of zero-knowledge proofs: .

- **Interactive ZKP:** In this approach, the Prover and Verifier carry out a series of exchanges. The verifier issues challenges or questions, and the prover replies in a way that demonstrates the statement’s validity without revealing any secrets. This ZKP method works best when both sides can stay actively involved.
- **Non-Interactive ZKP:** In this form, only one message is sent from the Prover to the Verifier. The Prover creates the proof using shared public parameters, and the verifier can confirm its validity on their own, without any further exchanges. Because it eliminates repeated back-and-forth, NIZKPs are far more efficient and scalable, making them ideal for large, decentralized systems like blockchain networks. The two most deployed non-interactive constructions are [24]:

- **zk-SNARKs (Succinct Non-interactive Arguments of Knowledge):** zk-SNARKs are a family of zero-knowledge proofs that allow a prover to convince a verifier they know a secret witness w satisfying some statement, defined by a circuit C , without revealing any information about that input. *Succinct* means the proof size is small and can be verified in milliseconds, making zk-SNARKs ideal for performance-sensitive blockchains. The *non-interactive* nature implies the prover sends one proof to the verifier, who then checks it without further back-and-forth. The term *Argument* (instead of “Proof”) reflects that soundness relies on certain computational assumptions, and *of-Knowledge* guarantees no one can forge a valid proof without actually knowing the witness.

Some zk-SNARK constructions, such as *Groth16* require an initial phase called *trusted setup*. This setup involves generating a pair of cryptographic keys, one for generating proofs and another for verifying them, based on a secret parameter usually denoted as τ . The secrecy of this parameter is crucial: if τ were ever exposed, an attacker could potentially generate fraudulent proofs that still pass verification. This risk is commonly referred to as the *toxic waste* problem. To reduce this risk, practical implementations of zk-SNARKs that involve a trusted setup often rely on secure *multi-party computation*. In this approach, multiple independent participants contribute to the generation of the setup parameters, ensuring that no single entity has access to the complete secret. At the end of the process, all temporary data are destroyed, and the resulting parameters can be safely used in the system. [25]

- **zk-STARKs (Scalable Transparent Arguments of Knowledge)** were introduced as an advanced alternative to zk-SNARKs. One of their main advantages is that they don’t require a trusted setup phase, making the system more transparent and easier to deploy. This is possible thanks to the use of symmetric cryptography and strong hash functions, which are

considered more secure against future threats such as quantum attacks. Unlike zk-SNARKs, where the complexity of the proof increases with the size of the circuit, zk-STARKs keep the cost nearly constant. This makes them faster and reduces the amount of data exchanged between the prover and the verifier. [26]

Use cases

Zero-knowledge proofs are one of the most flexible tools for enhancing security and privacy in machine-to-machine communication. In particular ZKPs enable:

- **Mutual authentication without key exposure:** each device proves possession of its cryptographic credentials without ever transmitting them in the clear, avoiding replay attacks.
- **Message-integrity:** proofs accompany every transmission, certifying that the data haven't been tampered with en route, all without revealing the underlying payload.
- **On-the-fly secure channel setup:** complex parameters (for example, cipher suites or session keys) can be established dynamically via concise proofs, minimizing latency and bandwidth consumption.

2.5 Blockchain and space applications

Blockchain, originally designed to make financial transactions with digital currencies secure and feasible, is now finding application in the space domain. By combining blockchain's low-cost, reliable, and decentralized cryptographic framework with the high intelligence and autonomous communication capabilities of satellite constellations, it becomes possible to create a powerful tool for managing space-based data. The following paragraphs introduce the fundamental principles of how blockchain works and then explain how the characteristics of this technology can be adapted to satellite communication contexts. [27]

2.5.1 Blockchain Fundamentals

Blockchain is a type of DLT (Distributed Ledger Technology) that consists of blocks linked to one another by cryptographic hashes, forming a “chain”—hence the name blockchain, or “chain of blocks.” Each block is composed of:

- **Blockheader:** contains all metadata, the hash of the previous block, information on the block's difficulty level, the timestamp and the nonce.
- **Body:** holds the transactions and events that need to be recorded.

In this way, blocks are chronologically linked, and altering even a single byte in any block would invalidate all subsequent blocks, thereby deterring tampering. Figure 2.9 shows a generic chain of blocks.

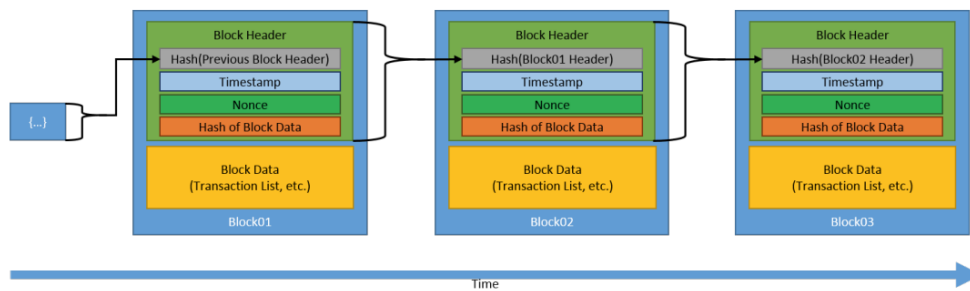


Figure 2.9: Chain of blocks

Consensus mechanisms

All nodes in the blockchain maintain a real-time updated copy of the ledger. To determine who will publish the next block in the chain—and thus which transactions will be considered valid—a *consensus mechanism* is employed. This is a procedure by which the peers in a blockchain network (miners) reach agreement on the network’s current state. The most common consensus mechanisms are:

- **Proof of Work (PoW):** In this model, the right to publish a block is granted to the first node that solves a complex cryptographic puzzle using high-powered computers. The solution to this puzzle, which can be easily verified by any node in the network, serves as the *proof* of the *work* performed.
- **Proof of Stake (PoS):** This method is based on a process called “staking.” In this case, miners lock up a certain amount of cryptocurrency and compete to be randomly selected to validate a new block and receive a reward.

Thus, having described these two consensus mechanisms, it is evident that PoW awards the reward to the first miner who solves the puzzle—leading to intense competition and high energy consumption—whereas PoS is far less energy intensive and allows for broader participation without requiring specialized hardware.

Technical Properties

The robustness of blockchain technology stems from a set of characteristics that render it highly versatile and applicable across a wide range of domains. The primary properties include: [28]:

- **Decentralization:** There is no central authority controlling the ledger; instead, each node in the blockchain network maintains an identical copy. A new block is appended only through a *consensus mechanism*.
- **Transparency:** every block on the blockchain, starting from the genesis block, remains permanently accessible to all participants, since each node holds the same copy of all transactions. Consequently, any node can verify the entire transaction history at any time, ensuring a fully transparent ledger.
- **Encryption:** trust in the blockchain is founded on cryptographic algorithms that prove the authenticity of each transaction. No third party is required to verify the validity of the information.
- **Immutability:** cryptography is also fundamental in this respect. Each block on the blockchain contains the hash of the previous block, linking them to-

gether to form a chain. If anyone attempted to alter a single byte anywhere in the shared ledger, the entire sequence of hashes would immediately change, and the network nodes would detect the tampering. Before a block is added, the entire network verifies its validity and only accepts it with majority consensus. As a result, once a block is appended, it becomes permanent and unalterable.

- **Programable:** blockchain technology enables the implementation of so-called *smart contracts*. A *smart contract* is a collection of code and data (often referred to as functions and state) that is deployed via cryptographically signed transactions on the blockchain network. These contracts are executed by all network nodes, each of which must arrive at the same result. The outcome is recorded on the ledger and becomes immutable, just like the blocks themselves. Smart contracts allow for the automation of payments, the management of multi-party processes, and the secure storage of information using a “if-then” paradigm. Execution of a smart contract proceeds sequentially: each step can only be executed after the immediately preceding step has completed successfully. Running a smart contract incurs a cost in terms of gas/fees and must stay within predefined computational limits. Thanks to this programmability, the blockchain becomes much more than a passive ledger; it evolves into a system capable of automatically enforcing complex business logic without the need for a central authority.
- **Events in Smart Contracts:** a key feature of programmable blockchains, especially in platforms like Ethereum, is the ability of smart contracts to emit *events*. Events are used to log specific actions or states within the contract’s execution. While they do not affect the blockchain’s state, they are recorded in the transaction logs and can be efficiently accessed by off-chain applications. This mechanism is essential for enabling interaction between smart contracts and external systems, such as user interfaces or monitoring tools. For example, a decentralized application (dApp) can listen for a particular event and automatically update its UI (User Interface) when the event is emitted. Events thus serve as a bridge between on-chain computations and off-chain responses, enhancing the responsiveness and usability of blockchain-based systems

Gas and Fees

Executing operations on a programmable blockchain—such as sending transactions or triggering smart contracts—incurs a computational cost. To prevent abuse and ensure fair allocation of network resources, many blockchains adopt a pricing system based on virtual computation units, often referred to as *gas*. [29] Gas measures the amount of computational effort required to complete an operation.

Each user requesting an operation on the network must specify how much gas they are willing to consume, and at what price per unit. The total cost, known as the *fee*, is paid in the network’s native cryptocurrency (for example, ether on Ethereum) and serves as compensation for the validators who execute and record the operation.

For example, on the Ethereum network, the actual cost of a transaction is calculated as:

$$\text{Gas fee} = \text{Gas used} \times (\text{Base fee} + \text{Priority fee})$$

where:

- the *base fee* is dynamically determined by the protocol based on network congestion;
- the *priority fee* (or *tip*) is an optional incentive for validators;
- the *gas limit* represents the maximum amount of gas the user is willing to spend.

A transaction will only be accepted if the sum of the base fee and the priority fee is less than or equal to the maximum cap set by the user (e.g., *maxFeePerGas* in Ethereum). Any unused portion is typically refunded.

This system is essential for the security and efficiency of the network, as it prevents infinite or malicious code execution, discourages spam, and enables dynamic regulation of network traffic.

