Journal of the Brazilian Computer Society, 2026, 32:1, doi: 10.5753/jbcs.2026.5568

A scheme based on Proof-of-Download blockchain for Digital Rights Management and Traitor Detection

João Tito do Nascimento Silva [**University of Brasília** | ***jt.mat@hotmail.com***]
Felipe Z. da N. Costa [**Federal University of Pernambuco** | ***felipe@zimmerle.org***]
João Gondim [**University of Brasília** | ***gondim@unb.br***]

Universidade de Brasília - Prédio da Faculdade de Tecnologia, Bloco E, Campus Darcy Ribeiro - Asa Norte, Brasília - DF, 70910-900, Brazil.

Received: 01 March 2025 • **Accepted:** 21 November 2025 • **Published:** 15 April 2026

Abstract The licensing of digital assets is an important aspect of modern markets like music and movies. In this context, Digital Rights Management (DRM) and Traitor Detection (TD) comprise a set of tools, techniques, and processes that try to ensure control of the use of licensed media. This work contributes to this area by making it possible for DRM and TD to happen in a public and highly decentralized blockchain under a proposed threat model by developing new cryptographic techniques, which, as shown by the results obtained, are very efficient without requiring expensive hardware.

Keywords: blockchain, consensus, distributed systems, digital rights management, traitor detection

1 Introduction

The licensing of assets is an important part of markets such as software, music, and entertainment, which are largely based on controlling digital media for monetization. The set of tools and techniques aimed at ensuring licensing and control over digital assets is called Digital Rights Management (DRM).

The objective of DRM policies is to ensure that assets do not lose their market value, which is achieved through the imposition of strict rules on the distribution and licensing of an asset. However, these mechanisms do not completely prevent the illegal sharing of assets. As a consequence, Traitor Detection (TD) mechanisms become essential, allowing the tracking of the origin of an unauthorized copy of an asset.

In this context, entertainment-related markets have shown growth over the past few years, influenced by various factors, both cultural and technological. The growth of these markets contributes to the expansion of the global DRM market, which in 2024 was valued at $4.96 billion, with forecasts showing an expected compound annual growth rate of 11.51% until 2029, when its value is projected to reach $8.55 billion (Mordor Intelligence [2024]).

Furthermore, there are projects with social impact that could leverage DRM mechanisms to facilitate the exchange and dissemination of information. One example is the Amazon Biobank (Kimura and Simplicio Junior [2023]), in which communities from high-biodiversity regions can contribute to research and industry by collecting and providing information on the genetic material of their regions through a blockchain-based system.

Given this, the need and demand for efficient DRM solutions become evident. As will be presented in Section 2, although there are already solutions available in the market, recent cryptographic techniques and distributed system mechanisms can be employed to develop new solutions. In particular, the use of blockchain for DRM is attractive as it enables an immutable and distributed record of asset transfers and licensing, which in turn brings transparency to content management and distribution.

However, the development of blockchain-based solutions that truly leverage the available benefits of decentralized systems poses a set of challenges to research in the area. In the face of this, as seen in Aldweesh [2024], previous works have developed techniques to deal with aspects of DRM. Some of these aspects include **content distribution and authentication**, **monetization**, **transparency and decentralization**. Other aspects found in Mishra *et al*. [2024] include **licensing** and **flexibility** (which is related to the simplicity and portability of the solution in terms of payments and content download).

Each of these aspects composes the whole spectrum needed for a robust DRM solution, but combining them introduces technical difficulties mainly related to security, which have led, in some cases, to the preference for consortium or private blockchains (Ciriello *et al*. [2023]; De León and Gupta [2017]), which do not take full advantage of the **transparency and decentralization of public blockchains**.

Other difficulties are related to the challenge of combining payments and content distribution to provide assurances to digital asset owners and consumers. The payments should be confirmed only if the correct assets are provided, which introduces the need for smart execution of currency transfers. Even though this is achievable through smart contracts, the execution fees of complex smart contracts are too high to make it feasible. Hence, it is important to develop content distribution mechanisms that are robust but also possible to embed into smart contracts with low execution cost, i.e., simple implementation and execution.

Besides that, some of the techniques previously proposed for content distribution, such as those based on watermarking,

rely on media-specific properties or focus too heavily on specific industries, which restricts the use cases for these solutions. A more flexible solution is needed, and this can be achieved through efficient cryptographic mechanisms.

1.1 Objectives

Thus, this work aims to propose a truly decentralized and media-agnostic solution for DRM that is capable of harmonizing content provisioning and monetization without compromising security properties and flexibility. These objectives may be summarized by the following points:

1. **Requirement analysis**: identify desired functional properties and security properties for DRM solutions, building a threat model that is simple and flexible to be adapted to multiple industries or use cases. This model will be used as the basis for the other parts of the work.
2. **Content distribution**: develop efficient media-agnostic techniques based on cryptography for verifiable distribution of digital content.
3. **Monetization**: propose payment methods to ensure safe monetization of the digital assets distributed through the solution. This requires the mechanisms used for verification of content distribution to be simple and efficient enough to be executed in smart contracts.
4. **Licensing and Traitor Detection**: propose a licensing mechanism that allows for verification of content access rights and also traitor detection and tracing. This mechanism should be flexible enough to be adapted for different use cases and, possibly, legal actions.
5. **Blockchain design**: propose a public and decentralized blockchain structure and consensus algorithms that allow for transparent verification and registry of content provisioning and payment proofs, ensuring safety for both digital content owners and consumers.

1.2 Outline

Section 2 will start by presenting and comparing similar work developed in the area, highlighting the advancements provided in this work. The text that follows Section 2 will treat the Objectives mentioned in Subsection 1.1, using these as a guide for each theme that needs to be covered for a full blockchain-based DRM solution.

The Section 3 will cover Objective 1 by identifying initial requirements for decentralized and public DRM solutions and developing a threat model, also taking into account functional requirements related to the industry's adoption of the solution.

Based on that, the work will proceed to approach Objective 2 in Section 4, which will develop a protocol for provable data provisioning using cryptographic mechanisms. This protocol will later be coupled and linked to payments in Section 5, which will develop techniques related to Objective 3 based on *smart contracts* to ensure atomicity between content provisioning verification and payment execution. Following that, Subsection 6.3 will treat the licensing mechanisms of Objective 4.

Finally, with these aspects covered, the text will discuss the blockchain design and consensus algorithms proposals in Section 6, covering Objective 5.

Sections 7 and 8 will show results obtained with experimentation until now, relate the proposals and results to the aforementioned objectives, and discuss possible lines for future research.

2 Related work

The idea of using blockchain for DRM has been discussed in many works. In fact, De León and Gupta [2017] lists the benefits of using blockchain technology for the music industry, which can be generalized, to some extent, for DRM systems. In this regard, it is pointed out that decentralization is a benefit of using this technology, as it eliminates single points of failure, enhancing system security. Additionally, transparency regarding pending transactions and payment settlements for the use of music assets is presented as an advantage, whether through micropayments or smart contracts. Thus, De León and Gupta [2017] contributes by identifying market opportunities for blockchain technology development and related challenges. In this work, we extend this analysis to identify functional and non-functional requirements for a robust DRM solution that can be adopted by possible markets.

In a similar line, but with a focus on more technical details, the article Ciriello *et al*. [2023] analyzes similar characteristics of DRM systems for the music industry and presents design principles for blockchain-based solutions. Transparency in payments is also seen as one of the major advantages of blockchain solutions, and the study revolves around discussions on how to process such payments efficiently. Therefore, it proposes design principles and valuable insights for developing solutions in this field.

However, in both these works, there is no use of public and permissionless blockchains due to the challenge of ensuring proper content provisioning and payment execution. This work addresses this challenge through cryptographic techniques and distributed systems to enable truly decentralized and secure solutions even in permissionless environments.

Another issue in both these works is that they focus on metadata distribution and do not address content distribution and consumption, which is a crucial aspect of DRM. In this regard, the present work aims to introduce verifiable download mechanisms that can be used for content distribution, in addition to metadata, in a way that aligns with payment execution.

Another aspect of these works is their focus on the music industry. In contrast, a study more directly applied to DRM in general is Ma *et al*. [2018], in which cryptographic techniques and digital watermarking are used to build a content distribution and verification system, not just metadata. The proposed model and mechanisms were designed with attention to many important aspects of DRM, and the work contributes relevant techniques based on elliptic curves. In a similar line, Thanh *et al*. [2024] mixes watermarking with decentralized file distribution to create a technique called "blockmarking", with interesting properties of data splitting and reconstruction. Xiao *et al*. [2023] also uses an advanced asymmetric watermarking technique to build a solution in which actors interact through smart contracts to distribute and license content without the need for intermediaries. Meanwhile, Siddique and

Related works	Content distribution	Monetization	Public & decentralized blockchain	Media-agnostic
Ma *et al*. [2018]	✓	×	×	×
De León and Gupta [2017]	×	✓	×	×
Ciriello *et al*. [2023]	×	✓	×	×
Xiao *et al*. [2023]	✓	✓	×	×
Thanh *et al*. [2024]	✓	×	✓	×
Siddique and Fatima [2022]	✓	×	✓	×
Shang and Yu [2023]	✓	×	Public + Consortium	×
Mishra *et al*. [2024]	✓	✓	×	✓
Yuan *et al*. [2024]	✓	×	✓	✓
This work	✓	✓	✓	✓

Table 1. Comparison of features in related work

Fatima [2022] uses image analysis and processing techniques to build similarity indices based on contrast and structure.

The issue with watermarking techniques is that these are often designed with a specific type of media in mind, making their adoption for other use cases, such as software and game distribution, harder or unfeasible at all. In contrast, this work aims to develop media-agnostic mechanisms, which also facilitate their adoption and adaptation to various markets.

Some works did not use watermarking and invested more heavily in a cryptographic approach, bringing more flexibility. In particular, it is possible to mention Mishra *et al*. [2024], in which a key distribution mechanism is used to allow the existence of multiple distributors of the same asset. The solution, however, relies on a trusted and centralized license server.

Other work with interesting techniques is Shang and Yu [2023], which uses a blockchain based on ChinaDRM to realize DRM in a decentralized manner. Yuan *et al*. [2024] also develops a technique based on zero-knowledge proofs to authenticate intellectual property owners. However, these works do not give sufficient attention to payments, which are also an important feature. In contrast, this work provides better harmonization between payments and downloads for a more robust DRM system and stronger guarantees by interoperating with smart contracts on well-known and trusted blockchains.

Therefore, by utilizing cryptographic techniques, this work proposes a public and transparent blockchain design with these properties. Table 1 shows an overview and comparison with other works in terms of media-agnostic content-distribution, monetization, and decentralization, and reaffirms a valuable contribution in conciliating different aspects of DRM in a single solution.

3 Requirement analysis - Actors and Threat Model

For the requirement analysis, it is important to identify the actors in a generic and cross-industry use case. This identification will also allow the construction of a threat model by working on the different ways that these actors may behave.

3.1 Actors and requirements

In this work, two main actors are considered: the **provider**, who is interested in granting access to the content of an asset in exchange for payment; and the **client**, who is interested in purchasing the rights to access and use an asset provided by a provider.

The interaction between them will take place in a public environment, where third parties may witness the process and verify the proofs issued to confirm the correct participation of these actors in the protocols. We then have two categories of proofs: payment confirmations made by clients in **payment proofs**; and confirmations that the providers delivered the assets to the clients with **download proofs**.

The provider will also issue a **license** that is *irrevocable* and linked to the client, along with the delivery of the file. The client will use it as proof of the right to use the asset. If the license is shared illegally, it should be possible to trace it back to the original client, revealing the **traitor**. This corresponds to the Traitor Detection (TD) mechanism.

Note that the **Confidentiality of the Assets** is important for both the providers and the clients, who want the asset they are paying for to retain its value in the future. Thus, although it is not possible to completely prevent the illegal sharing of assets, it is important that, while clients act correctly, the content of the assets remains secure and is not disclosed by the protocols developed in this work.

Furthermore, it is important to ensure that the protocols and mechanisms developed guarantee decentralization and fair competition, so that no central and trusted entity is required, and no provider can gain an advantage over their competition through network manipulation.

3.2 Threat Model

The initial immediate threats derived from the presented actors consist of the client or the provider attempting to gain advantages without fulfilling their obligations in the protocol, in which case they will be referred to as **ghost client** or **ghost provider**, respectively. There must be mechanisms to prevent or punish these actions using the public environment of the blockchain.

Another threat mentioned earlier is the **traitor**, a client who shares assets illegally, against whom we want to have TD mechanisms. Additionally, a provider could try to manipulate the blockchain to prevent transactions from their commercial

competitors from being validated, which we will call a **dishonest provider**, and a consensus mechanism is needed to prevent these actions.

Finally, we will consider verifiers under the **honest-but-curious (HBC)** model (Paverd *et al*. [2014]), meaning they execute the payment and download verifications correctly but attempt to extract as much information as possible from each execution of the protocol. This model is chosen because the design of the blockchain will include consensus mechanisms that ensure **Byzantine Fault Tolerance (BFT)**, meaning an attacker would need to gain control of a significant portion of some network resource to manipulate the verification effectively in a public environment (Xu *et al*. [2023]). Furthermore, the existence of HBC verifiers means that we need a **zero-knowledge content distribution protocol** that does not reveal any new information about the content of the assets from the download proofs. Considering that the verifiers will actually mediate an exchange of download for payment, they will also sometimes be referred to as **mediators**.

Given the different threats and requirements provided, it may be noticed that the solution may be split into two main parts: **data provisioning** (verifiable content distribution) and **payment verification** (verifiable monetization of digital content). The first part is used mainly to prevent the *ghost provider*, who will have to falsify a proof of download to succeed, and also to deal with the HBC verifiers with a zero-knowledge scheme. The second part is more concerned with the *ghost client*, which will have to falsify a proof of payment to succeed. These threats are treated in the following sections.

The threat of the *traitor*, who acts after download provisioning and payment, will be handled by the licensing system, and this will be done in Subsection 6.3.

After that, the *blockchain* mechanisms will be presented considering the protocols proposed. The main concern in this case will be related to manipulations of the blockchain history and block forgery, which are directly related to the correct execution of proof verification, and the prevention against *dishonest providers*.

These threats are briefly summarized in Table 2.

4 Content distribution - Provable Data Provisioning (PDPr) protocol

The content distribution protocol must allow the issuance of a verifiable download proof by the provider. This protocol must have the property of zero-knowledge, meaning it should not reveal any new information about the asset in each execution, in order to prevent asset disclosure by HBC verifiers.

This will be called the *Provable Data Provisioning* (PDPr) protocol, as it is similar to *Provable Data Possession* (PDP) (Walker *et al*. [2022]), except that the execution of the protocol issues a type of *receipt* used by the provider to prove, to the verifier, that authentic data was provisioned to the client, instead of proving that they possess some data.

4.1 Objectives for the PDPr protocol

First of all, it is important to state what are the objectives for the PDPr protocol, and this will be done now with a generic version of the protocol. Let $\Pi = (E, Dec, Gen)$ be an encryption scheme and H a *one-way* function (for example, a cryptographic *hash*) such that,

$$E_K(H(F)) = H(E_K(F)) \tag{1}$$

In this case, the pair (Π, H) can be used to build a protocol for safe and verifiable content distribution with the following phases.

- **Setup**: before the execution of the protocol, the provider announces that it has an asset with identifier $H(F)$
- **Execution**: the provider generates $K \leftarrow Gen$ **uniquely** and provides $E_K(F)$ to the client. The client returns the signed **receipt** $r = H(E_K(F))$.
- **Verification**: the provider provides (K, r) to the mediator, who can use equation 1 to verify

$$Dec_K(H(E_K(F))) = Dec_K(E_K(H(F))) = H(F)$$

- **Recovery**: the mediator provides K to the client, who can decrypt $Dec_K(E_K(F)) = F$

It is clear that from the Setup phase alone, no party can recover the value F, as H is assumed one-way. So it does not break the confidentiality of assets.

As for the Execution phase, assuming that the encryption system remains secure under **uniquely** generated keys, then all that could be leaked is the encryption $E_K(F)$, from which no adversary would be able to derive significant information about F, also keeping asset confidentiality.

However, one may notice that for the Verification and Recovery phases, the mediator has a very important role in the protocol, which also gives it the power to act maliciously. In this sense, during the Verification, the mediator could choose not to verify the receipt properly, or could choose not to provide the key to the provider to allow the Recovery phase to happen correctly. These issues are accepted in the PDPr design, but will be mitigated with incentives and fault tolerance mechanisms in the blockchain design in Section 6.

Also, it is important to note that the client may share the asset contents after the successful execution of the protocol. However, as mentioned before, the idea here is not to completely prevent this sharing, but to provide mechanisms to detect and trace the origins of an illegal copy of the asset. This will be treated later in Subsection 6.3.

But, assuming that all parties behave correctly in the protocol execution, the message received by the mediator in the Verification phase is a signature of the value $H(E_K(F))$. This message could be simulated, for a fictional client, by the mediator with the following steps:

1. Generates a pair of signing keys for the fictitious client.
2. Generates a simulation key $\mathcal{K} \leftarrow Gen$
3. To generate the receipt that would be signed by the client if the provider used this key $\mathcal{K}$, the mediator computes

$$E_{\mathcal{K}}(H(F)) = H(E_{\mathcal{K}}(F))$$

4. The mediator signs the result with the signing key, obtaining $Sign(H(E_{\mathcal{K}}(F)))$, which would be exactly the received proof.

Actor	Action	Related mechanisms
Ghost provider	Tries to receive a payment without providing the correct asset	Verifiable content distribution (Section 4)
Ghost client	Tries to obtain a file without paying for it	Payment mechanisms (Section 5)
Traitor	Obtains a file but then tries to share it illegally	Traitor detection (Subsection 6.3)
HBC verifiers	Tries to obtain information about the confidential assets	Verifiable content distribution (Section 4)
Dishonest provider	Tries to manipulate the network to obtain advantages	BFT (Section 6)

Table 2. Summarization of actors in the threat model

This means that, in fact, for each execution of the protocol, there is no information revealed to the mediator that it could not already simulate, except for the client signature, which is replaced with a fictional client. So each execution of the protocol does not contribute significantly for an adversary acting as the mediator to recover the contents of F. This composes the desired *zero-knowledge* property (Menezes *et al*. [2018]).

Hence, this model captures the properties, assumptions, and adversary abilities that are used as a basis to develop the PDPr protocol implementations. To implement the protocol, it is enough to find real algorithms that implement the pair (Π, H) satisfying the property in equation 1. The following Subsections will discuss multiple possible implementations that were considered. In practice, instead of having exactly the property from equation 1, an equivalent evaluation with homomorphisms will be used for each protocol.

Future work will formalize this generic protocol using an Ideal Functionality in the Universal Composability (UC) framework Canetti [2001]; Canetti *et al*. [2015].

4.2 PDPr with Homomorphic Encryption

Fully Homomorphic Encryption (FHE) (Acar *et al*. [2018]; Gentry [2009]) has been the foundation for several new advances in the cryptology community. Among these, the possibility of evaluating secure hash algorithms over encrypted data has been explored (Bendoukha *et al*. [2022]), and PDPr could be a potential application. In summary, the idea is to obtain a pair composed by an encryption algorithm E and a cryptographic hash function H such that, omitting and simplifying the details of homomorphic evaluation, it satisfies the property

$$H(E_K(F)) = E_K(H(F)) \tag{2}$$

Or an equivalent way to perform this same verification using available operations. Suppose the provider and the client have a reliable FHE scheme and a cryptographic hash algorithm that can be evaluated under this FHE scheme. The protocol, illustrated in Figure 1, would go as follows:

1. The provider generates a unique key for the FHE and uses it to encrypt the data. Then, it sends the encrypted data to the client.
2. The client homomorphically evaluates the hash on the encrypted data, signs the result, and sends this signature back to the provider.
3. The provider verifies the received signature and then publishes it to the mediator along with the key.