2.5.2 Blockchain in space

In recent years, the space industry has undergone significant growth: companies and nations now compete for lunar missions, space programs, and orbital expansions. The emergence of commercial LEO constellations, private lunar missions, and reusable launch platforms has ushered the sector into its Space 4.0 phase—an ecosystem that includes not only large corporations but also agencies, start-ups, investors, and digital service providers operating on a global scale. [30] This “democratization” of space introduces a need to protect an increasing number of assets from potential cyberattacks, certify payments in real time, and coordinate transactions among parties that often lack preexisting trust. In such a context, blockchain—with its smart contracts, decentralized consensus, and immutable ledger—provides a robust solution to these challenges, adding a “by design” layer of reliability to emerging business models. The following paragraphs examine the four primary advantages that blockchain offers to space activities. [31]

Trust

Collaboration between agencies, start-ups, and investors requires a ledger that anyone can consult at any time without relying on a central authority. In this regard, blockchain combines transparency and immutability to establish trust among parties that do not necessarily trust one another: every transaction remains visible and accessible to anyone at any time. This feature is critical for international collaborations and partnerships in the space sector, as it reduces the risk of disputes or mistrust.

Data Security and Integrity

One of the most critical concerns in the space sector is ensuring that data remains intact and unaltered during transmission, reducing the risk of tampering and interception. Blockchain provides a secure means of transmitting data between space assets and ground stations. By employing cryptographic hashes, each block in the blockchain is linked to its predecessor, forming an unbroken and secure chain. Even the slightest modification to the ledger is detected, as it will no longer match the

recorded hash, and the network immediately discards the corrupted copy.

Resilience

Resilience and availability are two characteristics of blockchain that enable this technology to continue operating even when some nodes disconnect for a period or cease functioning. In the space ecosystem, these traits offer advantages from multiple perspectives. There is no longer a single point of failure: even if a node stops working or a local malfunction occurs, the system continues to operate, ensuring continuity of operations that is not present in centralized systems. When nodes reconnect, they are able to re-synchronize autonomously without the assistance of a third party.

Operational Autonomy

Thanks to the use of smart contracts, blockchain introduces efficiency by automating payments, escrow, and agreements among different parties. It is possible to implement specific operational logics—such as a pay-per-use model—where the contract automatically releases funds as soon as the predefined condition is satisfied; otherwise, it applies a penalty or remains pending. This fosters the emergence of new cooperative business models, for example, automatically “leasing” space resources to multiple users simultaneously, regulating payments and priorities without human intervention and strengthening trust among collaborators.

2.5.3 Technological Challenges

Despite its rapid evolution, blockchain remains an emerging and, in certain respects, still immature technology; consequently, it presents challenges and limitations that must be addressed. Below are some of the constraints associated with this technology: [32]

- **Energy Consumption:** The use of consensus algorithms such as Proof of Work within blockchains like Bitcoin leads to very high energy consumption—approximately 125 TWh per year—resulting in a significant carbon footprint.
- **Privacy and security:** Although blockchain is theoretically robust and secure, privacy-related vulnerabilities still persist: for example, 51 % attacks, double-spending, and risks associated with publishing personal data that could be exploited by malicious actors.
- **Scalability, Latency, and Throughput:** Blockchain also operates predominantly in the technology and cloud storage sector. One of the limitations of the technology is scalability in relation to latency and throughput, since all transactions must be validated and stored. For example, Bitcoin could initially process only 7 transactions per second, resulting in long transaction times and high costs when the network is congested. Although there have been significant improvements in recent years, it remains a limitation because the goal is to ensure smooth transaction processing.
- **Regulatory Challenges:** The decentralization of blockchain can make it difficult to enforce laws that are often inadequate or unenforceable. This may lead to conflicts with local and global regulations, especially in the financial sector.

3 Architecture and Implementation of the Proposed Protocol

3.1 Scenario

In this chapter, we present a blockchain-based secure relay architecture that enables two ground stations, A and B, to exchange messages reliably via two satellite constellations—Alpha and Beta—operated by independent entities with no prior trust agreements.

3.1.1 Main Actors

Ground Station A

- Generates the message M to be transmitted and encrypts it.
- Is intermittently covered by one or more nodes of the Alpha constellation (e.g., LEO or MEO satellites).
- Has no direct visibility of any node in the Beta constellation, and therefore cannot deliver the packet to B without intermediaries.

Alpha Node (Alpha Constellation)

- May be a Low Earth Orbit (LEO) or Medium Earth Orbit (MEO) satellite, or any mobile platform that passes over A with short contact windows but low latency.
- Receives and temporarily stores the encrypted message from A, and forwards it via the blockchain as soon as it comes into view of a node in the Beta constellation.

Beta Node (Beta Constellation)

- May be a Geostationary (GEO) satellite, a MEO satellite, or another node with continuous coverage over the region where B is located.
- Has no direct visibility of A, but receives the message as soon as it comes into line of sight with Alpha. As the intermediary node, Beta then forwards the message to B once it reappears.
- Is financially incentivized to forward the message to B, as compensation is granted only upon successful completion of the operation. The node sends the message to B upon reentry into coverage (in the case of LEO/MEO), or immediately if continuous coverage is available (as with GEO satellites). Upon successful delivery, it generates a zero-knowledge proof (ZKP) and obtains a digital signature from B. The payment is automatically released by the smart contract only if the proof is valid and the signature is correctly verified.

Ground Station B

- Final recipient of the message M .

- Lies permanently within the coverage area of one or more nodes of the Beta constellation, receiving data without intermittent visibility windows (in the case of GEO) or with predictable windows (in the case of secondary MEO/LEO nodes).
- Confirms the reception of the message.

The main objectives of the proposed protocol are:

- *Guaranteed delivery and timeliness:* ensuring that a message originating from A reaches B even when the destination ground station is not immediately reachable by the same originating constellation. This is achieved by leveraging relaying through multiple constellations to minimize waiting times.
- *Trustless cooperation:* enabling two constellations—potentially operated by independent entities with no prior trust relationships—to collaborate securely and with proper incentives through the use of blockchain technology. Each phase of the relay process is recorded on a public and immutable ledger, ensuring transparency and minimizing the need for offline legal agreements or pre-established trust.

3.1.2 Role of the Blockchain

The blockchain serves as the foundational framework for immutably storing the hash of the encrypted message that A intends to send to B, and for managing an escrow mechanism in ETH. This mechanism is designed to incentivize the Beta constellation to correctly forward the data to B and to generate a zero-knowledge proof attesting to the successful relay. Through this system, ground stations are able to communicate securely, regardless of their geographical separation.

The proposed solution ensures an optimal trade-off among the following aspects:

- *Confidentiality* of sensitive information, achieved through asymmetric encryption and zero-knowledge proofs.
- *Integrity and authentication*, ensured via the use of cryptographic hashes and digital signatures.
- *Non-repudiation*, guaranteed by the application of digital signatures.
- *Economic escrow mechanism*, which locks the reward intended for Beta until the data relay is successfully completed, leveraging the trustless nature of the blockchain.

3.2 Development Environments and Tools

To implement the protocol, a suite of software tools was selected based on their integration with Zero-Knowledge Proofs, smart contract development, and interaction with the Ethereum Sepolia testnet.

Remix IDE

For the development of the Solidity smart contract *MessageRelayWithZkpAndSignature*, the Remix IDE (Integrated Development Environment) was used. Remix IDE is an open-source application available both as a web-based and desktop tool. This environment is specifically designed to facilitate the creation, testing, debugging,

and deployment of smart contracts written in Solidity on various blockchain platforms such as Ethereum. Remix stands out for its integrated compiler and native debugger, which simplify the smart contract development process. Thanks to its modular architecture and extensive plugin system, it allows users to customize the development environment according to their specific needs. The Remix interface is intuitive and user-friendly, aiming to streamline the workflow for smart contract creation. [33]

Zokrates and Docker

In this protocol, *ZoKrates*—a widely used open-source toolbox—was adopted to support the entire process of Zero-Knowledge Proof (ZKP) generation, from defining the off-chain circuit to generating the *verifier.sol* contract and performing the on-chain cryptographic proof verification. ZoKrates provides a comprehensive environment that includes a domain-specific language (DSL), a compiler, a *zkSNARK* proof generator, and a Solidity smart contract generator for blockchain verification. [34]

The framework is designed to simplify the inherent complexity of Zero-Knowledge Proofs by offering high-level tools that allow developers to express private computations in a human-readable form, which are then transformed into circuits compatible with the *Groth16* protocol. This protocol enables the creation of short, non-interactive proofs with constant verification costs, leveraging elliptic curve cryptography—specifically the BN128 curve. [35]

To ensure portability and isolation from the host operating system, all ZoKrates operations were performed inside a *Docker* container. *Docker* is an open-source platform designed for the rapid development, testing, and deployment of applications. It allows developers to separate applications from the underlying infrastructure, facilitating consistent and reproducible environments.

The core component of *Docker* is the *container*, a lightweight and isolated unit that includes everything needed to run an application—such as code, libraries, and dependencies.

Thanks to the isolation provided by containers, it is possible to run multiple applications simultaneously on the same host without conflicts. Containers guarantee that the software behaves consistently across different systems, regardless of host configurations. [36]

In this project, *Docker* was used to run the ZoKrates environment in a reproducible way, avoiding manual setup and ensuring compatibility across platforms. All ZoKrates commands—such as circuit compilation, witness computation, and proof generation—were executed inside a container based on the official *zokrates/zokrates* image. [37]

To make the local project directory accessible within the container, a volume mount was used, allowing files generated inside the container (e.g., *proof.json*, keys, *verifier.sol*) to be stored persistently on the host system.

The *Docker* container was launched with the following command:


```
docker run --rm -it -v C:\path\to\folder:/home/zokrates/workspace
-w /home/zokrates/workspace zokrates/zokrates:latest bash
```

The entire operational workflow within ZoKrates consists of the following steps (Figure 3.1):

- Compilation of the circuit written in the ZoKrates language (*proof.zok*)

```
zokrates compile -i proof.zok
```

The *proof.zok* file, which contains the computation logic, is translated into flattened code compatible with the R1CS (Rank-1 Constraint System) format—a standard representation for building zkSNARK circuits.

- Trusted setup

```
zokrates setup
```

In this phase, the initial setup required by the Groth16 protocol is performed, resulting in the generation of two essential keys: the *proving key*, used to generate the proof, and the *verification key*, which is embedded into the Solidity contract for on-chain verification.

- Witness computation

```
zokrates compute-witness -a <input1> <input2> ...
                           <inputN>
```

The witness generator executes the *.zok* circuit using the provided private and public inputs, producing a *witness*. This assignment represents a proof of the correct execution of the program and serves as the basis for generating the zero-knowledge proof.

- Generation of the zkSNARK proof

```
zokrates generate-proof
```

Using the proving key and the witness, a proof is generated, consisting of the values *a*, *b*, and *c*, which can subsequently be verified on the blockchain.

- Exporting the verifier.sol file

```
zokrates export-verifier
```

With this command, the *verifier.sol* contract is automatically generated. It contains the *verifyTx* function, which uses elliptic curve *pairings* to verify the correctness of the received proof.

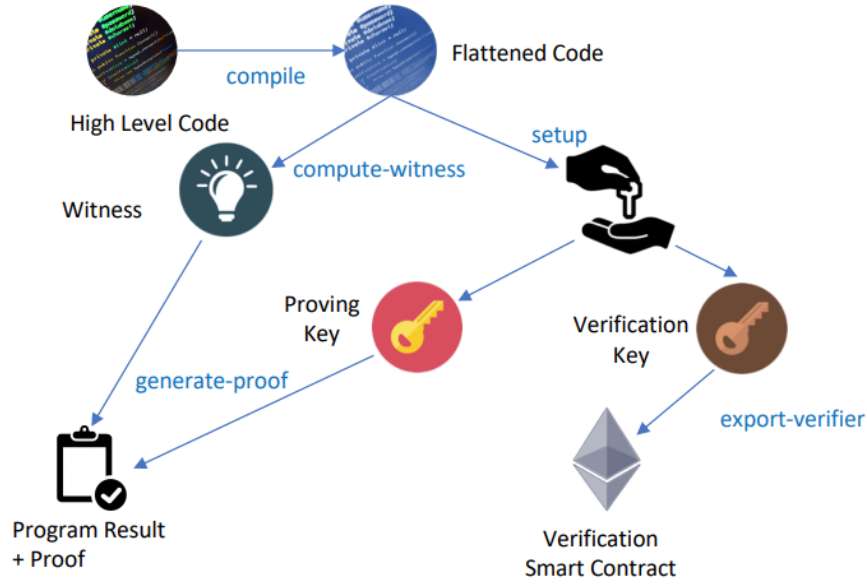


Figure 3.1: Zokrates process

In the implemented protocol, the proof is provided to the *MessageRelayWithZkpAndSignature* smart contract through the *proveAndRelease* function, which internally calls the *verifyTx* function. ZoKrates thus enables the integration of confidentiality, verifiability, and scalability.

Sepolia and Metamask

For the development and validation of the proposed protocol, the *Sepolia* network—one of Ethereum’s official testnets—was used in combination with the *MetaMask* wallet. Sepolia accurately simulates the behavior of the Ethereum mainnet in terms of smart contract execution, state management, and block structure, while relying on test tokens (*SepoliaETH*) that carry no economic value. The network adopts a *Proof-of-Stake* consensus mechanism, managed by a limited set of validators, which ensures a stable and controlled environment—ideal for development and testing activities. This approach enabled the deployment of complex smart contracts in a risk-free economic context, while still maintaining technical realism. *SepoliaETH* tokens can be obtained for free through faucets, which dispense small amounts of tokens directly to the user’s *MetaMask* wallet. An example is the faucet provided by Google, shown in Figure 3.2, which allows users to receive 0.05 *SepoliaETH* in their wallet by simply entering their Ethereum address. [38]

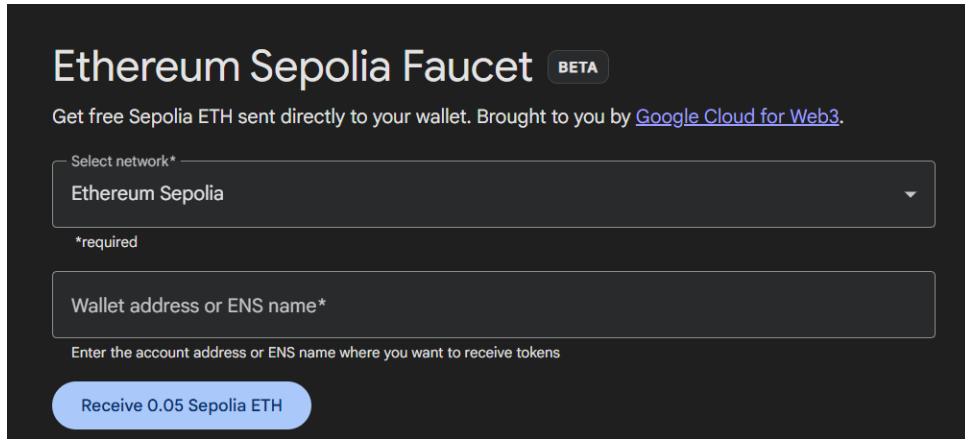


Figure 3.2: Sepolia faucet

MetaMask is one of the most widely used wallets in the Ethereum ecosystem, available as a browser extension (for Chrome, Firefox, Brave, and Edge) and as a mobile application. [39] In addition to managing private keys and performing cryptographic signatures, MetaMask acts as an *Ethereum provider* by injecting the *window.ethereum* object into the browser context. This object serves as the primary interface for interaction between a decentralized application (dApp) and the blockchain, and it implements the *JSON-RPC* communication protocol. Through *window.ethereum*, a dApp can request access to Ethereum accounts, sign transactions and messages, and communicate with remote nodes. DApps typically rely on libraries like *ethers.js* to build and send JSON-RPC calls through MetaMask’s injected provider; alternatively, developers can skip those abstractions and interact directly with the *window.ethereum* API to issue raw JSON-RPC requests. These requests are handled by MetaMask and forwarded to an Ethereum node, often via a remote provider like *Infura*, which serves as an access point to the blockchain network. As illustrated in Figura 3.3, the flow begins with the WebApp running in the browser, which leverages the injected *window.ethereum* object to securely interact with the blockchain, without the need to run a local node. [40]

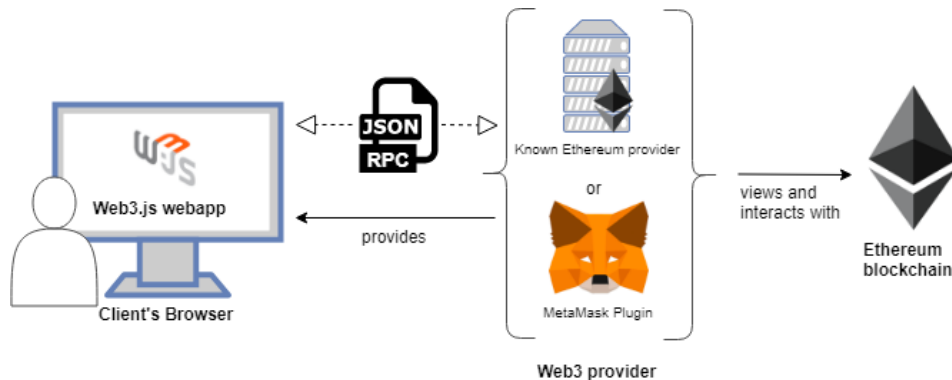


Figure 3.3: dApp-to-blockchain connection

3.2.1 Use cases

Below are two example use cases demonstrating how the secure relay can be applied:

- **LEO-GEO Hybrid Architecture:** In a hybrid satellite architecture, LEO satellites provide low-latency and temporary coverage over specific regions of the globe, while GEO satellites provide continuous service over large areas while maintaining a fixed position relative to the Earth's surface. In the Figure 3.4 the footprint of *Alpha* (satellite in LEO orbit) is highlighted in red, while the coverage of *Beta* (GEO satellite) is shown in blue. Ground station *A*, located within *Alpha*'s footprint, can only transmit data during direct visibility windows with the LEO satellite. However, the same station results permanently outside the coverage of the GEO *Beta* satellite, which has no direct visibility on it. The receiving ground station *B*, on the other hand, is permanently located within the coverage area of *Beta*. However, it cannot receive any data until the relay is activated. When *Alpha* simultaneously establishes line-of-sight with both *A* and *Beta*, the secure message routing is performed through the following chain:

$$A \rightarrow \text{Alpha} \rightarrow \text{Beta} \rightarrow B$$

The relay protocol requires each step to be tracked on-chain by committing a cryptographic hash to the blockchain, ensuring immutability, time verifiability, and economic incentives for relay nodes, even in multi-operator or trustless contexts.

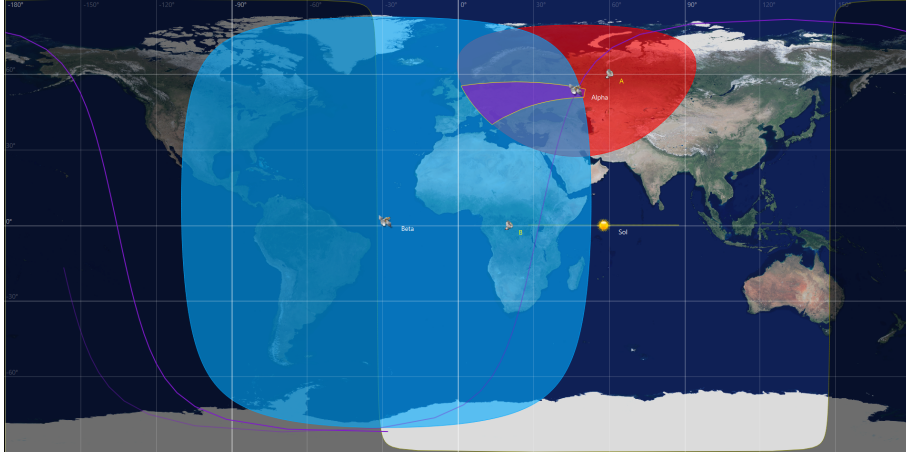


Figure 3.4: Communication architecture in LEO-GEO scenario

- **IoT in remote areas:** A low-power sensor *A* uses a LEO satellite named *Alpha* to send critical data to a gateway *B*, which is located in a region covered by a different satellite constellation called *Beta*. Since *A* has no direct visibility to the nodes of *Beta*, relaying through *Alpha* speeds up the transmission and, thanks to blockchain technology, ensures data integrity and provides economic incentives for the nodes involved in the relay.

3.3 Technical Implementation

The previous sections have outlined the high-level functioning of the secure relay protocol. This section delves into the details of its implementation: it will analyze the roles of each on-chain and off-chain actors, the structure and internal logic of the smart contract, the pipeline for generating Zero-Knowledge Proofs using ZoKrates

(executed within Docker container), and the use of MetaMask and Remix for off-chain signing and contract deployment.

3.3.1 Off-Chain Flow

The off-chain flow represents the portion of the protocol that remains invisible to the blockchain, yet is essential to ensure the correct transmission of the encrypted message among the actors involved: *A*, *Alpha*, *Beta*, and finally *B*. This flow encompasses operations such as encryption, data transmission, cryptographic proof generation, and digital signing. The following subsections describe each phase in detail.