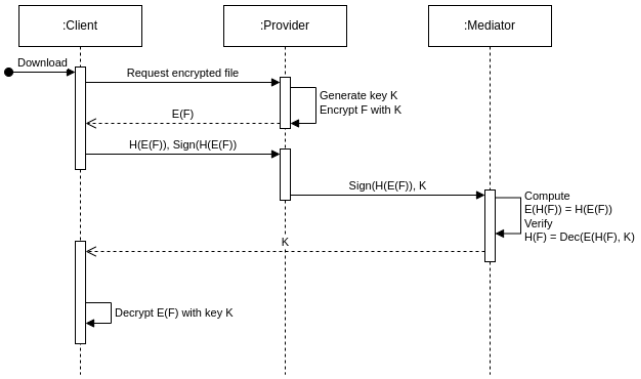

Figure 1. Proposal of PDPr implementation with homomorphic encryption

4. The mediator checks the received hash by using the property of the equation 2 and decrypting the result with the key.
5. The client retrieves the key through the mediator and uses it to decrypt the previously received data.

Where it is assumed that the hash of the file, $H(F)$, is publicly known. This protocol takes advantage of the FHE's ability to perform operations on encrypted data without requiring decryption, thus ensuring that the hash can be evaluated without revealing the content of the original data.

Assuming that the FHE scheme and the encryption algorithm are secure, the client will not be able to retrieve the encrypted data during the evaluation of the hash. Additionally, assuming that the chosen FHE algorithm does not reveal computation history over data, the mediator will not be able to recover the original F from its hash $H(F)$, which helps to ensure the expected zero-knowledge property.

The disadvantage of this method is that, for large files, the evaluation process becomes unfeasible. To circumvent this limitation, we propose encrypting the file with a secure symmetric cipher, such as AES, and distributing the encrypted file publicly using a network like BitTorrent (Johnsen *et al*. [2005]). Thus, it is enough to apply the PDPr to the symmetric encryption key, which will be much smaller (for example, 256 bits for AES-256). We will call this process **asset minification**.

However, even after minification, the evaluation is still not efficient, as shown in recent research using current FHE schemes (Bendoukha *et al*. [2022]). For the use case considered here, the execution latency of these algorithms would be prohibitive. Therefore, a more efficient protocol is needed. It is important to mention, in any case, that with the advancement of FHE techniques, this could become a viable application in the future.

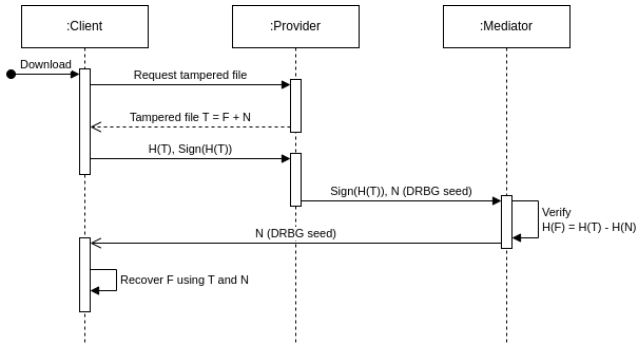

Figure 2. Proposal of PDPr implementation using homomorphic hash

4.3 PDPr with homomorphic hash - a first approach

The **homomorphic hash (HomHash)** is a cryptographic tool that allows the computation of the hash of a sequence of bytes after the insertion of a new block of bytes incrementally, that is, using only the previously known hash and the hash of the inserted block, without the need to recalculate the entire hash of the sequence. In this sense, the homomorphism occurs between the operation of concatenating bytes and the operation of combining hashes.

Some proposals that could be used to implement HomHash are provided in Bellare and Micciancio [1997], where a set of constructions is presented that make it possible to obtain incremental hash algorithms, including LtHash. Consider a file F and some data generated N inserted randomly into F, resulting in a new *tampered file* T. The construction of LtHash allows obtaining a HomHash with the property that

$$HomHash(T) = HomHash(F) + HomHash(N) \quad (3)$$

It is possible to note that if $HomHash(T)$, $HomHash(F)$, and N are known, the relationship between them can be verified by computing

$$HomHash(N) \stackrel{?}{=} HomHash(T) - HomHash(F) \quad (4)$$

Using this tool, a PDPr protocol can be constructed, as shown in Figure 2.

1. The provider generates a unique Deterministic Random Bit Generator (DRBG) seed.
2. The provider uses the seed to insert random byte sequences in random locations in the file and provides the tampered file to the client.
3. The client computes the HomHash of the received file, signs the result, and sends it back to the provider.
4. The provider sends the DRBG seed along with the client's signature to the mediator.
5. The mediator checks the relation between T, F, and N with the relation in equation 4.
6. The client retrieves the DRBG seed from the mediator and uses it to determine what should be removed from the tampered file to recover the original file.

This protocol is largely inspired by **Proof-of-Download (PoDl)** (Costa *et al*. [2022]). However, it has the disadvantage of requiring the mediator to compute all the *nonces* that were inserted into the file, which may demand more memory and computation in the case of larger files, making its use in smart contracts unfeasible due to the high fees it incurs. It also requires the provider to generate and distribute the tampered files to each interested client, increasing the workload.

There is also an attack that compromises the property of **Confidentiality of Assets** mentioned in Section 3. Consider an attacker A attempting to recover the original file from tampered copies without receiving the DRBG seed from the provider:

1. A requests N tampered versions of the file from the provider. A can use multiple blockchain addresses and IP addresses to avoid being detected as a single client.
2. A reads the copies and intersects their data blocks, removing the blocks that are not common.
3. A continues requesting tampered copies, removing more blocks in each iteration until a brute-force attempt to remove the remaining blocks and recover F becomes feasible.

With this attack, A can recover the original file without providing proof to the provider, possibly avoiding payment for the asset. This is due to the fact that the process proposed is not a safe encryption scheme, not even under unique keys, as mentioned in Subsection 4.1. Thus, this protocol is not suitable for the model considered in this work, and the development of alternative approaches is necessary.

Nevertheless, the protocol fits the threat model for PoDl and can be used in schemes where confidentiality is not essential, as it allows the execution of the download proof efficiently without requiring the transfer of entire copies of tampered files, but only their *hashes* and the DRBG seed.

4.4 PDPr through homomorphic hash - an improved approach

Consider **LtHash**, from Bellare and Micciancio [1997], as an implementation of HomHash. It is based on *lattices* to ensure security. In particular, it can be seen as a *one-way* function $LtHash : \mathbb{Z} \to \mathbb{Z}_p^n$, where p is the modulus used in LtHash. This algorithm can be leveraged as a subroutine that enables the construction of a pair of hash and an encryption scheme that interacts similarly to the previous protocol based on homomorphic encryption. Just for ease of identification in the following text, these algorithms will be named **GHash** and **GCrypt**, respectively.

4.4.1 Encryption Scheme - GCrypt

Let $\mathcal{M} = \mathcal{K} = \mathcal{C} = \mathbb{Z}_p^l$ be the message, key, and cipher spaces, respectively. Let also $GCrypt = (Enc, Dec, Gen)$ be an encryption scheme with $Enc : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ defined by

$$Enc(m, k) = m + k \quad (5)$$

where the addition is performed over the group obtained by the direct sum of $\mathbb{Z}_p$ groups. The decryption function $Dec : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$ is obtained by the inverse operation. If Gen selects unique keys from $\mathcal{K}$ with uniform probability,

then this scheme will have *perfect secrecy* (Shannon [1949]). Thus, it is the One-Time Pad (OTP) applied to the group $\mathbb{Z}_p^l$.

4.4.2 Hash - GHash

Consider LtHash as $LtHash : \mathbb{Z} \to \mathbb{Z}_p^n$ for the construction of a $GHash$ algorithm: $GHash : \mathbb{Z}_p^l \times \mathbb{Z}_p^n \to \mathbb{Z}_p^n$ defined by

$$GHash(m, m_\epsilon) = m_\epsilon + \sum_{i=1}^{l} m_i \times LtHash(i) \quad (6)$$

Where we use the entries of m as multipliers for the LtHash of their indices, taking the operation $\times$ as scalar multiplication in the vector space $\mathbb{Z}_p^n$. It is possible to note that the summation reduces to a matrix multiplication.

The value m_ϵ represents an error term added to the result of the sum. It is observed that the reversal of GHash without the knowledge of m_ϵ is assumed to be computationally difficult, as it also equates to decrypting an instance of OTP. Additionally, it can be stated that GHash exhibits a homomorphic property:

$$GHash(a + b, a_\epsilon + b_\epsilon) = GHash(a, a_\epsilon) + GHash(b, b_\epsilon) \quad (7)$$

By applying this algorithm to the GCrypt scheme presented earlier, it is possible to notice that, if GCrypt and GHash use the same moduli, then

$$GHash(c, c_\epsilon) = GHash(m, m_\epsilon) + GHash(k, k_\epsilon) \quad (8)$$

Where $c = Enc(m, k)$ and $c_\epsilon = m_\epsilon + k_\epsilon$. Clearly, GCrypt shares the same disadvantages as the *One-Time Pad (OTP)* scheme: the keys should never be used more than once, and the key must have the same size as the message to be encrypted. To overcome these limitations, it is proposed to use the *asset minimization* mentioned earlier. The symmetric key needs to be encoded into the message space $\mathcal{M}$, which can be done by simply encoding bits into odd or even numbers uniformly chosen from $\mathbb{Z}_p$. Despite having a high expansion factor, when applied to a small 256-bit key, it will still be feasible to transfer it between the provider and the client.

With this, as illustrated in Figure 3, it is possible to obtain a PDPr protocol as follows:

1. The provider symmetrically encrypts the asset with a key and publishes the encrypted asset along with the GHash of the encoded key, $GHash(m, m_\epsilon)$, for which they choose a random integer r_m and use $m_\epsilon = LtHash(r_m)$ as the error, **keeping it secret**.
2. The provider generates another unique encryption key k with a DRBG seed k_{seed} (which should be short for propagation in blockchain) and another error $k_\epsilon = LtHash(r_k)$. They compute $c = Enc(m, k)$ and $c_\epsilon = k_\epsilon + m_\epsilon$. Then, the provider sends (c, c_ϵ) to the client.
3. The client computes and then signs the GHash of the received ciphertext, using c_ϵ as the error.