Message Encryption (A)

Ground station A serves as the origin point of the message *M*, which is intended for ground station B. Since the content of the message may be sensitive, it is essential to employ an encryption mechanism that ensures end-to-end confidentiality, guaranteeing that only the legitimate recipient (B) can access its contents.

To achieve this goal, A employs a hybrid encryption scheme known as *ECIES* (Elliptic Curve Integrated Encryption Scheme), which combines the security of asymmetric cryptography with the efficiency of symmetric encryption. ECIES is not a single cryptographic algorithm, but rather a framework that can be implemented using various elliptic curves, key derivation functions, and encryption algorithms. For instance, it is possible to use curves such as *secp256k1* for public key computation, in combination with key derivation functions like *PBKDF2* and symmetric ciphers such as *AES-GCM* or *ChaCha20-Poly1305*. [41]

In all cases, the underlying idea remains the same and can be summarized as follows:

- The input to the ECIES encryption function consists of the recipient's public key and a plaintext message.
- The output is a packet containing: the sender's ephemeral public key, the encrypted message, the parameters used for symmetric encryption, and an authentication tag (MAC).

$$ECIES_encrypt(pubKey, plaintextMessage) \Rightarrow \{cipherTextPublicKey, encryptedMessage, authTag\}$$

The implementation adopted in this project, however, is based on:

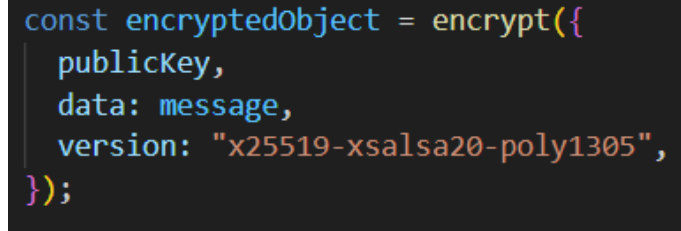
- *X25519* key exchange algorithm, an efficient and secure variant of the elliptic-curve *Diffie-Hellman* protocol, is used to compute a shared secret between the sender and the recipient.
- A key derivation function (KDF), based on *HKDF-SHA256*, which generates two separate keys: one for message encryption and one for authentication (MAC).
- The authenticated symmetric cipher *XSalsa20-Poly1305*, which simultaneously ensures the confidentiality, integrity, and authenticity of the encrypted message.

In the application context ground station B retrieved its own encryption public key using the standard *eth_getEncryptionPublicKey* method exposed by MetaMask.

This encryption key, used exclusively for the ECIES-based encryption process, is not the same as the public key associated with B’s on-chain address. This distinction reflects how, in practice, Ethereum accounts can derive separate keys for encryption and signing operations, ensuring the separation of confidentiality and authentication mechanisms.

Although this interaction is specific to the development environment, it effectively simulates a real-world scenario in which B generates its public key and transmits it to A through a secure out-of-band channel. Once obtained, A uses this key to encrypt a message intended for B, thus initiating the secure relay process.

The encryption process was implemented in a custom JavaScript dApp named *encryptandsign_dapp*, developed specifically for this thesis. The core logic is encapsulated in the component *EncryptAndSign.js*, as shown in Figure 3.5. The full source code is publicly available on GitHub at https://github.com/SaraSim98/encryptandsign_dapp.



```
const encryptedObject = encrypt({
  publicKey,
  data: message,
  version: "x25519-xsalsa20-poly1305",
});
```

Figure 3.5: Core encryption logic in *EncryptAndSign.js*

This call returns an encrypted object containing:

- The *ephemeral public key* generated by A (*ephemPublicKey*)
- The cryptographic nonce used for encryption (*nonce*)
- The encrypted message (*ciphertext*), encoded in Base64
- The authentication tag (included within the *ciphertext*)

Internally, the *encrypt()* function performs the following steps:

1. Generation of an ephemeral key pair by A
2. Computation of the shared secret using *ECDH* (Elliptic Curve Diffie-Hellman) between A’s ephemeral key and B’s public key.
3. Application of the *KDF* to the shared secret to derive the symmetric key k_{ENC} e k_{MAC}
4. Encryption of the message M using *XSalsa20* and computation of the authentication tag with Poly1305

The final output is a packet structured according to the following format:

$$\text{ECIES_encrypt}(\text{pubKey}, M) \Rightarrow \{\text{ephemPublicKey}, \text{ciphertext}, \text{nonce}, \text{version}\}$$

All these elements are necessary for the recipient B to reconstruct the same symmetric key and correctly decrypt the message. Once encrypted, the ciphertext is ready

to be transmitted to a node in the *Alpha* constellation for forwarding to ground station *B*.

Transmission Channel from A to Alpha

The transmission of the ciphertext from ground station *A* to the *Alpha* constellation primarily occurs through radio frequency (RF) links, which remain the most established means of communication in satellite systems. The choice of frequency band depends on the constellation’s architecture and regulatory constraints. In particular, the UHF and S bands are commonly used for communication with LEO/MEO satellites, due to their strong atmospheric penetration and reliable performance even under adverse weather conditions.

When higher bandwidth is required, higher-frequency bands such as X or Ka can be employed, offering greater throughput at the cost of increased sensitivity to atmospheric attenuation. Looking forward, in more technologically advanced contexts, optical links may also be adopted, as they provide very high transmission capacity, albeit under more stringent operational conditions. It is therefore clear that the choice of frequency band directly affects latency, throughput, and transmission stability.

Alpha Node’s Role: Forwarding and Hash Registration

Node *Alpha* is responsible for the initial relay step in the forwarding process. Upon receiving the *ciphertext* from *A*, it forwards it to *Beta* and computes its hash using the *Keccak-256* function, selected for its efficiency and compatibility with the Ethereum ecosystem. The resulting hash, obtained as

$$rawHash = keccak256(ciphertext)$$

will then be recorded on the blockchain, serving as immutable proof of the successful forwarding. The hash computation is performed in *EncryptAndSign.js*.

Transmission Channel from Alpha to Beta

Since both Alpha and Beta are satellites, the transmission occurs through inter-satellite links (ISLs). This type of connection enables the implementation of relay functionality directly in space, enhancing coverage and reducing the overall system latency. The proposed protocol is agnostic to the specific link technology used and can operate with:

- **RF-based ISLs:** Suitable for both LEO–LEO and LEO–GEO configurations, especially in scenarios where greater robustness or ease of integration is required.
- **Optical (laser) ISLs:** applicable in the same configurations, they offer advantages in terms of bandwidth and latency but require high-precision alignment between terminals and careful management of orbital dynamics—particularly in high relative velocity LEO–LEO scenarios.

The choice of technology therefore depends on the type of orbital architecture and the operational requirements, but the protocol remains compatible with both approaches.

Beta's Role: Zero-Knowledge Proof Generation and Message Relay

After receiving the encrypted message from *Alpha*, the *Beta* node forwards it to ground station *B*. To enable formal verification of the relay operation, in compliance with the timing rules defined by the protocol, Beta generates a *zero-knowledge proof* (ZKP). This cryptographic proof allows the validity of a statement to be confirmed *without revealing any of its confidential elements*.

The proof is constructed from a *zkSNARK* circuit, shown in Figure 3.6, which takes as input:

- **Private values**, accessible only to node *Beta*:
 - *timestamp*: a time marker representing the moment at which the forwarding took place;
 - *method*: an identifier specifying the channel or protocol used for the transmission.
- **Public values**, visible also to the verifier:
 - *rawHash*: the hash of the encrypted message, computed by *Alpha* and already recorded on the blockchain, associated with *Beta*;
 - *blockTimestamp*: the timestamp of the Ethereum block containing the latest transaction sent by *Alpha*;
 - *delta*: the maximum allowed time tolerance, which defines the window within which the message relay must occur.

```
import "hashes/poseidon/poseidon" as poseidon;

def main(
  private field timestamp,
  private field method,
  field rawHash,
  field zkpHash,
  field blockTimestamp,
  field delta
)
{
  assert(timestamp >= blockTimestamp);
  field upperBound = blockTimestamp + delta;
  assert(timestamp < upperBound);
  field[3] preimage = [rawHash, timestamp, method];
  field h = poseidon(preimage);
  assert(h == zkpHash);
}
```

Figure 3.6: zkSNARK circuit

In this protocol, the parameter *timestamp* represents a private value selected by Beta, indicating the moment the node claims to have forwarded the message to B. The Zero-Knowledge Proof generated by Beta enforces that this private *timestamp* falls within a valid time window relative to the *blockTimestamp*, ensuring that the declared relay operation is consistent with what has already been recorded on the blockchain. The *blockTimestamp* refers to the timestamp of the Ethereum block containing the *registerRawHash* transaction submitted by Alpha, in which the message's *rawHash* is recorded and linked to Beta's address.

This registration occurs through the emission of a specific smart contract event named *RawRegistered*, which is triggered during the execution of the *registerRawHash* function. When Alpha submits a message hash, the contract emits a *RawRegistered* event, which includes three indexed parameters: the *rawHashFull* (i.e., the hash of the encrypted message), the address of the *Beta* node, and the *escrow* amount committed for the relay.

These events are not stored in contract storage but are appended to the blockchain's logs, making them publicly accessible and efficiently retrievable by any off-chain client, such as decentralized applications or verification tools. In this protocol, the event allows an off-chain application to retrieve the timestamp of the block in which the registration occurred.

This structure allows for the cryptographically reliable simulation of a temporal constraint on the relay action. The system thus ensures temporal consistency between off-chain declared events and on-chain timestamps, while preserving the confidentiality of operational details.

In practice, the *BlockTimestamp* is retrieved off-chain by the custom JavaScript component *BlockTimestamp.js*, which is part of the dedicated web application *timestamp_dapp* developed for this thesis. This module queries the *RawRegistered* event and extracts the associated timestamp by accessing the corresponding block through the *getBlock* method of *ethers.js* (Figure 3.7). The full source code is publicly available on GitHub at: https://github.com/SaraSim98/timestamp_dapp.

This mechanism enables the system to bind the validity of the relay to a specific and verifiable blockchain state.

```
const block = await provider.getBlock(lastEvent.blockNumber);
const ts = block.timestamp;
```

Figure 3.7: Off-chain retrieval of block timestamp in *BlockTimestamp.js*

In the circuit, the *Poseidon* hash function is applied to the array *[rawHash, timestamp, method]*. The following constraints are applied:

1. The value of *timestamp* must fall within the interval $(blockTimestamp, blockTimestamp + \delta)$, ensuring that the relay operation occurred *after the hash was recorded on the blockchain* and *within an acceptable time window*, as required by the protocol.
2. The result of the Poseidon hash must match the value *zkpHash*, a public commitment published by *Beta*.