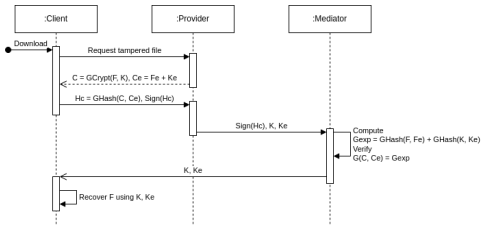

Figure 3. Proposal of PDPr implementation based on homomorphic hash with an improved approach

4. The provider publishes the DRBG seed k_{seed} and k_ϵ along with the signature as proof of provision to the mediator, who uses property in equation 8 to check.
5. The client obtains the DRBG seed from the mediator and uses it to generate k and recover $m = Dec(c, k)$.

The client cannot decrypt the ciphertext without the key obtained from the mediator, as this would be equivalent to breaking a scheme with perfect secrecy. Moreover, while the mediator can check that a certain key was used to encrypt the message, they cannot recover the content of the ciphertext from its GHash, nor recover the error m_ϵ from the information received in the proof, preventing the recovery of the message m.

However, GHash has a weak *binding* property. Thus, to use GHash and GCrypt, it would be necessary to add mechanisms to the blockchain that allow the client to prove that a provider did not act as they should, in case this occurs, and there should be ways to punish this malicious provider. To circumvent this deficiency, a protocol based on *homomorphic hidings* is presented.

4.5 PDPr through Homomorphic Hiding

Homomorphic Hidings (HH) are a cryptographic primitive that has been used in the design of *zero-knowledge proof* mechanisms (Ballesteros Rodríguez [2020]). The idea is to obtain a deterministic *one-way* function $HH : G \to G_{HH}$, where G and G_{HH} are groups, such that if $a, b \in G$, then $HH(a + b) = HH(a) * HH(b)$, where $+$ and $*$ are the operations of groups G and G_{HH}, respectively.

One way to obtain a function of this form would be to use the discrete logarithm. In this case, we have $HH : \mathbb{Z} \to (\mathbb{Z}/\mathbb{Z}_p)^\times$, where p is a large prime number to make the discrete logarithm difficult. A $g \in (\mathbb{Z}/\mathbb{Z}_p)^\times$ is chosen as the generator of the multiplicative group, and we define $HH(x) = g^x$. Note that, for any $x, y \in \mathbb{Z}$,

$$H(x + y) = g^{x+y} = g^x * g^y = H(x) * H(y) \quad (9)$$

It is then possible to use the *One-Time Pad* (OTP) to encrypt a given $F \in \mathbb{Z}$: $E(F, k) = F + k$, where $k \in \mathbb{Z}$ is the key used. When combining the algorithms, it is observed that

$$HH(E(F, k)) = HH(F + k) = HH(F) * HH(k) \quad (10)$$

In this way, it is possible to construct a new proposal for implementing the PDPr protocol, illustrated in Figure 4.

1. The provider encodes the asset F as an integer and encrypts it using OTP, where the key k will also be an integer and will be the download key.

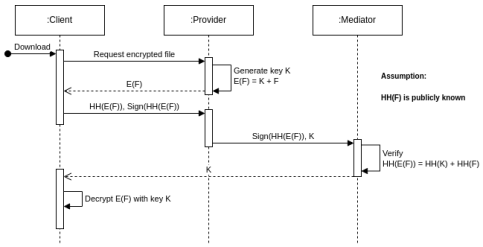

Figure 4. Proposal for implementing the PDPr protocol using Homomorphic Hiding

2. The provider sends the encrypted asset $c = E(F, k)$ to the client.
3. The client computes and signs the value $r = HH(c)$, which will act as the receipt. This value is sent back to the provider.
4. The provider provides (r, k) to the mediator, who can verify the provision through equation 10.
5. The mediator, after verifying that r is correctly signed by the client and that it is correctly related to k, provides k to the client.
6. The client uses k to compute $F = c - k$ and decrypt the asset.

In order to operate with larger assets, the minification process must be performed. Additionally, the value $HH(F)$ must be publicly known in advance. However, it is noted that this protocol has more robust properties. The encryption algorithm is, in fact, safe under unique keys and HH is a one-way function, as mentioned in Subsection 4.1.

Furthermore, the zero-knowledge property is asserted because the protocol can be simulated from the mediator's perspective (Menezes *et al*. [2018]). To perform an emulation without knowing the asset's content F, the mediator generates a $k \in \mathbb{Z}$ and takes the publicly known value $F_H = HH(F)$, computing the value $c_H = F_H * HH(k)$. Note that, in this case, $c_H = HH(c)$, where $c = F + k$. However, the mediator does not need to know either the value of c or the value of F, yet can still emulate the interaction between the provider and any client using the generated values c_H and k. It suffices to use c_H and k as the content that would be received as proof of download from the provider. Thus, each execution of the protocol does not reveal any information that could not already be simulated by the mediator.

From the mediator's perspective, the provider and client together are proving that the download occurred correctly. Thus, the client's signature on the value $H(c)$ ensures the *soundness* property. In other words, this signature proves that the client now knows a value c such that, if they come to know k, they could use it to obtain F, which proves the download. Finally, if the provider and client are honest, it is clear that the process will be successfully completed with high probability, proving the *completeness* property and that the protocol is a *zero-knowledge proof*.

However, it is important to note that a *Sybil* attack cannot be detected, meaning a scenario where the provider and the client are controlled by the same entity. This issue will be mitigated through blockchain mechanisms, as discussed in Section 6, with the payment of fees. Additionally, the protocol does not provide post-quantum resistance (Hhan *et al*. [2024]).

5 Monetization - Payment schemes

It is now necessary to consider payment verification, as monetization is also a crucial aspect of DRM. The text now presents interaction proposals using smart contracts to efficiently and effectively embed payment verification within the consensus mechanisms that will be proposed later.

5.1 Interoperating with other blockchain

Incorporating payments into a new blockchain is not an ideal design decision because (1) it requires users to trust a new currency solely for asset provisioning, and (2) it increases the coupling between payments and downloads, creating risks and necessitating a more complex consensus mechanism.

Therefore, it is proposed that a well-established blockchain, such as *Ethereum* (Buterin *et al*. [2014]), be used for payments, while downloads can take place on a separate blockchain designed as a ***sidechain* with *Simple Payment Verification* (SPV)** (Wang [2021]; Belchior *et al*. [2021]). This means that the blockchain will monitor other existing popular cryptocurrencies to verify payments or smart contracts.

SPV verification works by utilizing the Merkle tree of a block. To verify a transaction, the client—who, in this case, is the payer—provides a reference to the block containing the transaction and the Merkle tree path from their transaction to the root. This allows a verifier to confirm a given transaction using only the block headers.

Although implementing interoperability is required, this approach enhances the overall adoption of the solution within the community and provides greater flexibility. We will refer to the blockchain used for payments as the **payment chain** and the blockchain used for downloads as the **download chain**.

The payment method and the timing of its verification influence the behavior and implementation of the blockchain. Therefore, it is necessary to propose interaction models that ensure security for both the client and the provider. Given this, we now introduce proposals for **payment schemes**.

A payment scheme defines a method for executing payments while ensuring, when possible, that the exchange between payment and download is guaranteed. When this is not feasible, the scheme may specify a way to penalize malicious participants. Additionally, the payment scheme establishes the way by which fees are paid to the download chain forgers.

5.2 Pay-then-Download Scheme

This first proposal requires the client to first pay for the asset, present proof of payment, and only then proceed with the download. The network, acting as a mediator, will verify the payment proof, and upon confirmation, the provider will execute the download and issue a proof of download. The confirmation of the payment can be considered as the inclusion of the payment proof in a block.

If a provider fails to provide proof of download within n blocks after the payment verification, they will be considered a *ghost provider* and will be penalized. The penalization will be explored in greater depth later, within the discussion on consensus mechanisms in Section 6.

The fees must be included as transfers to the fee recipients' addresses within the transaction made by the client to the provider. As will also be detailed later, the fee recipient will be the forger of a previous block in the blockchain. The mediator must verify that this fee has been correctly paid to accept the proof of payment.

This method is simple and does not require smart contracts, which allows for lower execution fees on the payment chain and greater flexibility in choosing which payment chain to use. Additionally, it is highly efficient in verifying transactions. However, it requires trust in the provider, as they may act maliciously, and the punishment mechanisms available in the download chain are quite limited, with no way to reverse the payment.

5.3 Lock-then-prove scheme

To reduce the risk of a payment being made by the client without the provision of the download, one idea is to use a smart contract to ensure that the payment will only occur after the presentation of a valid proof of download. To achieve this, we introduce the concept of *zero-knowledge time-locked contracts*.

5.3.1 Zero-knowledge time locked contracts

The **Hash Time Locked Contracts** (HTLC) technique (Bitcoin Wiki [2021]) is used to implement interoperability between blockchains. In HTLC, the assets to be exchanged between blockchains are locked through smart contracts that require the revelation of a secret to unlock them within a time limit. By modifying the secret revelation process, a similar protocol can be constructed to exchange resources for confidential information while still allowing public verification and the execution of the smart contract.

Suppose Bob wants to obtain confidential information K from Alice in exchange for cryptocurrency. Alice reveals some information to Bob, P, from which Bob cannot directly derive K. Bob uses this information to derive a proof P_{proof}, which does not allow the recovery of P and encodes it in a smart contract requesting the revelation of some other information, R, from which he can construct the information K, and which can be verified using P_{proof}. Bob notifies Alice about this contract, and Alice reveals R publicly. With *zero-knowledge proofs*, the network can verify that Alice revealed K to Bob using P_{proof} and R, without actually gaining knowledge of K.

What makes this problem difficult is that the smart contract must be based on information that is **publicly** verifiable, but the information revealed, unlike the HTLC, is **confidential**. This will be called a **Zero-knowledge time-lock contract** (ZKTLC).

5.3.2 Lock-then-prove using ZKTLC

The idea is to create a ZKTLC that works like the previously presented download verifiers, encoding the verification of the download proof in the smart contract using the same algorithm employed by the blockchain nodes during verification.

The protocol will begin with the provider supplying an encrypted or modified copy of the asset, depending on the PDPr protocol used, and then the client will create a smart contract that requires the provider to reveal the necessary information for recovering the original asset in order to unlock cryptocurrency within a time limit, using information derived from the received copy of the asset, which will be the signature provided by the client to the provider in the presented PDPr protocols.

In this new protocol, there is no need to trust the provider. If the provider is interested in selling their asset, they must properly participate in the ZKTLC. There is no need to reverse the payment or implement punishment mechanisms, and the client is guaranteed that their currency will only be unlocked if the correct download proof that allows asset recovery is revealed. Furthermore, by working exclusively with *zero-knowledge proofs*, Asset Confidentiality is maintained.

However, the major drawback of the *lock-then-prove* scheme is that implementing the PDPr protocols with the presented zero-knowledge methods is complex and relies on algorithms that are not natively implemented in the possible payment chains. This causes contract execution fees to increase very rapidly.

5.4 Lock-then-prove with external validation

To use the *lock-then-prove* scheme with lower transaction fees, the blockchain forgers can be leveraged to unlock the contracts instead of embedding the verification execution within the smart contract.

As will be discussed later, **proof-based** consensus mechanisms will be used in the download chain. Thus, it is sufficient to require that the forgers present their proofs to unlock the transaction, including the payment of the fee.

In the case of *Proof-of-Work*-based mechanisms, the computational power of the download chain forgers can be utilized to verify the *zero-knowledge proofs* and produce smart contracts that verify only the solutions to these proofs.

The client first downloads the modified or encrypted asset and creates a smart contract that requires $Min_{verifiers}$ different addresses to reveal their solutions to the cryptographic puzzle of the download chain. If at least $Min_{verifiers}/2 + 1$ verify that the proof is correct, then the transaction is unlocked. The forgers also receive their rewards automatically after this confirmation.

Now the smart contract is much more secure, and the forgers have greater certainty that they will receive their reward after validating a download. Additionally, this increases the likelihood that the transaction corresponding to the ZKTLC will be published in the next block.

6 Public Blockchain design - decentralized consensus mechanisms for DRM

The blockchain mechanisms and the interaction between nodes to ensure the correct execution of the mediator algorithm during exchanges are now discussed, with proposals for

consensus mechanisms for DRM. After presenting these proposals, the text will briefly discuss a mechanism for licensing and Traitor Detection.

Consensus mechanisms on a high level might be divided in two main categories: **proof-based** and **committee-based**. On proof-based mechanisms, a node is elected the forger of the next block of the blockchain by emitting a proof that they've been chosen. On the other side, committee-based mechanisms have blocks forged by a committee composed of multiple nodes that dictate which transactions are included in the next block, often by a voting process.

Given that committee-based approaches require the committee composition to be known in advance for forgery, they are often deployed in permissioned environments. Hence, in this work, the mechanisms proposed are proof-based.

One mechanism proposed will use the **Proof-of-Work** structure, in which the node is elected by being the first to solve a cryptographic puzzle, and the solution is used as proof of election. Another mechanism will be proposed using the **Proof-of-Stake** structure, in which a node is elected by holding a high quantity of some stake, such as cryptocurrency.

As mentioned in Xu *et al*. [2023], these mechanisms will be analyzed under the properties of **persistence** and **liveness**, which are related to security. Persistence is a property by which a transaction k-blocks deep in an honest node's history will remain in the same position permanently, meaning that the history is stable and does not change after k blocks. Liveness refers to the property that an honest node transaction will eventually appear k-blocks deep in any honest node's history, meaning that the blockchain is always operational, creating new blocks and appending honest transactions.

Also from Xu *et al*. [2023], other important properties are **throughput**, which refers to the rate at which transactions are included in new blocks, and **transaction confirmation latency**, which refers to how long it takes for a transaction included in a new block to be considered irreversible with overwhelming probability.

6.1 Threshold Proof-of-Work (TPoW)

For the *Proof-of-Work*-based mechanism, one proposal is to rely on Nakamoto's mechanism (Nakamoto [2008]), in which a node must solve a cryptographic challenge to gain the right to forge the next block. In this case, any entity that wants to participate in the consensus can become a blockchain node and take part in the search for solutions.

6.1.1 Transactions in TPoW

In this structure, the provider will issue download proof transactions to the blockchain, and the client will issue payment proof transactions. The blockchain forgers will then verify and associate the generated proofs to confirm that the PDPr process was correctly executed by both parties, and each confirmed mediation key will be published in the blockchain blocks, making it available to the corresponding client. In this way, while the mediator in PDPr can be corrupted to withhold the mediation key, the blockchain as a whole will act as the mediator, with Byzantine Fault Tolerance (BFT) ensuring that the correct verification is carried out.

6.1.2 Including multiple solutions in block forging

One issue that can be observed is that a provider might be interested in participating in the forging process to omit transactions from competitors and gain an advantage. It is important to make it more difficult for such a *dishonest provider* to become the sole forger of a block. Therefore, a modification is made to Nakamoto's mechanism to include multiple cryptographic challenge solutions in the forging process.

To implement this inclusion, a security parameter N will be considered, indicating how many solutions will compose a new block. The N solutions will function as a type of committee, with each solution voting for or against the inclusion of a specific transaction in the block. If the transaction is included in the solution's Merkle tree, the vote is in favor. A transaction should be included in the next block of an honest node if it is considered in the majority of the N received solutions, meaning it must appear in at least $N/2 + 1$ solutions, where the division is integer-based. Due to this characteristic, this mechanism is called *Threshold Proof-of-Work* (TPoW).

While a block is being forged, more than N solutions may be received. Supposing that R solutions are received, it is possible to generate

$$C = \binom{R}{N} = \frac{R!}{(R-N)!N!} \tag{11}$$

different blocks from the solutions, because it means combining N solutions out of R without a specific order. These different possible blocks form a type of fork in the chain. We will call this a ***head fork***, as it occurs during the generation of the next blockchain block. The higher the value of N, the longer it will take to gather all the solutions needed to form a block, but it will be easier to prevent head forks. Additionally, the number of chains C grows rapidly with R, and assuming an asynchronous network model, a new solution might arrive after k blocks have already been produced. Because of this, there must be a rule to choose an authoritative chain among the C chains in a way that aligns with the blockchain's objectives while also ensuring persistence.

6.1.3 Reputation of provider nodes

In order to define an efficient rule for resolving forks, taking inspiration from the *Proof-of-Download* (Costa *et al*. [2022]) download count, a **provider node reputation** metric was introduced.

This metric is dynamically calculated based on the number of downloads completed by a provider in each block. Nodes with a high number of confirmed downloads in recent blocks have their reputation increased. This follows the formula below, which has the property of "memory loss":

$$R(p, b) = \sum_{i=0}^{W} Dl(b - i, p) * 2^{-i} \tag{12}$$

Where p is the provider, b is the index of the considered block, Dl counts the number of downloads of p in the indexed block, and W is a blockchain parameter. The larger W is, the longer it will take for the reputation of a provider node that stops operating to decrease.

Additionally, reputation is also used to penalize a provider node that fails to deliver a download correctly. If a client provides proof of payment and the provider does not supply a valid download proof after W blocks, its reputation will be permanently set to 0. This will indicate to other clients that the provider is unreliable, helping to prevent future attacks of the same type. To reflect this, the formula can be updated as follows:

$$R(p,b) = T(p) * \sum_{i=0}^{W} Dl(b-i,p) * 2^{-i} \tag{13}$$

Where T is a *trustness* function:

$$T(p,b) = \begin{cases} 1, & \text{if all downloads before block } b-W \\ & \text{were confirmed} \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

6.1.4 Fork resolution rule

With the reputation metric defined, the resolution of forks is carried out by considering two rules. First, the *Longest Chain Rule* (LCR) is applied, where the longest chain is chosen. This favors the persistence and liveness properties of the blockchain by ensuring that when a block is forged and other blocks are forged from it, even if new solutions are provided for this block, they will be ignored.

This discourages nodes from generating new solutions for a block after receiving enough solutions and reduces the chance of new head forks, which are harder to resolve. Without this, there would never be certainty that a particular block could not change, as in the asynchronous model considered, new solutions could arrive much later, which would directly harm persistence and also affect liveness by preventing the blockchain from advancing consistently.

Note that this does not resolve head forks, which occur when a fork happens due to the arrival of an additional cryptographic challenge solution for the block currently being forged. Therefore, after applying the LCR, in the case of a tie, the chain with the highest **popularity aggregation** is considered. The idea is to calculate, for each block of each chain, the accumulated reputation of the block:

$$R_{acc}(b) = \sum_{t \in T(b)} R(P(t), b) \tag{15}$$

Where b is the index of the considered block, T is the set of transactions in the block, and P is the provider of a transaction. The chain with the highest accumulated reputation sum for each block is the authoritative one. In the case of a head fork, this rule will ensure that one of the $C = \binom{R}{N}$ chains becomes the authoritative one.

6.1.5 Rewarding the forgers

Unlike cryptocurrency blockchains, the download chain does not have an asset with financial value. Therefore, it is necessary for the reward mechanism to incentivize the participation of forgers by receiving some type of payment within the payment chain.

Thus, it is proposed that after W blocks, where W is a parameter already mentioned, all transactions in the block can only be accepted if they include the payment of a fee to the N forgers of the current block.

It is important to note that W is a parameter that should be chosen based on transaction confirmation latency, to avoid rewarding nodes that later cease to be the forgers of the authoritative chain.

6.1.6 Security analysis

It has already been mentioned that the persistence property is preserved through the fork resolution rules. As for the liveness property, it is noted that any node can participate in the process, which does not require specific nodes to be online. Additionally, nodes have a clear incentive to participate in the forging process.

Although it is not possible to completely avoid Sybil attacks, the requirement for a payment fee as a reward to previous forgers makes this type of attack difficult. A provider could try to fake downloads to increase its reputation, but this would incur a financial cost, as fees would need to be paid for each verified transaction. Similarly, a Sybil attack attempting to manipulate the forging process is not trivial, as adding one more node does not increase the rate of cryptographic challenge solutions because it does not increase computational resources and does not facilitate the search for cryptographic challenge solutions.

Furthermore, to carry out an effective attack on the network and manipulate the block history, the attacker would need to subvert the block forging process by generating longer chains more quickly than the honest nodes, which would mean controlling the majority of the network's computational resources. In this way, the attack would not be profitable.