This scheme enables *Beta* to prove knowledge of a pair of values (*timestamp*, *method*) which, when combined with the public message hash *rawHash*, produce exactly the Poseidon commitment *zkpHash*.

The proof certifies *the consistency between private data and public constraints*, providing a cryptographic guarantee of the node’s correct behavior without compromising operational confidentiality. In addition to the ZKP, Beta also retains a digital signature provided by B as confirmation of message receipt. Both the ZKP and the signature will later be used to securely and transparently validate the successful relay and reception of the message directly on-chain.

Decryption and Signature (B)

The transmission of the ciphertext from node Beta to ground station B occurs via a direct satellite-to-ground link. The technological considerations regarding the transmission medium are similar to those discussed for ground-to-satellite links: both optical and radio-frequency links can be employed, depending on the architecture’s specifications and the desired performance. Ground station B receives the ciphertext and decrypts it using its private key, thus obtaining the plaintext:

$$plaintext = Decrypt(ciphertext, privKey_B)$$

Then, it recalculates the hash of the plaintext using the same function previously employed by Alpha, obtaining:

$$rawHash = keccak256(ciphertext)$$

To prove the successful reception of the message, and to ensure the authenticity of the sender, the integrity of the data, and non-repudiation, B digitally signs the freshly computed hash using their private key. The signature is generated using the ECDSA (Elliptic Curve Digital Signature Algorithm) standard, based on the *secp256k1* elliptic curve—commonly used in Ethereum.

In the context of this thesis, the signature is computed through the custom JavaScript dApp *encryptandsign_dapp*, previously introduced in the context of the encryption process. The signing logic is encapsulated in the *EncryptAndSign.js* component, which also handles the encryption functionality described earlier. The process is handled programmatically using JavaScript, as shown in the following snippet:

```
const address = await requestAccount();
const provider = new BrowserProvider(window.ethereum);
const signer = await provider.getSigner();
const flatSig = await signer.signMessage(getBytes(hashHex));
const sigObj = Signature.from(flatSig);
```

Figure 3.8: ECDSA signature generation via MetaMask in *EncryptAndSign.js*.

Here, *hashHex* represents the hexadecimal encoding of the Keccak-256 hash of the ciphertext. The *signer* object corresponds to ground station B’s on-chain identity

and is obtained via MetaMask. The method *signMessage* produces a flat signature *flatSig*, which is then parsed into a structured object *sigObj* using the Ethereum library. This object exposes the canonical ECDSA triplet:

- *r*: represents the x-coordinate of a point on the elliptic curve, derived from a random number *k* selected during the signature generation process.
- *s*: binds the message hash to the signer’s private key in a non-reversible manner.
- *v*: an auxiliary value that allows for the unique determination of the public key associated with the signer, useful for recovering the corresponding Ethereum address.

This tuple is sent to Beta, which will use these values to validate the signature on-chain and, consequently, to confirm the successful receipt of the message by B.

3.3.2 On-Chain Flow

The on-chain flow constitutes the public, immutable, and transparent component of the protocol, where all cryptographic operations and interactions between the various nodes are permanently recorded on the blockchain. This phase ensures the traceability of the relay, the formal validation of the proofs generated off-chain, and the secure distribution of economic incentives. All on-chain logic is implemented within the smart contract *MessageRelayWithZkpAndSignature*, written in Solidity and deployed, in the context of this project, on the Ethereum Sepolia testnet. The complete source code of the contract, including the contract *verifier.sol* generated via ZoKrates, is publicly available at <https://github.com/SaraSim98/smart-contract>. This contract serves as a decentralized arbiter that coordinates three main functionalities:

- to record the ciphertext hash on the blockchain and immutably associate it with an authorized Beta node
- to verify the validity of the Zero-Knowledge Proof generated by node *Beta* and the ECDSA digital signature provided by ground station *B*.
- to manage an economic incentive mechanism through an Ether-based escrow system

Contract Initialization

The contract is initialized through the special *constructor* function shown in Figure 3.9, which is executed only once at the time of deployment on the blockchain. Its purpose is to define the system’s initial parameters. Specifically, it performs three main tasks:

- Registration of the deployer’s identity: the address that deploys the contract is stored in the *alpha* variable using the *msg.sender* keyword. In this way, the Alpha node is assigned a privileged role within the protocol.
- Configuration of ground station B: the address of the ground station is passed as the *_groundStationB* parameter and assigned to the *groundStationB* variable. This address will later be used to verify the ECDSA digital signature from *B* on the hash of the received message.

- Whitelist of Beta nodes: the `_betas` array of addresses, passed as input to the function, is processed to validate each node (address) and register it within the `isBeta` data structure. This mapping acts as a whitelist of authorized Beta nodes allowed to perform data relay.

```

constructor(address _groundStationB, address[] memory _betas) {
    alpha = msg.sender;
    groundStationB = _groundStationB;
    for (uint256 i = 0; i < _betas.length; i++) {
        address b = _betas[i];
        require(b != address(0) && !isBeta[b], "Beta is invalid");
        isBeta[b] = true;
        betas.push(b);
    }
}

```

Figure 3.9: *Constructor* function of the smart contract *MessageRelayWithZkpAndSignatureFunzione*

Registration of the `rawHash` on the Blockchain and Escrow Deposit

Following the contract initialization, the first on-chain phase of the relay process is executed through the `registerRawHash` function, shown in Figure 3.10. This function is called exclusively by node Alpha, the only entity authorized to do so, and it performs three main actions:

- Register on the blockchain the `rawHash` of the `ciphertext` computed off-chain by Alpha. This operation immutably records the hash on-chain, making it publicly verifiable at a later stage.
- Associate the message hash with a specific Beta node selected for the relay from the `isBeta` whitelist defined at deployment. In addition to the hash, Alpha also specifies the Ethereum address of the Beta node `beta` as an input to the function. This association is mapped on-chain via the `byRawHash` data structure, effectively binding the relay of that particular message to a designated Beta node.
- Alpha deposits a certain amount of Ether (`msg.value`), which is stored in the `escrow` mapping linked to the `rawHash`. The deposited amount is not immediately accessible but remains locked within the smart contract as an economic guarantee. The release of these funds will only occur after the successful validation of the correct message forwarding by the Beta node. This validation is performed through the `proveAndRelease` function, which is invoked by Beta and executes all the protocol's required checks. Only if these checks are passed successfully, the previously locked amount is released and credited to Beta's internal balance.

The `registerRawHash` function concludes by emitting the `RawRegistered` event. This event serves to publicly notify the registration of a new `rawHash`, specifying both the Beta node assigned to perform the relay and the amount deposited in escrow. Although this data is already stored in the internal state of the smart contract, the event provides an optimized and secure channel for off-chain retrieval.

In this project, the event is leveraged by a web interface developed in React, which connects to the Sepolia testnet via the *ethers.js* library. This interface listens for events emitted by the smart contract and displays data related to the most recent event. In particular, the key piece of information is the timestamp of the block in which the last event was triggered, which is later used for the creation of the off-chain ZKP. This approach enables direct access to the relevant data without the need to query the internal state of the smart contract.

```
function registerRawHash(uint256 rawHash, address beta) external payable {
    require(msg.sender == alpha, "Only Alpha can call this function");
    require(isBeta[beta], "Beta address is not whitelisted");
    require(byRawHash[rawHash] == address(0), "rawHashFull already registered");
    require(msg.value > 0, "Must deposit a positive amount");
    byRawHash[rawHash] = beta;
    escrow[rawHash] = msg.value;
    emit RawRegistered(rawHash, beta, msg.value);
}
```

Figure 3.10: *registerRawHash* function of the *MessageRelayWithZkpAndSignature* smart contract.

Verification of the Relay and Escrow Release

The second phase of the on-chain flow is implemented through the *proveAndRelease* function, shown in Figure 3.11 and Figure 3.12. This function represents the core of the on-chain verification process for secure message relay. It can only be invoked by a previously authorized Beta node (present in the whitelist) and performs a sequence of essential cryptographic checks to ensure the integrity of the relay. The following list outlines the main operations carried out by the function:

- **Caller Authentication:** The function first verifies that the *msg.sender* address belongs to the *isBeta* whitelist and that the input *rawHash* is indeed associated with that node by checking the *byRawHash* mapping.
- **Unique Registration of the *zkpHash*:** The function ensures that the *zkpHash* computed off-chain by the Beta node has not been previously used. The *byZkpHash* mapping allows each *zkpHash* to be uniquely registered to the node that generated it. This prevents replay attacks by ensuring that the same proof cannot be reused to release funds multiple times.
- **Verification of the Zero-Knowledge Proof:** The *verifyTx* function (Figure 3.13), inherited from the *verifier.sol* contract generated by ZoKrates, is invoked. This function takes as input the proof components *a*, *b*, *c*, along with the two public values *rawHash* and *zkpHash*. Specifically, *rawHash* is reduced modulo a predefined prime number *FIELD_PRIME* to ensure it fits within the finite field required by the zkSNARK circuit generated via ZoKrates. The function returns *true* only if the proof provided by Beta is valid with respect to the off-chain defined zk-SNARK circuit; otherwise, it returns *false* and the transaction is reverted.
- **ECDSA Signature Verification:** After the Zero-Knowledge Proof is validated, the digital signature from ground station B is verified. This signature is generated off-chain. Specifically, the contract reconstructs the signed hash *aggHash*

by applying the keccak256 function to the concatenation of the Ethereum standard prefix `emph"\x19Ethereum Signed Message:\n32"` and the *rawHash*. At this point, the *ecrecover* function is used to verify that *B*'s ECDSA signature (*v*, *r*, *s*) was indeed produced by the private key associated with the ground station *B*'s Ethereum address. This step ensures that ground station *B* actually received the message, effectively preventing spoofing attempts by unauthorized nodes.

- **Escrow Release:** Upon successful validation of both the Zero-Knowledge Proof and the digital signature, the funds previously deposited by Alpha and locked to the *rawHash* are released. The corresponding amount is retrieved from the *escrow* mapping and credited to the internal balance of the Beta node by updating the *balances* structure. This mechanism prevents immediate withdrawal and enables precise tracking of the credits accrued by each node, reducing the risk of irreversible loss of escrow funds in case of errors.

The function concludes with the emission of the *zkpAndSigRelay* event, which publicly records on the blockchain the successful validation of the relay.

```
function proveAndRelease(  infinite gas
    uint256 rawHash,
    uint256 zkpHash,
    uint256 blocktimestamp,
    uint256 delta,
    uint256[2] calldata a,
    uint256[2][2] calldata b,
    uint256[2] calldata c,
    uint8 v,
    bytes32 r,
    bytes32 s
) external {
    address beta = msg.sender;
    require(isBeta[beta], "Beta is not authorized");
    require(byRawHash[rawHash] == beta, "Invalid Beta for specified rawHash");
    require(byZkpHash[zkpHash] == address(0), "zkpHash already used");
    byZkpHash[zkpHash] = beta;
    require(
        verifyTx(
            Proof({
                a: Pairing.G1Point(a[0], a[1]),
                b: Pairing.G2Point([b[0][0], b[0][1]], [b[1][0], b[1][1]]),
                c: Pairing.G1Point(c[0], c[1])
            }),
            [rawHash % FIELD_PRIME, zkpHash, blocktimestamp, delta ]
        ),
        "invalid proof"
    );
};
```

Figure 3.11: *proveAndRelease* function - verification of the beta node and the ZKP of the *MessageRelayWithZkpAndSignature* smart contract

```


bytes32 aggHash = keccak256(
    abi.encodePacked("\x19Ethereum Signed Message:\n32", bytes32(rawHash))
);
require(ecrecover(aggHash, v, r, s) == groundStationB, "Invalid signature from groundStationB");
uint256 amount = escrow[rawHash];
require(amount > 0, "No escrowed amount available");
escrow[rawHash] = 0;
balances[beta] += amount;

emit ZkpAndSigRelay(rawHash, zkpHash, beta, amount);
}

```

Figure 3.12: *proveAndRelease* function – Signature Verification and Escrow Release of the smart contract *MessageRelayWithZkpAndSignature*.

```

function verifyTx(  undefined gas
    Proof memory proof, uint[4] memory input
) public view returns (bool r) {
    uint[] memory inputValues = new uint[](4);

    for(uint i = 0; i < input.length; i++){
        inputValues[i] = input[i];
    }
    if (verify(inputValues, proof) == 0) {
        return true;
    } else {
        return false;
    }
}
}

```

Figure 3.13: Function *verifyTx* of the contract *verifier.sol*.

Funds Withdrawal (B)

The final stage of the off-chain flow involves the withdrawal of funds by the Beta node that successfully performed the relay. This operation is implemented through the *withdraw* function (Figure 3.14). Specifically, the function

- Retrieves the funds associated with the node from the *balances* mapping.
- Checks that the balance is positive.
- Resets the node’s internal balance before performing the transfer to prevent *reentrancy* attacks.
- Executes the transfer of Ether to the Beta node’s address.

Finally, the *Withdrawal* event is emitted, recording the node’s address and the amount withdrawn. This procedure completes the economic flow of the protocol,

ensuring that funds are released only in the presence of relays that have been cryptographically verified and approved.

```
function withdraw() external {  ⚙ infinite gas
    address beta = msg.sender;
    uint256 amount = balances[beta];
    require(amount > 0, "No balance available");

    balances[beta] = 0;

    (bool success, ) = payable(beta).call{value: amount}("");
    require(success, "Withdrawal failed");

    emit Withdrawal(beta, amount);
}
```

Figure 3.14: *withdraw* function of the *MessageRelayWithZkpAndSignature* smart contract.

3.3.3 Practical Implementation of the Protocol Using Development Tools

In this protocol, integration between the technologies described in Section 3.2 and the *React* framework was essential to develop a verifiable and consistent operational flow, both on-chain and off-chain. The goal was to implement a secure relay mechanism for forwarding messages between nodes, leveraging a simulated but realistic Ethereum environment such as Sepolia. The on-chain logic was implemented through the *MessageRelayWithZkpAndSignature* smart contract written in Solidity using the Remix IDE. Remix, in addition to providing integrated tools for writing and compiling Solidity code, allows direct connection to MetaMask via the *Injected Provider* option. This is shown in Figure 3.15.

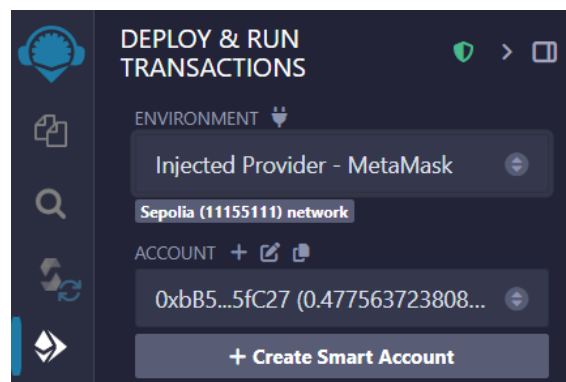


Figure 3.15: Option *InjectedProvider* on Remix

In this way, the contract was deployed directly on the Sepolia testnet using the account selected in the *Metamask* wallet, which signed and authorized the transaction. Each time a contract function is invoked, a JSON-RPC request is generated, This

request is then handled by MetaMask, which intercepts it and displays a signature confirmation window to the user. Here, the user can review the transaction details before approving: the contract address, the parameters passed, the estimated gas, and the cost. *fig.* This pop-up ensures control and transparency, as no transaction is ever sent to the network without the user's explicit consent as shown in Figure 3.16.

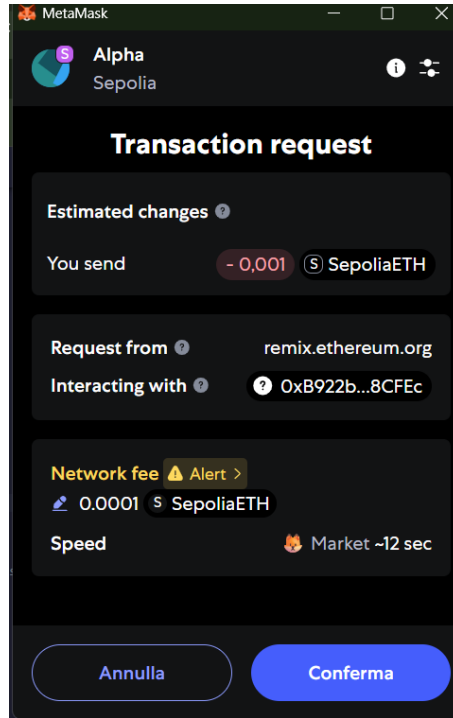


Figure 3.16: MetaMask pop-up for transaction confirmation.

Upon confirmation, MetaMask signs the transaction with the private key of the selected account. The transaction is then forwarded to the Ethereum network via an RPC provider on the Sepolia testnet (in this case, Infura), which serves as the externally accessible Ethereum node. Once the node receives the transaction, it is added to the mempool and subjected to validation and consensus. After confirmation, the transaction is recorded in a blockchain block and becomes visible on Sepolia through block explorers like Etherscan. (Figure 3.17)

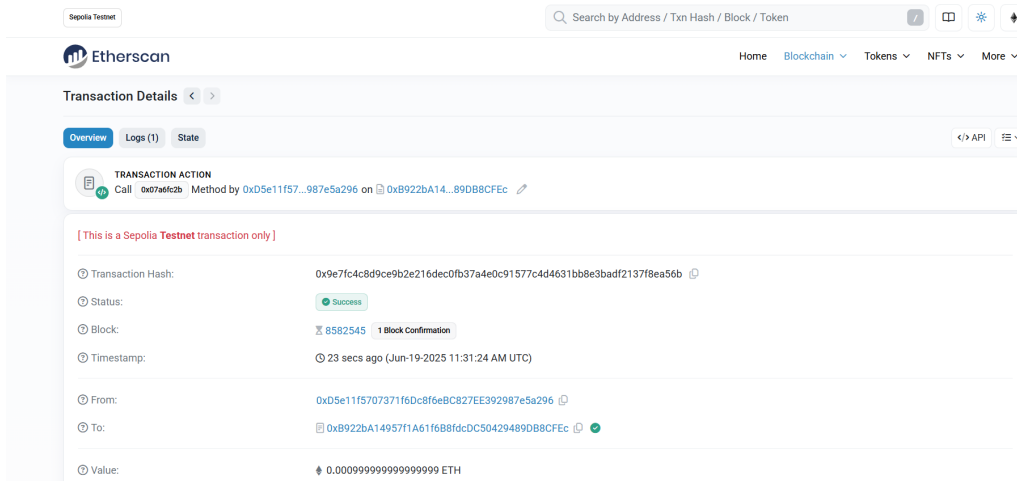


Figure 3.17: Transaction details on Etherscan

On the client side, two React-based web applications were developed to support the off-chain execution of the protocol. The first, *encryptandsign_dapp*, simulates the behavior of the nodes involved in the relay and handles encryption, Keccak-256 hashing, and ECDSA signature generation through MetaMask. It interacts with the Ethereum provider via the *window.ethereum* object and leverages the *ethers.js* library to perform cryptographic operations securely in the browser.

The second application, *timestamp_dapp*, is designed to retrieve, directly from the blockchain, the block timestamp associated with the *RawRegistered* event emitted by the Alpha node. This timestamp is then processed and displayed in a human-readable format, enabling the temporal verification step required by the Zero-Knowledge Proof.

4 Comparative Analysis

This chapter presents a comparative analysis between the protocol implemented in this thesis and the protocol described in the *SpaceCoin* white paper. [42] Although both architectures share the goal of incentivizing message forwarding in a trustless manner within decentralized environments, they differ in terms of application domain, architecture, communication flow, verification mechanisms, and design objectives. The analysis addresses both implementation and conceptual aspects, offering a structured and in-depth comparison that helps position the contribution of this thesis within the context of a broader, existing project such as *SpaceCoin*.

4.1 Spacecoin

SpaceCoin is a project born from the ambition to provide global Internet access through a decentralized infrastructure. The proposed solution is based on a network of low Earth orbit (LEO) satellites, integrated with a public blockchain that manages economic incentives for the nodes responsible for data transmission—referred to as transmitters—in a reliable manner. The goal is to decentralize global connectivity infrastructure, overcoming the limitations of traditional centralized networks, such as censorship and lack of access in remote areas.

The operational flow of the protocol is based on the interaction of three main actors (Figure 4.1):

- *Requester*: the entity that aims to obtain information and initiates the request
- *Trasmitter*: the node responsible for transmitting the requested data to the Requester
- *Smart Contract*: the blockchain component that autonomously mediates interactions and manages funds.