6.1.7 TPoW *Trade-offs*

There are some advantages to using TPoW. First, the considered network model is asynchronous, which is closer to what happens in practice. Additionally, the algorithm includes a reputation metric that is useful as a protection mechanism for clients. Finally, the algorithm ensures that even if a provider controls a large part of the network's computational power, other nodes that also have computational power will have the opportunity to participate and vote in favor of transactions that are in their interest, promoting fair competition.

However, there are also disadvantages. It is worth mentioning that the expected transaction confirmation latency is quite high, as there are different types of forks, and head forks generate multiple chains, which increases the chance that a transaction will not be included after a fork is resolved. Another considerable disadvantage is the high energy consumption, as each solution requires computational resources, and the trend is for these resources to become increasingly higher.

Given this, an alternative algorithm based on *Proof-of-Stake* is proposed.

6.2 Proof-of-Download-Exchange (PoDlE)

In order to use algorithms based on *Proof-of-Stake*, there needs to be some form of *stake* that is useful for the goals of the blockchain. Since, in this work, it is important that mediators have incentives to correctly perform the verification of a mediation, it was decided to use the count of mediations by a certain mediator as the *stake*, which will be called the *download token*.

Each time a mediator facilitates an exchange, they make an effort to participate in the protocol along with the provider and the client. After mediating the exchange, the mediator can send a transaction to the blockchain in which they prove the mediation performed by indicating the signed download and payment proofs, and in return, they receive a download token. For this reason, this mechanism will be referred to here as *Proof-of-Download-Exchange* (PoDlE).

In this case, unlike TPoW, the parties interested in sending transactions to the blockchain are the mediators themselves, rather than the client and the provider. While this requires the client and the provider to interact directly with a single mediator, the advantage is that they can use the blockchain as a service, without needing to participate directly in the network protocols of the blockchain. This interaction can occur through simpler interfaces, which are easier to integrate with other systems.

In this sense, it is sufficient for the client and the provider to mutually determine a reliable mediator to be used and proceed with the mediation. The chosen mediator has an incentive to perform the mediation correctly in order to receive download tokens, and if a mediator fails to perform the mediation correctly, another mediator can be chosen. This choice of mediator can take advantage of the blockchain records to determine which mediators are more active and reliable, so that, in a way, the download tokens of a mediator can also serve as a measure of their reputation.

The download token can only be created (from mediation transactions) or destroyed, but it cannot be transferred between nodes. This ensures that all download tokens for a given node were earned through useful effort performed.

6.2.1 Block Forgery in Epochs

Block forgery will occur, similarly to other PoS-based blockchains, in epochs (Grandjean *et al*. [2024]). Each epoch will consist of several stages, and the product of an epoch is a block that is propagated through the network.

For the election of the forger, a mechanism similar to the one used in *Proof-of-Download* (Costa *et al*. [2022]) was chosen, where a random number is used to select, from a subset of nodes, which will be the forger. To prevent manipulation of the lottery result, the process uses a distributed *random beacon* algorithm to generate the random number for the lottery.

The stages of an epoch, along with their respective durations, are as follows:

1. ***Commitment*** – 6 seconds.
2. **Revelation** – 6 seconds.
3. **Proposal and Voting** – 24 seconds.
4. **Propagation** – 24 seconds.

Honest nodes must strictly adhere to the deadlines of each stage, rejecting messages that arrive late. As a time reference, the *Unix timestamp* reduced modulo 60 is used, which precisely defines the intervals for each epoch, ensuring that each epoch lasts exactly 1 minute.

During the **commitment** period, each node generates a random number and publishes its hash as a commitment. Along with the hash, the node specifies how many of its download tokens it is committing to the current forging process. This information must be signed and verified by other nodes, including the committed number of tokens.

Immediately afterward, in the **revelation** stage, nodes publish the actual values they used in the commitment, also signed. Each value is verified, and the N highest commitments in terms of committed tokens are selected to form a committee, where N is a predefined parameter.

Locally, the received numbers are combined using an XOR operation along with the hash of the previous block's header. The result is then reduced modulo T, where T is the total number of tokens, producing a lottery number. The tokens are then sorted in ascending order based on the number of tokens committed by each node, and the lottery number is used to select a token. The owner of the selected token becomes the forger of the next block.

The block proposal and voting process then begins. The forger must publish a block proposal to the other committee nodes, which will verify the received proposal. If the node fails to publish the proposal within the time limit, it will be punished by having its committed tokens destroyed.

Each node will then cast a vote either in favor of or against the block. If the block receives a **supermajority** approval of 2/3 of the votes, it is accepted. However, if the block receives a supermajority rejection of 2/3 of the votes, it is rejected. In other cases, the block is simply ignored. In the case of a rejected or ignored block, the transactions from the current block must be included in the next block.

In cases where the block is accepted or rejected, the tokens committed by the forger are destroyed, and the committee members receive a reward of 1/10 of the number of tokens they had committed if they voted correctly. If the block is accepted, despite having its tokens destroyed, the forger receives transaction fee rewards within a few blocks, as proposed for TPoW. If the block is ignored, none of the nodes receive or lose tokens.

The mechanism of destroying the forger's tokens ensures that other nodes have an opportunity to be elected in future forges, preventing the same forger from immediately forging another block. The committee's reward incentivizes proper participation in the forging process and genuine validation of block transactions. Additionally, the transaction fee reward system provides a way to convert download tokens, which cannot be transferred between nodes and therefore cannot be used as a currency.

6.2.2 Handling forks in the Chain

Analyzing the proposed mechanism, it is evident that the system assumes a synchronous network model. Messages must arrive within the specified time frames for each epoch stage to ensure proper forging.

However, in practice, there will be asynchronous nodes. One possible scenario is that a given node does not receive a commitment or revelation message in time, leading to a different perception of the elected forger compared to other nodes. Consequently, the node may propose a different block. In such cases, the proposed block should be ignored. Similar situations, where a specific node within the group is asynchronous, are handled in the same manner—by disregarding the messages, as would be done with an adversary.

In a more complex scenario, a group of nodes may communicate with another group with a latency higher than the required epoch stage durations, potentially leading to an entirely separate forging process within each group. This situation results in a genuine fork.

To handle forks, we propose using the sum of the tokens committed by the nodes that voted during the forging of each block as a metric. By aggregating this value for each chain, the chain with the highest total commitment should be chosen. This approach makes sense from a network contribution perspective, as nodes with a higher number of tokens are those that contribute the most effectively. Moreover, this selection mechanism aligns with the ideal scenario where messages do not experience delays, ensuring that the nodes with the highest committed tokens are the ones forging the block.

6.2.3 Security Analysis

The characteristic of persistence in PoDlE is ensured through voting. Each vote acts as an attestation in favor of a specific block, and the more *stakes* are committed, the more secure the selection of that block becomes in the event of a fork. This suggests that persistence in PoDlE can be more stable than in TPoW, potentially leading to lower transaction confirmation latency and improved scalability.

However, the property of liveness depends on at least N nodes being willing to participate in the forging process and ensuring network synchronization. These conditions are stricter than those in TPoW, which may present challenges in practice. Nonetheless, nodes have strong incentives to participate in the forging process and maintain the necessary network conditions to ensure successful operation.

Regarding Sybil attacks, a node could attempt to forge mediations to increase its number of download tokens and gain advantages in the forging process. However, similar to TPoW, each of these mediations incurs a financial cost. To accumulate enough tokens to manipulate the forging process, an attacker would need to pay fees equal to or greater than the amount they would receive as a reward for becoming the forger. This makes the attack unprofitable.

6.2.4 *Trade-offs* in PoDlE

As mentioned earlier, PoDlE offers several advantages over TPoW. In particular, its persistence and expected scalability are significantly better and more stable. Additionally, the mechanism is simpler, making it easier to implement and use in practice. Finally, for providers and clients, the process is greatly simplified, as they do not need to join the network to use the solution. This enables the development of more user-friendly interfaces, such as *web* interfaces and others, to facilitate adoption and ease of use.

However, there are also disadvantages. In particular, this approach requires a higher level of trust in the mediator. Additionally, as demonstrated, the nodes participating in the forging process must operate within a synchronous network model.

6.3 Licensing and Traitor Detection mechanism

After covering aspects related to downloads, payments, and consensus, it is now possible to discuss mechanisms for licensing and traitor detection, considering the previously presented protocols.

As mentioned in Section 3, a threat that is also considered is the *traitor*, who, after receiving rights to access and consume the contents of a certain digital asset, shares it illegally. While this cannot be totally prevented, since the traitor is operating directly with the asset's content, there are ways of detecting a leakage using a **licensing system**. In this work, it is proposed that, during the asset provisioning, when the provider sends to the mediator the key or seed for verification, it should also send a signature that is used as a **license** by the client to prove that they have access to the provisioned asset.

This signature is provided as a companion to the key in the PDPr protocol and must include the asset hash and the client public key. The verifier may verify the license as part of the mediation by knowing the public key of the provider and the client. This verification may also be embedded in a smart contract, in case the lock-then-prove scheme is used. Figure 5 shows how the PDPr protocol based on homomorphic hiding would be adapted to include license generation. Other PDPr protocols could be modified similarly to include this process.

In this way, when a copy of a certain asset is found, it is possible to authenticate the user of the copy. For instance, consider the case of a user trying to play a song encoded in a file that was provided through PDPr. The media player software may check the asset contents, compute its hash, and request a license from the user together with a challenge. This challenge, as usual with authentication protocols (Menezes *et al*. [2018]), is a unique nonce that should be signed by the user with the correct private key. The software may then verify the challenge against the public key contained in the license.

This proves that the user has the key that was used during PDPr, but is not enough to prove that they are the true owner of the copy, since the license and the private key could be shared illegally together with the assets.

Because of this, it is allowed for the signature to include some metadata, which is specified by the client to the mediator and the provider during mediation, and must be included by the provider. This metadata is used to add more information about the client, and could use additional methods of verification of its validity. Figure 6 illustrates the process of license validation together with metadata validation for authentication.

This opens the possibility for a wide range of identity verification mechanisms. As an example, a mechanism could use the metadata to bind the license to the login of a certain user

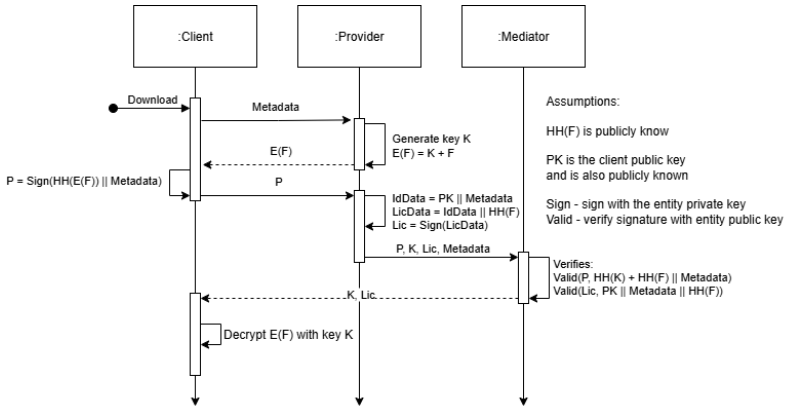

Figure 5. Example of license generation during PDPr with Homomorphic Hidings

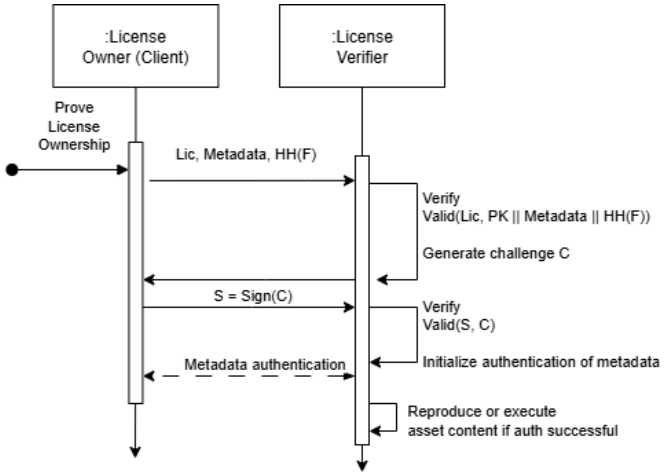

Figure 6. Example of license usage after generation

in a media player platform, using it later before playing the media to ensure a match of the license with the authenticated user.

In another example, the metadata chosen by the client could be a digital certificate with a trusted root such as ICP-Brasil (Guelfi [2021]), accompanied by a signature of the hash of some previous block of the blockchain, used for verification of ownership of the certificate. In this case, the license would be bound to the granted identity of a certain person or company recognized by Brazil's government, and could be used as proof in case of a judicialization.

This flexibility for identity verification design makes it possible to adapt the solution to different markets and use cases, being agnostic to the media type.

7 Results

Having presented the protocols and consensus mechanisms proposed, the text now provides results from experimentation. The experiments were mainly based on benchmarking the implementations of some of the PDPr proposals to verify the feasibility of usage in real-world scenarios with common hardware. While a great part of the PoDlE consensus has been implemented, it is not yet fully completed. Therefore, results for the consensus mechanisms will be included in future work, including tests of the traitor detection mechanism and block generation times.

7.1 Methodology and reproducibility

All results were gathered in this section using a 7th-generation Intel Core i7 processor with a 2.9GHz clock speed and a machine with 8GB of RAM, which corresponds to standard personal notebook hardware. The objective is to show that the algorithms proposed run in milliseconds or lower timescales using common hardware, demonstrating the feasibility of usage to build public and accessible environments for DRM.

For each function benchmarked, the test looped the execution of the corresponding function N times, where N is a parameter dynamically adjusted to ensure that the function runs long enough to stabilize the measurements obtained. The algorithm used to adjust this parameter is the default benchmarking algorithm from the Go Standard Library Go Official Documentation [2025].

The code for the PDPr implementations, the benchmarking tests, and instructions on how to reproduce the experiments can be found at `https://github.com/titosilva/pdpr-go`. The code is written in the Go language.

7.2 Benchmarks of LtHash

The LtHash algorithm uses a *randomize-then-combine* approach, in which the data to be hashed is split in blocks and each block is fed to a *randomize* function. The results for every block are then *combined*. In LtHash, the randomization function must output a vector of large integers, and the

combination is the addition of these integers modulo a public large integer.

The LtHash was implemented using the **BLAKE2b XOF** (Extendable Output Function) (Aumasson *et al*. [2013]) as the randomization function, dividing the message into blocks and applying the XOF to generate a fixed-size vector. As recommended in Bellare and Micciancio [1997], a vector of 500 entries of 128 bits was chosen to provide security while maintaining performance. Parallelization was not used because the data to be hashed in GHash was not large enough to see advantages.

Another important parameter for LtHash is the block size used. There is a trade-off in choosing the block size, as a smaller block will make the hash calculation slower, since the randomization function is used more times. However, a very large block loses incrementality, since it will only be possible to append large chunks of data.

In Table 3, the results for different parameters are presented. All cases use vectors of 500 entries, but with variable entry sizes.

File Size	Block Size	Vector Entry Size	ms/hash
1GB	1kB	256	108548
1GB	1kB	128	64202
1GB	4kB	256	26908
1GB	4kB	128	18770
1MB	1kB	256	116.7
1MB	1kB	128	72.2
1MB	4kB	256	30.4
1MB	4kB	128	19.2
1kB	512B	256	0.246
1kB	512B	128	0.187

Table 3. Benchmarks for LtHash

It is important, however, to emphasize that LtHash is only used as a subroutine for GHash in this work, and is not applied directly to the files, since the PDPr with LtHash was shown not to preserve asset confidentiality.

7.3 GHash and GCrypt Benchmarks

For GHash, the LtHash implementation was used as a subroutine, requiring only the addition of the block insertion operation with multiplication by a scalar using large modular integers. GHash uses the message bytes as blocks to be multiplied as scalars with the LtHash of each block's index. The benchmarks presented in Table 4 were obtained using a fixed-size vector of 500 for LtHash and other variable parameters. The values for 128 blocks and 256 blocks correspond to what would be used in PDPr for 128-bit and 256-bit messages, considering the expansion factor of GCrypt.

Table 4. Benchmark Results for GHash

Input (blocks)	Time (ms)
128	15.55
256	30.63

For GCrypt, the first step in encryption involves encoding a message by reading each bit and generating an odd or even number based on its value, using a non-deterministic pseudo-random number generator (PRNG). The second step is key expansion, which uses a variant of HASH-DRBG (Barker and Kelsey [2015]) based on iterating SHA-256 with the key as the seed. The expanded key is then added to the encoded message to obtain the ciphertext. The benchmarks for encryption and decryption are presented in Table 5.

Table 5. Benchmark Results for GCrypt

Input (bits)	Encryption (ms)	Decryption (ms)
128	1.130	0.580
256	2.280	1.152

7.4 Benchmarks of the Homomorphic Hiding implementation

The implementation of Homomorphic Hiding used a 1024-bit prime from the IKE specification (Harkins and Carrel [1998]) on the second Oakley group, with operations carried out using a library that provides big integer operations in constant time. The computation of the Homomorphic Hiding function involves a simple exponentiation, using 2 as the generator. The benchmarks for this operation are presented in Table 6.

Table 6. Benchmark Results for the Homomorphic Hiding Function

Input (bits)	Time (ms)
128	0.094
256	0.092

For encryption, which corresponds to the OTP (One-Time Pad), it is sufficient to add two large integers. Similarly, decryption is achieved by subtraction. Unlike GCrypt, there are no key or message expansion steps in this process. The benchmarks for these operations are presented in Table 7, with the values given in **nanoseconds**.

Table 7. Benchmark Results for OTP Used with Homomorphic Hiding

Input (bits)	Encryption (ns)	Decryption (ns)
128	823.7	843.6
256	815.0	848.0

7.5 Benchmarks of the PDPr Protocol with GHash and GCrypt

The PDPr protocol using GHash and GCrypt offers simplicity and security. In addition to the encryption and decryption of the asset, PDPr also utilizes proof generation, which corresponds to the calculation of GHash over the encrypted minified asset on the client, and proof verification, which the download verifiers perform. The benchmarks for these operations are presented in Table 8.

Table 8. Benchmarks of the PDPr protocol with GHash and GCrypt

Operation	Key size	Time (ms)
Proof generation	128	17.59
Proof generation	256	34.94
Proof verification	128	289.5
Proof verification	256	576.5

7.6 Benchmarks of the PDPr protocol with Homomorphic Hiding

Another possible implementation of the PDPr protocol uses Homomorphic Hiding. In this case, proof generation corresponds to the client taking the minified and OTP-encrypted asset and applying the Homomorphic Hiding function to it. Proof verification takes the encryption key, the result of the Homomorphic Hiding function applied to the minified asset, and the proof generated by the client, and verifies the mathematical relationship between these three. Since this process is more direct and does not require the computation of multiple *hashes*, its performance is expected to be superior to that of PDPr with GHash and GCrypt. The benchmark results for the performance of PDPr with Homomorphic Hiding are presented in Table 9.

Table 9. Benchmarks of the PDPr protocol with Homomorphic Hidings

Operation	Key size	Time (ms)
Proof generation	128	0.663
Proof generation	256	0.667
Proof verification	128	0.093
Proof verification	256	0.176

8 Conclusion

In relation to the objectives mentioned in Subsection 1.1, this work proposed various techniques to handle the most important aspects of DRM:

1. **Requirement analysis**: the work identified requirements and a threat model for a decentralized solution for DRM and TD. The model was developed through the analysis of real actors and covers the most relevant threats.
2. **Content distribution**: cryptography-based protocols for media-agnostic content distribution were proposed and implemented. The protocols were analyzed from the security and performance perspectives, with benchmarks provided. The benchmarks show that the protocols can be efficiently used and implemented in real-world cases without requiring high hardware standards.
3. **Monetization**: by using smart contract techniques, payment schemes were proposed to ensure payments are only executed if enough evidence of authentic content provisioning is given, providing more safety for content consumers.
4. **Licensing and Traitor Detection**: a licensing mechanism was proposed to allow authentication of digital rights and traitor detection and tracing under flexible types of identity bindings, making it possible for actions to be taken in case of illegal content sharing.
5. **Blockchain design**: consensus mechanisms were proposed to allow verification and history registration on a public and decentralized blockchain structure, ensuring safety for both digital content owners and consumers.