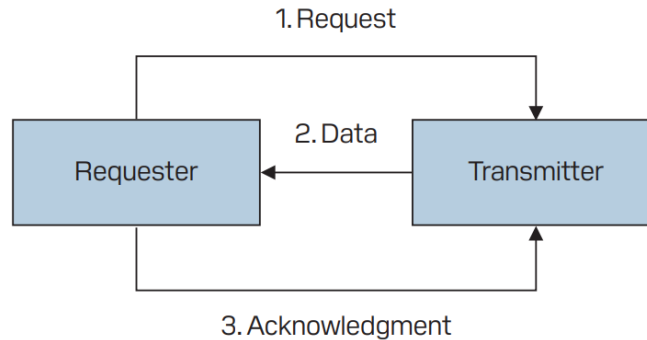


Figure 4.1: Data exchange flow in the Spacecoin protocol

The process begins with the Requester locking a sum of money in the smart contract, specifying the parameters of the request. An eligible *Transmitter* then takes charge

of the operation, collects the data, and transmits it to the Requester. Once the *Requester* receives the data it generates a digital signature (ACK) using its private key. This signature serves as a cryptographic confirmation of receipt. The *Transmitter* sends the ACK to the smart contract, which verifies the validity of the signature and its correspondence with the original request. If the verification is successful, the smart contract releases the funds to the Transmitter as a reward for the service provided. This scheme introduces a post-verification payment logic, in which the Requester’s signature serves as unequivocal proof that the data has been received. The blockchain ensures that all interactions are immutable, traceable, and free from external arbitration, guaranteeing transparency among the parties involved.

The SpaceCoin roadmap is structured into three main development phases, each with progressively more ambitious objectives:

- *Short Term – Space-to-Earth Connectivity*: establish connectivity between space and Earth to ensure a stable, resilient, and global connection.
- *Medium Term – Space-to-Space Connectivity*: extend the system’s capabilities to support direct communication between satellite nodes, introducing advanced services for the transmission and security of aerospace data.
- *Long Term – Interplanetary Communication*: establish a global standard for secure data transmission even in interplanetary environments, expanding the paradigm beyond Earth’s orbit.

The protocol proposed in this thesis aligns with the objectives outlined by *SpaceCoin*, particularly those related to the Medium-Term phase. In fact, the designed solution implements advanced cryptographic mechanisms such as Zero-Knowledge Proofs and the combined use of ECDSA signatures, anticipating the needs for authentication, security, and verifiability that arise in future scenarios of inter-satellite communication. At the same time, thanks to its well-defined structure, it is already compatible with the reliability and transparency requirements necessary for space-to-Earth connectivity.

From this perspective, the protocol can be considered an effective *proof of concept*: it demonstrates the feasibility of an incentivized and verifiable infrastructure for message forwarding among distributed nodes, and provides a model adaptable to the future developments outlined in the SpaceCoin roadmap. The thesis thus offers a concrete experimental contribution which, although still in a validation phase, reflects the technological ambitions of SpaceCoin.

4.1.1 Comparison with the Proposed Protocol

The following paragraph highlights the main differences between the two protocols:

- **Objectives**: SpaceCoin aims to ensure global and decentralized Internet access through LEO satellites. The proposed protocol focuses on the secure and verifiable implementation of message forwarding via satellites, offering a technical solution for distributed environments.
- **Actors Involved and Operational Model**: SpaceCoin involves two main actors, the Requester and the Transmitter, who interact through a smart contract. The proposed protocol adopts a more articulated flow, involving A,

Alpha, Beta, and B. This structure allows for a more granular management of roles, making the system suitable for multi-hop flows and more refined verifications.

- **Verification and Privacy Protection Mechanisms:** In SpaceCoin, successful transmission is confirmed through a signature from the Requester. In the thesis protocol, verification is twofold: it combines the recipient’s signature with a Zero-Knowledge Proof generated by the relay node. This architecture not only enhances security but also protects the privacy of participants by concealing sensitive information such as timing, message path, or the identity of the relay. The ZKP thus serves a dual purpose: ensuring the integrity of the communication flow and safeguarding confidentiality.
- **Incentive Management:** Both protocols adopt an escrow model: funds are locked in the smart contract at the beginning and released only after successful verification. However, while SpaceCoin adopts a binary and direct model, the proposed protocol binds escrow to additional cryptographic conditions, making the system more secure.

The analysis has shown that, although the two protocols operate in different contexts and at different levels, they share the common goal of building reliable, verifiable, and transparent communication infrastructures. SpaceCoin offers a broad, long-term vision aimed at providing global Internet access through a shared infrastructure coordinated by the blockchain. The thesis protocol, on the other hand, represents a concrete and targeted response to a specific issue: the secure transmission of messages between distributed nodes.

Although differing in scale, the two approaches can be seen as complementary. The protocol presented in this thesis serves as a practical building block that aligns with the broader vision of SpaceCoin, addressing some of its anticipated future needs. As a *proof of concept*, it demonstrates the technical feasibility of a secure, verifiable multi-hop relay system, and lays the groundwork for potential integration into larger-scale architectures. In the long term, solutions of this kind could become key enablers of reliability, security, and auditability in next-generation decentralized space networks.

4.2 Other Relevant Architectures

SpaceChain

SpaceChain is an open-source platform that integrates blockchain nodes into satellite infrastructure, enabling decentralized services to be executed directly in orbit. Unlike *SpaceCoin*, which focuses on incentivized message forwarding to support decentralized Internet access, *SpaceChain* provides a more general-purpose environment for executing smart contracts, signing transactions, and securely storing data in space.

The system deploys satellites equipped with Ethereum-compatible blockchain nodes capable of validating transactions and hosting decentralized applications (dApps). Security is ensured through multisignature protocols and encrypted communication with ground-based clients.

Among its key use cases are:

- Offline crypto transactions via satellite
- Storage of private keys in orbit
- Execution of smart contracts for Earth Observation or IoT applications

Although *SpaceChain* does not explicitly target incentivized relay between satellites, it shares with the proposed protocol the foundational goal of enabling secure and trustless interactions in decentralized infrastructures. In this sense, the two systems are complementary: the protocol developed in this thesis could theoretically operate on top of a *SpaceChain* node, using its infrastructure as a secure execution environment for relaying proofs and transactions. [43]

Blockstream Satellite

Blockstream Satellite is an initiative developed by Blockstream that enables users to receive the Bitcoin blockchain via satellite, removing the need for a traditional internet connection to stay synchronized with the network. The service continuously broadcasts Bitcoin blocks through a global network of geostationary satellites, making blockchain access possible even in remote, rural, or censorship-prone regions. While it operates as a unidirectional system that does not support the direct submission of transactions, the project stands as a compelling example of how space infrastructure can be integrated with blockchain technology. Although it differs significantly from the protocol proposed in this thesis—which focuses on multi-hop relay and on-chain cryptographic verification—Blockstream Satellite nonetheless demonstrates how decentralized technologies and satellite-based systems can be combined to enhance resilience and accessibility. [44]

5 Conclusions

5.1 Achieved Results

The work presented in this thesis fits within the rapidly evolving field of secure and autonomous space communications, proposing a decentralized protocol for the reliable relay of encrypted messages between satellite constellations operated by independent entities. Starting from an analysis of the main limitations of traditional communication models, a solution was designed that leverages blockchain technology to overcome the lack of prior trust between parties, while ensuring traceability, integrity, and economic incentives in an automated manner. The proposed architecture integrates advanced cryptographic techniques (ECIES, ECDSA signatures), zk-SNARKs, and Ethereum smart contracts to manage every phase of the relay process on a public blockchain. In particular, the protocol guarantees that each intermediate node is compensated only after a cryptographic proof of correct message forwarding to the final recipient is successfully verified.

Beyond the technical integration of multiple technologies, particular emphasis was placed on achieving an optimal balance between functionality and efficiency. The smart contract was designed with a deliberately minimal and essential set of functions, avoiding unnecessary state changes, external calls, and redundant verifications. This architectural choice significantly reduced the gas consumption and simplified the auditing of the contract’s logic, resulting in lower operational costs and a smaller attack surface—key aspects in blockchain-based environments where every on-chain computation has a financial impact.

These considerations guided the protocol’s development toward a solution that is not only secure and verifiable, but also cost-effective and practical to deploy. The result is a modular, verifiable, and economically sustainable protocol for secure message forwarding between distributed nodes, capable of operating in adversarial or trustless scenarios without sacrificing performance or traceability.

5.2 Limitations and Future Perspectives

Despite the positive results, the protocol presents some limitations, primarily related to the Ethereum infrastructure:

- *Scalability*: the Ethereum network still suffers from congestion and high gas costs under high traffic, limiting the efficiency and economic viability of the approach in large-scale operational scenarios
- *Intermittent Connectivity*: satellites—especially those in GEO or interplanetary orbits—cannot guarantee continuous connectivity to the Ethereum network.

The protocol developed in this thesis can be interpreted as a first functional prototype, which demonstrates the possibility of integrating security, automation and blockchain in a satellite context. However, the operational challenges associated

with the use of a terrestrial public blockchain such as Ethereum require technological evolution.

In the short term, one possible improvement lies in the adoption of Layer 2 scalability solutions such as optimistic or zero-knowledge rollups, which enable batch processing of multiple transactions off-chain before settling them on the main chain. These techniques drastically reduce gas consumption and increase throughput, making them particularly attractive for protocols like the one proposed in this thesis, where cost efficiency and verifiability are critical. Rollups could also mitigate congestion issues and support higher message relay volumes without compromising security guarantees. [45]

In the long term, more radical innovations may be required. In this regard, the paper “Beyond Earth’s Boundaries: Blockchain-Driven Autonomy for Satellites and Probes” by Thales Alenia Space proposes a completely new paradigm: a blockchain distributed natively in space, capable of supporting autonomous, secure and efficient machine-to-machine (M2M) communications between space entities such as satellites and probes.

This blockchain network solves the problem of isolation, allowing clusters of satellites to operate even in the absence of connection to the global network, through deferred synchronisation and sharing of resources. Among the main innovations are:

- *New consensus mechanism: Proof Of Useful Work*, which replaces the classic *Proof of Stake* or *Proof of Work* with a model that rewards the computational contribution that is truly useful to the mission, such as scientific data processing, image processing or packet compression. This allows the consensus mechanism to be integrated into the operational logic of the mission itself, lowering energy costs and making the system more efficient and sustainable
- *Sharding*: the division of the blockchain into fragments managed in parallel by different subsets of satellites, increasing scalability and reducing latency in environments where direct communication between all nodes is not always possible.
- *Dynamic clustering and adaptive relay mechanisms* that adjust to continuously evolving spatial configurations and the varying connectivity between orbiting nodes. This capability is essential to ensure resilience in scenarios where nodes may operate in isolation for extended periods and later reconnect to the network.