These aspects were satisfactorily combined to build a robust DRM and TD solution with demonstrated ability for decentralization and operation with Byzantine fault tolerance, as well as proven persistence and operational mechanisms for functioning in a public environment. Thus, it can be stated that the work achieved the expected objectives.

However, there are still advancements to be made and research opportunities. The complete implementation of the consensus algorithms is ongoing. Furthermore, the application to real-world environments involves additional investigations, such as royalty payments, metadata storage, stronger licensing mechanisms, and adaptation to specific business models (Ciriello *et al*. [2023]; De León and Gupta [2017]). Future work could also explore the applications of the protocols and mechanisms presented here to other contexts, such as the traceable distribution of sensitive data, such as medical or financial records.

Regarding cryptographic mechanisms, future research should find alternative solutions to the discrete logarithm used in Homorphic Hiding in order to provide post-quantum resistant solutions for the PDPr protocol. The recent advancements in quantum computing technologies have motivated NIST to consider several cryptographic techniques, including the use of Diffie-Hellman and elliptic curve Diffie-Hellman, as obsolete starting in 2030 (Moody *et al*. [2024]).

Other necessary advancements, which are already being made, involve the formalization of the presented protocols and techniques. In particular, it is necessary to analyze the PDPr protocol under a security *framework* such as *Universal Composability* (UC) (Canetti [2001]; Canetti *et al*. [2015]). This will clarify the intentions and scope of the proposed mechanisms. It is also necessary to improve the payment schemes to reduce the fees paid for the execution of contracts and transactions on the payment chain.

Declarations

Authors' Contributions

JG and FZ contributed to the conceptualization of this study. JG contributed with supervision. JS is the main contributor and writer of this manuscript. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The datasets (and/or softwares) generated and/or analysed during the current study are available in https://github.com/titosilva/pdpr-go.

References

Acar, A., Aksu, H., Uluagac, A. S., and Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35. DOI: 10.1145/3214303.

Aldweesh, A. (2024). The impact of blockchain on digital content distribution: a systematic review. *Wireless Networks*, 30(2):763–779. DOI: 10.1007/s11276-023-03524-0.

Aumasson, J.-P., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. (2013). Blake2: simpler, smaller, fast as md5. In *Applied Cryptography and Network Security: 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings 11*, pages 119–135. Springer. DOI: 10.1007/978-3-642-38980-1_8.

Ballesteros Rodríguez, A. (2020). zk-snarks analysis and implementation on ethereum. Available at:`https://www.semanticscholar.org/paper/zk-SNARKs-Analysis-and-Implementation-on-Ethereum/b1c72c598d9954362605f839f6b71a1a53013d17`.

Barker, E. and Kelsey, J. (2015). Nist special publication 800-90a revision 1: Recommendation for random number generation using deterministic random bit generators. *NIST, June 2025 SP*. DOI: 10.6028/NIST.SP.800-90Ar1.

Belchior, R., Vasconcelos, A., Guerreiro, S., and Correia, M. (2021). A survey on blockchain interoperability: Past, present, and future trends. *Acm Computing Surveys (CSUR)*, 54(8):1–41. DOI: 10.1145/3471140.

Bellare, M. and Micciancio, D. (1997). A new paradigm for collision-free hashing: Incrementality at reduced cost. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 163–192. Springer. DOI: 10.1007/3-540-69053-0_13.

Bendoukha, A. A., Stan, O., Sirdey, R., Quero, N., and Freitas, L. (2022). Practical homomorphic evaluation of block-cipher-based hash functions with applications. In *International Symposium on Foundations and Practice of Security*, pages 88–103. Springer. DOI: 10.1007/978-3-031-30122-3_6.

Bitcoin Wiki (2021). Hash Time Locked Contracts. Available at:`https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts` Accessed September 14, 2023.

Buterin, V. *et al*. (2014). A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1. Available at:`https://ethereum.org/whitepaper/`.

Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE. DOI: 10.1109/SFCS.2001.959888.

Canetti, R., Cohen, A., and Lindell, Y. (2015). A simpler variant of universally composable security for standard multiparty computation. In *Advances in Cryptology–CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II 35*, pages 3–22. Springer. DOI: 10.1007/978-3-662-48000-7_1.

Ciriello, R. F., Torbensen, A. C. G., Hansen, M. R. P., and Müller-Bloch, C. (2023). Blockchain-based digital rights management systems: Design principles for the music industry. *Electronic Markets*, 33(1):1–21. DOI: 10.1007/s12525-023-00628-5.

Costa, F. Z. D. N., De Queiroz, R. J., Bittencourt, G. P., and Teixeira, L. (2022). Distributed repository for software packages using blockchain. *IEEE Access*, 10:112502–112514. DOI: 10.1109/ACCESS.2022.3216569.

De León, I. L. and Gupta, R. (2017). The impact of digital innovation and blockchain on the music industry. *Inter-American Development Bank.(Nov 2017). Available online: https://publications. iadb. org/en/impact-digital-innovation-and-blockchain-music-industry (accessed on 23 June 2020)*. Available at:`https://publications.iadb.org/en/impact-digital-innovation-and-blockchain-music-industry`.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178. DOI: 10.1145/1536414.1536440.

Go Official Documentation (2025). Go testing package - benchmarks. Available at:`https://pkg.go.dev/testing#hdr-b_N_style_benchmarks` Accessed: 2025-05-03.

Grandjean, D., Heimbach, L., and Wattenhofer, R. (2024). Ethereum proof-of-stake consensus layer: Participation and decentralization. In *International Conference on Financial Cryptography and Data Security*, pages 253–280. Springer. DOI: 10.1007/978-3-031-69231-4_17.

Guelfi, A. R. (2021). *Análise de elementos jurídico-tecnológicos que compõem a assinatura digital certificada digitalmente pela Infra-estrutura da Chaves Públicas do Brasil (ICP-Brasil)*. PhD thesis, Universidade de São Paulo. Avaialble at:`https://repositorio.usp.br/item/001603527`.

Harkins, D. and Carrel, D. (1998). Rfc2409: The internet key exchange (ike). Available at:`https://www.rfc-editor.org/rfc/rfc2409.html`.

Hhan, M., Yamakawa, T., and Yun, A. (2024). Quantum complexity for discrete logarithms and related problems. In *Annual International Cryptology Conference*, pages 3–36. Springer. DOI: 10.1007/978-3-031-68391-6_1.

Johnsen, J. A., Karlsen, L. E., and Birkeland, S. S. (2005). Peer-to-peer networking with bittorrent. *Department of Telematics, NTNU*. Available at:`https://web.cs.ucla.edu/classes/cs217/05BitTorrent.pdf`.

Kimura, L. T. and Simplicio Junior, M. A. (2023). Amazon biobank: a blockchain-based genomic database for bioeconomyi. Master's thesis, Universidade de São Paulo. Available at:`https://repositorio.usp.br/item/003161088`.

Ma, Z., Jiang, M., Gao, H., and Wang, Z. (2018). Blockchain for digital rights management. *Future Generation Computer Systems*, 89:746–764. DOI: 10.1016/j.future.2018.07.029.

Menezes, A. J., Van Oorschot, P. C., and Vanstone, S. A. (2018). *Handbook of applied cryptography*. CRC press. DOI: 10.1201/9781439821916.

Mishra, A., Obaidat, M. S., and Mishra, D. (2024). Privacy

preserving content distribution framework for multidistributor drm systems. *Security and Privacy*, 7(1):e327. DOI: 10.1002/spy2.327.

Moody, D., Perlner, R., Regenscheid, A., Robinson, A., and Cooper, D. (2024). Transition to post-quantum cryptography standards. DOI: 10.6028/nist.ir.8547.ipd.

Mordor Intelligence (2024). Digital Rights Management Market Size and Share Analysis - Growth Trends and Forecasts (2025 - 2030). Available at:`https://www.mordorintelligence.com/industry-reports/digital-rights-management-drm-market` Accessed February 5, 2025.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*. Available at:`https://bitcoin.org/bitcoin.pdf`.

Paverd, A., Martin, A., and Brown, I. (2014). Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep*. Available at:`https://www.semanticscholar.org/paper/Modelling-and-Automatically-Analysing-Privacy-for-Paverd-Martin/92df4589efbbb2adb104f1346aab73e4d71627ef`.

Shang, W. and Yu, Z. (2023). A new media content trusted dissemination architecture based on av-blockchain and chinadrm. *Intelligent and Converged Networks*, 4(2):142–157. DOI: 10.23919/icn.2023.0015.

Shannon, C. E. (1949). Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715. DOI: 10.1002/j.1538-7305.1949.tb00928.x.

Siddique, S. S. and Fatima, N. S. (2022). Digital file rights management system using blockchain. *Procedia Computer Science*, 215:309–320. DOI: 10.1016/j.procs.2022.12.033.

Thanh, T. M. *et al*. (2024). A proposal of digital contents copyright protection by using blockmarking technique. *Computer Science*, 25(4). DOI: 10.7494/csci.2024.25.4.5377.

Walker, I., Hewage, C., and Jayal, A. (2022). Provable data possession (pdp) and proofs of retrievability (por) of current big user data. *SN Computer Science*, 3(1):83. DOI: 10.1007/s42979-021-00968-z.

Wang, G. (2021). Sok: Exploring blockchains interoperability. *Cryptology ePrint Archive*. DOI: 10.1145/3582882.

Xiao, X., Zhang, Y., Zhu, Y., Hu, P., and Cao, X. (2023). Fingerchain: Copyrighted multi-owner media sharing by introducing asymmetric fingerprinting into blockchain. *IEEE Transactions on Network and Service Management*, 20(3):2869–2885. DOI: 10.1109/tnsm.2023.3237685.

Xu, J., Wang, C., and Jia, X. (2023). A survey of blockchain consensus protocols. *ACM Computing Surveys*. DOI: 10.1145/3579845.

Yuan, S., Yang, W., Tian, X., and Tang, W. (2024). A blockchain-based privacy preserving intellectual property authentication method. *Symmetry*, 16(5):622. DOI: 10.3390/sym16050622.