This solution would enable a blockchain infrastructure that is autonomous, resilient, and inherently space-native, capable of supporting new forms of cooperation, commerce, and interplanetary communication. Within this framework, the protocol proposed in this thesis serves as an important experimental foundation for the development of such advanced architectures and represents a significant step toward a future of autonomy and reliability beyond Earth’s boundaries.

Bibliography

- [1] J. Zhang, C. McInnes, and M. Macdonald, *LEO Mega Constellations: Review of Development, Impact, Surveillance, and Governance*, Space Policy, vol. 61, Elsevier, 2022, [Online]. Available: <https://spj.science.org/doi/10.34133/2022/9865174>
- [2] S. Mansfield, *Private Sector Innovation and Its Impact on the Space Industry*, SpaceDaily, [Online]. Available: https://www.spacedaily.com/reports/Private_Sector_Innovation_and_Its_Impact_on_the_Space_Industry_999.html
- [3] Knowledge for Policy, *Space becomes a new area of expansion*, Knowledge for Policy (European Commission), 2024, [Online]. Available: https://knowledge4policy.ec.europa.eu/foresight/space-becomes-new-area-expansion_en
- [4] Festo Didactic Staff, *Principles of Satellite Communications: Courseware Sample 86311-F0*, 1st ed., Revision 05/2016, Festo Didactic Ltée/Ltd, Québec, Canada, 2014. ISBN 978-2-89640-417-9.
- [5] New Space Economy, “What are the Components of a Satellite?”, [Online]. Available: <https://newspaceeconomy.ca/2023/04/05/what-are-the-components-of-a-satellite/>
- [6] L. J. Ippolito, Jr., *Satellite Communications Systems Engineering: Atmospheric Effects, Satellite Link Design, and System Performance*, Wiley, 2008.
- [7] PrimaLuceSpace, *frequencybands*, PrimaLuceSpace Blog, Dec. 2024. [Online]. Available: <https://www.primalucespace.com/it/what-radio-frequencies-are-used-for-space-communication/>
- [8] National Aeronautics and Space Administration (NASA), *9.0 Communications*, in *State-of-the-Art of Small Spacecraft Technology*, Chapter 9, NASA SmallSat Institute, 2025. [Online]. Available: <https://www.nasa.gov/smallsat-institute/sst-soa/soa-communications/#9.2>
- [9] H. Kaushal and G. Kaddoum, *Free Space Optical Communication: Challenges and Mitigation Techniques*, Journal of Optical Communications and Networking, vol. 7, no. 11, pp. 1–20.
- [10] H. Wu, J. Ma, P. Guo, Q. Wang and J. Wu, *Secrecy outage probability analysis in a free-space optical system based on partially coherent beams through anisotropic non-Kolmogorov turbulent atmosphere*, Journal of the Optical Society of America B, vol. 39, no. 4, 2022, [Online] Available: <https://opg.optica.org/josab/viewmedia.cfm?uri=josab-39-5-1378&seq=0&html=true>
- [11] Basile, Cataldo, *Appunti di Sicurezza dei Sistemi Informativi*, Corso di Sicurezza dei Sistemi Informativi, Politecnico di Torino, Torino, 2025.

- [12] Libsodium, *XSalsa20 Stream Cipher - Libsodium Documentation*, [Online]. Available: https://libsodium.gitbook.io/doc/advanced/stream_ciphers/xsalsa20
- [13] Fortinet, *Message Authentication Code (MAC) - Definition*, [Online]. Available: <https://www.fortinet.com/resources/cyberglossary/message-authentication-code>
- [14] Xilinx, *Poly1305 - High-Performance MAC Algorithm*, [Online]. Available: https://xilinx.github.io/Vitis_Libraries/security/2019.2/guide_L1/internals/poly1305.html
- [15] SSL.com, *Cos'è la crittografia a curva ellittica (ECC)?*, [Online]. Available: <https://www.ssl.com/it/articolo/cos%27%C3%A8-la-crittografia-a-curva-ellittica-ecc/>
- [16] Encryption Consulting, *What is ECDSA? (Elliptic Curve Digital Signature Algorithm)*, [Online]. Available: <https://www.encryptionconsulting.com/education-center/what-is-ecdsa/>
- [17] Fiveable, *Elliptic Curve Integrated Encryption Scheme (ECIES)*, [Online]. Available: <https://library.fiveable.me/elliptic-curves/unit-3/elliptic-curve-integrated-encryption-scheme-ecies/study-guide/ld7deLPycGFCjeeR>
- [18] How.dev, *What is the Elliptic Curve Diffie-Hellman Algorithm?*, [Online]. Available: <https://how.dev/answers/what-is-the-elliptic-curve-diffie-hellman-algorithm>
- [19] Corporate Finance Institute, *Hash Function - Definition, Purpose, and Examples*, [Online]. Available: [https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/#:~:text=A%20hash%20function%20is%20a%20mathematical%20function%20that%20converts%20any,output%20\(called%20the%20hash\)](https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/#:~:text=A%20hash%20function%20is%20a%20mathematical%20function%20that%20converts%20any,output%20(called%20the%20hash))
- [20] GeeksforGeeks, *Applications of Hashing*, [Online]. Available: <https://www.geeksforgeeks.org/dsa/applications-of-hashing/>
- [21] freeCodeCamp, *MD5 vs SHA-1 vs SHA-2 - Which is the Most Secure Encryption Hash and How to Check Them*, [Online]. Available: <https://www.freecodecamp.org/news/md5-vs-sha-1-vs-sha-2-which-is-the-most-secure-encryption-hash-and-how-to-check-them/>
- [22] GeeksforGeeks, *Difference Between SHA-256 and Keccak-256*, [Online]. Available: <https://www.geeksforgeeks.org/ethical-hacking/difference-between-sha-256-and-keccak-256/>
- [23] Chainlink, *Zero Knowledge Proof (ZKP)*, [Online]. Available: <https://chain.link/education/zero-knowledge-proof-zkp>
- [24] C. Nightingale, *A Full Comparison: What are zk-SNARKs and zk-STARKs?*, Cyfrin Blog, Oct. 1, 2024. [Online]. Available: <https://www.cyfrin.io/blog/a-full-comparison-what-are-zk-snarks-and-zk-starks>

- [25] A. Nitulescu, *A Survey of SNARKs*, [Online]. Available: <https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf>.
- [26] Bit2Me Academy, *What Are zk-STARK?*, [Online]. Available: <https://academy.bit2me.com/en/what-are-zk-stark/>.
- [27] D. Yaga, P. Mell, N. Roby, and K. Scarfone, *Blockchain Technology Overview*, NISTIR 8202, National Institute of Standards and Technology, October 2018. [Online]. Available: <https://doi.org/10.6028/NIST.IR.8202>
- [28] The Linux Foundation, “*Blockchain: Understanding Its Uses and Implications (LFS170) – Course Information*”, Training Portal – The Linux Foundation, [Online]. Available: <https://trainingportal.linuxfoundation.org/learn/course/blockchain-understanding-its-uses-and-implications-lfs170/introduction/course-information>
- [29] Ethereum Foundation, *Gas and Fees*, [Online]. Available: <https://ethereum.org/en/developers/docs/gas/>
- [30] Aviate Labs, “*Blockchain and the New Space Age*”, Aviate Labs – Blog, [Online]. Available: <https://www.aviatelabs.co/post/blockchain-and-the-new-space-age>
- [31] K. L. Jones, *Blockchain in the Space Sector*, Center for Space Policy and Strategy, Game Changer Series, March 2020. [Online]. Available: file:///C:/Users/User/OneDrive/Desktop/blockchain%20x%20tesi/Jones_Blockchain_03052020.pdf
- [32] N. E. Villanueva, *Blockchain Technology Application: Challenges, Limitations and Issues*, Journal of Computational Innovations and Engineering Applications, vol. 5, no. 2, pp. 8–14, 2021.
- [33] D. Shah, *Remix IDE Explained: A Brief Guide*, [Online]. Available: <https://medium.com/@danishshahh22/remix-ide-explained-a-brief-guide-76ee27a411d0>
- [34] F. Eberhardt, *ZoKrates – Scalable Privacy-Preserving Off-Chain Computations*, Technische Universität Berlin, 2018. [Online]. Available: https://www.static.tu.berlin/fileadmin/www/10002238/projekte/2018_eberhardt_ZoKrates.pdf
- [35] Coinmonks, *Under the Hood of zkSNARK: Groth16 Protocol – Part 4*, [Online]. Available: <https://medium.com/coinmonks/under-the-hood-of-zksnark-groth16-protocol-part-4-53667944366f>
- [36] Docker Inc., *Docker Overview*, [Online]. Available: <https://docs.docker.com/get-started/docker-overview/>
- [37] ZoKrates, *Getting Started*, [Online]. Available: <https://zokrates.github.io/gettingstarted.html>
- [38] GetBlock, *What is Sepolia? A Beginner’s Guide to Ethereum Test Networks*, [Online]. Available: <https://getblock.medium.com/what-is-sepolia-a-beginners-guide-to-ethereum-test-networks-866663a26698>

- [39] CoinMarketCap Academy, *What Is MetaMask?*, [Online]. Available: <https://coinmarketcap.com/academy/article/what-is-metamask>
- [40] MetaMask Docs, *Wallet API*, [Online]. Available: <https://docs.metamask.io/wallet/concepts/wallet-api/>
- [41] S. Nakov, *ECIES: Public Key Encryption*, [Online]. Available: <https://cryptobook.nakov.com/asymmetric-key-ciphers/ecies-public-key-encryption>.
- [42] T. Oh, *SPACECOIN: A Decentralized Connectivity Network*, Gluwa Inc., Wilmington (US), 2024, [Online]. Available: <https://docsend.com/view/msj8249wcy9xxvwh>
- [43] SpaceChain Foundation, *SpaceChain Whitepaper v3.1*, SpaceChain Foundation, [Online]. Available: <https://www.spacechain.com/wp-content/uploads/2023/06/SpaceChain-Whitepaper.pdf>
- [44] Blockstream, *Blockstream Satellite*, Blockstream, [Online]. Available: <https://blockstream.github.io/satellite/>,
- [45] Coinbase, *What are Zero-Knowledge Rollups (ZK Rollups)?*, [Online]. Available: <https://www.coinbase.com/it/learn/crypto-glossary/what-are-zero-knowledge-zk-rollups>